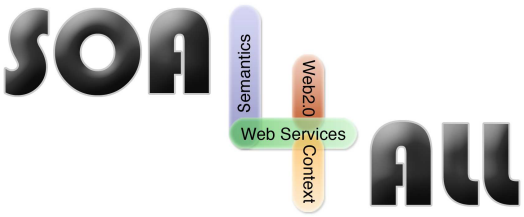

Project Number:	**215219**
Project Acronym:	**SOA4All**
Project Title:	**Service Oriented Architectures for All**
Instrument:	**Integrated Project**
Thematic Priority:	**Information and Communication Technologies**

D1.5.1 SOA4All Testbeds Specification And Methodology

Activity N:	Activity 1 – Fundamental and Integration Activities	
Work Package:	WP1 – SOA4All Runtime	
Due Date:		M12
Submission Date:		10/03/2009
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 Simeona H. Cruz HANIVAL Sven Abels TIE Tomás Pariente ATOS Marc Richardson BT	
Reviewer(s):	Michal Zaremba SEEKDA Clemens Blamauer HANIVAL	

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	01/2009	Draft version	Simeona H. Cruz (Hanival)
0.2	01/2009	Updates to all sections	Bernhard Schreder (Hanival)
0.3	02/2009	Contributions by TIE to sections 1,2,5	Sven Abels (TIE)
0.4	02/2009	Contributions by ATOS to section 2	Tomás Pariente (ATOS)
0.5	02/2009	Updates to all sections	Bernhard Schreder (Hanival)
0.6	02/2009	Contributions by BT to section 2	Marc Richardson (BT)
0.7	02/2009	Contributions to section 2	Simeona H. Cruz (Hanival)
0.8	02/2009	Updates to section 5 and 6	Bernhard Schreder (Hanival)
0.9.1	02/2009	Review	Michal Zaremba (seekda)
0.9.2	02/2009	Review	Clemens Blamauer (Hanival)
1.0	03/2009	Integration of review comments, Final version for submission	Bernhard Schreder (Hanival)
Final	10/2009	Overall format and quality revision	Malena Donato (ATOS)

Table of Contents

List of Figures

List of Listings

List of Tables

Glossary of Acronyms

Acronym	Definition
D	Deliverable
EC	European Commission
ES	Enterprise Search
FIRE	Future Internet Research and Experimentation
PUPPET	Pick UP Performance Evaluation Testbed
S-CRM	Simplified Coverage Relationship Model
SWS-Challenge	Semantic Web Service Challenge
WADL	Web Application Description Language
WP	Work Package
WS-FIT	Web Services-Fault Injection Tool
WSDL	Web Service Description Language

Executive summary

In this deliverable, we discuss the different components of a testbed environment for SOA4All, as well as suitable testing methodologies. The testbed infrastructure, which is being developed in the scope of Task 1.5, will enable testers and component owners to define configurable testbeds and services according to a collection of service templates, which will be made available to users of the testbed. The testbed infrastructure is delivered as part of the overall evaluation plan for SOA4All, which consists of three different types of evaluation – usability, fit-fo-purpose and technical evaluation. Task 1.5 is concerned with the technical evaluation, and the results can be used to validate the major technical objectives of SOA4All, including scalability and performance of the developed solutions. This document presents a survey and evaluation of different tools suitable for SOA testing and thus for the SOA4All testbed. From this finding, we develop a testbed infrastructure, which shall support the development of the technical work packages in SOA4All, as well as an environment to test the use case prototypes.

Different types of tools that are going to be used as the basis for the testbed infrastructure are surveyed and evaluated in this document. The selected tool(s) should meet the requirements that will be addressed in the requirement section. Besides a collection of functional and non-functional requirements, which were proposed according to discussions with the technical work packages in the project, the deliverable also addresses the alignment with the Use Cases in SOA4All, as the testbeds to be generated with the infrastructure should simulate realistic environments suitable for experiments of the Use Case prototypes. In addition, the testbeds should also be made available to other projects and initiatives outside of SOA4All, such as the SWS Challenge or the FIRE facilities.

Finally, a first design for the overall testbed infrastructure is detailed, and the next steps for setting up a testbed for both short-term experiments in SOA4All, as well as for general experiments with large-scale service oriented architectures are explained at the end of this document.

To summarize, in order to realize the proposed functionalities for a testbed infrastructure, this document will specify the following:

- A set of requirements for a testbed framework. Those requirements will be based on input from different work packages and from the Use Case work packages.
- A report on State of the Art in service testing.
- A survey and selection of tools and methodologies that will be reused.
- Based on the requirements and the feature list of existing tools: A precise description of the functionality that will be implemented and a specification of the development for the testbed infrastructure.
- A usage scenario for the testbed infrastructure, explaining the necessary steps for testers, component owners and other testbed users.

1. Introduction

The SOA concept provides a number of significant advantages compared to monolithic architectures. For example, the SOA approach allows a very flexible way of combining functionalities and overall a perfect example of what it means to separate different concerns by dividing a complex system into many different – yet simple – parts that can be combined into complex and powerful applications. The SOA4All project will use those advantages of SOA and combine them with the powerful Web 2.0 approach and with semantics and context awareness. It also provides an easy way of using services or combining services and it provides a comfortable user interface in the SOA4All Studio (WP2).

However, the high flexibility obviously also lead to a downside of the SOA approach: As systems get more and more composed by different parts, the maintenance and the testing of systems gets very complex. If one subsystem of an SOA based application has been changed, a lot of side effects might occur and each single service might influence the stability, correctness and performance of the whole application. For example, a currency-rate info service that has a delay of 30 seconds for each request and that answers with incorrect results might slow down an eCommerce application and might lead to wrong money transactions in the order process.

As such, a comprehensive testing process becomes crucial for turning the SOA4All vision into reality. However, testing in a distributed environment including many different services that interact with each other in a complex way is not a trivial task. In this deliverable, we discuss the different components of a testbed environment for SOA4All, as well as testing methodologies. Different types of tools that are going to be used as the basis for the testbed are surveyed and evaluated. The selected tool(s) should meet the requirements that will be addressed in the requirement section. Finally, a first design for the overall testbed infrastructure is detailed, and the next steps for setting up a testbed for both short-term experiments in SOA4All, as well as for general experiments with large-scale service oriented architectures will be explained at the end of this document.

1.1 Challenges

In SOA testing, there are certain issues and challenges in the testing and deployment of Web Services that should be taken into account. These characteristics of Web Services are [6]:

- Web Services are intrinsically distributed and are platform and language agnostic
- Web Services can expose dependencies to third party service providers, which can change without notice
- Web Services ownership is shared across various stakeholders
- Web Services client developers typically only have access to interfaces (WSDLs) and lack access to actual code

Thus, it is important to choose the right strategy to face those challenges and to provide a testbed framework enabling users to create proper testing in an SOA4All based application. This includes defining a concrete and precise methodology for testing, to analyse and consider the reuse of existing tools and to provide a solid implementation that is accomplished with an example on how to use the testbed for SOA4All users and/or developers.

1.2 Purpose and Scope

In this deliverable, we will address the challenges mentioned above. We will do this by defining a testbed framework for SOA4All. This testbed framework mainly targets developers, more precisely SOA4All project team members as well as developers using the SOA4All framework as a base. The functionalities that will be provided are:

- **Testbed Setup**

  Setting up a suitable testbed – specifically for the purposes of testing performance and scalability – needs automated tool support. Services, which express a specific behaviour, according to a given set of parameters, need to be generated and deployed automatically.

- **Test Case Definition**

  Test cases may be defined allowing developers to define different criteria that a service needs to fulfil in order to pass a test (e.g. a specific return value or a maximum response time).

- **Test Case Combination & Ranking**

  Test cases may be combined into sets, and may be executed together, with the results being used to reduce testing effort for future runs.

- **Test Case Scheduling**

  The Test bed may be executed on one or more services automatically (scheduling).

- **Test Case Event Handling**

  Developers may define automatic tests and may specify certain events that will be launched when a test is finished successfully or unsuccessfully. This way, developers might specify to receive an email if a certain service fails.

This document presents a survey and evaluation of different tools suitable for SOA testing and thus for the SOA4All testbed. From this finding, we develop a testbed infrastructure, which shall support the development of the technical work packages in SOA4All, as well as an environment to test the use case prototypes.

In order to realize this functionality, this document will specify the following:

- A set of requirements for a testbed framework. Those requirements will be based on input from different work packages and from the use case work packages.
- A report on State of the Art in service testing
- Based on the requirements and the feature list of existing tools: A precise description of the functionality that will be implemented and a specification of the architecture.
- A selection of tools that will be reused
- A specification of plug-ins to be developed
- A step-by-step description on how to use the testbed framework

Its scope is for the consortium to use as a reference for further development of the testbed infrastructure in the upcoming activities for Task 1.5.

1.3 Alignment to SOA4All Evaluation

Figure 1 (taken from the draft version of deliverable D2.5.1, the Formative Evaluation and user-centred Design) depicts the Evaluation Model, which has been developed to define three main types of evaluation to be performed in the scope of the project and describes the flow of results between work packages according to this evaluation plan. The different types of evaluation include a usability evaluation, which will be conducted to determine the usability of the interfaces produced by the project. A Fit-for-purpose evaluation will test the requirements of the three SOA4All Use Cases against the results produced by each work package. Finally, a technical evaluation will test the performance, scalability and other technical characteristics of the tools and techniques developed in the project.

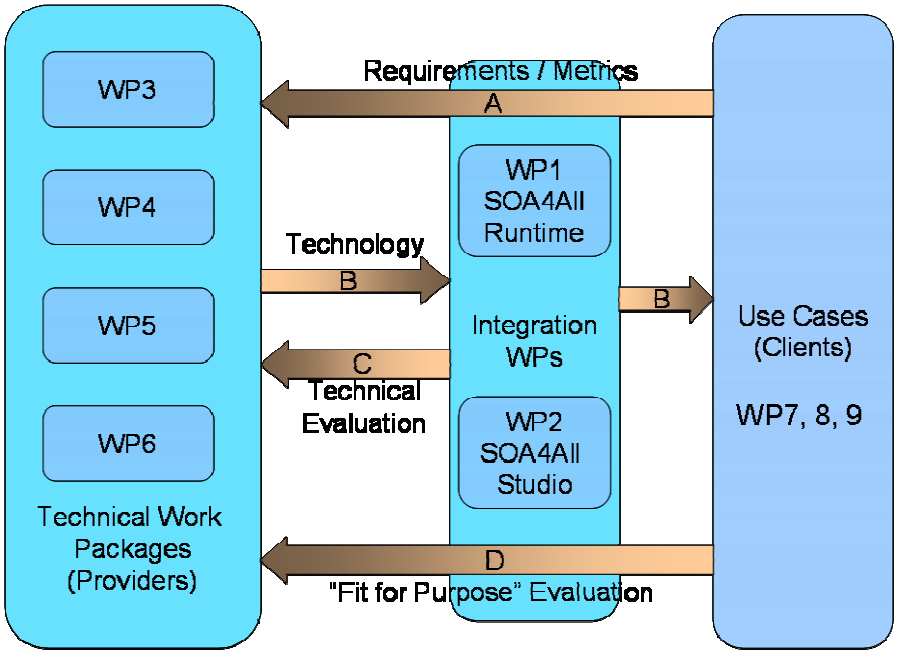

Figure 1: Internal Evaluation Model in SOA4All

Regarding the flow of project and evaluation results, the Use Case WPs provide requirements and evaluation metrics to the core technical work packages (arrow 'A' in the figure). Results from the technical work packages (arrow 'B') will be integrated through the SOA4All Runtime (WP1) and SOA4All Studio (WP2), respectively, who will provide Technical Evaluation (arrow 'C'). The Use Cases then conduct an evaluation of the integrated results concerning the suitability against their requirements and metrics. The results of this 'Fit for Purpose' evaluation (arrow 'D') will again provide feedback to the technical work packages.

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. Requirements Analysis

To define requirements for a testbed infrastructure in the SOA4All project, we should first examine what we would like to achieve by providing such an infrastructure and a matching testing methodology. The major benefit for the partners in the project would be to provide methods to automate parts of the testing process, which can include test case generation, the set-up of a test bed (by generating suitable test services according to a provided specification), or the actual execution of the test cases and subsequent analysis of results. In addition, the scalability of the testing process is of great importance, as SOA4All proposes to provide an architecture and tools, that are performing on a web of billions of services.

Current State-of-the-Art in the testing of SOA applications includes different methods to develop test beds for SOA environments (semi-)automatically. One possibility to do that is to use tools in the area of fault-injection testing, which is especially useful for Quality of Service based testing. A fault-injection testbed for SOAs must reflect a typical service-oriented architecture. Parameterization data for fault-injection testing is based on models, not raw data, due to the general unavailability of a sufficiently large set of statistical data for service behaviour.

In addition, research has been ongoing to develop methods to reduce the effort of testing, while maintaining test efficiency, which is of special concern for the testing of composite services. One such method is called Group Testing of Services. Group Testing of Services proposes a framework that can apply selection and ranking of test cases. By applying selection and ranking methods to the test cases for specific services (both atomic and composite), we can enhance the test efficiency while reducing the efforts of testing for future test iterations.

Based on these basic assumptions, we can define a set of requirements for the SOA4All testbed, which should be fulfilled by the selected tools and the overall testbed framework. Tools should be selected or developed that can provide support for automated testing for Web Services, in order to make the testing process more efficient. When evaluating composite Web Services, we should plan our testbed architecture carefully to enable the test case evaluation procedure performed in the right manner. We should also consider different phases of tests for SOA and the automated testbed environment should cover functional and integration tests. Finally, active (end) user contributions could also be utilised, by using Web 2.0 techniques to apply service ratings and rankings to enhance the test framework. Enumerated requirements are detailed next. These functional and non-functional requirements have been collected from discussions with the technical work packages in SOA4All, in order to enable us to provide a suitable testbed infrastructure.

2.1 Functional Requirements

The following table collects a set of functional requirements to be considered for the design of the testbed infrastructure. Section 4 in this document will specify an evaluation of existing technology concerning these requirements. The evaluation will thus guide the selection of tools and methodologies, which will be reused for the testbed infrastructure.

Table 1: Functional Requirements

ID	Description	Classification
R2.1.1	The testbed infrastructure should include tools to automatically generate and populate a testbed, given a set of configurations and parameters (please refer R2.1.9).	MUST
R2.1.2	The testbed should handle large-scale volumes of services that consist of different composite services.	SHOULD
R2.1.3	The testbed should deal with the complexity of Web Services that are comprised of heterogeneous technology and architectures (see also R2.1.2).	SHOULD
R2.1.4	The testbed infrastructure must be used to generate services according to the description of the specified behaviour or a template mechansims.	MUST
R2.1.5	The testbed should automatically identify and eliminate the test cases that do not meet certain selection criteria, in order to have a more efficient testing process.	SHOULD
R2.1.6	The testbed should cope with the third party Web Services that can change without notice.	SHOULD
R2.1.7	The testbed should be integrated with existing SOA infrastructures generally, specifically by supporting service compositions and other service enterprise features.	SHOULD
R2.1.8	The testbed infrastructure must employ a methodology that can be used to quickly and easily deploy testbeds.	MUST
R2.1.9	The testbed behaviour should be based on a parameterized model, which simulates the occurrence of failures and performance issues.	SHOULD
R2.1.10	The testbed should be independent within the scope of SOA4All architecture, in order to be usable in different contexts.	SHOULD
R2.1.11	The testbed infrastructure must provide support for RESTful Web Services.	MUST

2.2 Non Functional Requirements

The following table collects a set of non-functional requirements that should be considered when defining the testbed infrastructure.

Table 2: Non Functional Requirements

ID	Description	Classification
R2.2.1	The testbed infrastructure MUST be extendable and flexible to integrate it with other testbeds or to add more functionality.	MUST
R2.2.2	The testbed infrastructure tools SHOULD have an open source license[1] with the provided source code in order to achieve the intention of the consortium to reuse, extend and integrate it with other components within SOA4All.	SHOULD
R2.2.3	The testbeds SHOULD be deployable to different machine architectures and platforms.	SHOULD

2.3 Alignment with the SOA4All Use Cases

The testbed framework should serve as an environment in which to deploy the use case prototypes developed in the scope of the project. A suitable set of testing services should be available, in order to provide a realistic environment in which to conduct those tests. Therefore, the first services, which will be deployed on the testbed, will be services created from the Use Case requirements, detailed in the following sections. The testbed infrastructure will be used to create testbeds based on the services, which should be used as part of the Use Case prototypes, thus allowing Use Case partners to conduct suitable experiments on the SOA4All testbed environment.

This can include both services with an implementation, i.e. which can be deployed to the testbeds directly, but also services from third parties, where only the WSDLs are accessible. Service descriptions can then be used for automated service generation using service templates (see requirement R2.1.4 in Section 2.1).

2.3.1 End-user Integrated Enterprise Service Delivery Platform

By example of a public administration scenario, the Use Case “End-user Integrated Enterprise Service Delivery Platform” [14], developed in WP7, investigates how existing, heavyweight SOA platforms can interoperate with the open, dynamic, lightweight, and end

[1] For an overview of applicable licences see http://www.opensource.org/licenses/alphabetical

user driven service platform that is envisioned by SOA4All. WP7 is working around concrete scenarios implementing the “EC Services Directive” in an informal, narrative way. In order to analyze and clarify the desired functionality from the perspective of the different actors involved. In these scenarios, several administrative processes need to be completed triggered by citizens. Following the Services Directive, one public administration takes the responsibility to handle and guide through these processes based on the lightweight process modelling and execution environment developed by SOA4All.

Regarding specific requirements for the Testbed, WP7 has the peculiarity that uses some SAP Enterprise services. These SAP services are not easy to use. The WSDL files are quite complex and some of the data needed is not easy to get, because it is based on SAP codes. Therefore, the current version of services would probably evolve within this case study to simplified WSDL, which is not available now.

On the other hand, these services are not services deployed on the web, but in SAP machines. These SAP services are theoretically implemented and deployed on test servers by SAP in SOA4All. So the availability of these services for the testbed is limited.

It is expected to use third-party services, mainly coming from stub services (simulated services) available from public administration, but these services have not been identified yet.

The following list specifies the SAP services identified until now:

- ES Workplace
    - Create login: https://www.sdn.sap.com/irj/sdn/soareg
    - 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 storyboard scenario:

Table 3: Test Web Services from the WP7 storyboard

Provider	Service Name	URL	Description
SAP	BusinessPartnerBasic DataByNameAndAddress QueryResponse_In		Find Business Partner Basic Data by Name and Address
SAP	CitizenServiceProduct ERPByIDQuery Response_In	PS Permit Application and Approval	Read Citizen Service Product

SAP	CitizenServiceArrangementERPCreateRequestConfirmation_In	PS Permit Application and Approval	Create Citizen Service Arrangement
SAP	CitizenServiceProductMaintenanceCitizenServiceArrangementActionIn	PS Permit Application and Approval	Calculate Citizen Service Arrangement
SAP	ContractAccountsInvoiceProcessingManageContractAccountInvoicingTaskIn	PS Permit Application and Approval	Create Public Sector Contract Account Invoicing Task
3rd Party	(not available)		ValidateCreditCard
3rd Party	(not available)		ValidateAddress

2.3.2 W21C BT Infrastructure

The aim of the Web21c case study [15] is to investigate creating the future Web21c/Ribbit infrastructure based on SOA4All technology. As the Ribbit platform is important part in BT's transformation from a traditional telecommunications company to a converged software and services firm, this case study is focused around it.

Currently, use of Ribbit services requires detailed technical knowledge of programming languages (e.g. Flash and Flex) and understanding of voice protocols (e.g. SIP, Skype and Google Talk's XMPP) to call other Web-based phones, VoIP phones, or regular landline and mobile phones. Ribbit handles the calls and other voice-related services (call logs, voice messages, speech-to-text transcription, contact imports, directories, provisioning, billing, security and authentication) and provides the APIs to developers, who build their apps with Adobe's Flex and Flash development tools. It is a way to create voice apps in a Web application development environment that can easily be linked to other Web apps.

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 Ribbits' services can be done much more effectively.

As BT only recently acquired Ribbit (October 2008) it is in the process of integrating it with BTs systems and creating a new enhanced set of RESTful Services. These RESTful services will form the basis of the Ribbit services that are used in the case study. Currently they are scheduled for release in late March 2009, so at present there are no concrete Ribbit services to offer to the testbed platform. As they are made available, they will be added to the Testbed. The aim of the case study is to enable users to combine Ribbit services with other 3rd party services available on the web, so we can provide an example set of services that a user might consume, based on the scenario described in Deliverable 8.3 of creating a mashup to organise a group of friends meeting up.

For the deployment of the BT use case services on the testbed, it is of special importance to enable the creation of RESTful Services, as much of the Use Case is based on such services. This is reflected by the inclusion of requirement R2.1.11 in Section 2.1.

Table 4: Test Web Services from the WP8 storyboard

Provider	Service Name	URL	Description
3scale	ListEvents	http://www.3scale.net/happenr/happenr	Event search service - finds events (concert, comedy gig, etc.) in a given location by searching through a number of databases
Last.fm		http://www.last.fm/api/intro	Lists (predominately) Music Events
Plazes		http://plazes.com/api/docs#GET__plazes_xml	Nokia Plazes provides a list of Plazes to meet
Facebook API	ProvideContacts	http://developers.facebook.com/	Operation name: friends.get. This service returns the identifiers for the current user's Facebook friends
			Operation name: friends.getLists. This service returns the identifiers for the current user's Facebook friend lists
			Operation name: users.getInfo. This service returns a wide array of user-specific information for each user identifier passed, limited by the view of the current user
LinkedIn API		http://www.linkedin.com/static?key=developers_apis	not available as a public program, but possible to get access by request
Orange	LocationOfContact	http://www.orangepartner.com/site/enuk/access_orange_apis/advanced_apis/localisation_api/p_localisation_api.jsp	available to locate only Orange France users
O2 Litmus		http://www.o2litmus.co.uk/tools/apis#tabs-apis-3	restricted to O2 litmus users
open movilforum		http://open.movilforum.com/?q=node/308	available for Movistar users
GoogleMaps	ListLocal	http://code.google.com/apis/maps/	service provided by a map or routing provider
ViaMichelin		http://dev.viamichelin.com	
Textmefree	SendMessage	http://www.textmefree.com/	provides a list of free SMS options on the web

TFL	TravelRoute	http://www.journeyplanner.org/	service that provides travel option(s) to the meeting point based on the location of the contact
GoogleMaps		http://code.google.com/apis/maps/	
ViaMichelin		http://dev.viamichelin.com	
Met Office	Weather	http://www.metoffice.gov.uk/weather/uk/uk_forecast_weather.html	

2.3.3 C2C Service eCommerce

The C2C eCommerce use case [13] will provide a flexible eCommerce framework that will reuse most SOA4All functionalities in order to enable users to create a wide variety of C2C eCommerce applications. In this scenario, users may combine various services coming from three different domains:

1. Core Services from SOA4All work packages
2. Services provided by the WP9 eCommerce framework (e.g. Webshop services such as payment facilities)
3. Third party services provided by external parties in order to complete the C2C application.

Details on available and planned services are described in deliverable D9.1.1. According to the requirements for and planned work on the eCommerce Framework, the following additional requirements need to be fulfilled by the testbed framework in order to be of help for the WP9 use case:

- The testbed infrastructure should provide a way of setting up scheduled tests programmatically.
- It should also provide a UI allowing people that want to create a C2C application to define test cases and to schedule them
- The testbed framework should allow C2C application builders to define constraints that are used when testing. For example, it should be possible to define a range for return values or a maximum answering time.
- The testbed framework should allow the specification of notifications in order to notify a C2C application owner in case that a test was not passed successfully
- The test results should be passed to the SOA4All monitoring/analysis component and its results should be accessible/visible from the graphical UI of the monitoring/analysis component.

The following table shows the services involved in the current version of the WP9 storyboard scenario. As such, these services will be realised first and should be made available in the testbed environment.

Table 5: Test Web Services from the WP9 storyboard

Provider	Service Name	URL	Description
Deltavista	hanival_creditCheck: checkIndividual	https://www.deltavista-online.at/service/creditCheck	Credit check for individuals. Its function is to identify a person Deltavista needs either the set firstname, lastname and address or the set firstname, lastname, birthdate of customer. Webshop collects all of this data, so we could be sure that we get an accurate result of customer's credit status. It is a service from
	hanival_creditCheck: checkCompany		Credit check for companies, in case a company cannot be found in deltavista's database as a company, we should call the service for individual check. The reason for that is deltavista may store small companies as individuals in their database. It is a service from deltavista side.
Swsoft	SiteWebService	Internal service	This Web Service provides site management functions. Methods are invoked by the user whose credentials are specified in SOAP-header. Example some of the operations: - CreateSite - PublishSite - UpdateSite - ActivateSite
Hanival	OrderManager	Internal service	Operation name: executePayment. This service settles the payment at saferpay
			Operation name: startProvisioning. This service starts the provisioning of the customer products placed on an paid order
			Operation name: submitOrder. This service is started once a valid payment is registered for an order and the product provisioning process start when the order that paid the product is provisioned.

2.4 Alignment with other Projects and Initiatives

The SOA4All testbed should support both short-term experiments and continuously running applications, which demonstrate the uses of a service world in business scenarios. Thus, we will closely collaborate with existing initiatives in this area, including the Semantic Web Service Challenge (SWSC), and will propose the deployment of the SOA4All testbed on the Future Internet Research and Experimentation (FIRE) facilities.

2.4.1 SWS Challenge

Over the last three years and several workshops of the Semantic Web Services Challenge[2] (SWSC), various researchers working on SWS technologies, as a community, have discussed and experimented with the best way to evaluate technologies for the mediation, discovery, and composition of Web Services, and to understand the trade-offs among the various technical approaches.

The initiative and especially the development of its infrastructure have been actively supported until now by researchers from STI Innsbruck. The main effort in terms of work force, infrastructure and budget has been delivered by contributors from the University of Innsbruck. This is also causing the fact that the infrastructure cannot be properly maintained and developed since this has been mostly an effort that researchers have been doing in their spare time. To scale this initiative, one institution cannot be the only promoter and contributor of the initiative. To achieve and persistently provide a fully-fledged infrastructure for the testbeds, a more formal process is required to develop and maintain existing and further scenarios.

The other existing problem to be addressed is the scientific quality and relevance of the initiative. So far, it failed to produce any formal or evident way to evaluate SWS technologies, since the process for the evaluation is not at all well-defined, mostly ad hoc and not objective (e.g. the evaluation tables are changing every time together with meaning of evaluation symbols). The current evaluation methodology for the SWSC is described in more detail in [12].

In order to further develop and maintain this testbed infrastructure for the SWSC, it will be required to rethink the initiative and change its format, especially the evaluation part. Thus, it is planned to directly contribute to the SWSC within the scope of Task 1.5, and provide the testbed infrastructure to be developed within SOA4All also for the creation and maintenance of testbeds for the Challenge. In return, the services and scenarios, which have or will be published by the challenge, will also, provide useful feedback and test cases for the evaluation of SOA4All project results.

2.4.2 FIRE

Future Internet Research and Experimentation[3] (FIRE) is an initiative under the ICT theme of EU Framework Programme 7. The initiative has two related dimensions: Building a European Experimental Facility for Future Internet research, and supporting experimentally-driven

[2] http://sws-challenge.org/wiki/index.php/Main_Page

[3] http://www.ict-fireworks.eu/

advanced research, which defines the challenges for and takes advantage of the dynamically evolving facility.

The SOA4All consortium has already submitted a proposal to deploy its testbeds, generated by the testbed infrastructure described in this deliverable, to the FIRE facilities, in order to conduct experiments on a suitably large scale. This will provide an important benefit to SOA4All, and will enable the consortium to validate one of the main objectives of the project – enabling the Service Oriented Architecture (SOA) revolution on a worldwide scale. Results from this endeavour will be reported in the next deliverable of Task 1.5, D1.5.2.

3. State of the Art Tools and Methodologies

The creation of a testbed infrastructure in task T1.5 of SOA4All has three major goals. The first one is the deployment of a sufficiently large set of services to serve as a realistic environment in which to evaluate the objectives of SOA4All, such as the scalability of the developed solutions. Furthermore, an environment, which supports test automation, will be developed. Providing suitable tools for the testbed generation (creating parameterized services based on a statistical model), the generation of suitable test cases, as the execution of test case batches, will achieve this. Finally, a testing methodology will be developed, which aims to provide means for the ranking and selection of test cases, based on their potency and coverage of other test cases, in order to reduce test effort. The testbed infrastructure will enable the validation of SOA4All developments, demonstrating the achievement of project objectives and the advancement beyond current State of the Art.

In order to achieve these goals, we are going to develop a testbed infrastructure, based on and extending the current State-of-the-Art in SOA testing. Currently a number of open source testbed tools and methodologies are available, which can be reused and extended. This section details a number of existing tools in the mentioned areas, which will serve as a basis for further development. Section 4 contains an evaluation regarding the requirements for a SOA4All testbed. These tools mainly focus on the creation of suitable testbeds and for the configuration of the deployed service, in order to conduct a Quality of Service based analysis of Service-Oriented applications. These tools not only provide valuable insights into the typical challenges encountered when testing large-scale applications based on distributed services, but also serve as the basis for the creation of tools and components needed for the SOA4All testbed infrastructure, which will be detailed in Section 5.

3.1 Group Testing of Web Services

The Group Testing methodology for Web Services proposes a framework, which applies selection and ranking mechanisms to the test evaluation. By using this framework, the potentially overlapping coverage of the test cases is identified during an evaluation phase and used for the elimination of inefficient test cases. It also ranks newly added test cases and re-ranks existing test cases using updated coverage relationships and the recent test results. By applying the selection and ranking to the services, we can enhance the testing efficiency while at the same time reducing the efforts of testing of large scale and complex composite services.

The group testing service framework can be applied for both atomic and composite services. Regarding atomic services, the framework works by collecting those atomic services that are implementing the same specification. As an example, the “store handle” atomic service in the WP9 Use Case, several domain registration services are implementing the same specification. Given *m* test cases T_m ($T_1 \ldots T_m$) and n number of services S_n ($S_1 \ldots S_n$), the total number of tests that need to be executed is simply m*n (see Figure 2). The group testing mechanism broadcasts all test cases to all the atomic services under test. Following the execution of the tests, a voting service, which automatically generates an oracle for each test case, collects all outputs from the tests. Each service’s output is compared to the related oracle, and the results are collected in the form of a service profile, which includes the information on the service’s reliability, as well as on the test cases’ effectiveness. Based on this data, a ranking is applied to the services and test cases.

In the next test phase, the information is reused, by applying the highest ranking test cases first, thus eliminating services that fail as soon as possible. Testing time is thus reduced, and the results are again used to update the service and test case rankings.

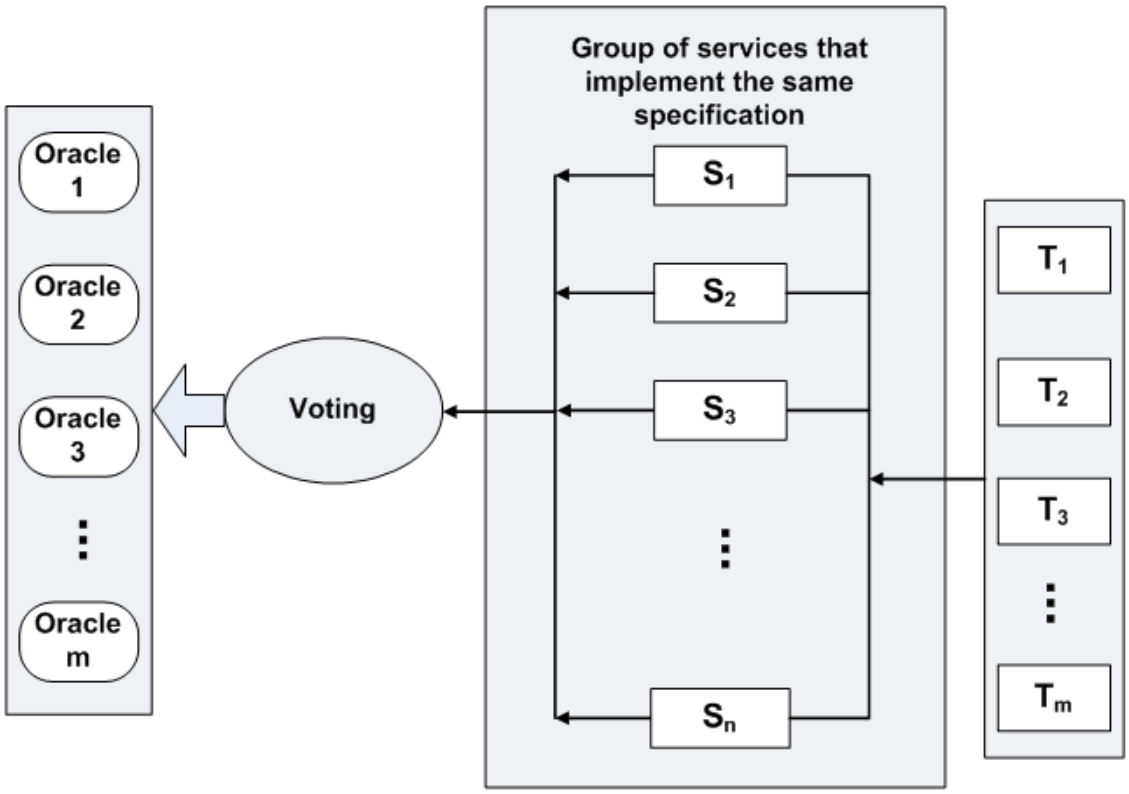

Figure 2: Group testing for atomic services

For composite services, the framework groups together the component services that have several equivalent services to determine the number of tests run. In a composite service, there is a set of *n* component services with each of these *n* component services having several equivalent services. A specific Set_i has m_i equivalent component service. Then the possible composite services that exist will be $m_1 * m_2 * ... * m_n$ if in each set of a composite service consists of one component service. Again, as an example from the eCommerce Use Case, the domain registration process for the webhosting platform (a composite service) consists of five component services where each of the services has three equivalent candidate services. Thus, the total number of number of tests run is 3^5. Figure 3 shows this example. Group testing collects the output from all equivalent component services, and establishes the oracles using the voting mechanism, which is subsequently used to identify and eliminate incorrect services.

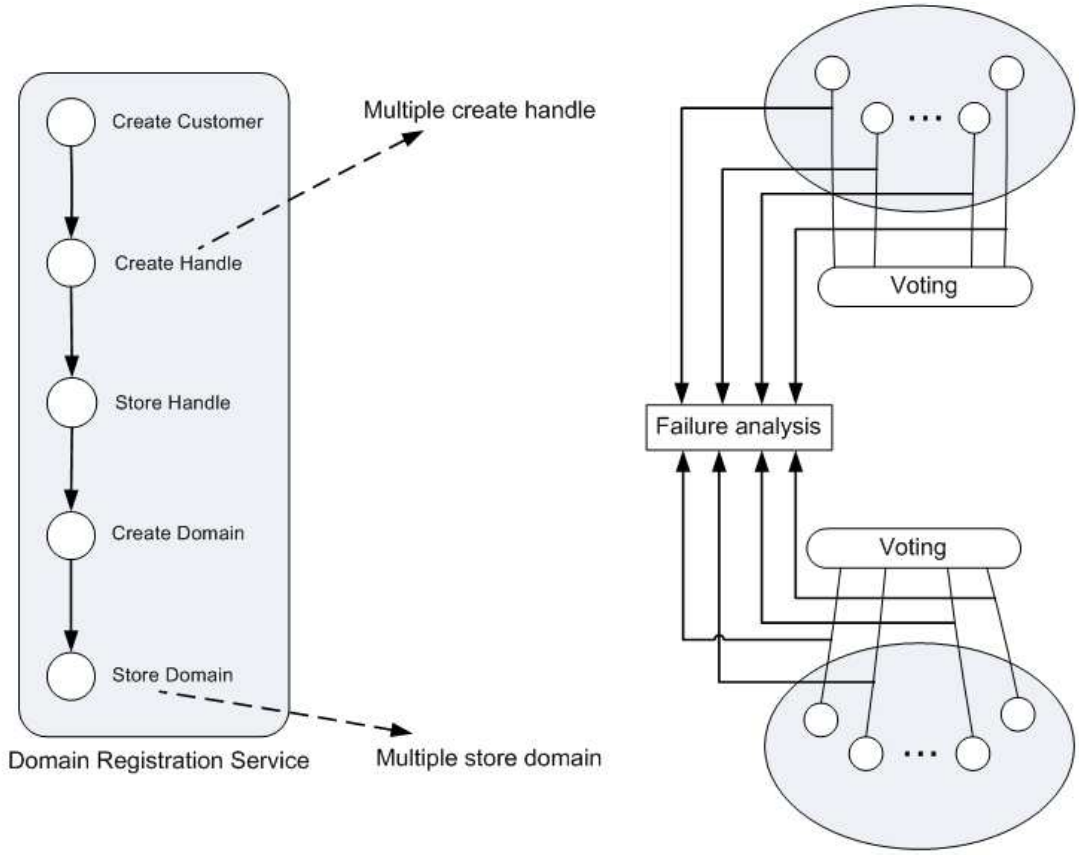

Figure 3: Applying group testing to composite services

Group testing services can also be used for integration testing by conducting the same approach. Only the best candidates from each atomic service are used for the integration testing using this mechanism.

Mentioned above, ranking and selecting the test cases make the execution of the test set more efficient for group testing services. Several criteria can be used for this selection and ranking mechanism. One of these criteria of ranking the test case is to calculate its potency. For this criterion, we could use the statistical data of its probability to detect faults in order to eliminate the incorrect services earlier and to reduce the overall test efforts. As an example, we could assume the potency by detecting the fault of 30 services out of the total 100 that are available. Thus, the probability of a detected fault is 30%.

The second ranking criterion, we can establish the coverage relationship among test cases, that is to identify the amount of additional coverage one test case provides, given that we have applied other test cases first. One way to identify the test case coverage relationships is to analyze how the test cases are developed. They are considered to have similar coverage when the test cases aim to evaluate the same software aspect (e.g., control flow), on the same software segment. A simplified coverage relationship model (S-CRM) has been proposed in [3], in order to reduce some of the computational complexity of constructing a full coverage model. This coverage relationship model is created by first collecting the value of the correct output & incorrect output that are generated by each of the services. From the

correct and incorrect output sets, we can determine the coverage between one test to the other test.

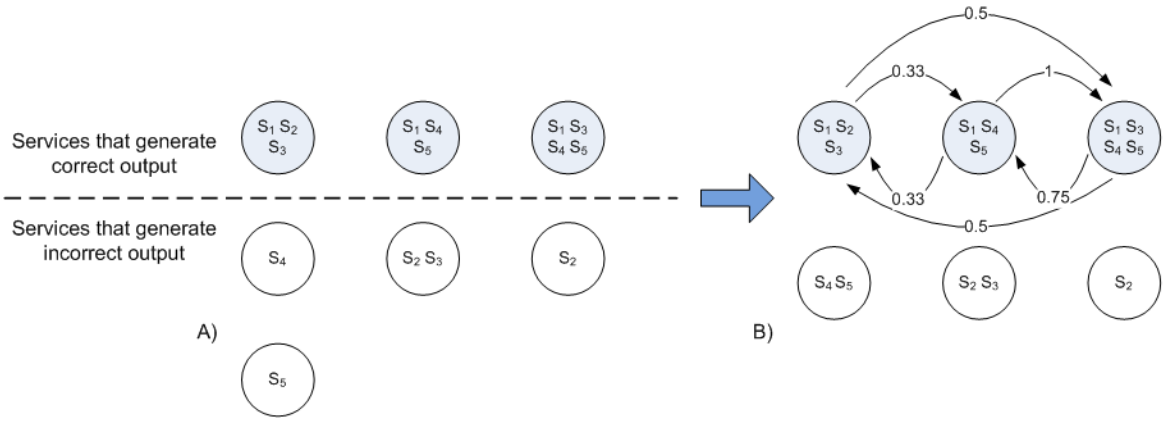

Figure 4: Testing and service selection using S-CRM

According to [3], when only ranking by potency, the SUTs are simply penalized for the same faults multiple times. Therefore ranking test cases by both potency and test case coverage relationships is recommended.

3.2 Fault Injection for a Service Oriented Architectures Testbed

A fault is a defect or an abnormal condition, which may lead to a failure. In fault injection testing, an error or fault is deliberately inserted into a software or hardware system in order to trigger and determine its response. Its target is not to recreate the conditions that produce the fault. By injecting faults into the Web Services, we can enhance the test coverage and simplify the testing. Moreover, fault injection techniques are also useful for the inspection of Web Service compositions.

Fault-injection testing is usually done for Quality of Service based testing. According to [5] fault injection is a well-proven method of assessing the dependability of a system. A fault-injection testbed for SOAs must reflect a typical service-oriented architecture. Parameterization data for fault-injection testing is usually based on models, not raw data. Raw data is usually sensitive, for both privacy reasons and from a business point of view. Raw data also requires a large place for storage. Furthermore, addressing a specific question in raw data is difficult, due to the scarcity of the raw data itself. As an example, in fault injection specifically, rate and the type of the fault occurrence are important for the testing. Therefore, parameterisation for the fault injection should be done based on suitable statistical models and not directly on the raw data [1].

According to [5] there are two types of methodologies for fault injections. Compile-time injection modifies the actual source code of the System Under Test – for example by using mutation code - to inject simulated faults into a system. The second type is Runtime injection where the faults are injected into a running system by using some kind of software trigger (such as time or interrupt based triggers). Between these two methodologies, the main drawback of Compile-time injection is that it requires the modification of the actual code, thus

this technique cannot be used when the actual service implementation is not available for modification, such as for commercial systems or third-party services.

In the next part, we will discuss a selection of current fault injection tools for SOA based systems. The following fault injection tools have been surveyed in the scope of this deliverable:

- GENESIS
- PUPPET (Pick UP Performance Evaluation Testbed)
- WS-FIT (Web Services Fault-Injection Tool)

3.2.1 GENESIS

Creator: **Vitalab – Vienna Internet Technologies Advanced Research Lab**
(http://berlin.vitalab.tuwien.ac.at/prototypes/genesis-generating-service-based-testbeds/)

GENESIS is an open source tool. The tool is available for download on request to the owner. It injects the faults during the actual service invocation, which allows delays and reliability problems to be simulated. It can also be used to generate test Web Services. Furthermore, the tool can be configured and extended by adding plug-ins for different purposes. This different plug-ins can be configured and added to the overall tool as needed. Some examples of plug-ins with its own purpose listed below are provided with the current version of the tool.

- QoSPlug-in: simulates performance- and dependability specific QoS metrics, such as processing time, scalability, throughput, availability, and accuracy.
- BPELPlug-in: integrates the bexee[4] BPEL engine into GENESIS and executes composed processes inside the Web service operations.
- LogPlug-in: logs the invocations of Web services and the interactions within the Web Service itself.
- RegistryPlug-in: registers and deregisters the Web service at a registry.

The GENESIS architecture consists of two major parts. The first part is a single front end where the components and charactheristics of the testbed are being defined and which allows for a centralized control of the test bed.

The second part is the distributed back-end, which generates the testbed infrastructure on a Web Service hosting environment. In this part, the incoming requests are encoded as Web Service descriptions. These descriptions are received by the Controller Web Service and forwarded to the Web Service generator. Then, the Web Service generator transforms them into real service instances. Plug-ins that are used as an extension for the functionalities of the created Web Services are collected at the Plug-in Container and are being controlled by changing their parameters in the local Plug-in Configuration Database. The Front-end and the back-end communicate using the SOAP and TCP based protocol. Figure 5 [2] shows the main components of the GENESIS architecture.

[4] bexee - BPEL Execution Engine, http://bexee.sourceforge.net/

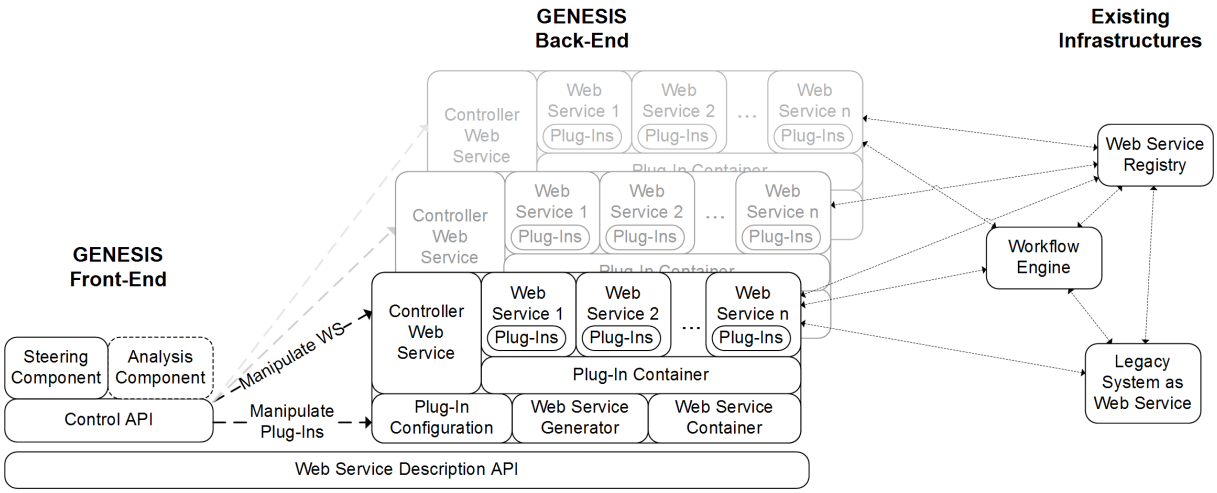

Figure 5: GENESIS Architecture

Both parts of GENESIS architecture share a common description model for Web Services, based on which they exchange data. The major elements of this Web Service model are hosts, services, operations, plug-ins and parameters. Host is defined by an URL which points to a running GENESIS instance. This host contains a set of Web Services. As usual, a service has a unique URL and a set of operations. These operations have a set of input types with a single output type, and the possibility to be extended by referencing a set of plug-ins. The service itself can also reference a plug-in, which is invoked during deployment or undeployment. The services can communicate via Remote Procedure Calls or in message-oriented manner. A parameter must be declared by a plug-in in order to be accessible. The behaviour of the Web Service can be controlled through setting up and declaration of suitable parameters, or by a plug-in.

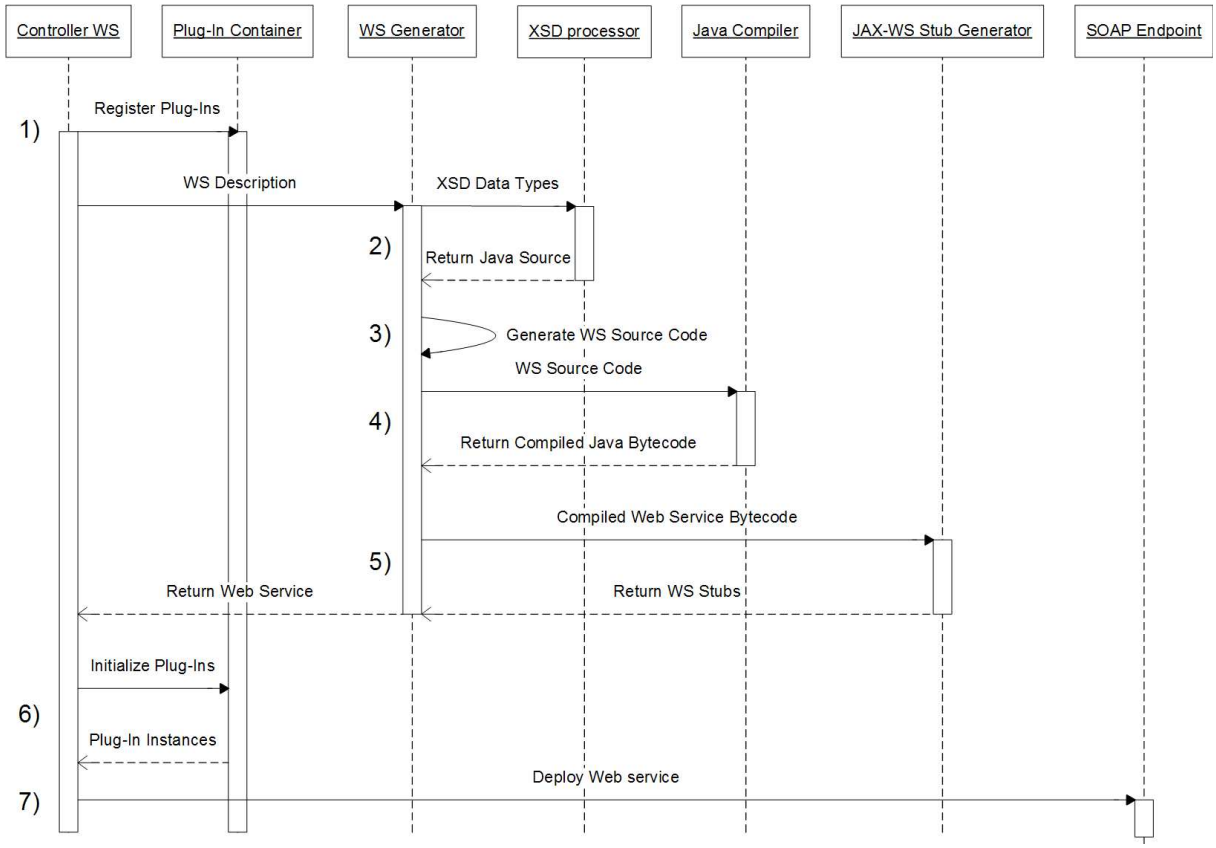

Figure 6: Web service generation in GENESIS

In Figure 6 [2], the sequence of activities for the generation of a Web Service in GENESIS is shown. At the remote back-end host the plug-ins declared in the Web Service description are being checked, to see whether the referenced plug-ins can be found. When these plug-in are not found in the plug-ins container, then they should be transferred and registered. The references are being checked inside the request and responses of the Web Service description. It checks whether the data types used are primitive or complex types. The complex data types are passed on to the XSD processor of JAX-WS, for generating corresponding Java classes. The JAX-WS-compliant source code of the Web Service is generated using Apache Velocity-based templates. The source code is then passed to the Java compiler. The compiled Web Service is passed to *wsgen*, which is again a part of JAX-WS, to generate the necessary stubs for deployment. The class loader reads in the compiled Web Service, instantiates it and initializes all plug-ins. Finally, the Web Service is deployed at the specified HTTP/SOAP endpoint.

In testing the services using GENESIS testbed tool, there are three parts where the parameters could be defined to suit our needs. First, we could define the parameters for the whole service generation as default parameters, second, in each of the operations for individual service template and third we could set the parameters for environment. During the deployment and undeployment of services, plug-ins that should be invoked at this stage, can be defined. The developer can configure these properties using the configuration facility of the API provided. In GENESIS, Web Services can be specified in two different ways. First, the Web Services could be declared as an abstract template, which can be, reused the instantiation of a set of services with common properties. Second, the Web Services are deployable as instances inside host declaration. For an example testbed configuration used by the GENESIS tool, please refer to Annex A Listing 2.

3.2.2 PUPPET (Pick UP Performance Evaluation Testbed)

Creator: **PLASTIC Consortium** (http://www.ist-plastic.org/)

Similar to GENESIS, Puppet validates the quality of service in the process of development of the service. It is an open source tool under GPLv3. The tool is available for download from http://plastic.isti.cnr.it/wiki/tools#puppet. Puppet generates a suitable testbed automatically, and can validate the implementation of a service before its deployment in the target environment. Puppet tests the specified quality of service after its deployment in the final environment using the specified quality of service parameters.

During the testbed creation process, Puppet conducts two different stages (as shown in Figure 7). In the first stage, Puppet generates a skeleton of the stubs. These skeletons are created according to the WSDL description without adding any logic to the service operations, which consist only of empty methods. These stub skeletons simulate the non-functional behaviour of the service in the composition based on the given Service Level Agreement (SLA) in Web Service Agreement format. After the skeleton is created then it is implemented by filling it with the behaviour according to the Web Service Agreement description and applying automatic code transformation. The second stage is to complete the implementation of testing the service. The provided quality of service is being tested by a service in a pre-specified service composition in WS-CDL (Web Services Coordination Language) or WS-BPEL (Web Services Business Process Execution Language). In this stage, the service composition requires the interaction of a human agent according to the specified composition in the orchestration or in the choreography and required services. Puppet fills the stubs with code to emulate the non-functional behaviour described in Service Level Agreement. Puppet generates an environment, in which to test whether the system under test provides the quality of service as described.

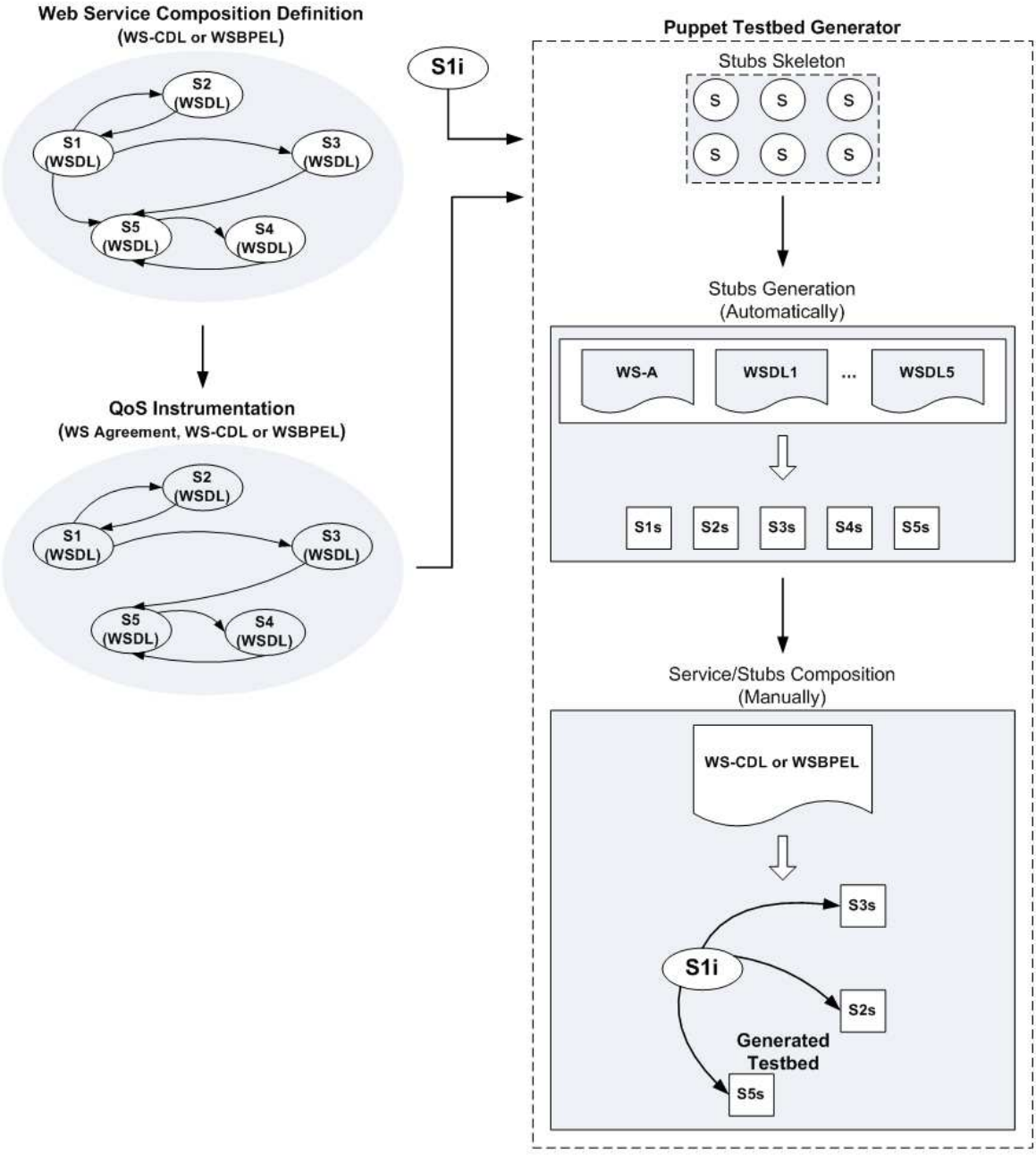

Figure 7: The Puppet testbed generator

Puppet provides the testbed for a reliable estimation of the exposed Quality of Service (QoS) properties of the Service Under Test (SUT). Some Quality of Service attributes supported by Puppet, are delay, reliability and workload. These three attributes can be parameterized. Delay could be emulated by inserting an appropriate sleep() instruction in the stub code. Workload is generated as calls to the remote service that are modelled on the client side, with the amount or frequencies of the calls defined in the Web Service agreement. Reliability refers to the rate with which calls to the Web Service fail. Failed calls are implemented by throwing remote exceptions within the stub.

As mentioned before, Puppet could be parameterized by introducing suitable parameters into the Web Service agreement definition. As an example, please refer to the code fragment shown in Listing 1 [8]:

```
...
<wsag:ServiceLevelObjective>
      <puppetSLO:PuppetSLO>
            <puppetSLO:Latency>
                  <value>10000</value>

                  <puppetSLO:Distribution>
                  <Gaussian>10</Gaussian>
                  </puppetSLO:Distribution>
            </puppetSLO:Latency>
      </puppetSLO:PuppetSLO>
</wsag:ServiceLevelObjective>
...

...
try{
      Thread.sleep(1000);
}
catch
      (InteruptedExcepiton e)
            {}
...
```

Listing 1: Service Level Objective Mapping for Latency

3.2.3 WS-FIT (Web Services Fault-Injection Tool) & GRID-FIT

Faults sometimes take a very long time to occur during testing, especially in an SOA system, where some services and their fault behaviour may not be directly observable. Therefore, WS-FIT aims in producing the fault itself and in injecting those faults on services deployed in on a wide variety of different platforms and machine architectures. This fault injection method is a modified version of general network-level fault injection. The tool injects faults to assess SOAP based Web Services by using network-level fault injection at runtime. It performs this operation on the middleware message layer (please refer to Figure 8). WS-FIT consists of two parts:

1. An instrumented version of the SOAP stack is used, by adding pieces of fault injector hook code to the SOAP API. This hook code can be installed in more than one SOAP stack, for example the SOAP stack of a client machine and the corresponding SOAP stack of the server machine whose services the client is consuming (as an illustration, please refer to Figure 8). The hook code consists of segments for both incoming and outgoing messages. The incoming message is intercepted by one hook. This hook then transmits the message to the fault injection engine via a specific socket and receives the modified message back from the fault injector engine. This modified message is then transmitted normally to the original destination. Another hook for outgoing messages processes SOAP messages in the same way.

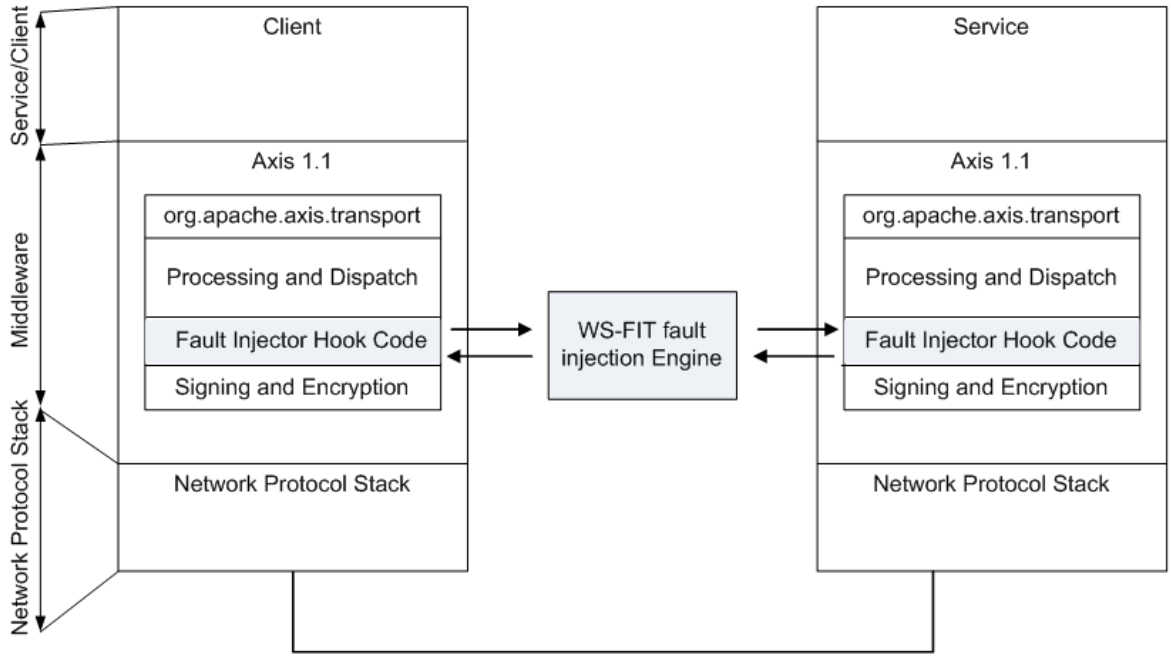

Figure 8: WS-FIT fault injection hook code placement.

2. The actual fault injector engine, which conducts the following sequence of activities (please refer to Figure 9).
    a. It receives the SOAP message encapsulated in an XML document from the hook code.
    b. It extracts the SOAP message, processes and logs the information from the encapsulating XML document.
    c. Two triggers now determine how the message should be further processed.
        i. In a first step, the so-called quick trigger determines if faults are required to be injected into the message supplied, in order to determine whether detailed processing is necessary at all.
        ii. In the second step, a message trigger determines where the faults should be injected. Two kinds of triggers are used, which react to the whole message, and forward the whole message body, or parameter triggers, which process the bodies of specific parameters.
    d. Conduct the actual fault injection on the message, based on a user script linked to the corresponding trigger.
    e. Transmit the message back to the hook code.

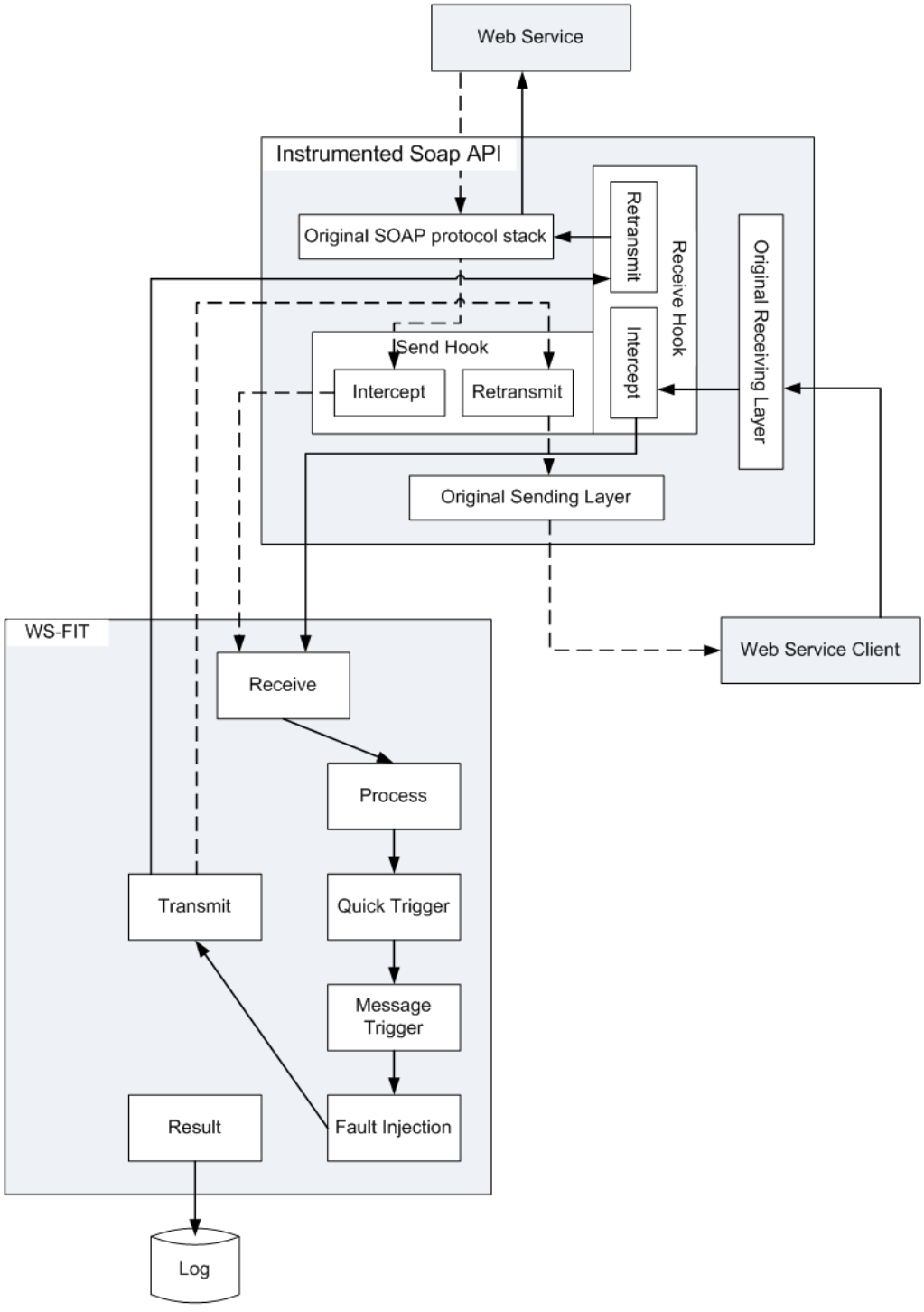

Figure 9: WS-FIT message modification

WS-FIT supports two fault classes. It supports communication faults such as delay and message loss, which are injected by delaying or discarding messages. Furthermore, a variety of faults can be injected by message modification, based on provided user scripts.

4. Evaluation

This section serves as an evaluation of the examined tools and methodologies with regard to the overall requirements for a SOA4All testbed, as previously specified in Section 2. The evaluation will be used to select an existing tool (or combination of tools and approaches) in order to provide a starting point for the development of the SOA4All testbed. The concrete decisions for the design of the testbed infrastructure are explained in more detail in Section 5.

Group Testing Service

There is currently no prototypical implementation of a group-testing tool available, however the framework and underlying methodology is described in detail. In the group testing service, the test could be done efficiently by eliminating unnecessary test cases by applying selection and ranking of test cases according to their coverage and potency. Thus, we will pick up the methodology described and implement it as a part of the overall tool chain in the SOA4All testbed infrastructure, which will fulfil the R2.1.5 requirement from Section 2. This methodology also covers the requirement R2.1.2, as the methodology is specifically suitable to handle a large-scale volume of services. We will use this method by applying it to one of the available tools. A description of how the method and tool are integrated will be explained further in the testbed infrastructure specification section.

GENESIS

This testbed generation tool fulfils some of the requirements specified in section 2 of this deliverable. GENESIS is a testbed generation tool (requirement R2.1.1), which can be integrated into existing SOA environments thus fulfilling requirement R2.1.7. With GENESIS, realistic Web Services with appropriate behaviour abstraction can be simulated, which fulfilled the R2.1.4 requirement. GENESIS also generates Java code automatically and can be controlled through the provided Java API conveniently. This testbed generates the services automatically on remote hosts and allows tool users to set global parameters for the testbed, fulfilling requirement R2.1.9. Regarding to the R2.2.1 requirement this testbed is also flexible and extendable by adding appropriate plug-ins. Furthermore, the tool is Open Source software, fulfilling requirement R2.2.2.

The flexible extension mechanism, based on additional plug-ins, also enables developers to provide support for functionalities that none of the examined tools currently provide. For example, a plug-in for the creation of simulated RESTful services can be included, which realises the described behaviour through the reactions to the usual HTTP commands.

Puppet

The puppet tool has been developed for specific application scenarios, specifically for service oriented mobile applications [9]. However, the tool and methodology can still be used as part of the envisioned testbed concepts.

Puppet has some features, which fulfil some of the requirements for a SOA4All testbed infrastructure:

– Automatic generation of a testbed, according to a set of configuration parameters (R2.1.1)

- Stubs are generated automatically based on WSDL & WS – Agreement (R2.1.2 & R2.1.7)
- Transform XML definitions into Java code (R2.1.9)

In addition, the tool is Open Source software, and the resulting testbeds deployable to different platforms, fulfilling the R2.2.2 and R2.2.3 requirements.

WS-FIT

The WS-FIT tool has seen another phase of development and is currently available as GRID-FIT, for the use in GRID computing [10]. WS-FIT has different features, which are:

- Simulates API level faults without the need for modifying code or running a test harness. This feature covers the R2.1.6 SOA4All requirement, as no access to sources of Web Services is necessary.
- It is performed in the middleware message level.
- The hook code can be installed in many different platforms, which are good for the Web Services, since their implementation is deployed from different platforms and machine architectures. This feature will cover the R2.2.3 SOA4All requirement.

WS-FIT is using Runtime fault injection techniques, where it does not need any code modification compared to the other techniques. However, according to [11] the WS-FIT design has a distinct disadvantage, since the trigger stage only returns simple truth values which determine for the process to continue to the second stage or not. This could complicate the inclusion of user script triggers.

As WS-FIT is based on a modification of the Web Service stack, it abstracts from the actual technology used to implement the services, fulfilling requirement R2.1.3. Requirement R2.1.9 is fulfilled, as different parameters and behaviour can be specified to simulate the occurrence of failures. The tool is available as Open Source (R2.2.2).

Table 6 below summarises all tools, the requirements fulfilled by each tool and the type of fault injection used by the tools.

Table 6: Summary of fault injection tools

Tool/Methodology	Fulfilled Requirements	Fault Injection Type
GENESIS	R2.1.1	Compile time fault injection
	R2.1.4	
	R2.1.7	
	R2.1.9	
	R2.2.1	
	R2.2.2	
	R2.2.3	
PUPPET (Pick UP Performance Evaluation Testbed)	R2.1.1	Compile time fault injection
	R2.1.2	
	R2.1.7	
	R2.1.9	
	R2.2.2	
	R2.2.3	
WS-FIT (Web Services Fault-Injection Tool)	R2.1.3	Runtime fault injection
	R2.1.6	
	R2.1.9	
	R2.2.2	
	R2.2.3	

The Fault Injection tools GENESIS, Puppet and WS-FIT have some similarities in the way they realise the injection of faults in Service based applications. Each tool is mostly concentrating on the Quality of Service, particularly in injecting delay. Aside injecting the delay, GENESIS also has some other QoS elements that are injected as a fault for QoS checking. GENESIS and Puppet use Compile-time fault Injection techniques, where the Web Services themselves are modified to simulate a particular behaviour. Still, as the tools enable the mocking of Web Services based on real WSDL definitions, they are still suitable to create realistic testbeds for SOA4All. Due to the extendibility of the GENESIS solution and the fulfilment of a large set of our requirements for the testbed infrastructure, we will base the SOA4All testbed infrastructure on this open source tool.

The next section will specify the development of the testbed infrastructure and the planned extensions for the chosen tool, GENESIS. A number of additional requirements will be fulfilled by extending the set of available plug-ins for this tool. Furthermore, a usage scenario for the testbed infrastructure based on this tool will be detailed.

5. Testbed Infrastructure Specification

In the remainder of this deliverable, we have provided details on the basic requirements for a SOA4All testbed. We have identified two methodologies for testing to support those requirements; including fault-injection based testing of Web Services, and group testing of services. We have also gathered some ideas or concepts that are used to test a SOA application in available Open Source tools, and - after an evaluation of these tools - we present a basic specification for the testbed infrastructure in this section.

As mentioned in Section 4, we will use the Open Source tool GENESIS as the basis for the development of a testbed infrastructure, more specifically as a tool to create testbeds which can then be used for experiments both within SOA4All, and in other contexts (such as the SWS Challenge).

Regarding licensing of the testbed infrastructure tools: As the basis for development, GENESIS, is using the GNU Lesser General Public License[5], as published by the Free Software Foundation, the same licence will be used for the SOA4All testbed tool.

5.1 Methodology Description and Infrastructure Design

Figure 10 shows the main use cases and actors of the SOA4All testbed infrastructure, based on the concepts and methodologies discussed in the previous sections. We will use the GENESIS testbed generator as a basic tool with the combination of Puppet fault injection functionalities and Group Testing Service concepts and methodologies. GENESIS offers a great degree of flexibility due to its plug-in mechanism and supports a large part of the core functionality for an automated testbed environment. We could use different plug-ins or create similar plug-ins for an extension of the tool. Furthermore, the group testing service methodology, including selection and ranking mechanisms, will be added as additional components to make the test evaluation more efficient. This plug-ins will then be integrated with the GENESIS tool.

The main actors of the testbed infrastructure include the technology providers, which include both component owners (creators of components such as the ranking component from WP5 or the composition component from WP6) and use case owners (the providers of the use case prototypes from WP7, WP8 and WP9). These actors will define service templates, i.e. create templates describing requested service behaviour, and store these templates in a repository. Test cases can be defined by each technology provider (including unit tests for specific functionalities of components or integration tests), but are also defined by testers, which conduct evaluation experiments (e.g. concerning scalability or performance of integrated components) independently from the specific component owners. In addition, testers would prepare testbed configurations, i.e. by defining suitable Quality of Service parameters and global settings like the definition of available hosts for deployment of services. Testers would also be in charge of deploying instances of the configure testbeds, which include selected service templates. Services will then be instantiated and deployed on the testbeds, according to the settings of the selected service templates. Finally, test cases can be executed, for example by using the SOA4All Runtime Environment. The execution of test cases produces not only evaluation data, but also useful ranking information useable for the future selection of test cases. A concrete usage scenario of the infrastructure is explained in Section 5.3.

[5] Available at http://www.gnu.org/licenses/

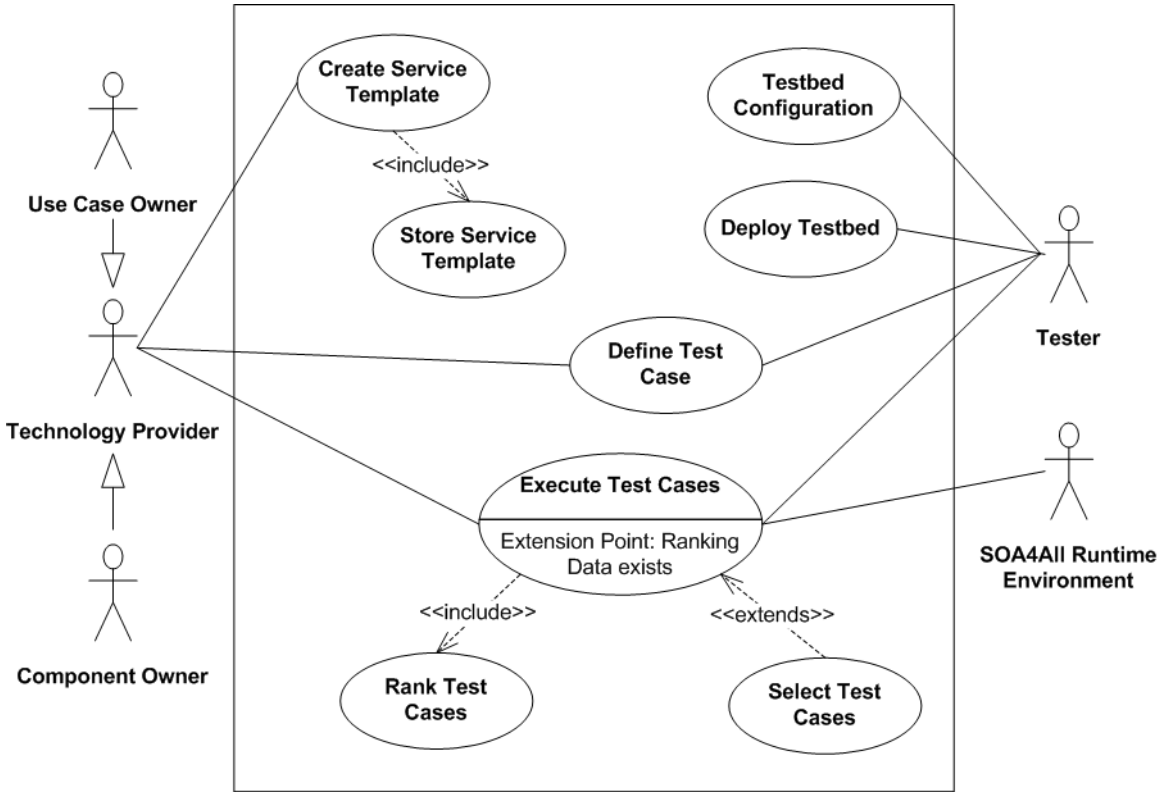

Figure 10: SOA4All Testbed Use Case diagram

5.2 Plug-ins Specification

In order to provide additional support for the requirements collected in Section 2, 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 future design and development of the testbed infrastructure tool, and will be considered as the basis for the extensions planned for deliverable D1.5.2.

The following plug-ins will be designed and implemented in D1.5.2:

- HTTP plug-in for the support of RESTful Web Services
- Orchestration plug-in for composite services
- Group-based testing plug-in for efficient testing

Further plug-ins could be defined based on the first experiments with the testbed generation tool and the service templates. Deliverable D1.5.2 will contain an updated section on the usage of the testbeds and the requirements for conducting evaluation experiments on the testbeds (based on the findings in the overall evaluation roadmap to be defined in D2.5.1).

5.2.1 REST plug-in

This 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 well-known Web Services testing tool soapUI[6] contains support for testing REST services. The services themselves can be created as mock objects, based on either a manual configuration of service name, endpoint and the definition of initial resources, or based on a WADL definition. The Web Application Description Language (WADL), specified in [16], has been designed to provide a machine process-able description of HTTP-based Web applications. By supplying a WADL document, soapUI can import all defined resources and methods (as requests). Associated XML Schemas are also imported and used for validations and form-generation. Additionally, code and documentation generation based on the WADL is possible as well.

In D1.5.2 we will propose a mechanism providing similar support in the form of plug-in. WADL style information will be based on an extension of the grammar used currently within GENESIS to define service templates.

5.2.2 Orchestration plug-in

A new version of an orchestration plug-in will be needed, supporting the lightweight process orchestration language developed in WP6. The existing work on the BPEL plug-in will be reused if possible, as it is expected that the SOA4All process definition language will use a mapable subset of the available language constructs from BPEL.

The light-weight process language will be defined in [17], which is due Month 12 of the project. The exact details of the process language are still open at the time of this writing, further details for the proposed orchestration plug-in will therefore be provided in the next deliverable for Task 1.5.

5.2.3 Group-based testing plug-in

In order to support the methodology for group based testing described previously, a plug-in will take care of administering test cases to different – potentially equivalent – services, and will realise the ranking and selection mechanism. Resulting data will be stored in a testing repository, and will be reused for future iterations of the test cases, thereby allowing the selection mechanisms to prioritize test cases based on their coverage probabilities and potency.

We will design and implement the simplified coverage relationship model proposed by the authors of [3]. The plug-in containing this methodology will collect test data generated over the whole testbed for which group-based testing was activated. Test data will be stored in a dedicated repository, and evaluated to create suitable oracles and provide data to the voting process. Test Cases will thus be ranked and selected accordingly for future test runs.

In addition to the testbed plug-in which collects and evaluates the data for the group-based testing, a suitable front-end for the selection of test cases will be needed. When users define which test cases should be part of a test suite to be executed, this ranking should be visible. Alternatively, the selection functionality could be automatically applied to suitable test suits, which are executed programmatically, e.g. through a continuous build system.

[6] Available at http://www.soapui.org/

5.3 Testbed Infrastructure Usage Scenario

The following systematic description exemplifies a typical usage scenario of the envisioned testbed infrastructure:

1. A prospective user of a testbed, like a SOA4All component owner or tester, or a use case partner, which wants to test the tools with services from their own use case service landscape, starts the testbed generator tool by selecting a suitable testbed configuration template.
2. The user further defines additional parameters of the testbed, such as Quality of Service parameters, host details and other global characteristics of the testbed.
3. The user now can start to select services, which are to be deployed to the testbed, by both selecting suitable prepared test service templates, writing their own service templates, or by simply deploying existing test services to the testbed hosts on their own.
4. The user can now start to execute test cases, or batches of test cases. These can be provided by the component owners, dedicated unit and integration testing teams, or by the use casework packages. A test case repository will enable users to gather suitable test cases quickly and to reuse and adapt already existing test cases.
5. Test case suites will be executed by the testbed infrastructure, according to the group testing methodology. This will enable the testbed user to collect useful test case ranking and rating data, allowing a future reduction in testing efforts, by the means of the group testing voting mechanism described previously.
6. Finally, the user persistently stores the collected test data in a testing repository, and alerts the concerned parties, like service creators or component owners, to the results from the test run.

The testing process (steps 4 to 6 in the description above) can of course also be included in a suitable continuous integration environment used for the development of SOA4All tools and components. The necessary alignment of the testbed infrastructure with continuous integration tools, based on an environment such as Apache Continuum[7], will be investigated in D1.5.2, the next deliverable in Task 1.5, concerned with the set-up of the testbed infrastructure for SOA4All. Apache Continuum is an enterprise-ready continuous integration server featuring automated builds, release management, role-based security, and integration with popular build tools and source control management systems, such as those already in use for the SOA4All developers.

[7] http://continuum.apache.org/

6. Conclusions

In this deliverable, we have described the major requirements of the testbed infrastructure environment for SOA4All. This infrastructure will be used as part of the overall efforts to evaluate SOA4All project results. The testbed infrastructure will provide a set of tools and methodologies to enable prospective users, such as component owners, use case partners and dedicated testers to generate testbeds, create test cases and execute those test cases on the testbed. Results from testing are not only be used for project evaluation, but also serve as the means to reduce testing efforts while maintaining efficiency.

We have surveyed a number of promising tools from two areas in SOA system testing, including fault injection tools and the group testing methodologies, which are used to reduce the testing effort for a large set of services. Based on this, we have described the setup of the SOA4All testbed infrastructure, which will consists of one of those tools – GENESIS – and a collection of plug-ins to support different functionalities in the testbed environment. The flexible architecture of this tool will enable the design and development of dedicated plug-ins, which provide support for specific areas of testing in SOA environments. Several plug-ins which are going to be developed in the scope of the next deliverable for Task 1.5 have been described, based on the set of collected requirements. These plug-ins include support for the creation of composed services, for the application of the group testing methodology to the testbeds and for the generation of RESTful services, respectively.

Finally, a usage scenario for the users of the testbed infrastructure has been described, and an outlook on the development of the different extensions for the testbed generator GENESIS has been given. Altogether this tool, along with the plug-ins and testing methodologies, should provide a way to reduce the testing effort in a large SOA environment, while maintaining test efficiency at the same time.

In the next deliverable of Task 1.5, we will describe the actual extensions of the GENESIS tool, and the set-up of the testbed architecture. The final deliverable in this task, D1.5.3, will then provide the results from experiments conducted on testbeds generated using the described infrastructure.

7. References

1. Philipp Reinecke, Katinka Wolter, Institut fuer Informatik Humboldt-Universitaet zu Berlin, “Towards a Multi-Level Fault-Injection Testbed for Service-Oriented Architectures: Requirements for Parameterisation”, Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems (at the IEEE SRDS 2008), October 6 2008-October 8 2008, available at http://www.amber-project.eu/srds-ws/.
2. 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.
3. W.T. Tsai, Xinyu Zhou, and Yinong Chen, Arizona State University, USA and Xiaoying Bai, Tsinghua University, China. On Testing and Evaluating Service-Oriented Software. 0018-9162/08 © 2008 IEEE, 40-46.
4. Eric Roch (Chief Technologist). The Service Oriented Architecture (SOA) Blog. Posted 3/22/2006.
5. N. Looker, M. Munro, and J. Xu, “WS-FIT: A Tool for Dependability Analysis of Web Services,” in COMPSAC ’04: Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts - (COMPSAC’04). Washington, DC, USA: IEEE Computer Society, 2004, pp. 120–123.
6. Rizwan Mallal, Mamoon Yunus, Crosscheck Networks, SOA Testing using Black, White and Gray Box Techniques access on: 3rd December, 2008 - http://www.crosschecknet.com/soa_testing_black_white_gray_box.php
7. Yugan Sikri, End-to-End Testing for SOA-Based Systems, available at http://msdn.microsoft.com/en-us/library/cc194885.aspx#anchor2
8. Antonia Bertolino, Guglielmo De Angelis, Andrea Polini, Consiglio Nazionale delle Ricerche, Pisa, Italy, Automatic Generation of Testbeds for PreDeployment QoS Evaluation of Web Services, Published as Technical Report at ISTI/CNR © 2006-07-31.
9. Antonia Bertolino, Guglielmo De Angelis, Francesca Lonetti, Antonino Sabetta Istituto di Scienza e Tecnologie dell’Informazione “A. Faedo” Consiglio Nazionale delle Ricerche, Pisa, Italy, Let The Puppets Move! Automated Testbed Generation for Service-oriented Mobile Applications, IEEE 3-5 Sept. 2008.
10. N. Looker, B. Gwynne, J.Xu, M. Munro "An Ontology-Based Approach for Determining the Dependability of Service-Oriented Architectures", 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems, Sedona, USA, 2005.
11. N. Looker "Dependability Assessment of Web Services," Durham University, Department of Computer Science, PhD Thesis, 2006.
12. Charles Petrie, W3C SWS Challenge Testbed Incubator Methodology Report, W3C Incubator Group Report 31 March 2008, available at http://www.w3.org/2005/Incubator/swsc/XGR-SWSC/
13. Schreder, B., Villa, M., Abels, S., Zaremba, M.; Deliverable D9.1.1: Future C2C eCommerce Requirements and Scenario Descriptions, SOA4All: Service Oriented Architectures for All - 215219.
14. Vogel, J., Schnabel, F., Mehandjiev, N.; Deliverable D7.2 Scenario Definition, SOA4All: Service Oriented Architectures for All - 215219.
15. Richardson, M., Martínez, I.; Deliverable D8.1. Web21c Requirements, SOA4All: Service Oriented Architectures for All - 215219.
16. Hadley, M.J.; Web Application Description Language (WADL), Sun Microsystems Inc., Specification available at https://wadl.dev.java.net/wadl20061109.pdf

17. Xu, L. (editor); Deliverable D6.3.1 Specification Of Lightweight, Context-aware Process Modelling Language, SOA4All: Service Oriented Architectures for All - 215219.

Annex A.

The following Listing presents a sample configuration for the GENESIS Testbed generation, explained in Section 3 of this document.

```
<configuration>

  <!-- necessary plugins to simulate QOS processing time and service
invocations -->
  <plugins>
        at.ac.tuwien.vitalab.genesis.server.plugin.QOSPlugin
      at.ac.tuwien.vitalab.genesis.server.plugin.InvocationPlugin
  </plugins>

  <!-- by default delay service operations by 2 seconds-->
  <defaultparameters
    qos_processingtime="2000"
  />

  <!-- by default we just simulate the delay -->
  <behavior>
    <QOS default="true">
      QOSPlugin.simulateDelay
    </QOS>
  </behavior>

  <schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <!-- types can be imported of defined inline -->
    <!-- <import name="SomeData" file="path/data.xsd"/> -->
    <xs:complexType name="somestructure">
    ...
    </xs:complexType>
  </schema>

  <servicetemplates>
    <service name="getAndCheckServiceTemplate">
      <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>

  <environment>
    <host address="http://localhost:8080/WebServices/GeneratorService" >

      <service name="dbService">
      <operation name="getData" >
        <!-- data retrieval takes 5 seconds -->
        <parameter name="qos_processingtime">5000</parameter>
          <input>
          <name type="string"/>
        </input>
          <output type="somestructure"/>
        </operation>
      </service>

      <service name="checkService">
      <operation name="checkData" >
        <!-- checking takes 0.5 seconds -->
          <parameter name="qos_processingtime">500</parameter>
          <input>
          <data type="somestructure"/>
        </input>
          <output type="void"/>
        </operation>
      </service>
    </host>
</environment>

</configuration>
```

Listing 2: Genesis Testbed Configuration