PLASTIC

IST STREP Project

Deliverable D4.2

Test Framework: Prototype Implementation

Release 1.0

http://www.ist-plastic.org

INRIA SIEMENS IBM Telefónica UCL UNIVERSITÀ DELLA SVIZZERA ITALIANA VIRTUAL TRIP Ltd Internet Technologies PRAGMA CONSULTORES

Project Number	:	IST-26955
Project Title	:	PLASTIC
Deliverable Type	:	Prototype

Deliverable Number	:	D4.2
Title of Deliverable	:	Test Framework: Prototype Implementation
Nature of Deliverable	:	Prototype
Dissemination Level	:	Public
Internal Document Number	:	D4.2
Contractual Delivery Date	:	01 August 2007
Actual Delivery Date	:	01 August 2007
Contributing WPs	:	WP4
Editor(s)	:	Antonia Bertolino
Author(s)	:	Antonia Bertolino, Domenico Bianculli, Guglielmo De Angelis, Lars Frantzen, Zsolt Gere Kiss, Carlo Ghezzi, Andrea Polini, Franco Raimondi, Antonino Sabetta, Giovanni Toffetti Carughi, Alexander Wolf
Reviewer(s)	:	

Abstract

This document is the second deliverable of PLASTIC Work Package 4 (WP4): Service Validation Methodology and Tools. It provides guidelines for installing and using the tools that have been developed within the work package. The first deliverable released by the work package complements this deliverable providing a more detailed view on the proposed PLASTIC validation framework. As described in the Deliverable 4.1, the framework is organised around two main phases, respectively called off-line and on-line, which corespond to two concecutive phases in the service lifecycle. These two phases are further divided in validation phases as described below.

The first phase, referred as *off-line* validation, concerns validation at development time. In this phase services are tested in a fake/simulated environment that reproduces functional and/or non-functional run-time conditions. In particular, the approaches defined for this stage are:

1. JAMBITION (see Chapter 2): this is a model-based testing tool that allows to automatically derive and execute invocation sequences on a service, checking whether the responses conform to a given specification, expressed as a Service State Machine (SSM). Indeed, it is possible to use JAmbition also for testing services in the subsequent phases. Nevertheless, in this case it is necessary to support JAmbition with additional mechanisms in order to prevent side-effects that may occur due to testing of a running system, in particular when stateful resources are involved. JAmbition is based on a model based testing engine called Ambition of which JAmbition is the a Java front-end.

2. PUPPET (see Chapter 3): this is a tool to automatically generate stubs implementing the services to be used by the service under development. The mock services generated by Puppet exhibit a correct behaviour with respect to given non-functional properties. At the same time Puppet can generate stubs making invocations on the service under evaluation according to certain workload profiles.

3. WEEVIL (see Chapter 4): it consists of a synthetic-workload generator coupled with an environment for managing the deployment and execution of experiments. Weevil is intended to facilitate experimentation activities for distributed systems by providing engineers with a flexible, configurable, automated and, thus, repeatable process for evaluating their software on a networked testbed.

The second phase of the validation framework is referred as *on-line* validation. This phase foresees testing of a service when it is ready for deployment and final usage. Two main stages have been identified within the on-line phase. The first stage is called *admission* and concerns testing of services when they ask to be included in a registry. Following the admission testing idea registration will be granted only to services that pass the testing phase. Chapter 5 presents WS-GUARD, which is the implementation of a registry that is capable to perform the admission phase before including services in the directory.

The last stage of the framework, part of the on-line phase, is the validation during the *Live Usage* stage. In this stage service behaviours are observed during real execution to reveal possible deviation from the expected behaviour. Also in this case validation can cover both functional and non-functional properties. Approaches developed to support this phase are:

1. DYNAMO-AOP (see Chapter 6): it is a framework for monitoring functional properties of external services which a BPEL process interacts with, to realize a composite service.

2. SLANGMON (see Chapter 7): it permits to detect violations of non-functional properties specified in SLAng, during real execution of the service. Events related to non-functional characteristics are logged and possibly used to redeem controversy.

Keyword List

On-line testing, Off-line testing, Audition testing, Model-Based Testing, QoS testing, Monitoring, Testbed Harness

Document History

Version	Type of change	Author(s)
0.1	Proposed Outline	CNR (with all)
0.2	Proposed Outline with updates from partners	CNR (with all)
1.0	First version for reviewers	CNR (with all)

Document Review

Date	Version	Reviewer	Comment
Date	1.0	Reviewer name	put comments here

Contents

Table of Acronyms

Acronym	Expanded Version
AMBITION	Automatic Model-Based Interface Testing in Open Networks
AOP	Aspect Oriented Programming
API	Application programming interface
ASF	Apache Software Foundation
BPEL	Business Process Execution Language
B3G	Beyond 3^{rd} Generation
DBMS	Database Management System
DYNAMO	Dynamic Monitoring
EMOF	Essential Meta Object Facility
GUI	Graphical User Interface
JAmbition	Java Ambition
HUTN	Human-Usable Textual Notation
JDBC	Java Database Connectivity
JML	Java Modeling Language
JRE	Java Runtime Environment
JSP	Java Server Pages
JVM	Java Virtual Machine
OCL	Object Constraint Language
PUPPET	Pick UP Performance Evaluation Test-bed
QoS	Quality of Service
SLA	Service Level Agreement
SLAngMon	SLAng Monitor
SOAP	Simple Object Access Protocol
SSM	Service State Machine
SSM	System Under Experimentation
UDDI	Universal Description, Discovery and Integration
WSCoL	Web Service Constraint Language
WSDD	Web Service Deployment Descriptor
WSDL	Web Service Description Language
WS-GUARD	WS-Guaranteeing Uddi Audition at Registration and Discovery
XPATH	XML Path Language
XSD	XML Schema Definition

List of Figures

List of Tables

1 Introduction

The PLASTIC project aims at enabling the development and deployment of mobile adaptable robust services for Beyond 3G (B3G) networks, by providing a comprehensive platform integrating both adequate software methodologies and tools, and the supporting middleware. Service provisioning and development over B3G distributed computing platforms faces numerous challenges in particular for what concerns the verification and validation phase.

The goal of Work Package 4 is to investigate, develop, and integrate with the PLASTIC platform new verification techniques for the B3G distributed computing domain. During the first period of the project we identified a set of verification stages to be exploited to derive a comprehensive testing strategy suitable for the specific application domain. As a concrete realization of this strategy, a set of testing techniques and supporting tools have been developed for each stage. Figure 1.1 provides a detailed view of the different testing stages that make up the PLASTIC testing framework.

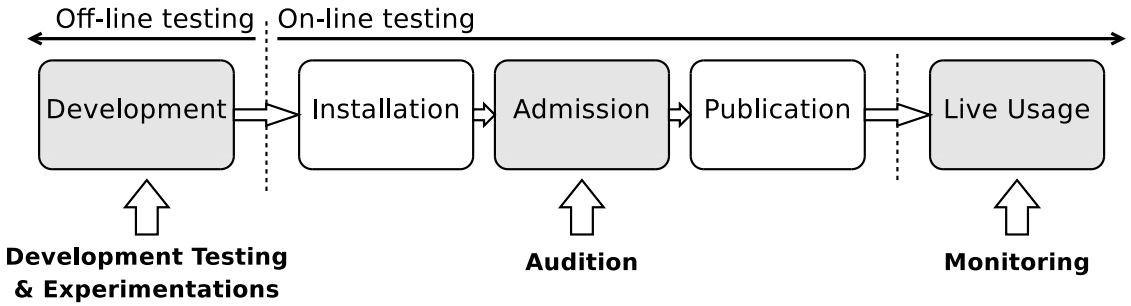

Figure 1.1: PLASTIC Testing stages

For each of these stages, we identified the peculiarities of the application domains that were not adequately covered by existing testing approaches.

An in-depth discussion of the approaches proposed has been given in Deliverable 4.1. For each of the approaches introduced in D.4.1, we recall here the main concepts and then we provide a detailed presentation of the tools that have been developed for each technique. In particular this chapter shortly illustrates the novel aspects of each proposed approach and the key reasons why each of them is particularly suitable for testing in the B3G and service-oriented domain.

1.1 Testing challenges and opportunities

Service-oriented systems and in particular software developed to be deployed over B3G network presents new challenges to software testers. In deliverable D4.1 this topic has been extensively discussed, nevertheless it can be useful to recall some of these challenges also here.

The first challenge that testing have to face in this domain certainly concerns the fact that service integration may not become apparent until run-time. According to the SOA paradigm, services can discover each other at run-time and they can select the partner to interact with based on parameters that are only defined at run-time. Therefore, it is difficult to anticipate which services will interact with a given service at a certain time in the future. At the same time, even if a service can be statically bound to an external service, the lifecycle of the latter could be under the control of a different organisation, so its implementation can change without notice.

Another important issue in the B3G and service-oriented domain is certainly the relevance of extra-functional characteristics. Service clients have no control on the services they use and on

the underlying platform, but at the same time require that certain requirements on the service extra-functional properties are met. This raises the necessity of languages to define agreements among parties. Also, the developers need techniques and approaches to evaluate extra-functional properties of systems built using web-services before the real system is actually deployed. Finally it is ncessary to develop mechanisms to check whether the agreeements are fulfilled at run-time.

Besides the new challenges that must be faced by this new development paradigm, new opportunities can be envisaged.

An extraordinary opportunity is the general increased availability of service specification in a machine-readable format. Certainly, at the moment, only the specification of syntax related specification have been fully standardised. Nevertheless, many works in the literature suggest that services should be complemented with behavioural specifications. In some cases, such specifications could be fruitfully applied in order to derive test cases automatically.

Finally the identification of another intermediate phase in the software lifecycle, i.e. the *registration* phase, suggests the introduction of an additional testing phase in the whole validation strategy.

According to the strategy shown in Figure 1.1, the next sections provide some detail on the tools developed within PLASTIC, highlighting how the challenges are tackled and the opportunities are exploited.

1.2 Development-time testing

Development-time testing refers to activities that are carried out by service developers. As remarked in the previos section, in a B3G service-oriented setting, where services are discovered and integrated only at run-time, it is difficult – if at all possible – to have guarantees on service behaviour. This is particularly true when extra-functional properties are considered. Nevertheless, developers need tools and techniques to assess the quality of a service before its final deployment.

Moreover, services in general may invoke other services in order to carry out the computation requested by the clients. If this invocation is directed to a service that does not refer to stateful resources, then for testing purposes it is possible to use existing and already running services. Conversely, if the invoked service accesses stateful resources, this option must be ruled out and the required services have to be simulated. Within PLASTIC the problems of reproducing predictable run-time environment is addressed providing two different tools, Puppet and Weevil, which allow the developers to reproduce different live usage scenarios.

In particular Puppet can be used to to automatically derive the elements necessary to recreate a predictable “live” environment that is suitable for the evaluation of extra-functional properties. Puppet allows to generate automatically the required services in such a way that they show a “correct extra-functional behaviour” with respect to a give specification. Chapter 3 discusses how to install and use Puppet.

Weevil is meant to ease the reproduction of distributed experimental environments. In particular it permits to recreate expected workload to stimulate the service under test, to remotely deploy the various element required by the experiment, and to collect data during the experiment. The usage of Weevil is illustrated in Chapter 4.

Finally the last testing tool targeting design-time testing of PLASTIC applications, called Ambition, is used to automatically derive test suites for services under development. The key idea of this tool is to exploit as much as possible the behavioural description often available for deployed services. The extreme dynamicity of the service domain suggested to augment service with operational specification in order to characterize services in a richer way. Ambition exploits this kind of information in order to derive test cases that are suitable for service evaluation. Ambition as-

sumes that such specification are available as Service State Machine (SSM), for which a sound theoretical foundation is also being developed. The concrete usage of Ambition is illustrated in Chapter 2.

1.3 Admission testing

As discussed in Section 1.1 one of the main challenges of testing is to test the service in the environment where it will operate at run-time. At the same time, the Service-Oriented Architecture (SOA) foresees the existence of a service broker that is used by services to search and obtain references to each other. The idea of the Admission testing is to have the service undergo a preliminary testing stage (also referred to as audition) whose results will decide the actual registration of the service in the directory.

The intuition of the Admission testing is that the quality of registered services can be increased by granting registration to the directory only to those services that pass the audition testing phase. At the same time this should provide better confidence in the fact that services will interact in a correct way even if they discover each other at run-time.

Admission testing clearly raises issues regarding the invocations to fully-operating services (as opposed to services being auditioned). This may be particularly dangerous if the services invoked are related to stateful resources. In order to avoid side effects resulting from invocations fired in the process of auditioning a service, suitable countermeasures must be taken. Chapter 5 shows how to install and use WS-Guard, which is an implementation of a directory service conforming the UDDI specification that permits to test services before their registration.

1.4 Live-usage verification

Difficulties in applying verification techniques before live usage, suggested to extend the verification phase till run-time. Within workpackge 4 two different activities, aiming at the development of monitoring mechanisms, have been activated. The idea is to add suitable mechanisms to the platform so as to detect violations with respect to the expected behaviour of services.

The first of these approaches, called Dynamo-AOP, focuses on functional behaviour of orchestrated services, and provides support to augment orchestrating services with checks, in order to verify that the orchestrated services bahave as expected. Chapter 6 describes how to install and use Dynamo-AOP.

Another approach in this category supports the monitoring and logging of extra-functional properties for running services. SLAngMon implements a mechanism to parse Service Level Agreement specifications defined in SLAng automatically generates the code of efficient checkers, whose operation is based on timed automata theory. Chapter 7 is devoted to the description of how to set and use SLAngMon.

1.5 Tool download

All the prototypes described in this document can be freely downloaded from:

`http://plastic.isti.cnr.it`

2 Jambition - Model-Based Testing of Web Services

Jambition is a prototype tool to automatically test the functionality of Web Services based on Service State Machine (SSM) specifications. SSMs are similar to UML state machines, but tailored to the domain of Web Services. They have a precise semantics and are especially suited to perform Model-Based Testing. SSMs specify a single port type. The Web Service to be tested is assumed to be passive, meaning that only `request-response` and `one-way` operations are defined in the corresponding WSDL. Jambition supports a subset of the XML-Schema data types.

The next sections briefly motivate and explain the tool and the specifications needed to perform model-based testing of a Web Service port.

2.1 What is Jambition Good for?

Every Web Service provides a set of operations to its potential users. To know which operations are available, an interface specification is needed. Commonly, Web Service interfaces are specified in the *Web Service Description Language (WSDL)*.

For instance, a Web Service representing a warehouse may offer operations to check the availability of products, and to order such. In the first case, a quote-request object is sent, and a quote object is returned. The corresponding WSDL file makes the signature of these operations public. This information is sufficient to connect to the Web Service, and to invoke the operations, but it does not give any kind of semantical information. For instance, the warehouse may only accept quote-requests of a certain quantity. Or it may only allow to order products when the availability has been requested beforehand. Or it may guarantee that every offered quote deals with the same quantity as the requested quote has indicated. WSDL files are not intended to provide such kind of information.

One natural way to extend a Web Service description in this direction is to use state machines. Jambition uses a dedicated variant of state machines which is especially useful for Model-Based Testing – *Service State Machines (SSM)*. Such a state machine can constrain the data as it is passed via the operations, and it gives a legal ordering of the invocations of operations. Hence, properties like the ones stated above for the warehouse can be expressed via SSMs.

Such an SSM model can suit several needs. For instance, it gives a specification of the dynamic aspects of the Web Service invocations. A user of the Web Service knows which operations are allowed to be invoked when. She also knows the restrictions on the date to be sent and received. SSMs are a valuable means to extend Web Service specifications.

Furthermore, an SSM can be used to do automatically test a Web Service. This is what Jambition does. It takes a WSDL and an SSM specification of a Web Service as an input. Based on these it fully automatically generates invocations to the Web Service, receives the returned messages, and checks if this data is conforming to the SSM specification.

Following the warehouse example, Jambition will respect the protocol as it is encoded in the SSM, meaning for instance that it will always check the availability before making an order. It will also respect the data constraints, for instance only making orders with a quantity of at least 3 products. With respect to testing, Jambition will receive the responses of the operation calls it makes, and then check if also the warehouse respects its constraints on the data. For instance, Jambition will check if the warehouse only makes offers for the products requested.

If Jambition spots a failure it will report so and stop the testing. While no failure is found, it does a random walk through the SSM, meaning that when there are several inputs specified, it chooses one randomly. With respect to the parameters of operations, Jambition only chooses data values which respect the constraints (like a quantity ≥ 3). Usually the first solution of the constraint is

chosen (like quantity = 3).

To visualize and to keep track of how the test proceeds, Jambition offers a GUI which monitors the testing events. It is also possible to log the ongoing testing to rotating log files. Furthermore, Jambition can be connected to a free UML sequence diagram editor, which displays the communication between Jambition and the Web Service in real time as a sequence diagram.

The execution of Jambition on a Web Service corresponds to the automatic generation and execution of thousands of test cases within minutes. The only thing needed in addition to the WSDL is the SSM specification. Having that, Jambition can be a great help in testing the functionality of Web Services.

2.2 Tool Architecture and Installation

The relevant concepts of Jambition are depicted in Fig. 2.1. The structural aspects (data types,

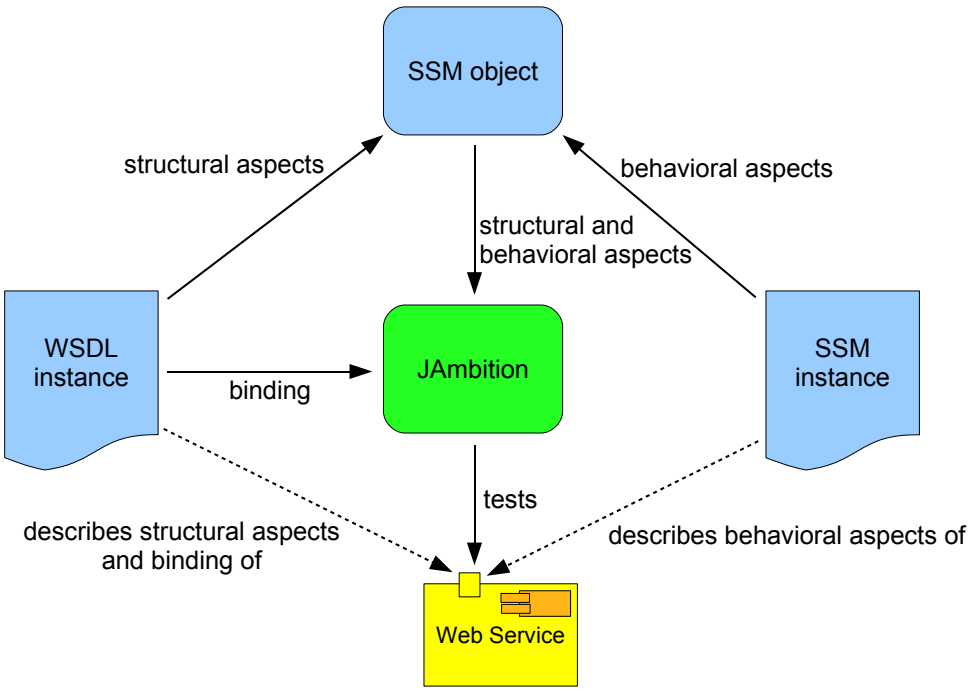

Figure 2.1: Jambition Concepts

messages, operations, port types) of the Web Service to be tested are described in a WSDL instance. The behavioral aspects (states of the Web Service, ordering of invocations, constraints on the data flow) are described in an SSM instance. Both aspects are combined together in an SSM object. This object is the main reference of Jambition to do the automatic testing. Furthermore, the binding information of the WSDL file (service, port) is used to physically connect to the Web Service.

Concretely, Jambition needs four inputs, see Fig. 2.2.

1. The URL of the WSDL instance
2. The service name of the Web Service to be tested as being given in the WSDL instance (since a single WSDL can define several services)
3. The port name of the Web Service to be tested as being given in the WSDL instance (since a single WSDL can define several ports per service)

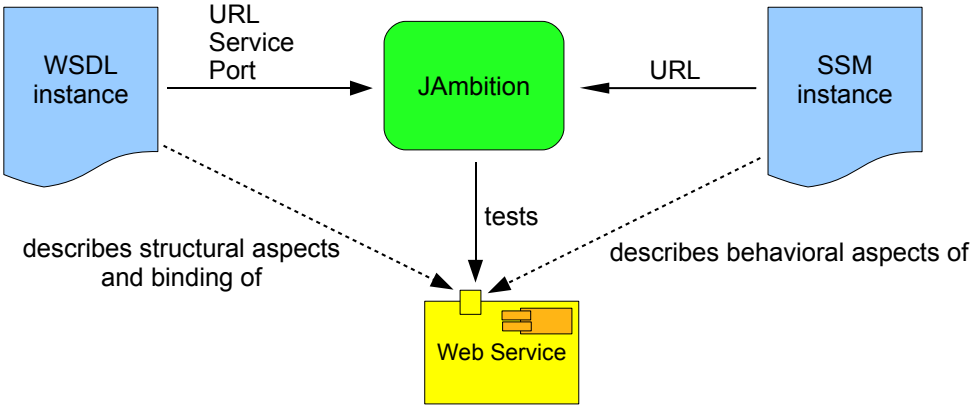

Figure 2.2: Jambition Inputs

4. The URL of the SSM instance

Based on these inputs, the SSM object is generated and the testing can be started.

2.2.1 Installation

Jambition is written in Java 6, hence it needs a Java 6 JRE installed. It comes as a `Jambition.zip` archive. Within the archive there is the `Jambition` directory. Just extract this directory to a desired place. Within the directory there is the `Jambition.jar` archive. A simple double click should start Jambition on all common operating systems.

Jambition needs access to a constraint solver to do the testing. It uses GNU Prolog for this purpose. It is necessary to install and run the `treeSolver` program, which opens a socket connection to the constraint solver. `treeSolver` can be freely downloaded here:

`www.cs.ru.nl/~lf/tools/treesolver/`

The web page provides executables for Windows and Linux. Also the source code is available, so it can be compiled for several other operating systems, see `www.gprolog.org/#platform`. The `treesolver` gets as only input the socket-port to open. Before starting Jambition, invoke `treeSolver` for instance with the port number `60002`. You should get something like:

```
user@pc-plastic:~$ treeSolver 60002
treeSolver V180607
By Lars Frantzen (lars@frantzen.info)
A socket interface to the constraint solver of GNU Prolog
GNU Prolog is copyright (C) Daniel Diaz
Listening on pc-plastic port 60002
```

A tool which is not mandatory for Jambition, but useful to visualize the testing process, is the `Quick Sequence Diagram Editor - sdedit`. It is written in Java and can be freely downloaded here:

`sdedit.sourceforge.net/`

After `sdedit` is launched, turn on its real-time server by choosing in the `File` menu `Start/stop RT server`. For the socket-port accept the suggestion `60001`.

Another tool which is not mandatory for Jambition, but useful to visualize the SSM, is `dot`. It is part of the `Graphviz` toolsuite. Binaries for Windows and Linux can be freely downloaded here:

`www.graphviz.org/`

Now Jambition can be launched, see the screenshot in Fig. 2.3. The first thing to tell Jambition is

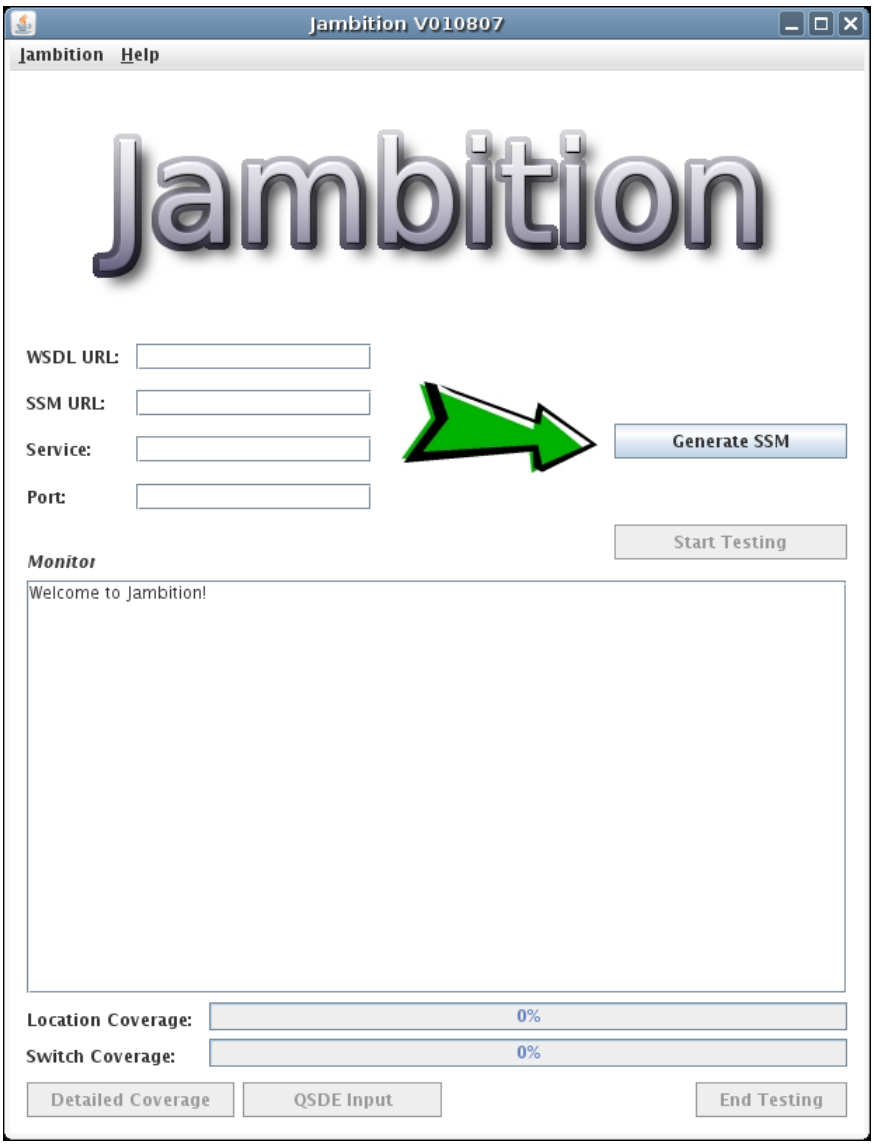

Figure 2.3: Jambition Initial Screenshot

where to find the WSDL and SSM specifications of the Web Service to test. The following sections will give further details about these files.

This sections ends with an explanation of the possible options of Jambition. Via the `Jambition` menu the `Preferences` window can be displayed, see the screenshot in Fig. 2.4. We explain the options from top to bottom.

At the top the socket of the `treeSolver` can be modified. The default values here are port `60002` on the local host. The option `Always New Inputs` is experimental, it means that whenever Jambition has to find a solution for a constraint, it tries to find a new solution. For instance, the constraint $\texttt{quantity} \geq 3$ will always be solved by choosing $\texttt{quantity} = 3$, since this is the first solution found. This may not always be desirable, turning this option on will find the solutions 3,4,5,et cetera. But this generates very huge and inefficient constraints, so this option should be treated with care.

Next the `sdedit` socket can be modified. The `Enabled` option allows to turn on and off the usage of `sdedit`. This means that Jambition does not send the testing messages to the `sdedit`

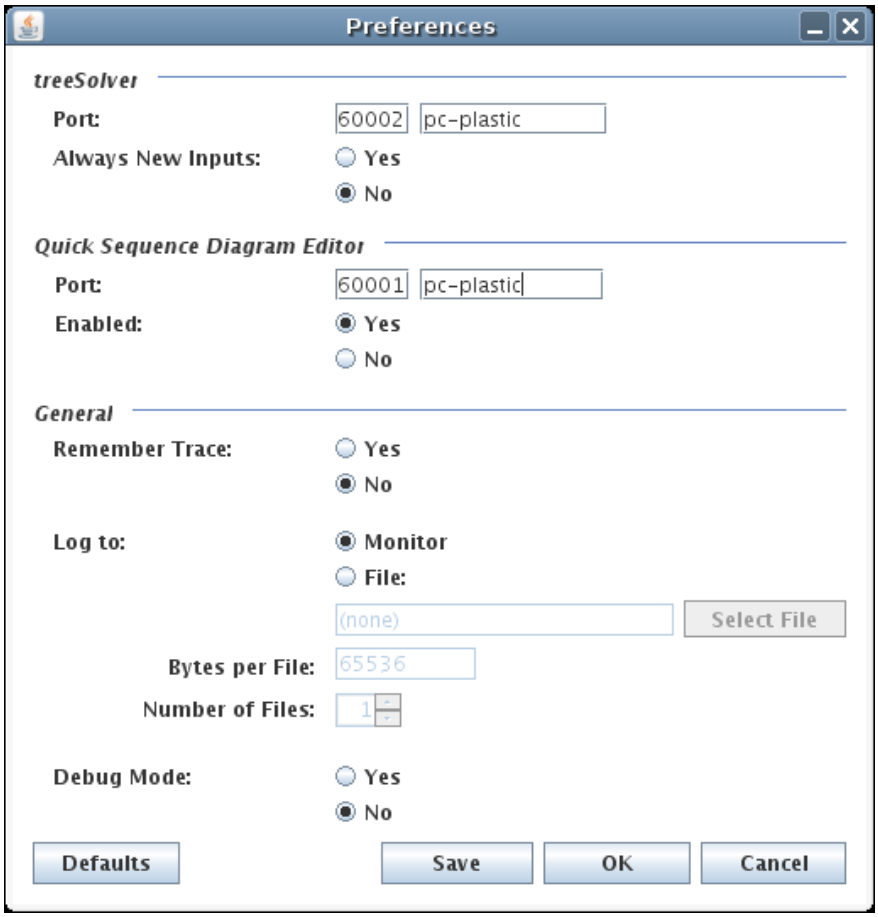

Figure 2.4: Jambition Preferences Screenshot

tool, which can nicely display those in form of a sequence diagram. Still, it is possible to generate the input for `sdedit` later, for instance when a failure was found. To do so it is necessary to turn on the `Remember Trace` option.

Jambition logs per default to its `Monitor` window. This can be turned off. Note that logging to the `Monitor` increasingly consumes memory since the window remembers all test events. Instead, or additionally, it is possible to log to rotating log files. The name of the file, the bytes per file, and the number of files can be set.

Finally, the `Debug Mode` can be turned on or off. This means that additional information is logged, for instance the exchanged SOAP messages. All options can be reset to its default values, and saved.

2.3 Web Service Description Language

The structural aspects (data types, messages, operations, port types) of the Web Service to be tested are described in a WSDL instance. Also the binding information is encoded here. Normally, WSDL files are automatically generated by the IDE which is used to develop the Web Service.

To define the data types of the message parameters, commonly the data types of XML Schema are used in the WSDL.

2.3.1 XML Schema Simple Types

Jambition supports the following simple types:

- xs:boolean
- xs:integer
- xs:string
- xs:enumeration (a restriction on xs:string)
- xs:double

Two limitations are important to note. Firstly, Jambition can only deal with positive integers. This is due to the fact that GNU Prolog (which is used by the `treeSolver`) can only solve constraints over positive integers.

Secondly, the support for doubles is very limited. This is due to the fact that constraint solving over double variables is not feasible in the way Jambition needs it. What Jambition does is supporting double variables with a fixed precision. For instance, doubles with a fixed precision of two positions after the decimal point can be used to model amounts of money. Such "fake" doubles can be mapped internally to integers. But this mapping easily leads to large integers, which may be too large again for the constraint solver. In its current version Jambition assumes a fixed precision of two positions after the decimal point.

2.3.2 XML Schema Complex Types

Complex types are used to represent classes of Object-Oriented languages in the WSDL. For instance, given a Java method which returns an object of class c. The class c has two fields. If this method is exposed as a Web Service method, the corresponding WSDL can define a complex type for c which is a sequence of two elements representing the two fields. Jambition supports such complex types.

Jambition does not support elements in a sequence which are of the kind `maxOccurs = "unbounded"`. Such elements are used to model, for instance, lists.

2.3.3 WSDL Operations

Four kinds of operations can be defined in a WSDL:

- request-response
- one-way
- solicit-response
- notification

Jambition does only support the first two kinds — request-response and one-way. We call such a service a passive service since it does never send messages actively, only after being requested.

2.3.4 SOAP Binding

The XML parameter mapping is defined by the `soap:body use` and the `soap:binding style` attributes. Jambition assumes that:

- `use = "encoded"` or
- `use = "literal"` and `style = "document"` and each message has exactly one part called `parameters`.

2.4 Service State Machines

The behavioral aspects (states of the Web Service, ordering of invocations, constraints on the data flow) are described in an SSM instance. An SSM is a state machine, consisting of typed variables, states, and transitions between the states. Every transition consists of these elements:

1. The state where the transition starts.
2. The name of a WSDL operation.
3. The kind of the corresponding message. This can be either `input` or `output` for request-response operations, and is always `input` for one-way operations.
4. A guard which restricts the conditions under which the transition can be executed. The language of the guard is described below.
5. An update of the variables.

Every SSM has in initial state.

2.4.1 The Language of the Guards

There are five kind of *literals*, i.e. constants of a certain type:

- **Integer literal** – a numberstring (e.g. 3423432)
- **Boolean literal** – `true` or `false`
- **Double literal** – numberstring followed by a dot . and optional another numberstring (e.g. 23.2324)
- **String literal** – string between double quotes, e.g., “Hello World”
- **Enumeration literal** – a string of the form *element@enumname*, where *enumname* is the name of an enumeration type, and *element* one of its elements, e.g. `BOOK@product`

An *identifier* is a name of a variable. If a variable has a complex type, then one can refer to the fields by the dot-notation, e.g. `q.product`. Every literal and identifier are expressions which have a corresponding type, i.e. one of integer, boolean, float, string, complex or enumeration. One can combine literals and identifiers via *operations*, yielding more complex expressions, which again have a type. Next we mention the operations currently supported by the parser. By *number* we mean integers or doubles. Note that here by writing for instance *boolean* or *number* we refer to any expression of this type.

First we mention the operations which lead to an expression of type boolean:

- == compares booleans, numbers, strings and enumeration instances for equality
- != compares booleans, numbers, strings and enumeration instances for inequality
- < compares numbers for being *less than*
- <= compares numbers for being *less than or equal*
- > compares numbers for being *greater than*
- >= compares numbers for being *greater than or equal*
- && the logical *and* of two booleans
- || the logical *or* of two booleans
- ! the logical *not* of a boolean

Next we mention the operations which lead to expressions of type integer or double:

- ++ increments (+1) an integer
- -- decrements (-1) an integer
- + adds numbers or strings (i.e., a concatenation of two strings)
- - subtracts two numbers
- * multiplies two numbers
- % The remainder operator for numbers
- / divides two numbers

A guard is an expression of type boolean. Sometimes one wants to express an empty guard, meaning an expression which has the logical value *always true*. This cannot be done by giving an empty string, nor by writing just the boolean literal `true`, one has to write `true == true` here. One also cannot just mention the name of a variable of type boolean like `p` or `!p`, it is necessary to write `p == true` or `p == false`.

2.4.2 Variable Updates

There is another operator we have not mentioned yet, it is the assignment operator =. Here we assign a value to a variable. An update can consist of several such assignments, but one can also have no assignment at all by giving the empty string “”, meaning that all variables remain unchanged. All assignments must be terminated by a semicolon ;.

2.4.3 Designing SSM in MagicDraw

All the files mentioned in this section can be found in the directory `MagicDraw`.

Using the MagicDraw modeling and CASE tool one can develop SSMs graphically, by drawing the corresponding SSM UML diagram and transforming the exported data. Note that the data structures used in the SSM and even the services themselves can be modeled in UML complying with the PLASTIC conceptual model developed in WP2. Currently it is a requirement for the data structures to have an UML representation.

Configuring MagicDraw
Before starting the design process, MagicDraw should be configured to use the specialized profiles and plugins.

- copy the file SSMProfile.mdzip in `<MAGICDRAW_HOME>/profiles`.
- copy the folder `MP SSM Diagram` and its contents from the distribution to the `<MAGICDRAW_HOME>/data/diagrams` directory. Edit the `descriptor.xml` just copied and set the modules location for the `SSMProfile` and `UML_Standard_Profile` correctly.
- (re)start MagicDraw.

Creating the Basic Classes and Interfaces
The complex (not simple) datatypes and the service ports used in the SSM have to be defined and accessible from MagicDraw. There are two options:

- they can be defined in MagicDraw in the same project, in another project, or
- they can be imported in MagicDraw, from another case tool, or, in the future, from a WSDL file.

In this section we describe the first method. The complex datatypes currently supported are literal enumerations and classes, with a recursive definition for the class attributes (they can be simple types, literal enumerations and classes.) These should be defined in standard class diagrams. In addition, for each service port an interface has to be defined, which contains the methods published by the service. In the case that one is going to generate the service stubs from the model, you should apply some stereotypes here, and give some additional information.
Create the interface and/or open its specification dialog as follows:

- Set the `Applied stereotype` property to `<<SSMInterface>>` (defined in SSMProfile)
- Select the `Tag` tab on the dialog. You will see in the first part of the right pane the list of tags (properties) associated with the SSMInterface stereotype. By clicking on a tag, its value will be displayed, if it has one, or you will get the option to create a new value. For example, to assign a value to the `Namespac` tag property, click on the property, then choose the `Create` button. In the right part you can then enter the namespace value. Clicking the `+` button will set the tagged value entered.
- Now you can add the operations to the interface. For each operation, you have to enter the following information in the specification:
  - On the General tab
    - Name
    - Applied stereotype: select `<<WebMethod>>` (defined in SSMProfile)
    - Type – should be set to the value of the return type of the call. This is the type of the response message.
  - Parameters tab. Here you specify the parameters (of the input message). For each parameter, you define the name, type and direction (in, out, inout or return). Note: you have to define a parameter for the return value.
  - Tags tab On the `<<WebMethod>>` tag group, optionally you can set:
    - inputMessage: the name of the input SOAP message associated with the operation
    - outputMessage: the name of the corresponding output SOAP message

Creating the Service State Machine

The steps of creating an SSM diagram are explained next. In the Containment tree (Containment tab in the upper side of the Browser) select the `Data` folder, or the folder in which you want to embed the SSM. From the context menu (right click), select `New Diagram - MP SSM Diagram`. Alternatively, from the `Diagrams` menu, choose `Custom Diagrams | MP SSM Diagram`. If MP SSM Diagram is not found within the list of diagrams, then MagicDraw was not configured correctly (see the section *Configuring MagicDraw*).

If everything is ok, a new SSM object is created and a new MP SSM diagram is also built as its child object in the containment tree. You can rename both the object and the diagram in the containment tree.

In the right pane, a new tab is created with an empty MP SSM diagram. The middle toolbar will contain 2 button groups: `Common` and `MP SSM`. You should draw the diagram using the shapes found here. There are 2 basic shapes you will use: the State and the SSM Transition.

The State is a normal state used in standard state diagrams. We represent here the states of the corresponding web service. A synchronous request-response message is represented by 2 states, corresponding to one request and one response transition.

The SSM Transition is a state transition having the `<<SSMTransition>>` stereotype applied. When creating such a transition, the following information should be entered:

- Operation (in the Trigger group)
  This is the operation used in the SOAP interface. To see this property, first choose `CallEvent` as the trigger EventType. Then the operation should be selectable from the dialog (activated by the right menu button on the value), as an operation of an SSMInterface (defined at the class level, as discussed above)
- Guard (in the Transition group)
  This is the guard of the transition. By opening the Edit dialog (with the left-most button), you can write a large guard using a text box. Choose `Language = English`. For better display, you might want to format the text using new-lines.
- In the `Tags` menu (select `Tags` in the left tree), edit the `kind` and `update` properties.
  - `Kind` is the message kind, and a mandatory property. You can choose a value of `INPUT`, `OUTPUT` or `UNOBSERVABLE`.
  - `Update` represents the variable update command. Enter its value as a string.

After creating the MP SSM diagram, export it in XMI for further use. It is a good idea to create a directory first, where to save the files exported by the project. Select `File | Export | EMF UML2 (v1.x) XMI`. The files exported will be used by the Minerva tool to build an SSM, which will be passed to Jambition for testing the web service.

2.4.4 Defining SSM directly in XML

Jambition expects an instance of an XML schema for SSM called `SP-SSM`. After giving the the full schema it is further explained below.

An XML Schema for SSM

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace=
            "http://frantzen.info/testing/ssmsimulator/schema/"
            xmlns:tns="http://frantzen.info/testing/ssmsimulator/schema/"
            elementFormDefault="qualified">

    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            A schema for Singleport Symbolic State Machines (SP-SSM).
            Version: 010807
            Copyright (C) 2007 Lars Frantzen
        </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="spssm" type="tns:SPssm"></xsd:element>
    <xsd:complexType name="SPssm">
        <xsd:sequence>
            <xsd:element name="wsdlURI" type="xsd:anyURI"/>
            <xsd:element name="porttype" type="xsd:token"/>
            <xsd:element name="states">
                <xsd:complexType>
                    <xsd:sequence maxOccurs="unbounded">
                        <xsd:element name="state" type="xsd:token"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="initialState" type="xsd:token"/>
            <xsd:element name="variables">
                <xsd:complexType>
                    <xsd:sequence maxOccurs="unbounded">
                        <xsd:element name="var" type="tns:Variable"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="switches">
                <xsd:complexType>
                    <xsd:sequence maxOccurs="unbounded">
                        <xsd:element name="switch" type="tns:Switch"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Variable">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:token"/>
            <xsd:element name="type" type="xsd:token"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Switch">
```

```
        <xsd:sequence maxOccurs="1">
            <xsd:element name="from" type="xsd:token"/>
            <xsd:element name="operationName" type="xsd:token"/>
            <xsd:element name="kind" type="tns:Kind"/>
            <xsd:element name="restriction" type="xsd:string"/>
            <xsd:element name="update" type="xsd:string"/>
            <xsd:element name="to" type="xsd:token"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="Kind">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="input"/>
            <xsd:enumeration value="output"/>
            <xsd:enumeration value="unobservable"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

Explanation of the Schema
We describe its elements:

- `<wsdlURI>` Every SP-SSM instance specifies a port-type which is given in an external WSDL file. This external file is referenced by this tag via an URI. For Jambition, this tag has no relevance.
- `<porttype>` Here the specific port-type from the referenced WSDL is named (there could be several in the WSDL). For Jambition, this tag has no relevance.
- `<states>` Here all the state-names are listed.
- `<initialState>` The name of the initial state is given here.
- `<variables>` Here all the variables are listed.
  - `<name>` Every variable has a name.
  - `<type>` Every variable has a type. This is either one of the XML schema simple types like `xs:boolean`, or the name of a complex type or an enumeration. In case of a complex type or an enumeration the namespace indicator must be the same as the target namespace of the referenced WSDL.
- `<switches>` Here the SSM transitions, called switches, are given.
  - `<from>` This is the state-name where the switch starts.
  - `<operationName>` This is the name of the operation (as defined in the WSDL) where the switch-message belongs to.
  - `<kind>` This is he kind of the message which is transported by this switch. For every operation this can be either `input` or `output`. A special third kind is `unobservable` which denotes an internal message which is not observable at the interface.
  - `<restriction>` This is guard.
  - `<update>` This is the variable update.
  - `<to>` This is the state-name where the switch leads to.

Note that especially when dealing with `<restriction>` and `<update>` elements there may be the need to map special characters like '&' or '<' to a different representation in the schema instances.

2.5 Future Improvements

SSMs can also be used to model the communication between several Web Services. To do that they have to embrace the message flow at several ports. Testing based on such multi-port SSM would allow to test more complex scenarios like coordinated and composed Web Services.

Many Web Services use lists as data types. As mentioned above, lists correspond to an unbounded sequence in XML. Supporting lists is another eligible future step.

With respect to the testing itself, the purely random approach might not always be satisfactory. Specific coverage criteria can be conceived.

The constraint solving can be more flexible and general. For instance, dealing with negative integers can be achieved by changing the underlying constraint solver.

3 Puppet

This chapter is supposed to be a guideline on the use of the PUPPET . Please refer to [12] for the detailed description of the whole approach, the architectural description of the proposed implementation, the tools and the standard that have been used, or for any kind of motivation of the work.

3.1 Installation

Unzip the archive **PuppetD4.2.tgz**.

The directory structure is the following:

- PuppetD4.2
  - puppetLibs
  - externalLibs
  - puppet.jar
  - example
  - xml

The directory **PuppetD4.2/xml** holds the definitions mapping of the WS-Agreement statements into th Java code. The directories **PuppetD4.2/externalLibs** and **PuppetD4.2/puppetLibs** hold the libraries required by PUPPET in order to run. As your preference, you would append the name of the **.jar** files in **PuppetD4.2/puppetLibs** and **PuppetD4.2/externalLibs** to the Java CLASSPATH variable.

3.2 Required Libs

The following JAR files are required in order to compile and to launch PUPPET :

Apache Axis Libs : axis-ant.jar, axis.jar, commons-discovery.jar, commons-logging.jar, jaxrpc.jar, jsr173_1.0_api.jar, log4j-1.2.8.jar, wsdl4j.jar, saaj.jar [1]

Apache XMLBeans Libs : xbean.jar [4]

String Template Libs : stringtemplate.jar [6]

INI4J Libs : ini4j.jar, ini4j-compat.jar [5]

Other Libs : antlr-2.7.7.jar, xercesImpl.jar, xmlsec.jar, mail.jar, activation.jar, java-getopt-1.0.13.jar

PuppetD4.2.tgz archive includes a version of these libraries in the directory **PuppetD4.2/externalLibs**

3.3 Usage

Let us assume that the variable **CLASSPATH** of the JVM you are executing includes the JAR files listed in Section 3.2 and those contained in **PuppetD4.2/puppetLibs**. PUPPET usage is:

java -cp $CLASSPATH:puppet.jar puppet.Puppet <IniConfigurationFile>

3.4 Writing an Agreement

In PLASTIC there are two possible way to write out a WS-Agreement specification for PUPPET . The former is to write it directly according to the indications given in [12]. The latter is to exploit the PLASTIC's editor of SLA as explained in the following.

According to the PLASTIC conceptual model definition [22], SLAng is adopted as the reference SLA language in the project. This means that the specific implementations of the various environments should consider to manage at least QoS annotations expressed in SLAng.

SLAng [24] defines an abstract syntax for the agreement. Such syntax would be instantiated in several concrete syntax. Each concrete syntax refers to a given kind of specification. For example in [17] the SLAng concepts were expressed using the HUTN (Human-Usable Textual Notation) as a concrete syntax.

The concrete syntax of SLAng could also refers to other languages for SLA specification. In that sense, a WS-Agreement specification could be seen as a concrete instantiation of the SLAng's abstract syntax. Note that such association is valid under the assumption that the two specifications predicate on the same kind of concepts.

In deliverable D2.2, the consortium presents a tool support for SLAng. In particular, it is an Eclipse-based editor for SLAng, in the form of an Eclipse plugin. The joint work between WP2 and WP4 developed an extension to the SLAng editor including a syntactic translation engine that generates WS-Agreement specification. In this case, the domain specific parts we developed for the WS-Agreement container language describe the same concepts on which SLAng predicates with the same semantics defined in [17]. The output produced by the plugin extension of the SLAng editor could be used as input for PUPPET .

3.5 Terms in the Agreement and Generation Process

In WS-Agreement [23], an agreement specification is composed by one or more terms. These terms are groupped by logic connectors. Connectors express whether how may of the contained terms must be fulfilled to take the agreement as a whole fulfilled. The connectors that are currently included in the WS-Agreement specification are: All (logic AND), OneOrMore (logic OR) and ExactlyOne (logic XOR).

Different scenarios could be considered defining a set of terms in the agreement that are assumed fulfilled. This set of terms enables in PUPPET the generation of the Java code emulating the extra functional behavior. PUPPET loads the list of these terms parsing an XML file. The value of each element in the XML file refers to the name of a term in the WS-Agreement specification. The example in Table 3.1 shows how to enable the code generation for term named **WarehouseGT** in the agreement specification.

3.6 Configuration File

The input PUPPET required by pupped have to be specified in an input configuration file. The configuration file is supposed to compile with the standard INI File Format. In such file, PUPPET looks for the section named **[mainSection]**. PUPPET loads his parameters as specified in the configuration holded by this section.

The parameters that could be specified into the input configuration file are:

wsdlPath : It is the path to the directory holding the WSDL specifications of all the services whose emulators would be generated by means of PUPPET . Required.

```
<wsag:AgreementOffer xsi:schemaLocation="..." xmlns:wsag="...">
...
    <wsag:GuaranteeTerm wsag:Name="WarehouseGT"
     wsag:Obligated="ServiceProvider">
     ...
    </wsag:GuaranteeTerm>
...
</wsag:AgreementOffer>
```

```
<tns:TrueGTList xmlns:tns="..." xmlns:xsi="..." xsi:schemaLocation="...">
...
 <tns:GTItemName>WarehouseGT</tns:GTItemName>
...
</tns:TrueGTList>
```

Table 3.1: Enabling the code generation in PUPPET

targetPath : It is the path to the directory where PUPPET will dump the generated stubs. Required.

JarPath : It is the path to the required Libs in Sec. 3.2[1]. Required.

wsaFilename : It is the name of the WS-Agreement file describing the agreements among the considered services. If it is not specified, PUPPET would look for a file named: **agreement.xml**. Optional

trueTermsFilename : It is the name of the file holding the terms of the agreement that have to be considered as fulfilled. As described in Sec 3.5, for each term in the agreement, PUPPET will generate code that emulates an extra functional behavior if and only if the term is fulfilled. If this filename is not specified, PUPPET would look for a file named: **gtTrueItemList.xml**. Optional

wsaPath : It is the path to the directory holding the WS-Agreement file. Required.

trueTermsPath : It is the path to the directory holding the *trueTermsFilename*. If it is not specified, PUPPET would assign to this parameter the path to the directory holding the WS-Agreement file (*wsaPath*). Optional.

qcMappingFilename : It is the path to the file holding the template mapping of the Qualifying Conditions in WS-Agreement on to the the Java code that will be generated. PUPPET already includes a predefined mapping file. Even though it is possible to change this mapping, we strongly discourage from changing it. Optional.

sloMappingFilename : It is the path to the file holding the template mapping of the Service Level Objectives in WS-Agreement on to the Java code that will be generated. PUPPET already includes a predefined mapping file. Even though it is possible to change this mapping, we strongly discourage from changing it. Optional.

3.7 Example

In this section, an example scenario on how to describe a WS-Agreement specification for the eHealth application domain.

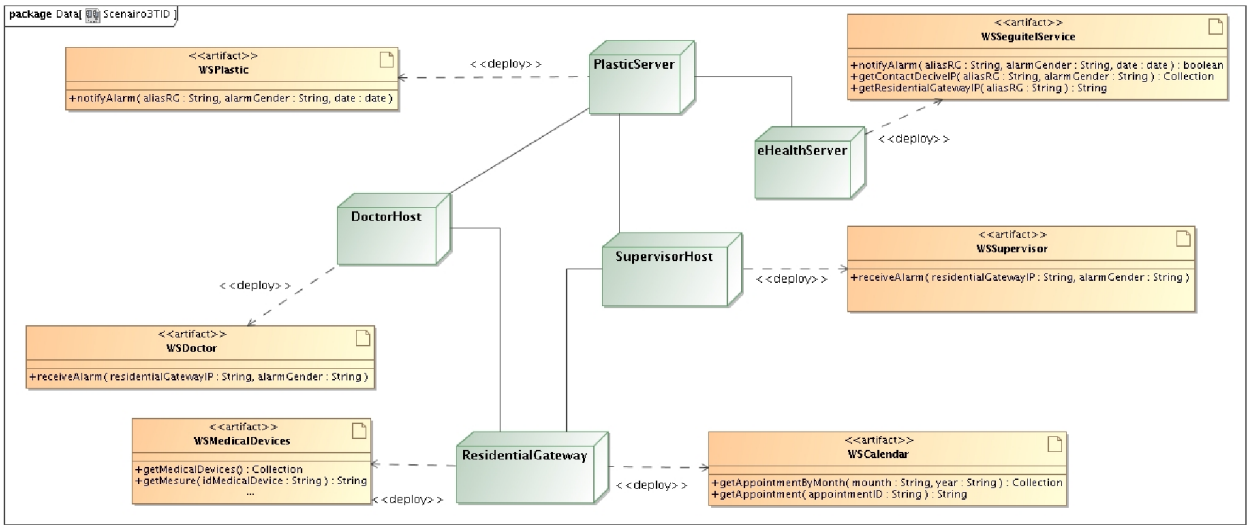

Figure 3.1: Scenario 3 Deployment Diagram

3.7.1 Scenario Description

The scenario is structured as depicted in Figure 3.1. Five web services are involved in this example:

WSPlastic : Is the Web Service that interfaces the current eHealth system with the new Plastic environment. In this scenario description we only refers to one of the possible operations and feature that it exports. Specifically the operation **notifyAlarm** is aimed at both collect and process the alarm messages coming from the eHealth part of the application that runs on Plastic. It is invoked when an alarm is raised. It takes as input the name of the Residential Gateway where the alarm comes from, the gender of the alarm and the date when it was raised.

In the scenario, the WSPlastic represents the new service that have to be tested. Puppet here is used to automatically build that portion of the system that interact with the service under test (i.e. WSPlastic)

WSSeguitelService : This Web Service represents the current eHealth application. It exports the following operations: **notifyAlarm**, **getResidentialGatewayIP**, **getContactDeciveIP**. **notifyAlarm**, takes as input name of the Residential Gateway where the alarm comes from, the gender of the alarm and the date when it was raised. Both the other two exported operations take as input the logic name of a Residential Gateway. Thus **getResidentialGatewayIP** returns the IP address associated with the input label while **getContactDeciveIP** the list of IP addresses that should be contacted in case of alarm.

WSDoctor : It is a service deployed on the device that a doctor use to interface with the eHealth system. The hosting device could be either a usual wired device as a PC or a mobile and wireless device such as a smart phone. The Web Service exports the **receiveAlarm** operation. Such operation take as input both the IP address of the Residential Gateway where the alarm was raised and the gender of the alarm.

WSSupervisor : It is a service deployed on the device that a supervisor use to interface with the eHealth system. The supervisor of a patient is a person that could assist the patient for not

[1] In the future releases it would be deprecated

critical situation. In those cases that are classified as not critical, some kind of alarm could be forwarded to the supervisor instead of the doctor.

As in the case of the doctor, also here the hosting device could be either a usual wired device as a PC or a mobile and wireless device such as a smart phone. The Web Service exports the **receiveAlarm** operation. Such operation take as input both the IP address of the Residential Gateway where the alarm was raised and the gender of the alarm.

WSMedicalDevice : Each Residential Gateway controls several hardware medical devices. Each medical device it is identified on the Residential Gateway by means of a unique identification code.

This Web Service interfaces the medical devices hosted by the Residential Gateway on the Plastic Network. It exports two operations. **getMedicalDevices** returns a collection of the controlled medical devices. To control an eHealth parameter monitored by a medical device, the web service has to be invoked on the **getMeasure** operation providing the id code of the medical device as input.

WSCalendar : Each Residential Gateway holds the lists of the periodic appointments that a supervisor plans with the patient. This Web Service interfaces the Plastic Network to this feature exporting the methods : **getAppointmentByMonth**, and **getAppointment**. The former gives information on the appointments already scheduled in a given month of a year. The latter is used to create a new one.

In scenario, the Web Services described above are supposed to be deployed on different and distributed platforms. Specifically, WSPlastic is deployed on a **PlasticServer**, while the WSSequitelService is supposed to run on the **eHealthServer**. Nevertheless, in principle the two server could be also the same.

3.7.2 Actors Interactions in the Scenario

This section provide a brief description on how the Web Services described in Section 3.7.1 interact. Figure 3.2 reports a UML Sequence Diagram describing these interactions.

When a alarm is notified to the **WSPlastic**, as first step it forwards the event to the **WSSequitelService** invoking the **notifyAlarm** method. The Plastic Web Service also invokes the **getResidentialGatewayIP** obtaining the IP address of the Residential Gateway where the alarm where raised. Thus, it gets the list of the IP addresses that must be contacted invoking the **getContactDeciveIP** on **WSSequitelService**. The obtained list depends on the gender of the received alarm.

Due to the kind of contact list the **WSPlastic** starts to invokes the appropriate Web Service. The Web Services to invoke are supposed to be deployed and reachable by means of the address list. This phase it will continue until any confirmation of the handled alarm is obtained either by the doctor or the supervisor.

In the following, the case where the alarm gender is an emergency (i.e. “EMERGENCY”). The **WSPlastic** extracts an IP address from the address list. Then, it invokes the **receiveAlarm** on **WSDoctor** using the IP address as endpoint. If the target Web Service decide to accept the alarm handling the **WSPlastic** ends considering the problem solved. On the other hand, if **WSDoctor** does not accept to handle the notification or due to a QoS agreement violation (see Section 3.7.4), **WSPlastic** proceeds extracting the next endpoint of the target Web Service.

When the doctor agrees on handle the alarm, they contact the **WSMedicalDevices** deployed on the referred Residential Gateway. So far, the **WSDoctor** collects the list of the medical devices controlled by the Residential Gateway. The monitoring of the patient parameters is performed

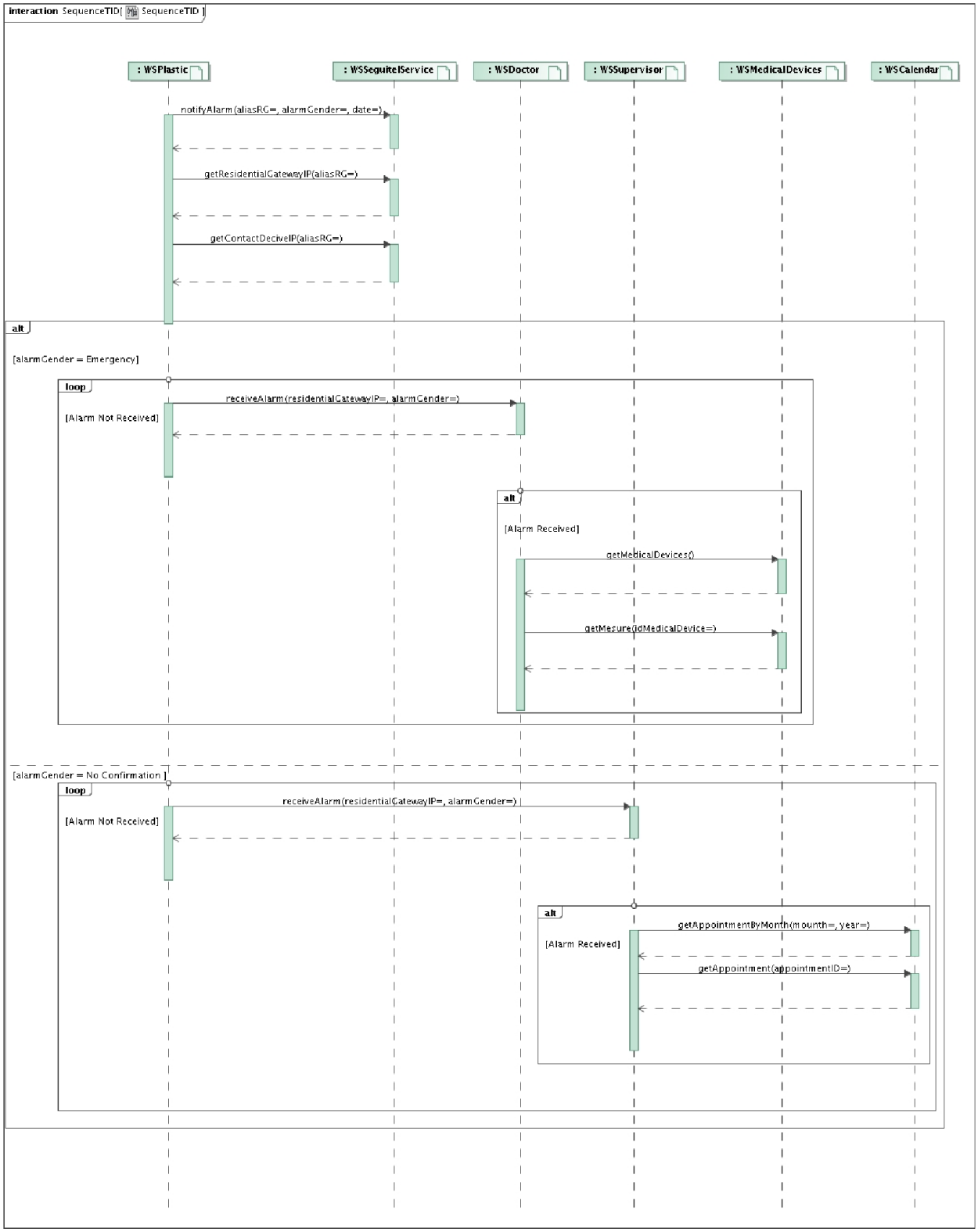

Figure 3.2: Scenario 3 Sequence Diagram

invoking the **getMeasure** method on those devices that are considered important for the clinical status.

In the following, the case where the alarm gender is not critical (i.e. “NOT CONFIRMATION”). The **WSPlastic** extracts an IP address from the address list. Then, it invokes the **receiveAlarm** on **WSSupervisor** using the IP address as endpoint. If the target Web Service decide to accept the alarm handling the **WSPlastic** ends considering the problem solved. On the other hand, if **WSSupervisor** does not accept to handle the notification or due to a QoS agreement violation (see Section 3.7.4), **WSPlastic** proceeds extracting the next endpoint of the target Web Service.

When the supervisor agrees on handle the alarm, they contact the **WSCalendar** deployed on the referred Residential Gateway. So far, the **WSSupervisor** queries the Residential Gateway to schedule an appointment with the patient.

3.7.3 Abstract WSDL Description

In the following, the WSDL abstract definition of the services described in Figure 3.1.

3.7.3.1 WSDoctor

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions targetNamespace="http://localhost:8080/axis/services/WSDoctor" xmlns:apachesoap
       ="http://xml.apache.org/xml-soap" xmlns:impl="http://localhost:8080/axis/services/WSDoctor"
       xmlns:intf="http://localhost:8080/axis/services/WSDoctor" xmlns:soapenc="http://schemas.
       xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="
       http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3  <!--WSDL created by Apache Axis version: 1.4
4  Built on Apr 22, 2006 (06:55:48 PDT)-->
5
6    <wsdl:message name="receiveAlarmRequest">
7
8      <wsdl:part name="residencialGatewayIP" type="soapenc:string"/>
9
10     <wsdl:part name="alarmGender" type="soapenc:string"/>
11
12   </wsdl:message>
13
14   <wsdl:message name="receiveAlarmResponse">
15
16   </wsdl:message>
17
18   <wsdl:portType name="WSDoctor">
19
20     <wsdl:operation name="receiveAlarm" parameterOrder="residencialGatewayIP␣alarmGender">
21
22       <wsdl:input message="impl:receiveAlarmRequest" name="receiveAlarmRequest"/>
23
24       <wsdl:output message="impl:receiveAlarmResponse" name="receiveAlarmResponse"/>
25
26     </wsdl:operation>
27
28   </wsdl:portType>
29
30   <wsdl:binding name="WSDoctorSoapBinding" type="impl:WSDoctor">
31
32     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
33
34     <wsdl:operation name="receiveAlarm">
35
36       <wsdlsoap:operation soapAction=""/>
37
38       <wsdl:input name="receiveAlarmRequest">
39
40         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
               http://wsScenario" use="encoded"/>
```

```
      </wsdl:input>

      <wsdl:output name="receiveAlarmResponse">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
            http://localhost:8080/axis/services/WSDoctor" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

  </wsdl:binding>

  <wsdl:service name="WSDoctorService">

    <wsdl:port binding="impl:WSDoctorSoapBinding" name="WSDoctor">

      <wsdlsoap:address location="http://localhost:8080/axis/services/WSDoctor"/>

    </wsdl:port>

  </wsdl:service>

</wsdl:definitions>
```

3.7.3.2 WSCalendar

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/services/WSCalendar"
    xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://localhost:8080/axis/
    services/WSCalendar" xmlns:intf="http://localhost:8080/axis/services/WSCalendar"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.
    org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.
    org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema targetNamespace="http://localhost:8080/axis/services/WSCalendar" xmlns="http://www.w3.
      org/2001/XMLSchema">
   <import namespace="http://xml.apache.org/xml-soap"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="ArrayOf_xsd_anyType">
    <complexContent>
     <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:anyType[]"/>
     </restriction>
    </complexContent>
   </complexType>
  </schema>
  <schema targetNamespace="http://xml.apache.org/xml-soap" xmlns="http://www.w3.org/2001/XMLSchema
      ">
   <import namespace="http://localhost:8080/axis/services/WSCalendar"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="Vector">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="xsd:anyType"/>
    </sequence>
   </complexType>
  </schema>
 </wsdl:types>

  <wsdl:message name="getAppointmentByMonthRequest">

    <wsdl:part name="month" type="soapenc:string"/>

    <wsdl:part name="year" type="soapenc:string"/>

  </wsdl:message>

  <wsdl:message name="getAppointmentRequest">
```

```
37
38      <wsdl:part name="apointmentID" type="soapenc:string"/>
39
40    </wsdl:message>
41
42    <wsdl:message name="getAppointmentByMonthResponse">
43
44      <wsdl:part name="getAppointmentByMonthReturn" type="impl:ArrayOf_xsd_anyType"/>
45
46    </wsdl:message>
47
48    <wsdl:message name="getAppointmentResponse">
49
50      <wsdl:part name="getAppointmentReturn" type="soapenc:string"/>
51
52    </wsdl:message>
53
54    <wsdl:portType name="WSCalendar">
55
56      <wsdl:operation name="getAppointmentByMonth" parameterOrder="month␣year">
57
58        <wsdl:input message="impl:getAppointmentByMonthRequest" name="getAppointmentByMonthRequest"
              />
59
60        <wsdl:output message="impl:getAppointmentByMonthResponse" name="
              getAppointmentByMonthResponse"/>
61
62      </wsdl:operation>
63
64      <wsdl:operation name="getAppointment" parameterOrder="apointmentID">
65
66        <wsdl:input message="impl:getAppointmentRequest" name="getAppointmentRequest"/>
67
68        <wsdl:output message="impl:getAppointmentResponse" name="getAppointmentResponse"/>
69
70      </wsdl:operation>
71
72    </wsdl:portType>
73
74    <wsdl:binding name="WSCalendarSoapBinding" type="impl:WSCalendar">
75
76      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
77
78      <wsdl:operation name="getAppointmentByMonth">
79
80        <wsdlsoap:operation soapAction=""/>
81
82        <wsdl:input name="getAppointmentByMonthRequest">
83
84          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
                http://wsScenario" use="encoded"/>
85
86        </wsdl:input>
87
88        <wsdl:output name="getAppointmentByMonthResponse">
89
90          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
                http://localhost:8080/axis/services/WSCalendar" use="encoded"/>
91
92        </wsdl:output>
93
94      </wsdl:operation>
95
96      <wsdl:operation name="getAppointment">
97
98        <wsdlsoap:operation soapAction=""/>
99
100       <wsdl:input name="getAppointmentRequest">
101
102         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
                http://wsScenario" use="encoded"/>
```

```
      </wsdl:input>

      <wsdl:output name="getAppointmentResponse">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
            http://localhost:8080/axis/services/WSCalendar" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

  </wsdl:binding>

  <wsdl:service name="WSCalendarService">

    <wsdl:port binding="impl:WSCalendarSoapBinding" name="WSCalendar">

      <wsdlsoap:address location="http://localhost:8080/axis/services/WSCalendar"/>

    </wsdl:port>

  </wsdl:service>

</wsdl:definitions>
```

3.7.3.3 WSSupervisor

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/services/WSSupervisor"
    xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://localhost:8080/axis/
    services/WSSupervisor" xmlns:intf="http://localhost:8080/axis/services/WSSupervisor"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.
    org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.
    org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->

  <wsdl:message name="receiveAlarmResponse">

  </wsdl:message>

  <wsdl:message name="receiveAlarmRequest">

    <wsdl:part name="residencialGatewayIP" type="soapenc:string"/>

    <wsdl:part name="alarmGender" type="soapenc:string"/>

  </wsdl:message>

  <wsdl:portType name="WSSupervisor">

    <wsdl:operation name="receiveAlarm" parameterOrder="residencialGatewayIP␣alarmGender">

      <wsdl:input message="impl:receiveAlarmRequest" name="receiveAlarmRequest"/>

      <wsdl:output message="impl:receiveAlarmResponse" name="receiveAlarmResponse"/>

    </wsdl:operation>

  </wsdl:portType>

  <wsdl:binding name="WSSupervisorSoapBinding" type="impl:WSSupervisor">

    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="receiveAlarm">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="receiveAlarmRequest">
```

```
39
40          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
                http://wsScenario" use="encoded"/>
41
42        </wsdl:input>
43
44        <wsdl:output name="receiveAlarmResponse">
45
46          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
                http://localhost:8080/axis/services/WSSupervisor" use="encoded"/>
47
48        </wsdl:output>
49
50      </wsdl:operation>
51
52    </wsdl:binding>
53
54    <wsdl:service name="WSSupervisorService">
55
56      <wsdl:port binding="impl:WSSupervisorSoapBinding" name="WSSupervisor">
57
58        <wsdlsoap:address location="http://localhost:8080/axis/services/WSSupervisor"/>
59
60      </wsdl:port>
61
62    </wsdl:service>
63
64  </wsdl:definitions>
```

3.7.3.4 WSPlastic

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions targetNamespace="http://localhost:8080/axis/services/WSPlastic"
       xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://localhost:8080/axis/
       services/WSPlastic" xmlns:intf="http://localhost:8080/axis/services/WSPlastic" xmlns:soapenc=
       "http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
       xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/
       XMLSchema">
3  <!--WSDL created by Apache Axis version: 1.4
4  Built on Apr 22, 2006 (06:55:48 PDT)-->
5
6    <wsdl:message name="notifyAlarmRequest">
7
8      <wsdl:part name="aliasRG" type="soapenc:string"/>
9
10     <wsdl:part name="alarmGender" type="soapenc:string"/>
11
12     <wsdl:part name="date" type="xsd:dateTime"/>
13
14   </wsdl:message>
15
16   <wsdl:message name="notifyAlarmResponse">
17
18   </wsdl:message>
19
20   <wsdl:portType name="WSPlastic">
21
22     <wsdl:operation name="notifyAlarm" parameterOrder="aliasRG␣alarmGender␣date">
23
24       <wsdl:input message="impl:notifyAlarmRequest" name="notifyAlarmRequest"/>
25
26       <wsdl:output message="impl:notifyAlarmResponse" name="notifyAlarmResponse"/>
27
28     </wsdl:operation>
29
30   </wsdl:portType>
31
32   <wsdl:binding name="WSPlasticSoapBinding" type="impl:WSPlastic">
33
34     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
35
```

```
    <wsdl:operation name="notifyAlarm">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="notifyAlarmRequest">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
            http://wsScenario" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="notifyAlarmResponse">

        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
            http://localhost:8080/axis/services/WSPlastic" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

  </wsdl:binding>

  <wsdl:service name="WSPlasticService">

    <wsdl:port binding="impl:WSPlasticSoapBinding" name="WSPlastic">

      <wsdlsoap:address location="http://localhost:8080/axis/services/WSPlastic"/>

    </wsdl:port>

  </wsdl:service>

</wsdl:definitions>
```

3.7.3.5 WSSeguitelService

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/services/WSSeguitelService"
    xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://localhost:8080/axis/
    services/WSSeguitelService" xmlns:intf="http://localhost:8080/axis/services/WSSeguitelService
    " xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.
    xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://
    www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema targetNamespace="http://localhost:8080/axis/services/WSSeguitelService" xmlns="http://
      www.w3.org/2001/XMLSchema">
   <import namespace="http://xml.apache.org/xml-soap"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="ArrayOf_xsd_anyType">
    <complexContent>
     <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:anyType[]"/>
     </restriction>
    </complexContent>
   </complexType>
  </schema>
  <schema targetNamespace="http://xml.apache.org/xml-soap" xmlns="http://www.w3.org/2001/XMLSchema
      ">
   <import namespace="http://localhost:8080/axis/services/WSSeguitelService"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="Vector">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="xsd:anyType"/>
    </sequence>
   </complexType>
  </schema>
 </wsdl:types>

  <wsdl:message name="getResidencialGatewayIPResponse">
```

```
29
30      <wsdl:part name="getResidencialGatewayIPReturn" type="soapenc:string"/>
31
32    </wsdl:message>
33
34    <wsdl:message name="getResidencialGatewayIPRequest">
35
36      <wsdl:part name="aliasRG" type="soapenc:string"/>
37
38    </wsdl:message>
39
40    <wsdl:message name="notifyAlarmRequest">
41
42      <wsdl:part name="aliasRG" type="soapenc:string"/>
43
44      <wsdl:part name="alarmGender" type="soapenc:string"/>
45
46      <wsdl:part name="date" type="xsd:dateTime"/>
47
48    </wsdl:message>
49
50    <wsdl:message name="getConnectedDeviceIPRequest">
51
52      <wsdl:part name="aliasRG" type="soapenc:string"/>
53
54      <wsdl:part name="alarmGender" type="soapenc:string"/>
55
56    </wsdl:message>
57
58    <wsdl:message name="notifyAlarmResponse">
59
60      <wsdl:part name="notifyAlarmReturn" type="xsd:boolean"/>
61
62    </wsdl:message>
63
64    <wsdl:message name="getConnectedDeviceIPResponse">
65
66      <wsdl:part name="getConnectedDeviceIPReturn" type="impl:ArrayOf_xsd_anyType"/>
67
68    </wsdl:message>
69
70    <wsdl:portType name="WSSeguitelService">
71
72      <wsdl:operation name="notifyAlarm" parameterOrder="aliasRG␣alarmGender␣date">
73
74        <wsdl:input message="impl:notifyAlarmRequest" name="notifyAlarmRequest"/>
75
76        <wsdl:output message="impl:notifyAlarmResponse" name="notifyAlarmResponse"/>
77
78      </wsdl:operation>
79
80      <wsdl:operation name="getConnectedDeviceIP" parameterOrder="aliasRG␣alarmGender">
81
82        <wsdl:input message="impl:getConnectedDeviceIPRequest" name="getConnectedDeviceIPRequest"/>
83
84        <wsdl:output message="impl:getConnectedDeviceIPResponse" name="getConnectedDeviceIPResponse
              "/>
85
86      </wsdl:operation>
87
88      <wsdl:operation name="getResidencialGatewayIP" parameterOrder="aliasRG">
89
90        <wsdl:input message="impl:getResidencialGatewayIPRequest" name="
              getResidencialGatewayIPRequest"/>
91
92        <wsdl:output message="impl:getResidencialGatewayIPResponse" name="
              getResidencialGatewayIPResponse"/>
93
94      </wsdl:operation>
95
96    </wsdl:portType>
```

```
97
98  <wsdl:binding name="WSSeguitelServiceSoapBinding" type="impl:WSSeguitelService">
99
100   <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
101
102   <wsdl:operation name="notifyAlarm">
103
104     <wsdlsoap:operation soapAction=""/>
105
106     <wsdl:input name="notifyAlarmRequest">
107
108       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://wsScenario" use="encoded"/>
109
110     </wsdl:input>
111
112     <wsdl:output name="notifyAlarmResponse">
113
114       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://localhost:8080/axis/services/WSSeguitelService" use="encoded"/>
115
116     </wsdl:output>
117
118   </wsdl:operation>
119
120   <wsdl:operation name="getConnectedDeviceIP">
121
122     <wsdlsoap:operation soapAction=""/>
123
124     <wsdl:input name="getConnectedDeviceIPRequest">
125
126       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://wsScenario" use="encoded"/>
127
128     </wsdl:input>
129
130     <wsdl:output name="getConnectedDeviceIPResponse">
131
132       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://localhost:8080/axis/services/WSSeguitelService" use="encoded"/>
133
134     </wsdl:output>
135
136   </wsdl:operation>
137
138   <wsdl:operation name="getResidencialGatewayIP">
139
140     <wsdlsoap:operation soapAction=""/>
141
142     <wsdl:input name="getResidencialGatewayIPRequest">
143
144       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://wsScenario" use="encoded"/>
145
146     </wsdl:input>
147
148     <wsdl:output name="getResidencialGatewayIPResponse">
149
150       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://localhost:8080/axis/services/WSSeguitelService" use="encoded"/>
151
152     </wsdl:output>
153
154   </wsdl:operation>
155
156 </wsdl:binding>
157
158 <wsdl:service name="WSSeguitelServiceService">
159
160   <wsdl:port binding="impl:WSSeguitelServiceSoapBinding" name="WSSeguitelService">
161
```

```
      <wsdlsoap:address location="http://localhost:8080/axis/services/WSSeguitelService"/>

    </wsdl:port>

  </wsdl:service>

</wsdl:definitions>
```

3.7.3.6 WSMedicalDevice

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/services/WSMedicalDevice"
    xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://localhost:8080/axis/
    services/WSMedicalDevice" xmlns:intf="http://localhost:8080/axis/services/WSMedicalDevice"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.
    org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.
    org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema targetNamespace="http://localhost:8080/axis/services/WSMedicalDevice" xmlns="http://www.
      w3.org/2001/XMLSchema">
   <import namespace="http://xml.apache.org/xml-soap"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="ArrayOf_xsd_anyType">
    <complexContent>
     <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:anyType[]"/>
     </restriction>
    </complexContent>
   </complexType>
  </schema>
  <schema targetNamespace="http://xml.apache.org/xml-soap" xmlns="http://www.w3.org/2001/XMLSchema
      ">
   <import namespace="http://localhost:8080/axis/services/WSMedicalDevice"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="Vector">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="xsd:anyType"/>
    </sequence>
   </complexType>
  </schema>
 </wsdl:types>

   <wsdl:message name="getMedicalDevicesRequest">

   </wsdl:message>

   <wsdl:message name="getMedicalDevicesResponse">

      <wsdl:part name="getMedicalDevicesReturn" type="impl:ArrayOf_xsd_anyType"/>

   </wsdl:message>

   <wsdl:message name="getMeasureRequest">

      <wsdl:part name="idMedicalDevice" type="soapenc:string"/>

   </wsdl:message>

   <wsdl:message name="getMeasureResponse">

      <wsdl:part name="getMeasureReturn" type="soapenc:string"/>

   </wsdl:message>

   <wsdl:portType name="WSMedicalDevice">

      <wsdl:operation name="getMedicalDevices">

        <wsdl:input message="impl:getMedicalDevicesRequest" name="getMedicalDevicesRequest"/>
```

```
55
56        <wsdl:output message="impl:getMedicalDevicesResponse" name="getMedicalDevicesResponse"/>
57
58      </wsdl:operation>
59
60      <wsdl:operation name="getMeasure" parameterOrder="idMedicalDevice">
61
62        <wsdl:input message="impl:getMeasureRequest" name="getMeasureRequest"/>
63
64        <wsdl:output message="impl:getMeasureResponse" name="getMeasureResponse"/>
65
66      </wsdl:operation>
67
68    </wsdl:portType>
69
70    <wsdl:binding name="WSMedicalDeviceSoapBinding" type="impl:WSMedicalDevice">
71
72      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
73
74      <wsdl:operation name="getMedicalDevices">
75
76        <wsdlsoap:operation soapAction=""/>
77
78        <wsdl:input name="getMedicalDevicesRequest">
79
80          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://wsScenario" use="encoded"/>
81
82        </wsdl:input>
83
84        <wsdl:output name="getMedicalDevicesResponse">
85
86          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://localhost:8080/axis/services/WSMedicalDevice" use="encoded"/>
87
88        </wsdl:output>
89
90      </wsdl:operation>
91
92      <wsdl:operation name="getMeasure">
93
94        <wsdlsoap:operation soapAction=""/>
95
96        <wsdl:input name="getMeasureRequest">
97
98          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://wsScenario" use="encoded"/>
99
100       </wsdl:input>
101
102       <wsdl:output name="getMeasureResponse">
103
104         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="
              http://localhost:8080/axis/services/WSMedicalDevice" use="encoded"/>
105
106       </wsdl:output>
107
108     </wsdl:operation>
109
110   </wsdl:binding>
111
112   <wsdl:service name="WSMedicalDeviceService">
113
114     <wsdl:port binding="impl:WSMedicalDeviceSoapBinding" name="WSMedicalDevice">
115
116       <wsdlsoap:address location="http://localhost:8080/axis/services/WSMedicalDevice"/>
117
118     </wsdl:port>
119
120   </wsdl:service>
121
```

```
122 </wsdl:definitions>
```

3.7.4 QoS Properties Definition

As described in Section 3.7.1, the Web Services considered in this scenario could be deployed on different kind machines. In particular, both the **WSDoctor** and **WSSupervisor** could be deployed on a usual wired device as a PC or a mobile and wireless device such as a smart phone.

	Latency (msec)	Reliability		Conditions
WSSeguitelService:getContactDeciveIP	2000			alarmGender=“Emergency”
WSSeguitelService:getContactDeciveIP	1000			alarmGender=“No Confirmation”
WSDoctor:receiveAlarm	6000	WinSize Max Fails in Win	30000 5	deployedOn=“MobileNode”
WSDoctor:receiveAlarm	2000	WinSize Max Fails in Win	30000 1	deployedOn=“WiredServer”
WSSupervisor:receiveAlarm	10000	WinSize Max Fails in Win	30000 5	deployedOn=“MobileNode”
WSSupervisor:receiveAlarm	6000	WinSize Max Fails in Win	30000 1	deployedOn=“WiredServer”
WSMedicalDevice:getMeasure	3000			idMedicalDevice=“device_1”
WSMedicalDevice:getMeasure	10000			idMedicalDevice=“device_2”

Table 3.2: QoS Properties

In those cases, the some QoS properties of the service could depend on where the service in actually deployed. On the one hand, if a the Web Service is deployed on a wireless node for example is not always given that is it possible to reach it. On the other hand, if a Web Service is deployed on a standard PC it operates at higher performances than one deployed on a smart phone.

Other cases are given when a method behaves differently depending on the parameters it receives. For example, the processing time of **getMeasure** on **WSMedicalDevice** directly depends on the kind of measurement that is performed and the medical device that is used.

The QoS levels admitted in the scenario here considered are formalized in an agreement. Table 3.2[2] reports a short version of the extra functional properties that are supposed to be respected in the scenario. Time are given in milliseconds. At the end of the section, the listings reports the complete version of the XML document expressing the agreement.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsag:AgreementOffer xsi:schemaLocation="http://www.ggf.org/namespaces/ws-agreement"
3                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
5                  xmlns:wsag="http://schemas.ggf.org/graap/2005/09/ws-agreement"
6                  xmlns:puppetScope="http://setest0.isti.cnr.it/puppetScope"
7                  xmlns:puppetSLO="http://setest0.isti.cnr.it/puppetSLO"
8                  xmlns:puppetQC="http://setest0.isti.cnr.it/puppetQC"
9                  xmlns:ns="http://setest0.isti.cnr.it/puppet">
10   <wsag:Name>Telefonica_Example_Scenario_3</wsag:Name>
11
12   <wsag:Context />
13
14   <wsag:Terms>
15     <wsag:All>
16       <wsag:GuaranteeTerm wsag:Name="ContactDeciveIP-Term1"
17         wsag:Obligated="ServiceProvider">
18         <wsag:ServiceScope wsag:ServiceName="WSSeguitelService">
19           <puppetScope:PuppetScope>
20             <puppetScope:Method>
21 <!--           <NameMethod>getContactDeciveIP</NameMethod> -->
22             <NameMethod>getConnectedDeviceIP</NameMethod>
23             </puppetScope:Method>
```

[2]The values in this table has to be considered just as an example.

```
        </puppetScope:PuppetScope>
      </wsag:ServiceScope>

      <wsag:QualifyingCondition>
        <puppetQC:PuppetQC>
          <puppetQC:StringBinaryExpression>
            <puppetQC:StringType>
              <Variable>alarmGender</Variable>
            </puppetQC:StringType>

            <op>equal</op>

            <puppetQC:StringType>
              <Value>Emergency</Value>
            </puppetQC:StringType>
          </puppetQC:StringBinaryExpression>
        </puppetQC:PuppetQC>
      </wsag:QualifyingCondition>

      <wsag:ServiceLevelObjective>
        <puppetSLO:PuppetSLO>
          <puppetSLO:Latency>
            <value>2000</value>

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

      <wsag:BusinessValueList>
        <wsag:Penalty>
          <wsag:AssessmentInterval>
            <wsag:Count />
          </wsag:AssessmentInterval>

          <wsag:ValueExpression> 2 </wsag:ValueExpression>
        </wsag:Penalty>
      </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
    <wsag:GuaranteeTerm wsag:Name="ContactDeciveIP-Term2"
      wsag:Obligated="ServiceProvider">
      <wsag:ServiceScope wsag:ServiceName="WSSeguitelService">
        <puppetScope:PuppetScope>
          <puppetScope:Method>
<!--          <NameMethod>getContactDeciveIP</NameMethod> -->
            <NameMethod>getConnectedDeviceIP</NameMethod>
          </puppetScope:Method>
        </puppetScope:PuppetScope>
      </wsag:ServiceScope>

      <wsag:QualifyingCondition>
        <puppetQC:PuppetQC>
          <puppetQC:StringBinaryExpression>
            <puppetQC:StringType>
              <Variable>alarmGender</Variable>
            </puppetQC:StringType>

            <op>equal</op>

            <puppetQC:StringType>
              <Value>No Confirmation</Value>
            </puppetQC:StringType>
          </puppetQC:StringBinaryExpression>
        </puppetQC:PuppetQC>
      </wsag:QualifyingCondition>

      <wsag:ServiceLevelObjective>
        <puppetSLO:PuppetSLO>
          <puppetSLO:Latency>
```

```
          <value>1000</value>

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

      <wsag:BusinessValueList>
        <wsag:Penalty>
          <wsag:AssessmentInterval>
            <wsag:Count />
          </wsag:AssessmentInterval>

          <wsag:ValueExpression> 2 </wsag:ValueExpression>
        </wsag:Penalty>
      </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
    <wsag:ExactlyOne>
      <wsag:GuaranteeTerm wsag:Name="AlarmDoctor-Term1"
        wsag:Obligated="ServiceProvider">
        <wsag:ServiceScope wsag:ServiceName="WSDoctor">
          <puppetScope:PuppetScope>
            <puppetScope:Method>
              <NameMethod>receiveAlarm</NameMethod>
            </puppetScope:Method>
          </puppetScope:PuppetScope>
        </wsag:ServiceScope>

        <wsag:QualifyingCondition>
          <puppetQC:PuppetQC>
            <puppetQC:StringBinaryExpression>
              <puppetQC:StringType>
                <Variable>deployedOn</Variable>
              </puppetQC:StringType>

              <op>equal</op>

              <puppetQC:StringType>
                <Value>MobileNode</Value>
              </puppetQC:StringType>
            </puppetQC:StringBinaryExpression>
          </puppetQC:PuppetQC>
        </wsag:QualifyingCondition>

        <wsag:ServiceLevelObjective>
          <puppetSLO:PuppetSLO>
            <puppetSLO:Latency>
              <value>6000</value>
              <puppetSLO:Distribution>
                 <Gaussian>10</Gaussian>
              </puppetSLO:Distribution>
            </puppetSLO:Latency>
            <puppetSLO:Reliability>
              <Reliabilitywindow>30000</Reliabilitywindow>
              <ReliabilityPerc>5</ReliabilityPerc>
              <puppetSLO:Distribution>
                <Gaussian>100</Gaussian>
              </puppetSLO:Distribution>
            </puppetSLO:Reliability>
          </puppetSLO:PuppetSLO>
        </wsag:ServiceLevelObjective>

        <wsag:BusinessValueList>
          <wsag:Penalty>
            <wsag:AssessmentInterval>
              <wsag:Count />
            </wsag:AssessmentInterval>

            <wsag:ValueExpression> 2 </wsag:ValueExpression>
```

```
        </wsag:Penalty>
      </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
    <wsag:GuaranteeTerm wsag:Name="AlarmDoctor-Term2"
      wsag:Obligated="ServiceProvider">
      <wsag:ServiceScope wsag:ServiceName="WSDoctor">
        <puppetScope:PuppetScope>
          <puppetScope:Method>
            <NameMethod>receiveAlarm</NameMethod>
          </puppetScope:Method>
        </puppetScope:PuppetScope>
      </wsag:ServiceScope>

      <wsag:QualifyingCondition>
        <puppetQC:PuppetQC>
          <puppetQC:StringBinaryExpression>
            <puppetQC:StringType>
              <Variable>deployedOn</Variable>
            </puppetQC:StringType>

            <op>equal</op>

            <puppetQC:StringType>
              <Value>WiredServer</Value>
            </puppetQC:StringType>
          </puppetQC:StringBinaryExpression>
        </puppetQC:PuppetQC>
      </wsag:QualifyingCondition>

      <wsag:ServiceLevelObjective>
        <puppetSLO:PuppetSLO>
          <puppetSLO:Latency>
            <value>2000</value>
              <puppetSLO:Distribution>
                <Gaussian>10</Gaussian>
              </puppetSLO:Distribution>
          </puppetSLO:Latency>
          <puppetSLO:Reliability>
            <Reliabilitywindow>30000</Reliabilitywindow>

            <ReliabilityPerc>1</ReliabilityPerc>

            <puppetSLO:Distribution>
              <Gaussian>100</Gaussian>
            </puppetSLO:Distribution>
          </puppetSLO:Reliability>
        </puppetSLO:PuppetSLO>
      </wsag:ServiceLevelObjective>

      <wsag:BusinessValueList>
        <wsag:Penalty>
          <wsag:AssessmentInterval>
            <wsag:Count />
          </wsag:AssessmentInterval>

          <wsag:ValueExpression> 2 </wsag:ValueExpression>
        </wsag:Penalty>
      </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
  </wsag:ExactlyOne>
  <wsag:ExactlyOne>
    <wsag:GuaranteeTerm wsag:Name="AlarmSupervisor-Term1"
      wsag:Obligated="ServiceProvider">
      <wsag:ServiceScope wsag:ServiceName="WSSupervisor">
        <puppetScope:PuppetScope>
          <puppetScope:Method>
            <NameMethod>receiveAlarm</NameMethod>
          </puppetScope:Method>
        </puppetScope:PuppetScope>
      </wsag:ServiceScope>
```

```
237         <wsag:QualifyingCondition>
238           <puppetQC:PuppetQC>
239             <puppetQC:StringBinaryExpression>
240               <puppetQC:StringType>
241                 <Variable>deployedOn</Variable>
242               </puppetQC:StringType>
243
244               <op>equal</op>
245
246               <puppetQC:StringType>
247                 <Value>MobileNode</Value>
248               </puppetQC:StringType>
249             </puppetQC:StringBinaryExpression>
250           </puppetQC:PuppetQC>
251         </wsag:QualifyingCondition>
252
253         <wsag:ServiceLevelObjective>
254           <puppetSLO:PuppetSLO>
255             <puppetSLO:Latency>
256               <value>10000</value>
257
258                 <puppetSLO:Distribution>
259                   <Gaussian>10</Gaussian>
260                 </puppetSLO:Distribution>
261             </puppetSLO:Latency>
262             <puppetSLO:Reliability>
263               <Reliabilitywindow>30000</Reliabilitywindow>
264
265               <ReliabilityPerc>5</ReliabilityPerc>
266
267               <puppetSLO:Distribution>
268                 <Gaussian>100</Gaussian>
269               </puppetSLO:Distribution>
270             </puppetSLO:Reliability>
271           </puppetSLO:PuppetSLO>
272         </wsag:ServiceLevelObjective>
273
274         <wsag:BusinessValueList>
275           <wsag:Penalty>
276             <wsag:AssessmentInterval>
277               <wsag:Count />
278             </wsag:AssessmentInterval>
279
280             <wsag:ValueExpression> 2 </wsag:ValueExpression>
281           </wsag:Penalty>
282         </wsag:BusinessValueList>
283       </wsag:GuaranteeTerm>
284       <wsag:GuaranteeTerm wsag:Name="AlarmSupervisor-Term2"
285         wsag:Obligated="ServiceProvider">
286         <wsag:ServiceScope wsag:ServiceName="WSSupervisor">
287           <puppetScope:PuppetScope>
288             <puppetScope:Method>
289               <NameMethod>receiveAlarm</NameMethod>
290             </puppetScope:Method>
291           </puppetScope:PuppetScope>
292         </wsag:ServiceScope>
293
294         <wsag:QualifyingCondition>
295           <puppetQC:PuppetQC>
296             <puppetQC:StringBinaryExpression>
297               <puppetQC:StringType>
298                 <Variable>deployedOn</Variable>
299               </puppetQC:StringType>
300
301               <op>equal</op>
302
303               <puppetQC:StringType>
304                 <Value>WiredServer</Value>
305               </puppetQC:StringType>
306             </puppetQC:StringBinaryExpression>
307           </puppetQC:PuppetQC>
```

```
      </wsag:QualifyingCondition>

      <wsag:ServiceLevelObjective>
       <puppetSLO:PuppetSLO>
        <puppetSLO:Latency>
         <value>6000</value>

          <puppetSLO:Distribution>
           <Gaussian>10</Gaussian>
          </puppetSLO:Distribution>
        </puppetSLO:Latency>
        <puppetSLO:Reliability>
         <Reliabilitywindow>30000</Reliabilitywindow>

         <ReliabilityPerc>1</ReliabilityPerc>

         <puppetSLO:Distribution>
          <Gaussian>100</Gaussian>
         </puppetSLO:Distribution>
        </puppetSLO:Reliability>
       </puppetSLO:PuppetSLO>
      </wsag:ServiceLevelObjective>

      <wsag:BusinessValueList>
       <wsag:Penalty>
        <wsag:AssessmentInterval>
         <wsag:Count />
        </wsag:AssessmentInterval>

        <wsag:ValueExpression> 2 </wsag:ValueExpression>
       </wsag:Penalty>
      </wsag:BusinessValueList>
     </wsag:GuaranteeTerm>
    </wsag:ExactlyOne>
    <wsag:GuaranteeTerm wsag:Name="MedicalDevice-Term1"
     wsag:Obligated="ServiceProvider">
     <wsag:ServiceScope wsag:ServiceName="WSMedicalDevice">
      <puppetScope:PuppetScope>
       <puppetScope:Method>
        <NameMethod>getMeasure</NameMethod>
       </puppetScope:Method>
      </puppetScope:PuppetScope>
     </wsag:ServiceScope>

     <wsag:QualifyingCondition>
      <puppetQC:PuppetQC>
       <puppetQC:StringBinaryExpression>
        <puppetQC:StringType>
         <Variable>idMedicalDevice</Variable>
        </puppetQC:StringType>

        <op>equal</op>

        <puppetQC:StringType>
         <Value>device_1</Value>
        </puppetQC:StringType>
       </puppetQC:StringBinaryExpression>
      </puppetQC:PuppetQC>
     </wsag:QualifyingCondition>

     <wsag:ServiceLevelObjective>
      <puppetSLO:PuppetSLO>
       <puppetSLO:Latency>
        <value>3000</value>

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

```
      <wsag:BusinessValueList>
       <wsag:Penalty>
        <wsag:AssessmentInterval>
         <wsag:Count />
        </wsag:AssessmentInterval>

        <wsag:ValueExpression> 2 </wsag:ValueExpression>
       </wsag:Penalty>
      </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
    <wsag:GuaranteeTerm wsag:Name="MedicalDevice-Term2"
     wsag:Obligated="ServiceProvider">
     <wsag:ServiceScope wsag:ServiceName="WSMedicalDevice">
      <puppetScope:PuppetScope>
       <puppetScope:Method>
        <NameMethod>getMeasure</NameMethod>
       </puppetScope:Method>
      </puppetScope:PuppetScope>
     </wsag:ServiceScope>

     <wsag:QualifyingCondition>
      <puppetQC:PuppetQC>
       <puppetQC:StringBinaryExpression>
        <puppetQC:StringType>
         <Variable>idMedicalDevice</Variable>
        </puppetQC:StringType>

        <op>equal</op>

        <puppetQC:StringType>
         <Value>device_2</Value>
        </puppetQC:StringType>
       </puppetQC:StringBinaryExpression>
      </puppetQC:PuppetQC>
     </wsag:QualifyingCondition>

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

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

     <wsag:BusinessValueList>
      <wsag:Penalty>
       <wsag:AssessmentInterval>
        <wsag:Count />
       </wsag:AssessmentInterval>

       <wsag:ValueExpression> 2 </wsag:ValueExpression>
      </wsag:Penalty>
     </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>

   </wsag:All>
  </wsag:Terms>
</wsag:AgreementOffer>
```

4 Weevil

4.1 Framework

A particular experiment is related to three primary concepts: the system under experimentation (SUE), the testbed and the actor. An actor maps a client to its system access point and actuates the SUE as dictated by the client's workload. An experiment can be understood as a two-phase process in which (1) a workload is generated for the actors, (2) the SUE is deployed on the testbed with the actors actuating it based on the workload and the results are collected afterward. Weevil supports the two-phase process through a central-controller architecture in which a master script controls the whole process. Workload generation is conducted on the master's host as presented in Section 4.2. In the experiment deployment and execution phase (described in Section 4.3) the master generates all the controls, deploys them together with the workload and the SUE on the testbed, coordinates all the system components and the actors to execute the experiment, and gathers data after the experiment terminates. The two program *weevilgen* and *weevil*, are for workload generation and experiment deployment and execution respectively.

Throughout the following sections, we refer to the Siena example and the Squid/Apache example included in the prototype distribution under *share/weevil-1.0.0/examples*.

4.2 Simulation-Based Workload Generation

Weevil's simulation-based workload generation process supported by the package *weevilgen* is illustrated in Figure 4.1.

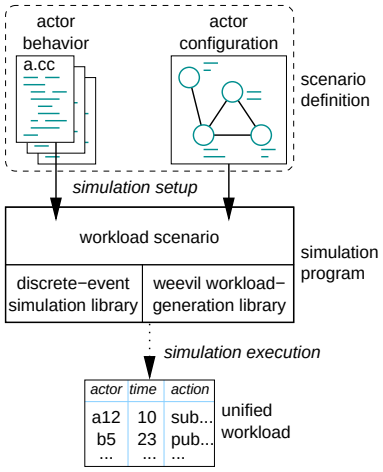

Figure 4.1: Simulation-Based Workload Generation

In detail, the following steps need to be taken in order.

4.2.1 Weevilgen Configuration (Workload Registration)

Create a name for the workload (refered to as <*workload name*> below), like "`MyWorkload`" in the Siena example; add the name into "*weevil.conf*" file after "`weevilgen_WORKLOADS:=`". You could have more than one workload registered in "*weevil.conf*" separated with space, as indicated in the apachesquid example. To start a new line, \ is needed to end the current line. Weevilgen only generates workloads that have been registered in "*weevil.conf*".

For example, in file "*weevil.conf*" of the Squid/Apache example we registered a series of workloads:

```
weevilgen_WORKLOADS := wkld_10_10_10 wkld_20_20_20 wkld_30_30_30 \
wkld_50_50_50 wkld_100_100_100 wkld_150_150_150 wkld_200_200_200 wkld_30 \
wkld_60 wkld_90 wkld_150 wkld_300 wkld_450 wkld_600 wkld_tp wkld_gsl
```

Then, to generate each workload, Weevil must be provided with programmed actor behavior models as illustrated in Section 4.2.2 and an actor configuration file named "<*workload name*>*.m4*" including all the actor configurations as illustrated in Section 4.2.3.

4.2.2 Actor Behavior Model Programming

One or more types of actor behaviors could be programmed in C++ with the support of Weevil's workload-generation library and SSim simulation library, and may therefore execute arbitrary functions and maintain arbitrary state. SSim is a discrete-event simulation library that supports message communication between simulated processes, so actor behavior programs may specify interactions with other actors. Weevil's workload-generation library extends the SSim's library by providing a workload-output method and convenient methods for dealing with processes by ids. Please refer to the *Weevil's workload-generation library* and the *SSim's simulation library* in the reference manual when programming your actor behavior model.

Basicly, actor behavior is encapsulated in a subclass of the `weevil::WeevilTProcess` class or the `weevil:: WeevilProcess` class. `weevil::WeevilTProcess`, which is a *sequential process*, is the extension of the `ssim::TProcess` class. A sequential actor behavior is defined simply by programming the `main` method, which defines the body of the process directly. A process can use the `WeevilGen::broadcast_event`, `WeevilGen:: self_signal_event` and `WeevilGen::signal_event` method defined in Weevil's library to signal itself or other processes, and the `wait_for_event` method to receive events.

In contrast, `weevil::WeevilProcess`, which is a *reactive process*, is the extension of the `ssim::Process` class. It is used when the process may have other reactive behaviors in parallel to its routine behaviors. A reactive actor behavior is defined by programming the `init` and the `process_event` methods. `Init` initiates the process just before the simulation starts. `Process_event` is called automatically by an event and defines the functions executed by the process in response to an event. Same as the `weevil::WeevilTProcess`, the event can be signaled using the `WeevilGen::broadcast_event`, `WeevilGen::self_signal_event` and `WeevilGen::signal_event` methods.

Both of the above two types of processes can use the method `workload_output()` to output actions to the workload. As examples of the usage of these two types, we consider the following two types of interdependent behaviors.

4.2.2.1 Intra-Actor Interdependent Behavior

One type of interdependent behavior is one where future requests need to incorporate data from a previous request. As an example, consider a distributed publish/subscribe system where actors use an access point to publish notifications and subscribe for notifications of interest. A common behavior for a subscriber actor could be to periodically subscribe and unsubscribe, with every unsubscription matching the previous subscription, then continue with the next subscription. We implemented this behavior in *examples/siena/subscriber.cc*. (It is used to generate workload *wkld_sequential.wkld*.) Notice that in this case, the behavior is represented as a sequential process class and is defined simply by programming the `main` method of that class.

Similarly, Weevil can easily generate workloads for various types of session-oriented protocols, where actors maintain a virtual connection with the system by providing an immutable session identifier throughout their interaction with the system.

4.2.2.2 Inter-Actor Interdependent Behavior

Another type of interdependent behavior is one where actors communicate and coordinate their interactions with the SUE. We consider a scenario in which three siena clients are involved. They have their routine behaviors. C1 subscribes 50 times, C2 publishes 40 times, and C3 publishes 20 times. Besides, they also communicate and may have reactive actions triggered by others.

To generate a workload, we consider each client as a reactive workload process `SienaClient`, the subscription, notification, and unsubscription as the possible actions of that process to be outputed to a workload file. The communication between clients is represented by a `Trigger` object. The configuration of each workload process consists of the number of its routine actions, the intervals between actions, those parameters used to generate the content of subscriptions and notifications, and the list of clients to whom the current client will trigger after its own action.

The code to implement this behavior model could be found in *examples/siena/sienaclient.cc*. (It is used to generate workload *MyWorkload.wkld*.) Notice that in this case, the behavior is defined as a reactive process by implementing the `init` and `process_event` methods. The `Trigger` class represents an event signaled to a simulation process. The `init` method initializes workload processes just before the simulation starts but after the assignment of process properties. Notice also that the configuration of each workload process is implemented with the help of actor configurations described later in Section 4.2.3. Corresponding to each property declared in the actor configurations, a class variable and a class function need to be defined in the behavior program, such as,

```
//class variable
unsigned int m_constr_min;
//class function
void constr_min( unsigned int cmin )
{ m_constr_min = cmin; }
```

in the file "*sienaclient.cc*" corresponds to the property configuration

```
WVL_SYS_WorkloadProcessProp( 'C1', 'constr_min', '1')dnl
WVL_SYS_WorkloadProcessProp( 'C2', 'constr_min', '2')dnl
WVL_SYS_WorkloadProcessProp( 'C3', 'constr_min', '1')dnl
```

in the actor configuration file "*MyWorkload.m4*".

4.2.3 Actor Configurations

After programming the actor behavior models, you can populate a scenario consisting of many actor instances specified in the actor configuration file named "<*workload name*>*.m4*". The actor behavior models and the actor configurations make up a workload scenario definition.

Figure 4.2 shows the portion of the Weevil conceptual model concerning the definition of workload scenarios. You need to parameterize the following GNU m4 declaration macros in the actor configuration file. The order of the declarations does not matter. A macro can be used before it is defined as long as it is defined somewhere in the actor configuration file. Weevil supports the engineer during this activity by performing extensive checks on the syntax and consistency of the configurations and by providing detailed error messages about any problems it encounters.

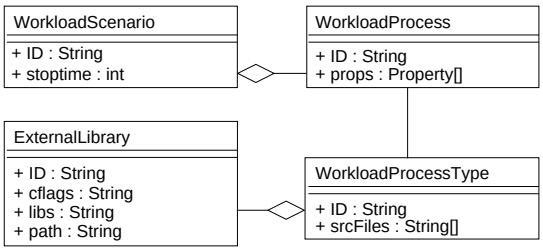

Figure 4.2: Workload Scenario Conceptual Model

- `WVL_SYS_WorkloadScenario(<ID>, <length>, <processes>)`

  A *WorkloadScenario* has an identifier and a collection of *processes*. The argument `<length>` defines the number of clock ticks in the simulation. If you define this argument, the simulation will end as long as the clock-tick number is reached even if the actors have not finished their work. For example,

```
define('clients', 'C1, C2, C3')dnl
WVL_SYS_WorkloadScenario( 'MyWorkload', '10000', 'clients')dnl
```

- `WVL_SYS_WorkloadProcessType(<ID>, <sourceFiles>, <libraries>)`

  A *WorkloadProcessType* defines an actor behavior model in a list of *sourceFiles* as described in Section 4.2.2, and may be associated with a collection of *libraries* that contain external dependencies of the implementation. Please note that none of the file names of the .cc files in the *sourceFiles* list can be the same as the workload name you registered in the *weevil.conf* since a <*workload name*>*.cc* will be generated in the workload generation process. The source files should list all the C++ files in the order of file inclusion. For example,

```
WVL_SYS_WorkloadProcessType('SienaClient','gen_message.h, gen_message.cc, sienaclient.cc',
'')dnl
```

  In this example, instead of adding the statement `#include ''gen_message.h''` in both *gen_message.cc* and *sienaclient.cc*, we only need to put *gen_message.h* before *gen_message.cc* and *sienaclient.cc* in the <*source files*> parameter of this declaration macro.

  The Squid/Apache's *wkld_gsl* example that generates workloads based on analytic models uses the GNU scientific library *libgsl* declared later:

```
WVL_SYS_WorkloadProcessType( 'AnalysisBrowser', 'analysisbrowser.cc', 'libgsl')dnl
```

- `WVL_SYS_ExternalLibrary(<ID>, <cflags>, <libs>, <path>)`

  This declaration defines the usage of an external library, including its cflags, its libs, and the path to add into the environment variable *LD_LIBRARY_PATH*. For example, the external library "*libgsl*" used above is declared as follows:

```
WVL_SYS_ExternalLibrary( 'libgsl', '-I/usr/include -L/usr/lib', '-lgsl -lgslcblas -lm',
 '/usr/lib')dnl
```

- `WVL_SYS_WorkloadProcess(<ID>, <type>)`

  Each *WorkloadProcess* represents an actor, which is an instance of a *WorkloadProcessType*. For example,

```
define('clients', 'C1, C2, C3')dnl
WVL_SYS_Foreach('i', 'WVL_SYS_WorkloadProcess( i, 'SienaClient')', clients)dnl
```

  defines three instances of the *WorkloadProcessType* `SienaClient` (`C1, C2, C3`).

- `WVL_SYS_WorkloadProcessProp(<workloadProcessID>, <name>, <value>)`

  This declaration macro is used to configure each defined process. Corresponding to each property, a class variable and a class function need to be defined in the actor behavior program to assign the value of the property to the class variable. Besides the example we gave in Section 4.2.2, another example could be the property"`sub_neighbors`":

```
WVL_SYS_WorkloadProcessProp( 'C1', 'sub_neighbors', '"C2 PUB 3 0.02|C3 SUB 2 0.01|"')dnl
WVL_SYS_WorkloadProcessProp( 'C2', 'sub_neighbors', '"C3 SUB 2 0.02|"')dnl
WVL_SYS_WorkloadProcessProp( 'C3', 'sub_neighbors', '"C1 PUB 5 0.01|"')dnl
```

  corresponds to the class variable "`m_sub_neighbors`" and function "`sub_neighbors`" in program "*sienaclient.cc*". The function "`sub_neighbors`" parses the string in the parameter `<value>` to assign the variable "`m_sub_neighbors`".

4.2.4 workload generation

With the actor behavior models programmed and actor instances configured for a workload scenario, Weevil checks them for consistency and then translates them into an executable simulation program that is linked with the libraries and then executed to produce the desired workload named "<*workload name*>*.wkld*". The workload consists of all interactions between actors and the SUE. These interactions represent service calls that must be applied to the SUE during experiment execution. This process is accomplished through the command "*weevilgen gen-*<*workload name*>".

```
examples/apachesquid> ${execPath}/weevilgen help
Welcome to WEEVIL WORKLOAD GENERATOR.

The following commands are available:

  check-all            - checks all workloads.
  check-<workload>     - checks a workload's settings.
  clean-all            - cleans files for all workloads.
  clean-<workload>     - removes all generated files for a workload.
  help                 - prints this help message
  gen-all              - generates all workloads.
  gen-<workload>       - generates a workload.
  version              - prints the version of weevil workload generator

The following workloads are available:

wkld_10_10_10 wkld_20_20_20 wkld_30_30_30 wkld_50_50_50
wkld_100_100_100 wkld_150_150_150 wkld_200_200_200 wkld_30 wkld_60
wkld_90 wkld_150 wkld_300 wkld_450 wkld_600 wkld_tp

examples/apachesquid> ${execPath}/weevilgen gen-wkld_10_10_10
```

Besides the "*gen-<workload name>*" option, you can also use the other options listed above for different functions.

4.3 Experiment Deployment and Execution

Weevil directly supports experiment deployment and execution. The overall process is depicted in Figure 4.3. Actions are represented by rectangles and are labeled by circled numbers. Input and output data for those actions are represented by ovals. Dark ovals represent input models provided by the engineer. White ovals represent control scripts and data files generated by Weevil. The cross-hatched ovals represent data generated by the SUE during an experiment. Solid arrows represent normal input/output data flow, whereas dotted arrows represent the execution of scripts.

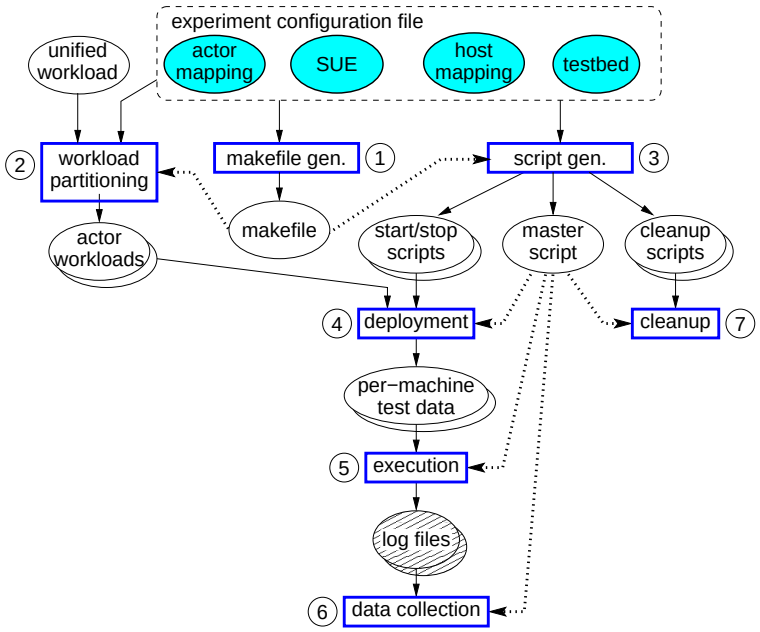

Figure 4.3: Weevil Experimentation Process

The following steps need to be taken to setup and conduct an experiment.

4.3.1 Weevil Configuration (Experiment Registration)

Create a name for the experiment (refered to as <*experiment name*> below), like "`experiment`" in the siena example; add the name into "*weevil.conf*" file after "`weevil_EXPERIMENTS:=`". You could have more than one experiment registered in "*weevil.conf*" separated with space, as indicated in the apachesquid example. To start a new line, \ is needed to end the current line. weevil only conducts experiments that have been registered in "*weevil.conf*".

For example, in file "*weevil.conf*" of the Squid/Apache example we defined a series of experiments:

```
weevil_EXPERIMENTS := exp_10_10_10 exp_20_20_20 exp_30_30_30 \
exp_50_50_50 exp_100_100_100 exp_150_150_150 exp_200_200_200 exp_30 \
exp_60 exp_90 exp_150 exp_300 exp_450 exp_600
```

For each experiment, Weevil must be provided with a workload file (refer to Section 4.3.2) and an experiment configuration file named "<*experiment name*>*.m4*" including all the experiment configurations as illustrated in Section 4.3.3.

4.3.2 Workload File

You can generate the workload file with *weevilgen*. Of course, you can always use workloads from other workload generators or just use a real trace as the workload. But please note that the workload should be made up of workload lines with the following format:

```
event(<time stamp>, <workload process ID>, '<event content>')
```

4.3.3 Experiment Configurations

These configurations are represented by the dark ovals along the top of Figure 4.3. They are programmed in GNU m4 by parameterizing Weevil-defined declaration macros (Please refer to the "Weevil's Experiment Environment Declaration Macros") to instantiate elements of two conceptual models (SUE and testbed) and necessary mappings (Please refer to the "Weevil's Experiment Environment Conceptual Models"). In other words, these declaration macros will define a set of macros serving as properties of an experiment. (Please refer to the "Weevil's Experiment Environment Declaration Macros" and the "Some Other Weevil-Defined Property Macros") for these property macros.) The order of the declarations does not matter. A macro can be used before it is defined as long as it is defined somewhere in the experiment configuration file. Weevil supports the engineer during this activity by performing extensive checks on the syntax and consistency of the configurations and by providing detailed error messages about any problems it encounters. These declaration macros are defined below:

Experiment `WVL_SYS_Experiment(<ID>, <workload>, <actors>, <testbed>, <sue>, <componentHosts>, <clean>)`

This declaration macro defines the experiment to be conducted, including its workload, actor mapping, testbed, SUE, and host mapping. All of these properties will be specified using further declaration macros. Note that the `ID` of an experiment can be different from the experiment name you registered in the "*weevil.conf*" file. The last parameter *clean* specifies if the workspace of the experiment is to be removed after the experiment terminates. For example, the experiment with name "*experiment*" in Siena example is defined as:

```
WVL_SYS_Experiment('Exp_0', 'MyWorkload', 'D0, D1, D2', 'Bed0', 'Siena', 'CHMap0, CHMap1, CHMap2',
 'Y')dnl
```

Workload `WVL_SYS_Workload(<ID>, <filename>, <processes>)`

This declaration macro further specifies the workload. It accociates the workload ID with a specific workload file. It also points out the workload processes included in the workload. For example, the workload in Siena example is defined as:

```
WVL_SYS_Workload('MyWorkload', 'MyWorkload.wkld', 'C1, C2, C3')dnl
```

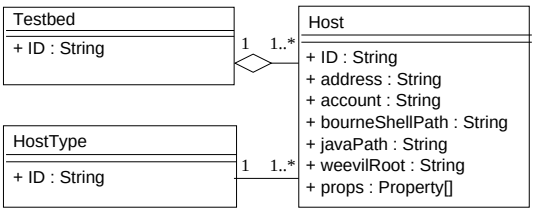

Figure 4.4: Testbed Conceptual Model

Testbed Model Weevil makes minimal assumptions about the testbed. It only requires an account on each host accessiable through user-level remote shell access. Our experience testified that it supports local network testbed as well as the PlanetLab wide-area testbed.

The declaration macros included in this model are:

- `WVL_SYS_Testbed(<ID>, <hosts>)`

  As shown in Figure 4.4, a *Testbed* has an identifier and a collection of *Host*s. For example,

```
WVL_SYS_Testbed( 'Bed0', 'H0, H1, H2')dnl
```

- `WVL_SYS_HostType(<ID>)`

  The *hostType* attribute is used to partition the hosts into categories that each of them has its own binary package of the SUE. For example, a software may have different binary packages if it is compiled on different operating systems. So you should just create an ID for the host type and define it through this declaration macro. Then this ID could be used later in the *WVL_SYS_Host* macro. For example,

```
WVL_SYS_HostType( 'FreeBSD' )dnl
WVL_SYS_HostType( 'Linux' )dnl
```

- `WVL_SYS_Host(ID>, <address>, <account>, <bourneShell>, <java>, <weevilRoot>, <type>`

  Each host in a testbed is an *account* on a network *address*. In PlanetLab the account is actually the PlanetLab slice name to which the engineer is assigned. Weevil uses the Bourne shell for experiment framework execution. Thus, each *Host* has the attribute *bourneShellPath* to provide its local path to the program *sh*. The *javaPath* attribute is needed if there is any Java command or Java implementation in component start/stop scripts or actor binary distribution, as described below. *weevilRoot* specifies the workspace on the host assigned for the experiment. At last, through assigning a host to a specific *HostType*, the proper software binary distribution could be used.

- `WVL_SYS_HostProp(<hostID>, <propertyName>, <propertyValue>)`

  Besides the above required properties, A *host* can also have an optional list of properties that can be used to contain any system-specific information that must be known for each host.

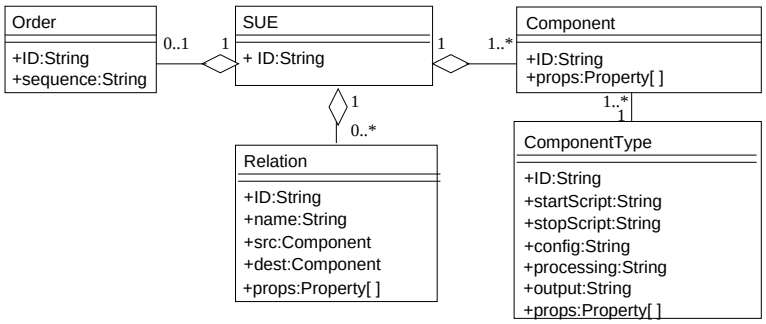

Figure 4.5: SUE Conceptual Model

SUE Model The conceptual model of an SUE is shown in Figure 4.5.

The declaration macros included in this model are:

- `WVL_SYS_SUE(<ID>, <components>, <relations>, <order>)`

  An *SUE* is comprised of typed *Components*, *relations* between them, and the *order* to start up. The *relations* and the *order* are optional arguments. For example,

```
WVL_SYS_SUE( 'Siena', 'S0, S1, S2', 'R10, R20', 'Order')dnl
```

- `WVL_SYS_ComponentType(<ID>, <startScript>, <stopScript>, <config>, <processing>, <logs>)`

  There could be different types of components in an SUE, such as Apache servers and Squid proxies in the Squid/Apache experiments. All the instances of a particular ComponentType tend to share the same formats of start/stop commands and configuration files. They also share the same log file names. Thus, for each *ComponentType*, you should define its *startScript* and *stopScript*, and optionally a *config* attribute that contains the contents of a configuration file. This design allows common attributes to be shared among all components of a type. For different experiments targeting the same system, an engineer would typically need to make minor changes to other entities without having to modify *ComponentType* attributes. The attributes *processing* and *output* are used to specify post-processing of experiment output: a *processing* script is first executed on the component log files, and then the *output* of the script is copied back from each component's workspace to the master. All the Weevil-defined property macros can be referenced in these parameters, and are resolved during script generation using m4's macro expansion. (Please refer to the reference manual for the Weevil-defined property macros.)

  The following example is for the experiments of Squid/Apache system. There are two types of components, `ApacheServer` and `SquidProxy`. Apache could be started under its *bin* directory through the command "*apachectl -k start*", while Squid could be started under its *sbin* directory through command "*squid -N -d 1*". Thus we defined `ApacheServer_startScript`, `SquidProxy_startScript` with some macros to customize to different components. Similar for their stop scripts. Apache and Squid are both configured through configuration files. Thus, we changed their default configuration files by adding macros to it to customize to different components. (Please refer to files *examples/apachesquid/httpd.conf* and *examples/apachesquid/squid.conf*.)

```
WVL_SYS_ComponentType(‘ApacheServer’, ‘ApacheServer_startScript’, dnl
‘ApacheServer_stopScript’, ‘‘include(‘httpd.conf’)’’, ‘’, dnl
‘theCompPath/WVL_Experiment_Name‘_’WVL_Component_ID‘_error.log’, dnl
theCompPath/WVL_Experiment_Name‘_’WVL_Component_ID‘_access.log’’)dnl

WVL_SYS_ComponentType(‘SquidProxy’, ‘SquidProxy_startScript’, ‘SquidProxy_stopScript’, dnl
‘‘include(‘squid.conf’)’’, ‘’, dnl
‘theCompPath/WVL_Experiment_Name‘_’WVL_Component_ID‘_access.log’, dnl
theCompPath/WVL_Experiment_Name‘_’WVL_Component_ID‘_cache.log’, dnl
theCompPath/WVL_Experiment_Name‘_’WVL_Component_ID‘_store.log’’)dnl

dnl
dnl -----ApacheServer Start/Stop Script-----
dnl
define(‘ApacheServer_startScript’, dnl
‘cd WVL_SYS_Echo(‘WVL_Host_’WVL_SYS_Echo(theHost)‘_apacheRoot’)/bin
./apachectl -k start -f theCompPath/WVL_Experiment_Name‘_’WVL_Component_ID.conf &’)dnl

define(‘ApacheServer_stopScript’, dnl
‘cd WVL_SYS_Echo(‘WVL_Host_’WVL_SYS_Echo(theHost)‘_apacheRoot’)/bin
./apachectl -k stop -f theCompPath/WVL_Experiment_Name‘_’WVL_Component_ID.conf’)dnl

dnl
dnl -----SquidProxy Start/Stop Script-----
dnl
define(‘SquidProxy_startScript’, dnl
‘rm -rf WVL_SYS_Echo(‘WVL_Host_’WVL_SYS_Echo(theHost)‘_squidRoot’)/var/cache
WVL_SYS_Echo(‘WVL_Host_’WVL_SYS_Echo(theHost)‘_squidRoot’)/sbin/squid -z
WVL_SYS_Echo(‘WVL_Host_’WVL_SYS_Echo(theHost)‘_squidRoot’)/sbin/squid dnl
-N -d 1 -f theCompPath/WVL_Experiment_Name‘_’WVL_Component_ID.conf &’)dnl

define(‘SquidProxy_stopScript’, dnl
‘read pid <theCompPath/WVL_SYS_Echo(WVL_Experiment_Name)‘_’WVL_Component_ID.pid
echo $pid
kill $pid’)dnl
```

- ,WVL_SYS_ComponentTypeProp(<componentTypeID>, <propertyName>, <propertyValue>)

  ComponentType allows system-specific properties to be assigned to all the instances of this type.

- WVL_SYS_Component(<ID>, <Type>)

  This declaration just defines each component of the SUE as an instance of a componenttype. For example,

  ```
  WVL_SYS_Component(‘S0’, ‘ApacheServer’)dnl
  define(‘Proxies’, ‘P0, P1, P2’)dnl
  WVL_SYS_Foreach(‘i’, ‘WVL_SYS_Component(i, ‘SquidProxy’)’, Proxies)dnl
  ```

- WVL_SYS_ComponentProp(<componentID>, <propertyName>, <propertyValue>)

  Component allows system-specific properties to be assigned to it.

  For example,

  ```
  WVL_SYS_ComponentProp(‘S0’, ‘Port’, 4321)dnl
  WVL_SYS_ComponentProp(‘S1’, ‘Port’, 5432)dnl
  WVL_SYS_ComponentProp(‘S0’, ‘Protocol’, ‘ka’)dnl
  ...
  ```

- `WVL_SYS_ComponentRelation(<ID>, <name>, <src>, <dest>)`

  The *Relations* contained in an SUE model are used to represent any binary associations between components. These are optional and entirely system specific. For instance, in the Squid/Apache example, the proxies works cooperatively. Thus, each proxy component has all the other proxy components as its siblings.

```
WVL_SYS_ComponentRelation(‘RP0P1’, ‘sibling’, ‘P0’, ‘P1’)dnl
WVL_SYS_ComponentRelation(‘RP0P2’, ‘sibling’, ‘P0’, ‘P2’)dnl
WVL_SYS_ComponentRelation(‘RP1P0’, ‘sibling’, ‘P1’, ‘P0’)dnl
WVL_SYS_ComponentRelation(‘RP1P2’, ‘sibling’, ‘P1’, ‘P2’)dnl
WVL_SYS_ComponentRelation(‘RP2P0’, ‘sibling’, ‘P2’, ‘P0’)dnl
WVL_SYS_ComponentRelation(‘RP2P1’, ‘sibling’, ‘P2’, ‘P1’)dnl
```

- `WVL_SYS_ComponentRelationProp(<componentRelationID>, <propertyName>, <propertyValue>)`

  This declaration macro defines any other properties related to a *Relation*.

- `WVL_SYS_ComponentOrder(<ID>, <sequence>)`

  The *order* contained in an SUE model are used to represent the necessary (or preferred) order to start all the components. This is optional and entirely system specific. Some SUEs require some components to be ready before other components, such as Siena. It requires a parent server to be ready before its children start (since they need to communicate with their parent server as soon as they start). Then you should specify the start order of all the components like,

```
WVL_SYS_ComponentOrder(‘Order’, ‘S0, 3, S1, 0, S2, 0’)dnl
```

  In this example, `S0` is the parent of `S1` and `S2`. Please refer to the format of the second argument of this declaration macro. Components are listed in the start order separated with commas. The number between successive component IDs represents the time to wait to start the next component. The number after the last component ID represents the time to wait to start actors. It is useful if a component needs some time to get ready. It can be zero if the start process is very quick or the order of the successive components does not matter (like `S1` and `S2` in this example).

  But for other SUEs, the order does not matter. For instance, in the Squid/Apache example, the proxies works cooperatively. But it does not require a component's sibling to start before it. Then you need not declare the order with this declaration macro. Weevil will start all the components in the order they are defined. Of course, you can always specify a preferred order in such cases.

Mappings The two models described above and the workload are largely independent. This independence means that they can be composed together in many different combinations by providing two mappings between them.

The first mapping simply associates each *component* in the SUE with a *host* in the testbed.

- `WVL_SYS_ComponentHost(<ID>, <component>, <host>)`

  For example,

```
WVL_SYS_ComponentHost('CHMap0', 'S0', 'H0')dnl
```

- WVL_SYS_ComponentHostProp(<componentHostID>, <propertyName>, <propertyValue>)

  If there is any extra limitation related to a component-host mapping. You could specify it here.

- WVL_SYS_ComponentHostType(<componentTypeID>, <hostTypeID>, <binaryDistDir>)

  This declaration macro maps a ComponentType to a HostType. Since the same type of components share the same binary distribution for the same *HostType*. Thus, For each ComponentType-HostType pair, if the software is installed through copying its binary distribution from the master machine to the testbed hosts, this macro needs to be called to assign a different binary distribution for each ComponentType-HostType pair. For example,

```
WVL_SYS_ComponentHostType( 'SienaServer', 'FreeBSD', '/scratch/software/Siena')dnl
WVL_SYS_ComponentHostType( 'SienaServer', 'Linux', '/scratch/software/Siena')dnl
```

  The values of <binaryDistDir> are the same since Siena has the same binary distribution for FreeBSD machine and Linux machine.

- WVL_SYS_ComponentHostTypeProp(<componentHostID>, <hostTypeID>, <propertyName>, <propertyValue>)

  This declaration macro defines any other properties related to a ComponentType-HostType mapping.

The second mapping associates each workload process with a component through an actor. Each workload process must be represented by a single component, but any number of workload processes can be associated with a particular component through actors.

- WVL_SYS_ActorProgram(<ID>, <style>, <binaryDistDir>, <program>, <argument>, <classpath>)

  An actor is implemented as a system-specific program that understands how to actuate the SUE as dictated by the workload line. In other words, you are expected to implement the function of parsing a workload line and sending out a corresponding system command. The program can also receive arguments as configured in the <argument> parameter. You can choose from Java and Shell for the <style> parameter to implement the program in Java or in shell script. Weevil provides a library to support its implementation in Java. (Please refer to the reference manual.) Otherwise, if you are implementing a shell script actor program, and you have *n* arguments, then *$i (i=1,. . . ,n)* represents the *i*th argument. $(n+1) represents the workload line to parse. (Please refer to the *examples/apachesquid/actor/apachesquidactor.sh* as an example.) The <binaryDistDir> is to specify the location of the actor program. If it is a *Java* actor, then you should put the Java classname in the <program> parameter; otherwise, you should put the shell script file name in it. The last parameter <classpath> is used when the execution of the Java actor program needs extra classpaths.

  For example, the shell script actor program declaration in Squid/Apache example is like:

```
WVL_SYS_ActorProgram('DrProgram0', 'Shell', './actor', 'apachesquidactor.sh',dnl
 'ApacheSquidActor_args')dnl
dnl
define('ApacheSquidActor_args', 'dnl
WVL_SYS_Echo('WVL_Host_'theHost'_executablePath')/bin dnl
WVL_SYS_Echo('WVL_Host_'theHost'_address') dnl
WVL_SYS_Echo('WVL_Component_'theComponent'_Port') theActorPath')dnl
```

The Siena's Java actor program needs the class definition in Siena software, thus:

```
WVL_SYS_ActorProgram('DrProgram1', 'Java', './actor', 'SienaActor', 'SienaActor_args', dnl
'SienaActor_classpath')dnl
dnl
define('SienaActor_args', 'WVL_SYS_Echo('WVL_Component_'theComponent'_Protocol') dnl
WVL_SYS_Echo('WVL_Host_'theHost'_address') dnl
WVL_SYS_Echo('WVL_Component_'theComponent'_Port')dnl
WVL_SYS_Echo('WVL_Actor_'WVL_Actor_ID'_Port')')dnl
dnl
define('SienaActor_classpath', 'theSoftwarePath/siena-1.5.1.jar')dnl
```

- `WVL_SYS_Actor(<ID>, <workloadProcess>, <program>, <component>, <dir>)`

  This declaration macro maps a `<workloadProcess>`to a `<component>` through an actor `<ID>` implemented as the actor program `<program>`. The last parameter `<dir>` is the source directory of all the other files the actor needs for experiment execution. For example, the Chord actor *D0* can be defined as,

  ```
  WVL_SYS_Actor('D0', 'C1', 'DrProgram1', 'S2', 'files/C1')dnl
  ```

  The directory `files/C1` is the source directory where all the files to be inserted by `C1` are located.

- `WVL_SYS_ActorProp(<actorID>, <propertyName>, <propertyValue>)`

  You can specify extra actor properties through this macro. For example,

  ```
  WVL_SYS_ActorProp('D0', 'Port', '4322')dnl
  ```

Others Besides the above configurations of conceptual models, there may be some other configurations, such as the macro extension included in the workload. For example, in the Squid/Apache examples, their workloads have the format like

```
event(10,br10,GET(file76))
...
```

The file IDs like `file76` included in the workload are not real filenames. It can be instantiated through the configuration

```
define('files', 'WVL_SYS_Range('', 1, 200)')dnl
WVL_SYS_Foreach('i', 'define('file'i, dnl
'http://''WVL_SYS_Echo('WVL_Host_'WVL_SYS_Echo('WVL_ComponentHost_'S0'_host')'_address')':''dnl
WVL_SYS_Echo('WVL_Component_'S0'_Port')'/test/'i'.txt')', files)dnl
```

in the experiment configuration file.

4.3.4 Setup and Script Generation

The goal of the setup phase is to "compile" the experiment configurations into the control scripts that will be used for experiment deployment and execution. Initially, the configurations are checked for consistency, and a per-experiment Makefile is generated to control the rest of the process, which consists of actions 2 and 3 in Figure 4.3.

Action 2 tailors the workload based on the experiment configurations (such as the file instantiation discussed above) and partitions the unified workload to per-actor workloads. Following that, in action 3, three tailored scripts, a start script, a stop script, and a cleanup script, together with a tailored configuration file are generated for each component in the SUE by applying macro expansion to the contents of the relevant attributes of *ComponentType*. Additionally, a single master control script is created. This whole process is accomplished through the command "*weevil setup-<experiment name>*".

```
examples/apachesquid> ${execPath}/weevil help
Welcome to WEEVIL.

The following commands are available:

  check-all            - checks all experiments.
  check-<experiment>   - checks an experiment's settings.
  clean-all            - cleans all experiments.
  clean-<experiment>   - removes all generated files for an experiment.
  help                 - prints this help message
  run-all              - runs all experiments.
  run-<experiment>     - executes an experiment.
  setup-all            - sets up all experiments.
  setup-<experiment>   - generates all files for an experiment.
  version              - prints the version of weevil

The following experiments are available:

exp_10_10_10 exp_20_20_20 exp_30_30_30 exp_50_50_50 exp_100_100_100
exp_150_150_150 exp_200_200_200 exp_30 exp_60 exp_90 exp_150 exp_300
exp_450 exp_600

examples/apachesquid> ${execPath}/weevil setup-exp_10_10_10
```

4.3.5 Deployment and Execution

```
examples/apachesquid> ${execPath}/weevil run-exp_10_10_10
```

At this point, the engineer can perform an experiment by simply executing the master control script. The master control script deploys the components of the SUE, per-actor workloads, actors, and control scripts to the hosts (action 4), then starts all the components and actors (action 5).

By estimating the round-trip time between the master host and testbed hosts, the master script intelligently decides when to have actors begin processing their workloads. The master script waits for all actors to complete processing their workloads and then causes execution of the stop scripts for each of the components. After all the components terminate, post-processing scripts are executed and output is copied back to the master machine (action 6). Finally, the testbed machines are cleaned up as necessary (action 7).

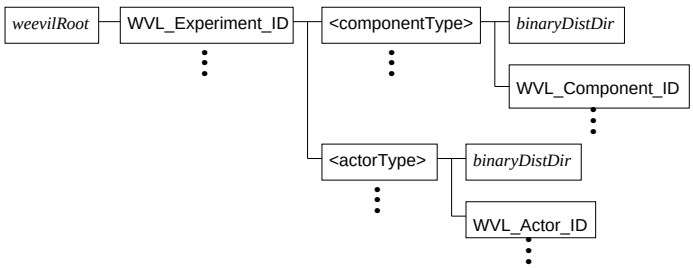

Figure 4.6: Run-Time Directory Structure

The directory structure on each testbed host when weevil executes an experiment is shown in Figure 4.6. The property macros *theWeevilRoot*, *theCompPath*, *theSoftwarePath*, and *theActorPath* are defined correspondingly. (Please refer to the "Some Other Weevil-Defined Property Macros" reference manual.)

4.3.6 Clean up

After the experiment terminates, it will clean up its workspaces either on all the testbed hosts automatically specified by the value *'Y'* of the parameter *clean* of the SUE, or one by one with the generated clean scripts like,

```
examples/apachesquid> sh exp_10_10_10_S0_clean.sh
examples/apachesquid> sh exp_10_10_10_S1_clean.sh ...
```

Besides, the master directory could be cleaned up through

```
examples/apachesquid> ${execPath}/weevil clean-exp_10_10_10
```

5 WS-Guard – Enhancing UDDI Registries with testing capabilities

WS-Guard[1] (WS-Guaranteeing Uddi Audition at Registration and Discovery) provides an implementation of the Audition Framework previously described in the deliverable 4.1. Scope of this chapter is to give technical detail on how the current implementation of the framework can be installed and used. However for a detailed description of the framework please refer to deliverable 4.1.

This short installation and usage manual assumes that the reader has a good knowledge on the basic standards enabling the WS vision. In particular to use the tool the reader has to be acquainted with technologies and standards such such as, XML, SOAP, WSDL and UDDI concepts.

5.1 Overview

The audition framework has been described in detail in the deliverable 4.1 and in a couple of scintific papers [13, 14]. Furthermore in deliverable 4.1 it has been illustrated the architecture of a real implementation of the framework with reduced capabilities with respect to the one presented in [13, 14]. The simplification concerns checks carried on by services receiving invocation by the service under evaluation.

Implementing the framework one of the main objective has been to reuse as much as possible available technologies and to refer to affirmed standards in the WS domain. To set up and use WS-Guard it is then necessary to have a basic knowledge and experience with the following technologies, in addition to the basic ones mentioned above:

- **Tomcat** [3]: Apache Tomcat is a web container developed at the Apache Software Foundation (ASF). Tomcat implements the servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. Tomcat also includes its own internal HTTP server.
- **jUDDI** [2]: this is an Apache open source Java implementation of the UDDI 2.0 specification. The standard UDDI functionality are exposed using servlet based technologies requiring to set up a servlet container at priori. To store all the data structure foreseen by the UDDI specification jUDDI relies on the availability of a DBMS. In principle any DB for which JDBC drivers are available can be used for such purpose. Nevertheless the WS-Guard implementation has been tested only using a MySQL server.
- **UDDI4J** [7]: this is a Java class library that provides an API to interact with a UDDI registry. Using such a library it becomes much mode easier to interact with a UDDI server to publish and discover services.
- **Axis2** [1]: Axis2 is the Apache implementation of a SOAP container. Using Axis the developer will not be overwhelmed with the generation and parsing of SOAP messages. Development and deployment of a Web Service is extremely simplified using Axis. In particular service can be directly derived by a java class through the generation of a corresponding WSDL description (using an enclosed tool called Java2WSDL) and the definition of a WSDD (Web Service Deployment Descriptor) that provides directives to Axis on how to expose the defined interface and to invoke the implemented methods. At run time Axis is mainly composed of a servlet that then have to be deployed within a servlet container such as Tomcat.

[1] part of this work has been developed by Federica Ciotti working at her Master Thesis [16]

WS-Guard applies known methodologies for model based testing derivation and execution. In particular it assumes that a Service State Machine is available for the service under registration. Such SSM will be provided to a service encapsulating the Ambition library and that at the same time will be able to make invocations on the servive under evaluation.

5.2 Technical description

In order to set up the environment to run WS-Guard all the technologies referred above have to be downloaded and correctly installed. In particular WS-Guard assumes the availability of a correct installation of the Apache Tomcat servlet container, on which jUDDI and Axis2 have been previously deployed. At the same time, in order to correctly derive test cases and execute them against the service under evaluation, it is assumed that an instance of a tester service, encapsulating Ambition, is available at the following address:

`http://pc-dispo5.isti.cnr.it:8080/Ambition2/AmbitionService`

As described in deliverable 4.1 WS-Guard relies on an augmented description model for registered services providing, in particular, behavioural definitions of the service. In the current implementation the model is specified using Service State Machine (SSM). Such models are stored in the WS-Guard registry through the use of tModel data structures as defined in the UDDI 2.0 specification. For detail on SSM specification see Chapter 2.

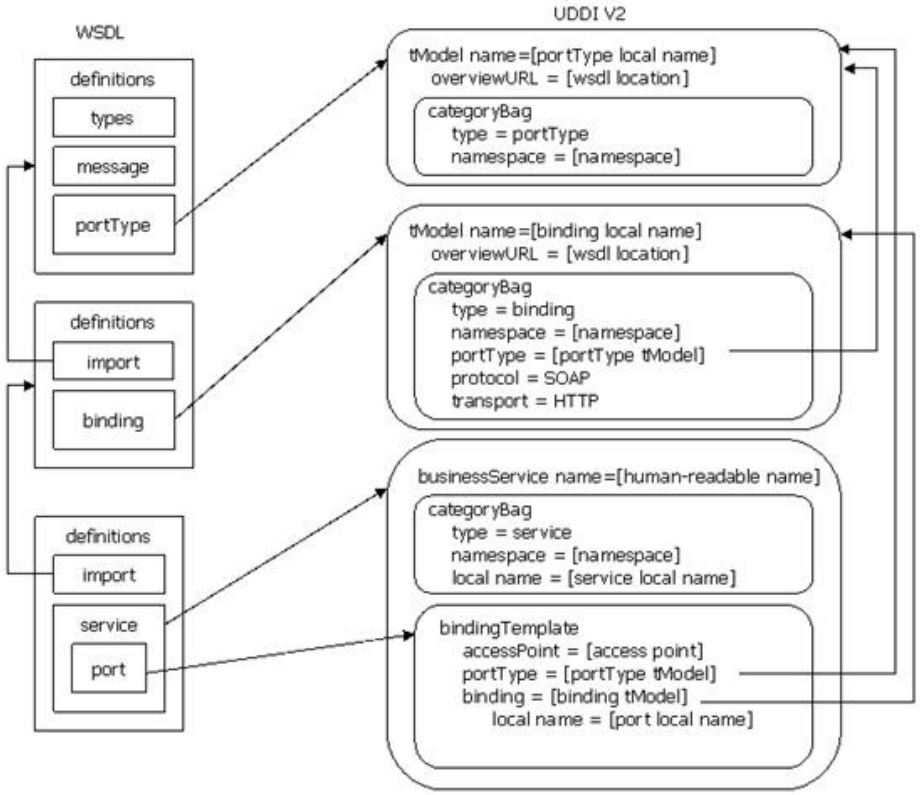

Figure 5.1: WSDL to UDDI mapping

SSM format contains information that put in relation a behavioural specification with a service port implementing it. Figure 5.1 shows the usual mapping of WSDL data to UDDI data structures. However in order to explicitly include support for SSM specification within a UDDI registries it has

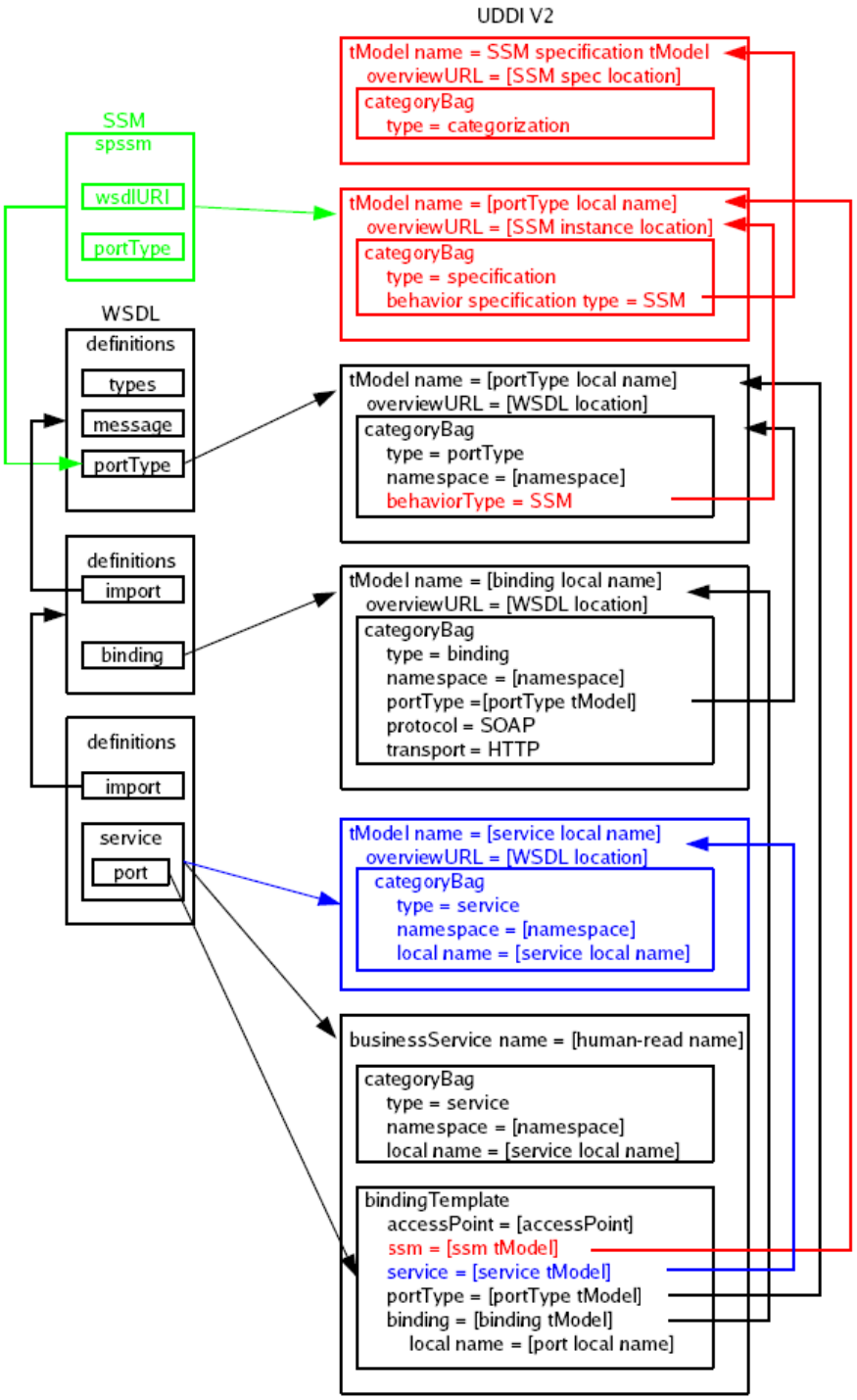

Figure 5.2: SSM/WSDL to UDDI mapping

been necessary to revise the mapping that has been redefined according to what is shown in Figure 5.2.

The additional information and relation are managed by the WS-Guard implementation and no additional effort is required to developers of service to be registered within a WS-Guard registry.

Finally in order to recognize invocations on the inquiry interface fired by a service under evaluation WS-Guard needs to recognize the sender of the inquiry message. In general SOAP messages are sent without including this information and the sender of a message can be recognized

only at the transport layer, for instnace through specific fields of the HTTP packet. Nevertheless WS-Guard require to manipulate this information at the application level. To do this it assumes that each message exchange with possible clients follow formatting rules defined by the WS-Addressing specification [26]. Figure 5.3 shows how, according to the "WS-Addressing - SOAP binding" W3C recommendation [27], the sender can be specified within the header of a SOAP message.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://www.w3.org/2005/08/addressing">
   <S:Header>
    <wsa:To>http://firm1.com/ValutationService</wsa:To>
    <wsa:Action>http://firm1.com/ValutationService</wsa:Action>
    <wsa:MessageID>6B29FC40-CA47-1067-B31D-00DD010662DA</wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://holdingCompany.com/clientService</wsa:Address>
    </wsa:ReplyTo>
   </S:Header>
   <S:Body>     ...
   </S:Body>
</S:Envelope>
```

Figure 5.3: Sender reference specification within a SOAP header

The current implementation of WS-Guard has been tested using the following configuration:

- Sun Java version jdk 1.5.0_11 (with version 1.6.* we experimented some problems with the combined use of UDDI4J)
- Apache Tomcat version 6.0.7
- Apache Axis2 version 1.1.1
- Apache jUDDI version 0.9rc4
- UDDI4J version 2.0.5
- MySQL version 5.0

5.2.1 Installation

Given the pre-requisites specified in the previous sections, concerning the availability of a running tomcat server on which required libraries have been deployed, the installation of WS-Guard is extremely easy. WS-Guard has been implemented as a standard Web Service, therefore using it, will only require to deploy the file containing the service on the correct tomcat directory. In particular to run WS-Guard you have to perform the following steps.

1. deploy the service into Tomcat copying the `UDDIProxyService.aar` in the folder:
   `apache-tomcat-6.0.7/webapps/axis2/WEB-INF/services`
2. Check that the deployment has been correctly performed. If your Tomcat configuration allows hot-deployment just open the browser at the address:
   `http://localhost:8080/axis2/services/listServices`
   In case hot-deployment is not supported restart the server and access the page previously indicated.

3. Since `UDDIProxyService.aar` is an Axis2 service group containing both the Publish and the Inquiry UDDI Web service, the two services should be present in the service list as two separate services, as show in the screenshots of Figure 5.4 and 5.5.

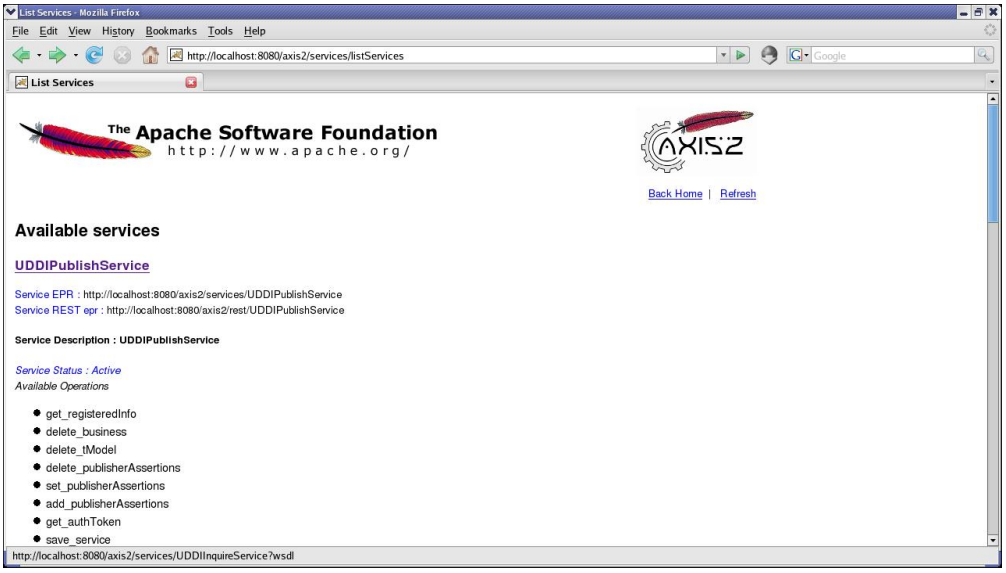

Figure 5.4: WS-Guard publish interface

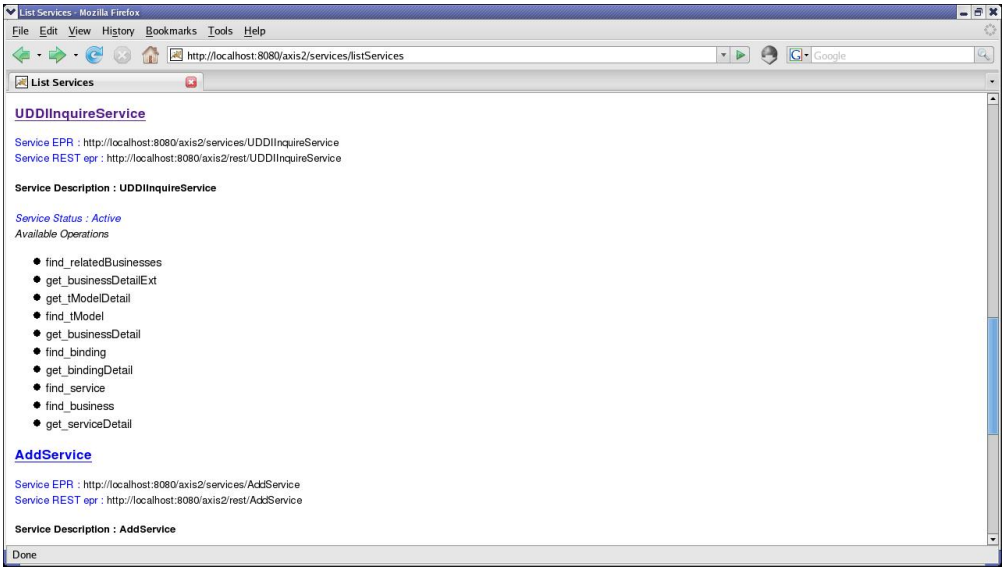

Figure 5.5: WS-Guard inquiry interface

The content of the `UDDIProxyService.aar` can be explored using any program suitable for compressing files. The content is structured in 5 main directories. The `lib` directory contains the library required by the WS-Guard service to correctly run. The directories `org`, `uddi_org`, and `utils` contain the implementation of the service. The structure of the file follows the guidelines defined for the deployment within the Axis2 container. The deployment into another WS technology will probably require to revise the file structure.

Finally in order to create, within the DB, the necessary data structure to store and manipulate SSM information, it is necessary to run the script:

`ssmTmodels.sql`

5.2.2 Usage

WS-Guard provides a ready to use UDDI registries with testing capabilities. Differently from a jUDDI registries functionality are exposed directly with web service interfaces. The interaction with WS-Guard have to follow the rules defined above in particular SOAP messages must contains the sender address in the header section.

The directory `client` contains the code for a generic simple client that can be used to acquire a better understanding on the usage of the WS-GUard registry. Under Linux the client can be launched using the command:

```
sh startClient.sh
```

available in the directory `client`

5.3 Planned improvements

WS-Guard will not be further developed within PLASTIC. Currently among the various case studies under development within PLASTIC, no one seems to show the necessity for having a directory service with testing capabilities. This chapter has been enclosed mainly for completeness with respect to the testing phases illustrated in the deliverable 4.1. Nevertheless there are several possibilities to extend the current implementation of the framework in particular in order to include and improve mechanisms for usage based checking. Such mechanisms refer to the possibility that services invoked by the service under audition check if it make correct invocation with respect to the protocol (order of messages) and/or expected pre-conditions. Exptensions/improvements to the current implementation of the framework will have to be planned in case WS-Guard will be adopted in one or more case studies.

6 Dynamo-AOP - Dynamic Monitoring of BPEL processes

Dynamo-AOP[1] is a framework for monitoring functional properties of external services which a BPEL[9] process interacts with, to realize a composite service. It is based on the conceptual model proposed in [11], but, with respect to the original design, its architecture is based on the dynamic aspectization of the BPEL engine executing the monitored service compositions, achieved by using AspectJ [19] as an AOP [20] language.

In the rest of this chapter, we first describe the language used to specify the properties to monitor and the architecture of the monitoring framework; than, we provide a short installation and usage guide and we conclude by presenting future improvements of the tool.

6.1 WS-CoL

WSCoL (Web Service Constraint Language) is the language used inside Dynamo-AOP to define monitoring properties; it is based on JML [21], with some conceptual and syntactical differences due to the adaption to the world of web services. Its main features are:

- Allowing to define and predicate on variables containing the data originating both within and outside the monitored BPEL process, and to retrieve data previously stored in a storage component.
- Using predefined variable functions for data manipulation.
- Using typical boolean, relational and arithmetic operators.
- Predicating on sets of variables through the use of the universal and existential quantifiers, and aggregate operators.

WSCoL allows to attach monitoring properties to the activities of a BPEL process that interact with external services. Properties can be pre- and post-conditions. Indeed, one can attach a pre-condition to an *invoke* activity, and a post-condition to an *invoke*, a *receive* and a *pick* activity.
All monitoring properties have three parameters:

- *priority*: it represents the “importance” of the rule and can be an integer ranging from 1 to 5. Each process can then define a threshold value that makes monitoring properties active or not, allowing for dynamically changing the amount of activities performed for monitoring the process.
- *validity*: it defines time constraints on *when* a monitoring properties should be considered.
- *trusted providers*: it’s a list of service providers for which monitoring is not necessary.

In its simplest definition, a monitoring property states relationships that must hold between variables. WSCoL supports three kinds of variables:

- *internal*: an internal variable corresponds to a datum that originates within the process being monitored. Usually, WSCoL internal variables define a form of *data extraction* from complex

[1] Part of this work has been developed by Lorenzo D’Ercole and Luca Gallupi, as part of their master theses, under the supervision of Luciano Baresi and Sam Guinea.

BPEL variables, by using XPATH [25] expressions. For example, to extract the value of a sub-element `easting` from the sub-element `start` of the element `parameters` of a complex BPEL variable named `getRouteIn`, one can use the notation `$(getRouteIn/parameters/start/easting)`, i.e. by concatenating the variable name prefixed by a dollar sign, with the XPATH expression matching the value of interest.

- *external*: An external variable indicates a monitoring datum that originates outside of the process in execution, such as a contextual datum. WSCoL assumes that the data source of an external variable can be queried through a web service interface and provides a function to invoke it: `(return<X> (W, O, Ins, Out))` where
  - `X` indicates the XSD type returned by the data source web service; it can be `Num`, `String`, `Bool`.
  - `W` represents the location of the WSDL of the data source web service.
  - `O` represents the name of the web method of the data source web service.
  - `Ins` represents the input message that you have to send when calling the data source web service.
  - `Out` represents an XPATH expression indicating the data extraction to apply to the output message returned from the data source web method, to get the desired value.
- *historical*: Historical variables consist of monitoring data that are related to previous activations of the Dynamo-AOP framework, related either to other processes or previous steps in the same process. Historical variables are defining using the `store` construct, as follows: `store $east_historical=($getRouteIn/parameters/start/easting);`. Previously stored historical variables can be retrieved using the `retrieve` function `(retrieve(pID,uID,iID,kID,type,alias,n))` where `pID` identifies the process family, `uID` is the user-id of the user who run the processes, `iID` identifies the instance within the process family, `kID` identifies an *invoke* activity in the process, `type` specifies if the historical variables was stored in a pre- or post-condition; `alias` is the name of the variable used in the `store` operation, `n` is the maximum number of results that should be returned by the query.

Moreover, variables may be aliased. This feature is provided both for allowing for less verbose expressions and for referring multiple times to a variable, whose value has been collected only once. Aliases are defined using the `let` keyword as shown below:
`let $east=($getRouteIn/parameters/start/easting);`.

Variables can be manipulated using data-type specific functions, invoked on the variable using the dot notation. Numeric functions include `abs()`, `ceiling()`, `floor()` and `round()`. The available string functions are: `compare(string)`, `replace(pattern, replace)`, `substring(start, len)`, `length()`, `contains(string)`, `startsWith(string)`, `endsWith(string)`. For example, to get the length of the string value referred by the alias `east`, one can write `$east.length()`.

WSCoL allows to use universal and existential quantifiers to express constraints over sets of value. The syntax is: `(quantifier $alias_name in range_def; constraint_def)` where `quantifier` can be `forall` or `exists`. A universal quantifier indicates a parametric constraint that must be true for each and every value (at least one, in the case of the existential quantifier) the parameter can assume in a given range. `alias_name` and `range_def` defines respectively a variable alias and a finite range for the values it can assume; `constraint_def` defines the parametric constraint that must hold.

Aggregate operators allows to define assertions on set of values. The syntax is
`(operator $alias_name in range_def; assertion_def)`

where `operator` is one among `max`, `min`, `avg`, `sum`, `product`; `range_def` is a variable that returns a set of values, and `alias_name` is an alias that can be used as a parameter in `assertion_def`.

When a violation of a property is detected, some sort of recovery action should be performed. In our framework we have not investigated any specific recovery strategy but, by simplifying the work presented in [18], we just provides three simple primitives: `ignore`, which ignores the violation and allows the process to continue, `halt`, which stops the execution of the process, and `notify(msg, addr)`, which sends an email with the text `msg` to the recipient `addr`. These strategies can be structured using an `if-elseif-else` statement and/or boolean operators. For example, if a value is found to have a value below a certain threshold, we might want to differentiate the recovery strategy on the basis of the difference with respect to the threshold, as shown below:

```
if ($hres < 80)
   { halt() and notify("Low resolution, dba@localhost)}
elseif ($hres > 80 && $hres < 120)
  {ignore ()  and notify("Medium resolution, dba@localhost)}
else {ignore()}
```

In the next section, we list the complete grammar of WSCoL.

6.1.1 WS-CoL grammar

⟨analyzer⟩ ::⇒ ⟨rules⟩ | ⟨recovery⟩
⟨recovery⟩ ::⇒ ⟨complete-strategy⟩ | strategy
⟨complete-strategy⟩ ::⇒ ⟨ifStrategy⟩ ⟨elseIfStrategy⟩* ⟨elseStrategy⟩?
⟨ifStrategy⟩ ::⇒ **if** ⟨condition⟩ ⟨strategy⟩
⟨elseIfStrategy⟩ ::⇒ **elseif** ⟨condition⟩ ⟨strategy⟩
⟨elseStrategy⟩ ::⇒ **else** ⟨strategy⟩
⟨condition⟩ ::⇒ **(** ⟨rules⟩ **)**
⟨strategy⟩ ::⇒ { ⟨steps⟩ }
⟨steps⟩ ::⇒ ⟨rec-step⟩ (**or** ⟨rec-step⟩)*
⟨rec-step⟩ ::⇒ ⟨actions⟩
⟨actions⟩ ::⇒ action (**and** ⟨action⟩)*
⟨action⟩ ::⇒ ⟨identifier⟩ **(** ⟨list⟩? **)**
⟨rules⟩ ::⇒ ⟨sub-rule⟩ ((**==>** | <**==** | <**==>**) ⟨sub-rule⟩)* **;**
⟨sub-rule⟩ ::⇒ ⟨and-expression⟩ (|| ⟨and-expression⟩)*
⟨and-expression⟩ ::⇒ ⟨equals-expression⟩ (**&&** ⟨equals-expression⟩)*
⟨equals-expression⟩ ::⇒ ⟨relational-expression⟩ ((**==** | **!=**) ⟨relational-expression⟩)?
⟨relational-expression⟩ ::⇒ ⟨operator-expression⟩ ((> | >**=** | < | <**=**) ⟨operator-expression⟩)?
⟨operator-expression⟩ ::⇒ ⟨basic-expression⟩ ((**+** | **-** | ***** | / | **%**) ⟨basic-expression⟩)*
⟨basic-expression⟩ ::⇒ ⟨dot-expression⟩ | ⟨variable⟩ | ⟨exists⟩ | ⟨forall⟩ | ⟨let⟩ | ⟨store⟩ | ⟨avg⟩ | ⟨min⟩ | ⟨max⟩ | ⟨sum⟩ | ⟨product⟩ | **true** | **false** | ⟨NUMBER⟩ | ⟨string-value⟩
⟨forall⟩ ::⇒ **(forall $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨exists⟩ ::⇒ **(exists $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨dot-expression⟩ ::⇒ ⟨variable⟩ **.** ⟨identifier⟩ **(** ⟨list⟩? **)**
⟨let⟩ ::⇒ **let $** ⟨identifier⟩ **=** ⟨sub-rule⟩
⟨store⟩ ::⇒ **store $** ⟨identifier⟩ **=** ⟨sub-rule⟩
⟨avg⟩ ::⇒ **(avg $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨sum⟩ ::⇒ **(sum $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨min⟩ ::⇒ **(min $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨max⟩ ::⇒ **(max $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨product⟩ ::⇒ **(product $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨variable⟩ ::⇒ **(** (⟨ivar⟩ | ⟨evar⟩ | ⟨hvar⟩) **)** | (⟨ivar⟩ | ⟨evar⟩ | ⟨hvar⟩)
⟨ivar⟩ ::⇒ **$** ⟨identifier⟩ ⟨xpath-expression⟩?
⟨evar⟩ ::⇒ ⟨returnType⟩ **(** ⟨string-value⟩ **,** ⟨string-value⟩ **,** ⟨string-value⟩ **,** ⟨xpath-expression⟩ **)**

⟨returnType⟩ ::⇒ **returnInt** | **returnBool** | **returnString**
⟨hvar⟩ ::⇒ **retrieve (** ⟨string-value⟩ (**,** ⟨string-value⟩)$^{?}$ (**,** ⟨NUMBER⟩) **,** ⟨xpath-expression⟩ **,** ⟨NUMBER⟩ **, $** ⟨identifier⟩ (**,** ⟨NUMBER⟩) **)**
⟨alias⟩ ::⇒ **$** ⟨identifier⟩
⟨list⟩ ::⇒ ⟨sub-rule⟩ (**,** ⟨sub-rule⟩)*
⟨string-value⟩ ::⇒ ⟨sub-string-value⟩ (**+** ⟨sub-string-value⟩)*
⟨sub-string-value⟩ ::⇒ ⟨identifier⟩ | ⟨literal⟩ | ⟨variable⟩
⟨xpath-expression⟩ ::⇒ ⟨union-expression⟩
⟨location-path⟩ ::⇒ ⟨absolute-location-path⟩ | ⟨relative-location-path⟩
⟨absolute-location-path⟩ ::⇒ (/ | //) (⟨i-relative-location-path⟩ | ε)
⟨relative-location-path⟩ ::⇒ ⟨i-relative-location-path⟩
⟨i-relative-location-path⟩ ::⇒ ⟨step⟩ ((/ | //) ⟨step⟩)*
⟨step⟩ ::⇒ ((⟨axis⟩ | ε) ((((⟨identifier⟩ **:**)$^{?}$ (⟨identifier⟩ | ***))) | ⟨special-step⟩) ⟨predicates⟩*) | ⟨abbr-step⟩ ⟨predicates⟩*
⟨special-step⟩ ::⇒ **processing-instruction (** ⟨identifier⟩$^{?}$ **)** | (**comment** | **text** | **node**) **( )**
⟨axis⟩ ::⇒ ⟨identifier⟩ **::** | **@**
⟨predicate⟩ ::⇒ { ⟨predicate-expr⟩ }
⟨predicate-expr⟩ ::⇒ ⟨expr⟩
⟨expr⟩ ::⇒ ⟨or-expr⟩
⟨primary-expr⟩ ::⇒ ⟨variable-reference⟩ | **(** ⟨expr⟩ **)** | ⟨literal⟩ | ⟨number⟩ | ⟨function-call⟩
⟨literal⟩ ::⇒ ⟨LITERAL⟩
⟨number⟩ ::⇒ ⟨NUMBER⟩
⟨variable-reference⟩ ::⇒ **$**⟨identifier⟩
⟨function-call⟩ ::⇒ ⟨identifier⟩ **(** ⟨arg-list⟩$^{?}$ **)**
⟨arg-list⟩ ::⇒ ⟨argument⟩ (**,** ⟨argument⟩)*
⟨argument⟩ ::⇒ ⟨expr⟩
⟨union-expr⟩ ::⇒ ⟨path-expr⟩ (| ⟨path-expr⟩)*
⟨path-expression⟩ ::⇒ ⟨location-path⟩ | ⟨filter-expr⟩ ⟨absolute-location-path⟩$^{?}$
⟨filter-expr⟩ ::⇒ ⟨primary-expr⟩ ⟨predicate⟩$^{?}$
⟨or-expr⟩ ::⇒ ⟨and-expr⟩ (**or** ⟨and-expr⟩)*
⟨and-expr⟩ ::⇒ ⟨equality-expr⟩ (**and** ⟨equality-expr⟩)*
⟨equality-expr⟩ ::⇒ ⟨relational-expr⟩ ((**=** | **!=**) ⟨relational-expr⟩)$^{?}$
⟨relational-expr⟩ ::⇒ ⟨additive-expr⟩ ((< | <**=** | > | >**=**) ⟨additive-expr⟩)$^{?}$
⟨additive-expr⟩ ::⇒ ⟨mult-expr⟩ ((**+** | **-**) ⟨mult-expr⟩)$^{?}$
⟨mult-expr⟩ ::⇒ ⟨unary-expr⟩ ((* | /) ⟨unary-expr⟩)$^{?}$
⟨unary-expr⟩ ::⇒ ⟨union-expr⟩ | **-** ⟨unary-expr⟩

6.2 Architecture

Figure 6.1 depicts the Dynamo-AOP monitoring framework, by illustrating the dependencies existing between the various components and the technologies used in the implementation. The Configuration Manager is a storage component for all the properties that have to be monitored. The ActiveBPEL engine is a modified version of ActiveBPEL [8] in which we embed monitoring. This is achieved by following an aspect oriented programming approach [20]. The engine is a Java program in which we weave the cross-cutting monitoring features via AspectJ [19]. ActiveBPEL works by creating an internal tree representation of the process being executed. In this tree, each node represents a single BPEL activity in the process definition, and is an appropriate extension of the `AEActivityDefinition` class. Each node contains the information necessary to perform the particular activity it is associated with. At run time, the tree is visited and the definition classes are used by the engine to instantiate appropriate `AEActivityImpl` classes, all of which implement a common interface. Amongst other things, this interface provides an `execute` method where the activity's primary action is performed. For example, a *scope* activity will set up its internal variables, while an *invoke* activity will perform the appropriate external invocation. To perform

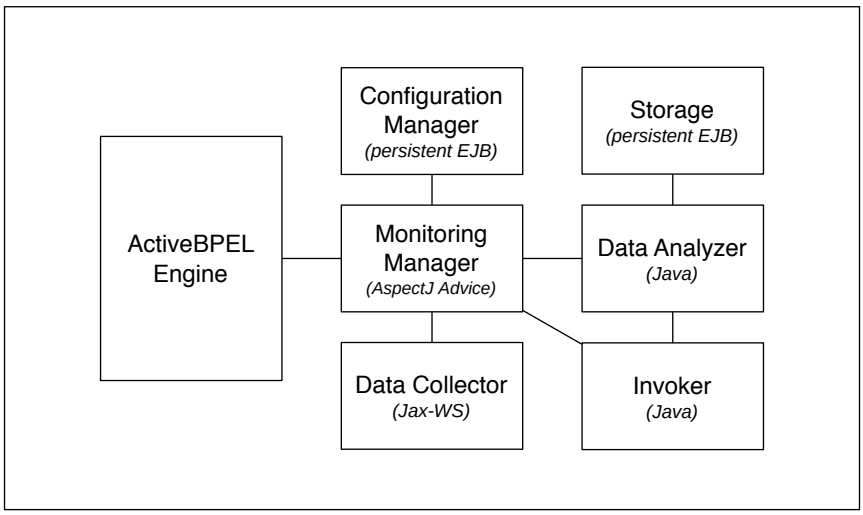

Figure 6.1: Dynamo-AOP components architecture

monitoring, we intercept the process after the `execute` method is called for the various BPEL activities. These are the points where the Monitoring Manager (implemented as an AspectJ advice) is activated. Its main responsibility is data collection, both from within the process and from the outside world, through the Data Collector. The collected data, together with the monitored property, are sent to the Data Analyzer, which first retrieves external data and/or historical variables by calling the Invoker and the Storage, respectively, and then analyzes the property, passing the result of the evaluation back to the Monitoring Manager. At this point, before returning control to the process, the Monitoring Manager executes the recovery code if a monitoring rule has been violated.

6.3 Installation and usage

6.3.1 Installation

The following components need to be installed in order to run Dynamo-AOP:

- the Apache Tomcat servlet container (assumed to be installed in the directory `$TOMCAT-DIR` and running on port `7080`), available at `http://tomcat.apache.org/`
- the JBoss application server (assumed to be installed in the directory `$JBOSS-DIR` and running on port `8080`), available at `http://www.jboss.org/`
- the ActiveBPEL BPEL engine, available at `http://www.active-endpoints.com`
- (optional) a mail server, to support the `notify` recovery strategy.

The current version of Dynamo-AOP has been tested using Apache Tomcat ver. 5.5.23, JBoss Application Server ver 4.2, ActiveBPEL ver 2.1.
The following libraries are also required:

- ANTLRv2, available at `http://www.antlr2.org/`
- Apache Axis, available at `http://ws.apache.org/axis/`
- Apache XMLBeans, available at `http://xmlbeans.apache.org/`

- Apache Xerces 2, available at http://xerces.apache.org/xerces2-j/index.html
- Castor, available at http://www.castor.org
- Jakarta Commons Discovery, available at http://commons.apache.org/discovery/
- Jakarta Commons Logging, available at http://commons.apache.org/logging/
- CommonJ Timer and Work Manager for Application Servers, available at http://dev2dev.bea.com/wlplatform/commonj/twm.html
- Javamail, available at http://java.sun.com/products/javamail/
- JAXP, available at https://jaxb.dev.java.net/
- Jaxen, available at http://jaxen.org/
- JAX-RPC, available at https://jax-rpc.dev.java.net/
- JBoss EJB3, available at http://labs.jboss.com/jbossejb3/
- JBoss Web Services, available at http://labs.jboss.com/jbosssws/
- Saxon XSLT processor, available at http://saxon.sourceforge.net/
- StAX, available at http://stax.codehaus.org/
- Sun Java Streaming XML Parser (JSR 173), available at http://java.sun.com/webservices/docs/1.5/sjsxp/ReleaseNotes.html
- WSDL for Java API, available at http://sourceforge.net/projects/wsdl4j
- XML Pull Parser (XPP), available at http://www.extreme.indiana.edu/xgws/xsoap/xpp/
- XSUL, available at http://www.extreme.indiana.edu/xgws/xsul/

To actually install the Dynamo-AOP monitoring framework you have to:

- copy from the Dynamo-AOP distribution `ConfigurationManager.jar`, `HistoricalVariable.jar`, `MonitorLogger.jar` in the directory `$JBOSS-DIR/server/default/deploy`.
- copy from the Dynamo-AOP distribution `ae_rtbpel.jar`, in the directory `$TOMCAT-DIR/shared/libs`, overwriting the original ActiveBPEL file.
- copy from the Dynamo-AOP distribution `invoker.jar` in the directory `$TOMCAT-DIR/common/libs`.
- install a copy of `antlr-2.7.6.jar`, `antlrdebug_1.0.0.jar`, `jaxb-api.jar`, `jaxb-impl.jar`, `jaxb-xjc.jar`, `jaxb1-impl.jar`, `mail.jar`, `jsr173_api.jar`, `saxon8-dom.jar`, `saxon8-jdom.jar`, `saxon8-sql.jar`, `saxon8-xom.jar`, `saxon8-xpath.jar`, `saxon8.jar`, `xbean_path.jar`, `xbean.jar`, `xmlpublic.jar`, `aspectjrt.jar`, `xpp3-1.1.3.4.M.jar`, `xsul-2.0.9_2.jar`, `stax-1.1.1.jar` in the directory `$TOMCAT-DIR/common/libs`.
- install a copy of `axis.jar`, `commons-logging.jar`, `commons-discovery-0.2.jar`, `jaxrpc.jar`, `saaj.jar`, `wsdl4j-1.5.1.jar`, `castor-0.9.6-xml.jar`, `commonj-twm.jar`, `jaxen-1.1-beta-8.jar`, `ssaj-api.jar`, `xercesImpl.jar`, `saxon8-dom.jar` in the directory `$TOMCAT-DIR/shared/libs`.

6.3.2 Usage

1. launch the JBoss application server using the command `$JBOSS-DIR/bin/run.sh` and wait for the completion of the starting phase; check that JBoss is running by typing in your browser `http://localhost:8080`: you should see the start page of the application server.

2. launch the Apache Tomcat servlet container using the command `$TOMCAT-DIR/bin/catalina.sh run` and wait until the message "ActiveBPEL In-Memory Configuration Started" is displayed on the console. Check that Tomcat is running by typing in your browser `http://localhost:7080`. To check that ActiveBPEL is running, visit `http://localhost:7080/BpelAdmin`: you should see the ActiveBPEL administration tool. On it, click on Configuration and uncheck the box labelled "Validate Input/Output messages against schema".

3. deploy your BPEL process, as illustrated in ActiveBPEL user guide.

4. configure monitoring for the deployed process. This step assumes an interaction with the ConfigurationManager, which — in the first prototype of Dynamo-AOP — is done by directly accessing the API of the component, as shown in the "Dynamo Supervision Manager" application, also distributed within the framework (see next section).

6.4 Example

In the software distribution, you will find a complete web application that can be used to see how the monitoring framework works.

The application contains a BPEL process, `PizzaDeliveryCompany`, which you should deploy in the BPEL engine; at the end of the deployment you should see the process listed in the Deployed Processes section of the ActiveBPEL control panel, as shown in Figure 6.2. Before using the demo application, you should deploy some web services in the application server, by copying all the `*.jar` file from the `demo` directory to `$JBOSS-DIR/server/default/deploy`. In the same directory, you should also copy `dynamo.war`, which contains the "Dynamo Supervision Manager", a web application developed to to control the behavior of the monitoring framework; it can be reached at `http://localhost:8080/dynamo`.

The main page of this application contains two links: one to set the properties of the monitored processes and the other to access the Dynamo-AOP demo, also available directly at `http://localhost:8080/dynamo/DemoManager.jsp`. This page contains the various steps through which you can interact with the BPEL process. The execution of step #1 will attach some monitoring rules to the process, as shown in Figure 6.3. The execution of the monitor process can then be monitored on the output console of Tomcat, as shown in Figure 6.4. The structure and the priority of the rules attached to a monitored process can be modified using the "Dynamo Supervision Manager", as shown in Figure 6.5.

6.5 Future improvements

The main limitation of the current version of the Dynamo-AOP framework is that it monitors only functional properties of stateless services; on the basis of this consideration, we envision two different directions of improvement.

The first one will concern the kind of properties that can be monitored: the specification language should be extended to allow for expressing also extra-functional properties of services; an initial attempt to define such a specification language is described in [10]. Moreover, this extension

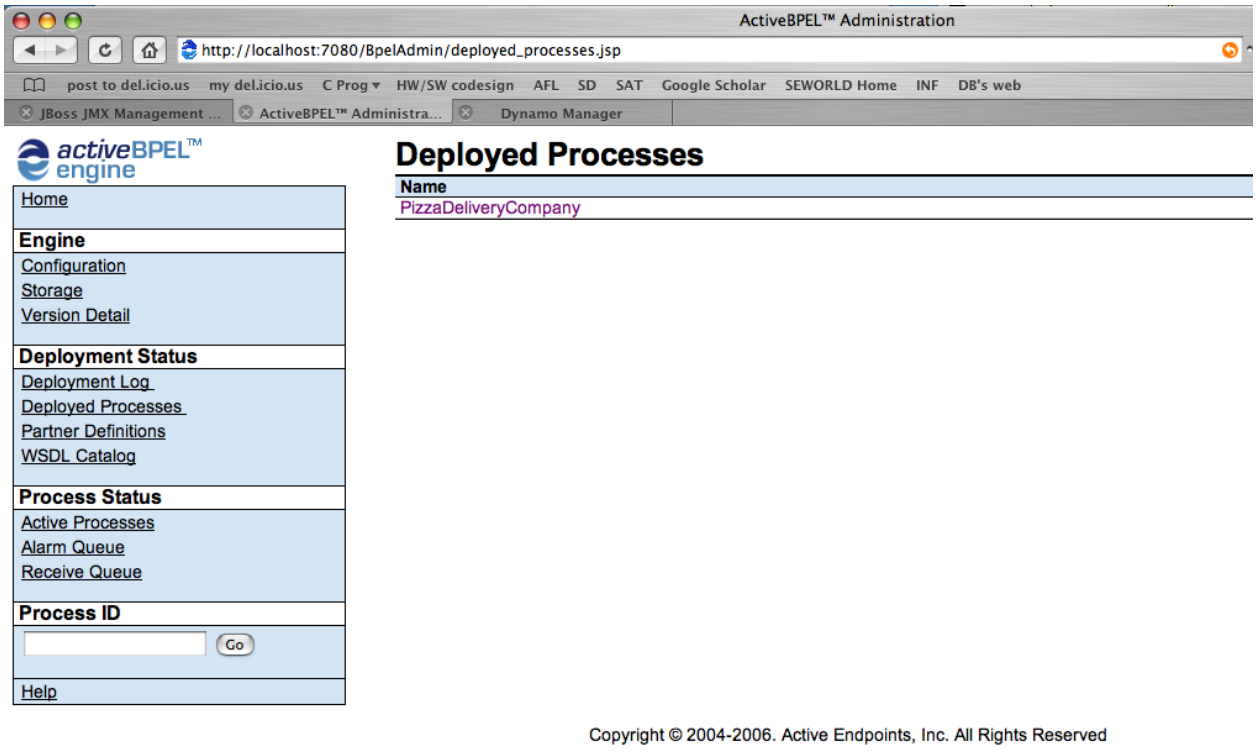

Figure 6.2: Process `PizzaDeliveryCompany` deployed successfully.

should also be integrated with the existing approaches on monitoring QoS properties, developed within WP4.

The second extension will focus on extending both the monitoring framework and the specification language to support monitoring conversational web services, a specific class of stateful services. This extension will be based on the work described in [15].

Other improvements will deal with the usability of the monitoring framework: mainly, we plan to develop a graphical user interface, blended within the ActiveBPEL control panel, to configure the parameters and the monitored properties of a process.

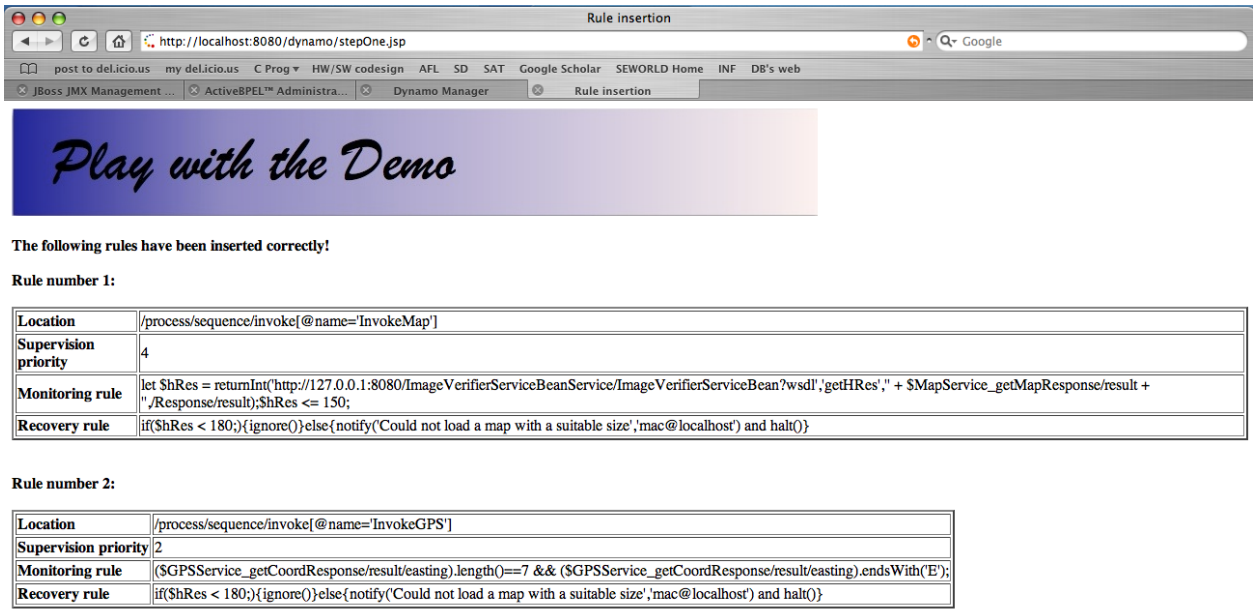

Figure 6.3: The result of inserting two monitoring rules.

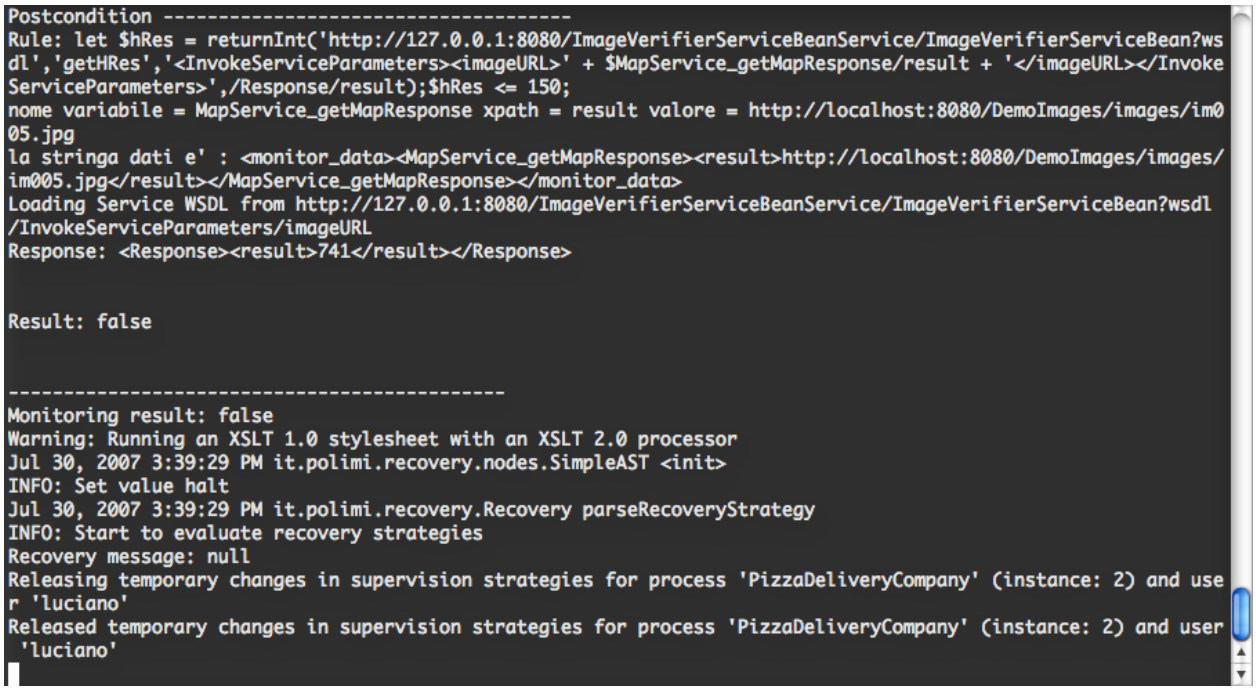

Figure 6.4: Output console of the monitored process

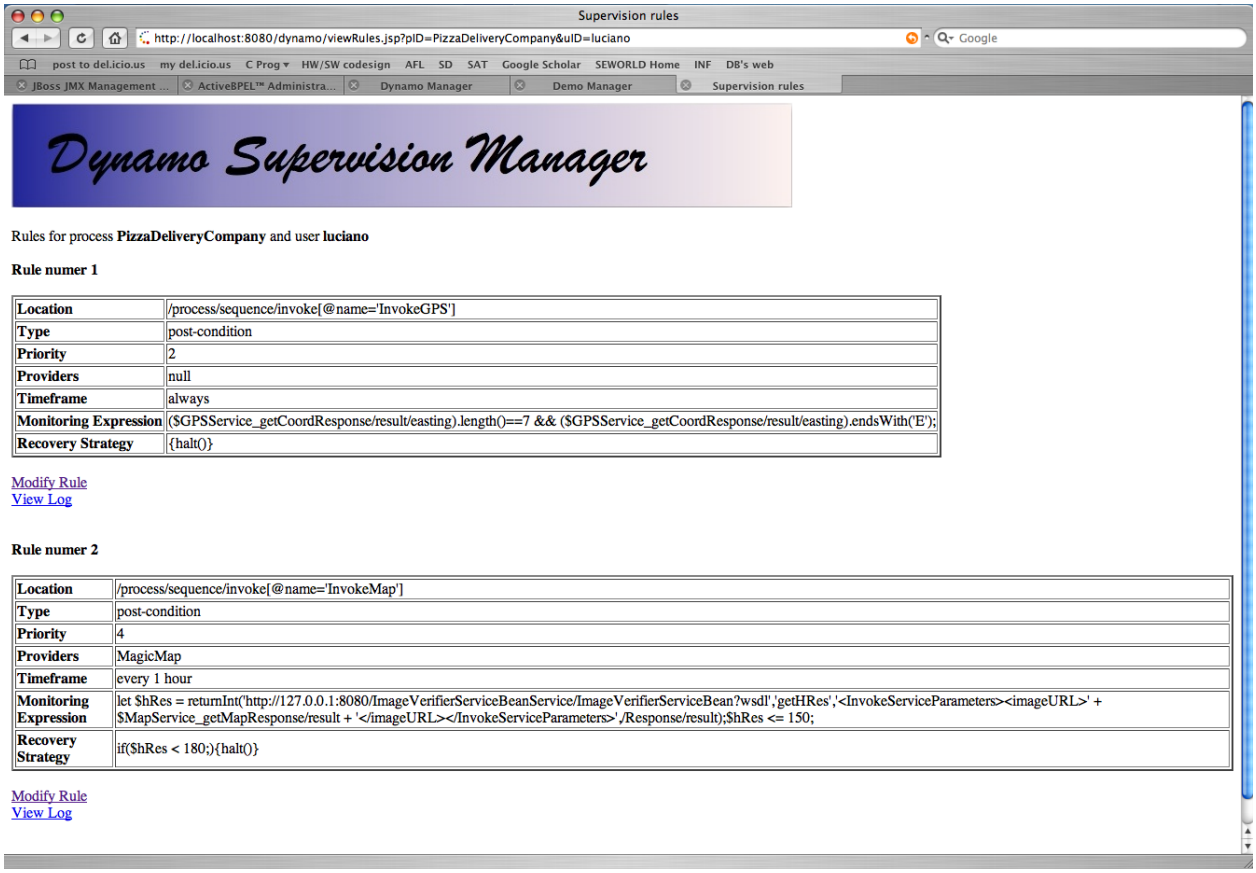

Figure 6.5: Monitoring rules modified with the “Dynamo Supervision Manager”

7 SLAngMon – Efficient On-line Monitoring Service Level Agreements

SLAngMon is an Eclipse plugin to generate Java monitors for Service Level Agreements defined using the SLAng language. We refer to Deliverable 2.1 for a detailed description of SLAng.

SLAngMon implements the automata-based technique presented in D4.1 to monitor agreements by introducing AXIS handlers to intercept messages exchanged and to check their conformance with respect to the established SLAs.

7.1 Overview

SLAngMon output includes

- Java classes implementing the automata corresponding to checkers for SLAs.
- Java classes extending AXIS Basic Handlers, which should be invoked by appropriately modifying the chain of invocations.

For the purposes of this document, it is assumed that the reader is familiar with the following technologies:

- Apache, Tomcat, and AXIS.
- Eclipse and Eclipse plugins (with Ant builds)
- Meta-modelling, EMOF and OCL: please see Deliverables 6.2 and 2.1 for further details.

The remainder of this section illustrates how to install the plugin and how to use it.

7.2 Installation instructions

These are the steps required to install SLAngMon:

1. Open Eclipse.
2. Add the following CVS repositories:

   ```
   Host: uclmda.cvs.sourceforge.net
   CVS path: /cvsroot/uclmda
   User: anonymous (no password)
   ```

   In HEAD, checkout the following: UCL, GTL, EMOFOCL2, EMOFOCLPlugin
3. Compile UCL:
   - Rename project.properties.example in project.properties and edit
   - Execute build.xml
4. Compile GTL:

- You need to obtain gnu.regexp-1.1.4.tar.gz, unzip, and place it in the lib/ directory (the archive can be obtained by looking for it in Google).
- Execute build.xml

5. Compile EMOFOCL2
   - Rename project.properties.example in project.properties and edit
   - Execute build.xml
6. Compile EMOFOCLPlugin
   - Rename project.properties.example in project.properties and edit
   - Execute build.xml
7. Create a new general project with name SLAng
   - Unzip SLAng.zip and import everything into new project SLAng
   - Edit project.properties
   - Run ant target ”clean”
   - Run ant target ”all”
8. Create a new general project with name SLAngTA
   - Unzip SLAngTA.zip and import everything into new project SLAngTA
   - Edit project.properties
   - Run ”all”

The plugin is at this point compiled and ready to be used.

7.3 Structure of the plugin and usage instructions

This is a brief overview of the structure of the project:

- SLAng is defined in the file `src/uk/ac/ucl/cs/slangta/specification/slang.emof`. By modifying this file it is possible to define / modify SLAng clauses to define custom SLAs
- The SLA editor with the SLAngMon plugin is defined in src/uk/ac/ucl/cs/slangta/editor/SLAngTAEditor.java.
- The actual pluging to generate the checkers is defined by the files in src/uk/ac/ucl/cs/slangta/editor/action and src/uk/ac/ucl/cs/slangta/editor/tautils (this is the automata stuff)

By modifying any of the previous entities, SLAngMon can be easily customized to suit many applications scenarios.

Usage. After compiling the software as described in the previous section, right-click on SLAngTA, Run As → Run... Run as Eclipse application. A new Eclipse instance should start. In this new instance, create a new General project. Then, create a new file, for instance `test.slangtaxmi` (notice: it is important to use `.slangtaxmi` extensions to activate the plugin). As an example, complete an InputThroughput clause, right click, and choose a destination directory. An example of the plugin at this point is depicted in Figure 7.1. This will generate a directory structure. The actual checker (the automaton) is generated in server/tautils. The handler for AXIS is generated in server/slamonitor

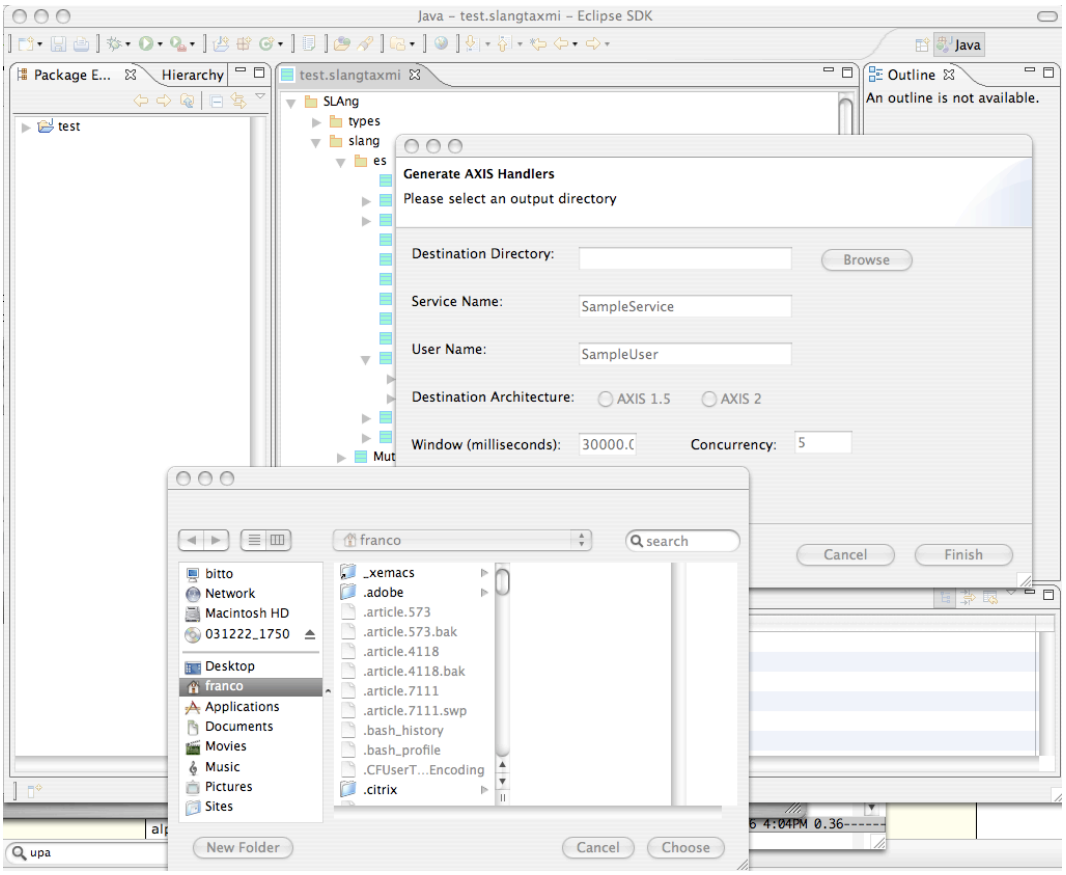

Figure 7.1: Plugin screenshot for SLAngMon: generation of checkers.

The generated java files can be compiled using the appropriate classpath for AXIS libraries. Copy the files and their container directories in `$AXIS_HOME/classes`

To install the handler, modify the file `$AXIS_HOME/server-config.wsdd`, for instance:

```
<handler name="slamonitor" type="java:slamonitor.SLAMonitor">
  <parameter name="filename" value="SLAMonitor.log"/>
  <parameter name="wsdlURL" value="/axis/SLAMonitorService-impl.wsdl"/>
  <parameter name="scope" value="Session"/>
  <parameter name="serviceName" value="SLAMonitorService"/>
  <parameter name="namespace"
     value="http://tempuri.org/wsdl/2001/12/SOAPMonitorService-impl.wsdl"/>
  <parameter name="portName" value="Demo"/>
</handler>
```

Finally, add the handler to you service (in `server-config.wsdd`):

```
<service name="MyService" provider="java:RPC">
    <requestFlow>
       <handler type="slamonitor"/>
    </requestFlow>
    <responseFlow>
       <handler type="slamonitor"/>
    </responseFlow>
[...]
</service>
```

The handler is now fully operational. Violations of the SLA clause are reported in the file `SLAMonitor.log`.

Bibliography

[1] Apache axis. on-line at: http://ws.apache.org/axis/.

[2] Apache jUDDI. on-line at: http://ws.apache.org/juddi/.

[3] Apache tomcat. on-line at: http://tomcat.apache.org/.

[4] Apache xmlbeans. on-line at: http://xmlbeans.apache.org/.

[5] Java api for handling windows ini file format. on-line at: http://ini4j.sourceforge.net/.

[6] Stringtemplate template engine. on-line at: http://www.stringtemplate.org/.

[7] UDDI4J. on-line at: http://uddi4j.sourceforge.net/.

[8] Active Endpoints. Activebpel engine architecture. http://www.activebpel.org/docs/architecture.html, 2007.

[9] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1, May 2003.

[10] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *IET Software*, 2007. submitted for publication.

[11] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In *ICSOC 2005: Proceedings of the 3rd International Conference on Service Oriented Computing*, volume 3826 of *Lecture Notes in Computer Science*, pages 269–282. Springer, 2005.

[12] A. Bertolino, D. Bianculli, A. Carzaniga, G. De Angelis, I. Forgacs, L. Frantzen, Z. Gere, C. Ghezzi, A. Polini, F. Raimondi, A. Sabetta, and A. Wolf. Test Framework Specification and Architecture. Technical Report Deliverable D4.1, PLASTIC Consortium, March 2007. IST STREP Project.

[13] A. Bertolino, L. Frantzen, A. Polini, and J. Tretmans. Audition of Web Services for Testing Conformance to Open Specified Protocols. In R. Reussner, J. Stafford, and C. Szyperski, editors, *Architecting Systems with Trustworthy Components*, LNCS 3938, 2004.

[14] A. Bertolino and A. Polini. The Audition Framework for Testing Web Services Interoperability. In *Proc. 31st EUROMICRO Conf. on Sw Eng. and Advanced Applications (EUROMICRO–SEAA 2005)*, pages 134–142. IEEE Computer Society, 2005.

[15] D. Bianculli and C. Ghezzi. Monitoring conversational web services. In *Proceedings of the 2nd International Workshop on Service-Oriented Software Engineering (IW-SOSWE'07), co-located with ESEC/FSE 2007*, September 2007. to appear.

[16] F. Ciotti. WS-Guard —Enhancing UDDI Registries with on-line Testing Capabilities. Master's thesis, Department of Computer Science, University of Pisa, June, 8th 2007.

[17] W. Emmerich, F. Raimondi, J. Skene, V. Cortellessa, P. Inverardi, M. Tivoli, D. D. Ruscio, M.Autili, R. Mirandola, V. Grassi, A. Sabetta, J. Gonzales, P. Mazzoleni, and S. Tai. SLA language and analysis techniques for adaptable and resource-aware components. Technical Report Deliverable D2.1, PLASTIC Consortium, March 2007. IST STREP Project.

[18] S. Guinea. *Dynamo: a Framework for the Supervision of Web Service Compositions*. PhD thesis, Politecnico di Milano, 2007.

[19] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *ECOOP 2001 - Object-Oriented Programming, 15th European Conference, Proceedings*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer, 2001.

[20] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP'97 - Object-Oriented Programming, 11th European Conference, Proceedings*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer, 1997.

[21] G. T. Leavens, A. L. Baker, and C. Ruby. JML: A notation for detailed design. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Behavioral Specifications of Businesses and Systems*, pages 175–188. Kluwer Academic Publishers, Boston, 1999.

[22] F. Liotopoulos, S. Tai, J. Sairamesh, H. Eikerling, J. Gonzalez, J. Barra, M. Jazayeri, J. Wuttke, P. Inverardi, V. Cortellessa, A. Di Marco, and M. Autili. Scenarios, Requirements and initial Conceptual Model. Technical Report Deliverable D1.1, PLASTIC Consortium, June 2006. IST STREP Project.

[23] H. Ludwig. WS-Agreement Concepts and Use - Agreement-Based Service-Oriented Architectures. Technical Report RC23949, IBM, May 2006.

[24] J. Skene and W. Emmerich. Engineering runtime requirements: monitoring systems using MDA technologies. 2005.

[25] W3C. XML path language (XPATH). on-line at: http://www.w3.org/TR/xpath, 1999.

[26] W3C. Web service addressing (ws-addressing) specfication. on-line at: http://www.w3.org/Submission/ws-addressing/, 2004.

[27] W3C. Web Service Addressing 1.0 - SOAP Binding. on-line at: http://www.w3.org/TR/ws-addr-soap/, 2006. W3C Recommendation.