Private Public Partnership Project (PPP)
Large-scale Integrated Project (IP)

D.9.4.3: Application Testing and Deployment Support Tools

**Project acronym:** FI-WARE  
**Project full title:** Future Internet Core Platform  
**Contract No.:** 285248  
**Strategic Objective:** FI.ICT-2011.1.7 Technology foundation: Future Internet Core Platform  
**Project Document Number:** ICT-2011-FI-285248-WP9-D.9.4.3  
**Project Document Date:** 2014-01-31  
**Deliverable Type and Security:** Public  
**Author:** FI-WARE Consortium  
**Contributors:** FI-WARE Consortium
1 Executive Summary

This deliverable provides documentation for the third release of the software prototypes of the tools developed within Task 9.3: "Testing Automation and Deployment Support" of the Tools Chapter. The documentation includes general information, architectural description, download and installation guide, and usage guide (with screenshots) for each tool. The tools provide support to the end users, developing Future Internet Applications by using the FI-WARE platform, in the deployment, monitoring and testing activities. The tools can be used across different phases of the applications lifecycle, both at design time and run-time, and support both functional and non-functional testing (i.e. performance).
About This Document

The purpose of this document is to provide to the end users of the FI-WARE platform a set of tools that support the deployment and testing activities of the Future Internet application under development.

2.1 Intended Audience

This document and the described tools are mainly oriented to developers and system architects, and in general, to any developers who want to take benefits from a set of tools to deploy and test the Future Internet application under development or during their operation.

2.2 Chapter Context

The Tools chapter, in the context of FI-WARE project, is in charge of providing a set of tools and practices in order to address requirements and support the FI Application developers to manage the development of their Future Internet applications based on a FI-WARE Platform instance. More precisely, those Future Internet applications that will benefit from the services made available by the FI-WARE Platform (Generic Enablers instances).

To support the various activities, which compose the lifecycle of Future Internet applications, such set of tools provides support for collaboration, development, training, and for testing applications running on top of a Core Platform Instance.

2.3 Structure of this Document

The document is generated out of a set of documents provided in the public FI-WARE wiki. For the current version of the documents, please visit the public wiki at http://wiki.fi-ware.eu/

The following resources were used to generate this document:

D.9.4.3 Application Testing and Deployment Support Tools front page

**Tools.PROSA**

PROSA is an online testing and monitoring tool targeted to ensure the constant availability of the QoS monitoring data for a given service in a service composition.

**Tools.Trace Analyzer**

Trace Analyzer is an Eclipse plugin for graphical and numerical analysis of performance traces.

**Tools.SoPeCo**

Software Performance Cockpit is a framework for systematic performance evaluations of software systems, based on systematic measurements, statistical methods, and machine learning.

**Tools.ETICS**

ETICS is a tools for Continuous Integration and for managing the configuration (and complexity) of a Future Internet application.

**Tools.UFT Framework**

A framework that allows to test your REST services directly from the Eclipse IDE.

**Tools.NGSITestServer**

The NGSITestServer allows you to test your client software for use with IoT Generic Enablers that make use of the FI-WARE NGSI10 binding.
2.4 Typographical Conventions

Starting with October 2012 the FI-WARE project improved the quality and streamlined the submission process for deliverables, generated out of the public and private FI-WARE wiki. The project is currently working on the migration of as many deliverables as possible towards the new system.

This document is rendered with semi-automatic scripts out of a MediaWiki system operated by the FI-WARE consortium.

2.4.1 Links within this document

The links within this document point towards the wiki where the content was rendered from. You can browse these links in order to find the "current" status of the particular content.

Due to technical reasons not all pages that are part of this document can be linked document-local within the final document. For example, if an open specification references and "links" an API specification within the page text, you will find this link firstly pointing to the wiki, although the same content is usually integrated within the same submission as well.

2.4.2 Figures

Figures are mainly inserted within the wiki as the following one:

```
[[Image:....|size|alignment|Caption]]
```

Only if the wiki-page uses this format, the related caption is applied on the printed document. As currently this format is not used consistently within the wiki, please understand that the rendered pages have different caption layouts and different caption formats in general. Due to technical reasons the caption can't be numbered automatically.

2.4.3 Sample software code

Sample API-calls may be inserted like the following one.

```
http://[SERVER_URL]?filter=name:Simth*&index=20&limit=10
```

2.5 Acknowledgements

The current document has been elaborated using a number of collaborative tools, with the participation of Working Package Leaders and Architects as well as those partners in their teams they have decided to involve.

2.6 Keyword list

FI-WARE, PPP, FI-CoDE, Future Internet, PROSA, TraceAnalyzer, Software Performance Cockpit, test, performance, Quality of Service, QoS, System Activity Report, TCPdump, Eclipse, Deployment, Continuous Integration, ETICS

2.7 Changes History

<table>
<thead>
<tr>
<th>Release</th>
<th>Major changes description</th>
<th>Date</th>
<th>Editor</th>
</tr>
</thead>
</table>

D.9.4.3 Application Testing and Deployment Support Tools
2.8 Table of Contents

1 Executive Summary ........................................................................................................... 2

2 About This Document ....................................................................................................... 3

2.1 Intended Audience ......................................................................................................... 3

2.2 Chapter Context ............................................................................................................. 3

2.3 Structure of this Document .......................................................................................... 3

2.4 Typographical Conventions ......................................................................................... 4

2.5 Acknowledgements ....................................................................................................... 4

2.6 Keyword list .................................................................................................................. 4

2.7 Changes History ............................................................................................................ 4

2.8 Table of Contents ......................................................................................................... 5

3 PROSA ............................................................................................................................. 7

3.1 Introduction .................................................................................................................. 7

3.2 Architecture ................................................................................................................ 8

3.3 Installation .................................................................................................................... 8

3.4 User Guide ................................................................................................................... 10

4 Trace Analyzer ................................................................................................................ 27

4.1 Introduction .................................................................................................................. 27

4.2 Architecture ................................................................................................................ 27

4.3 Installation .................................................................................................................... 29

4.4 User Guide ................................................................................................................... 29

4.5 Examples ...................................................................................................................... 34

5 SoPeCo ............................................................................................................................ 40

5.1 Introduction .................................................................................................................. 40

5.2 Architecture ................................................................................................................ 40

5.3 Installation .................................................................................................................... 41

5.4 User Guide ................................................................................................................... 41

6 ETICS ............................................................................................................................... 49

6.1 Introduction .................................................................................................................. 49

6.2 Architecture ................................................................................................................ 49

6.3 Installation .................................................................................................................... 52

6.4 User Guide ................................................................................................................... 53

6.5 Use Case ....................................................................................................................... 54

6.6 Improvements .............................................................................................................. 56

D.9.4.3 Application Testing and Deployment Support Tools ............................................. 5
3 PROSA

3.1 Introduction

PROSA is an online testing framework targeted to ensure the constant availability of QoS monitoring data for a given service (used by a FI App). The runtime solution monitors the (real) usage of a given service and additionally allows performing additional service invocations (online tests) where necessary. PROSA thereby allows users to ensure that a minimum set of QoS data points is available for the given service for every time interval.

Specifically, PROSA lists the services used in a deployed service composition. Then, the user selects a process instance or a constituent service and views the available monitored response times for the instance or the service. The user can also configure PROSA to test the selected service (e.g., when there is sparse monitoring data). The test configuration includes: specifying an operation, test input, and test frequency. The results of the tests (i.e., response times) are visualized using the same graph that shows the monitoring data. PROSA enables the user to export the available QoS data for a particular service or instance to a csv file.


3.1.1 Information

Name: PROSA
Version: 3.2.0
License: EPL v1.0
Download: http://forge.fi-ware.eu/frs/?group_id=15#title_prosa
Scope: QoS Testing and Monitoring
3.2 Architecture

The above figure shows the architecture of PROSA. The key components of PROSA are the Client, the Online Testing Module and the Monitoring Module. The Client is an Eclipse plugin and is used interact with the tool such as connecting PROSA with service composition engines; SCE; e.g., Apache ODE and Activiti and selecting the process/service/task in interest. It is also used to specify the online testing configuration (e.g., service, input, and test frequency) for testing constituent services of the compositions (e.g., BPEL). The Monitoring Module interfaces with the composition engines, through the composition adapters, to collect Meta Data such as deployed compositions and Monitoring Events resulting from executing them. The current version of PROSA has adaptors for the following composition engines: Apache ODE and Activiti, and can be easily integrated with other adaptors. Monitoring data (e.g., QoS Data) obtained from the collected events are stored in the Monitoring Data repository. The PROSA Client visualizes the QoS Data. Based on the Test Configuration, the Online Testing Module invokes the constituent services of a composition for collecting additional QoS data when needed. The collected QoS Data is stored in the Monitoring Repository. The Client allows the user to export the available QoS Data for instances, services and tasks to a csv file.

3.3 Installation

3.3.1 SYSTEM REQUIREMENTS

The tool is an Eclipse plugin. The current version requires Eclipse 4.2+, JDK 7, Apache Tomcat (tested with version 7.0), Apache ODE (tested with version 1.3.5), and Activiti (tested with version 5.1.4) to be installed. The tool should be compiled using the JDK instead of the JRE. Thus, please make sure that the "$PATH" points to the "${JDK-LOCATION}/bin" and "$JAVA_HOME" points to the "${JDK-LOCATION}" (but note the slight difference of having "/bin" only for "$PATH").

D.9.4.3 Application Testing and Deployment Support Tools
3.3.2 FILES

From the PROSA binary releases, you need to download the following files that contain the binary distribution of the PROSA tool. The tool has two components: client side and service side.

- **Client side**
  - The jars of the plugin.

- **Server side**
  - **For Apache ODE**
    - adapter.ode-1.0-SNAPSHOT.jar
    - adapter.ode-1.0-SNAPSHOT-all.jar
    - ode-axis2.properties
  - **For Activiti**
    - prosa-activiti-adapter.jar
    - common-1.0-SNAPSHOT.jar
    - activiti-standalone-context.xml

3.3.3 ENVIRONMENT SETUP

**Setting up the PROSA Plugin:**
- Copy all the PROSA plugin jars into the “dropins” folder of your eclipse instance.

**Setting up Apache Tomcat:**
- Install the Apache Tomcat as explained in the related documentation.

**Setting up Apache ODE:**
- Install the Apache ODE Engine as explained in the related documentation.
- Copy the files “adapter.ode-1.0-SNAPSHOT-all.jar” and “adapter.ode-1.0-SNAPSHOT-all.jar” into “{TOMCAT_HOME}/webapps/ode/WEB-INF/lib”.
- Copy the file “ode-axis2.properties” into “{TOMCAT_HOME}/webapps/ode/WEB-INF/conf”.

**Setting up Activiti:**
- Install the Activiti engine as explained in the related documentation.
- Copy the file "common-1.0-SNAPSHOT.jar" into "{TOMCAT_HOME}/webapps/activiti-explorer/WEB-INF/lib".
- Replace the file "activiti-standalone-context.xml" in "{TOMCAT_HOME}/webapps/activiti-explorer/WEB-INF/" with the delivered version.
- Copy the file "prosa-activiti-adapter.jar" into "{TOMCAT_HOME}/webapps/activiti-explorer/WEB-INF/lib".
3.4 User Guide

3.4.1 PROSA IDE
The PROSA IDE is documented in deliverable 9.3.3. There we provided detailed explanation of the GUI of the PROSA tool. Please refer to this document for further details.

3.4.2 Starting PROSA
Start the Tomcat application server as explained in the related documentation. Then, open your Eclipse instance. If everything works, you should see a menu for the PROSA tool as shown below.

![PROSA Menu](image)

**PROSA Menu**
The menu has two options: (1) "Start PROSA" (for launching PROSA) and (2) "Export QoS Data" (for exporting the available QoS data). You can also open the PROSA perspective by going to "Window"--"Open Perspective"--"Other", and then choosing "Prosa Perspective".
Selecting PROSA Perspective
PROSA Perspective Welcome Page

To launch PROSA, select the “Start” option from the PROSA menu. Once launched, the following view appears.
Starting PROSA

3.4.3 Connecting PROSA to Apache ODE and Activiti

Click on the "Update" button in order to connect to the Apache ODE and Activiti composition engines. Once clicked, PROSA tries to connect to the Apache ODE and Activiti composition engines. If the connections are successfully established, PROSA will list the deployed processes on those engines as well as their constituent services using a tree structure (see the following figure).
Successful Connection to the Engines

Otherwise, the name of the composition engine will be marked in red colour with exclamation mark at the end (see the following figure).
3.4.4 Viewing Historical QoS Data

PROSA stores the collected QoS data for the instances and services/tasks permanently. Thus, PROSA allows viewing the historical QoS Data at different levels. The first level is at the process instance level. For any deployed process, you can use the PROSA tool to see the past response times of all the executions of those process (i.e., instances). To perform this, you should click on the process from the tree structure as shown in the following figures.
The QoS History of Process Instances Deployed in the Activiti Engine (in this case, no instances are available)
The QoS History of Process Instances Deployed in the Activiti Engine

The second level is at the constituent services/tasks used in a process deployed in an engine. For any particular service/task available for a process, you can view all its historical QoS data as invoked in all the instances of the process. Additionally, you can view the historical QoS data available from testing the service using the PROSA tool, as we will see later. However, for this release, testing is supported for only the services used in the processes deployed in Apache ODE. In order to see past historical data collected from previous usage (monitoring) of a particular operation of a service, click on the service:operation in the list on the left hand side as shown in the following figures.
The QoS History of a Task Deployed in the Activiti Engine (in this case, no QoS data is available)
The QoS History of a Task Deployed in the Activiti Engine

The last level is at the constituent services/tasks of a particular instance. For any particular instance available for a process, you can view the historical QoS data of all the services/tasks that were invoked in this instance. To perform this, you should click on invocation point (in the instance graph) and the QoS of the services/tasks will be displayed in a below graph, as shown in the following figure.
The QoS History of a Task Invoked in Instance of a Process Deployed in the Activiti Engine

3.4.5 QoS Testing of Services

PROSA allows testing services in a deployed composition. The current version supports only testing of services using in a composition deployed in Apache ODE. You can test a service through the "Test Web Service" view. Once the "Test Web Service" view is selected, you will see a form for specifying the test configuration labelled with "Configure WS for Testing". The test configuration includes: selecting input "Parameters" and providing "New Value" to each parameter. PROSA allows several values to be added for a particular input parameter and uses random selection during the test execution (i.e., random testing). The last configuration is the "Test Frequency" (i.e., "Number of Executions" and "Time Interval (ms)"). The steps are shown in the following figures.
PROSA Test Web Service View
Test Configuration

To execute the tests according to the specified configuration, click on the "Start Testing" button. The results of the tests (i.e., response times) are then displayed in a plot (see the below figures). The test results are also stored and will be displayed when choosing to view the historical data of the service.
Test Execution
3.4.6 Exporting QoS Data

PROSA allows to export the available QoS data for a particular instance, task, or operation of a service to a csv file. To perform this, select the "Export QoS Data" option from the PROSA menu. Once the "Export QoS Data" option is selected, a new view will automatically open.
The view includes a menu which lists all the available processes, date and time intervals for specifying the date and time of the "First Execution" and "Last Execution", a menu for listing the available operations used in the selected composition, a menu for selecting the source of the QoS data (i.e., Monitoring, Testing, or Monitoring and Testing together), and finally a field for providing the full path of the file where the results should be saved (see the following figure).

View for Exporting QoS Data to a CSV File

Once the fields are filled (see the figure below), the QoS data are shown on the right hand side such that the user can check his selections before exporting the QoS data. If everything is fine, the user can save the results by clicking on the "Export" button, and the QoS data will be stored in a csv file accordingly.
Configuring the Export of QoS Data

The following information are exported: the process instance ID, the execution values (input used during invocation), timestamp (of invocation), response time, invocation source (monitoring or testing), and the result of the invocation. The following figure shows the result of opening, with MS Excel, an example csv file exported with the PROSA tool. The csv file can be post analysed using tools like the statistical package R.

Result of Exporting QoS Data to a CSV File
4 Trace Analyzer

You can find the content of this chapter as well in the wiki of FI-WARE.

4.1 Introduction

Trace Analyzer is a set of Eclipse plug-ins for graphical and numerical analysis of performance traces.

Trace Analyzer accepts input traces from a plethora of tools/sources, and displays time-based graphical visualizations of the program execution as well as a list of trace contents and the event details for selection.

On Linux, supported data sources include:

1. tcpdump (or equivalent - format may need 'massaging') tool for network information
2. SAR (System Activity Report) tool for operating system information
3. pthread_mon - an IBM proprietary tool for monitoring the behavior of threads

In addition, Trace Analyzer provides a remote invocation plug-in that can run the above tools on a remote machine, bringing the output to the user's machine.

4.1.1 Information

Name: Trace Analyzer
Version: 2.2
License: IBM License File (login required)
Documentation: Press F1 in Eclipse and navigate to Trace Analyzer
Download: Two methods are available:

- https://forge.fi-ware.eu/frs/?group_id=23#tools-traceanalyzer-trace-analyzer-2-2 (login required)
- Send an email to Marcel Zalmanovici (marcel [at] il.ibm.com) for a link to the Eclipse plugins.

4.2 Architecture

Trace Analyzer plug-in has two major modules:

- The **Trace Modeler** that reads the trace, generates internal model of the trace and provide data on demand.
- The **Visualizer**, the UI of Trace Analyzer that visualizes the data in the trace model, according to the selected view and/or statistical query.

Each of those modules is further divided into a sub-module per each of the supported input, network, operating system and threading.

4.2.1 Trace Modeler

- buildTraceModel(trace)
- getViewData(view_identifier)
- getRecordDetails(record_id)

D.9.4.3 Application Testing and Deployment Support Tools
4.2.2 Trace Analyzer

- `getStatistic(parameters)`

4.2.3 System Monitoring

In order to collect monitoring data, Trace Analyzer uses the tcpdump and sar tools in the following way:

- `tcpdumpReq(interface, portNum)` is implemented by
  ```
  tcpdump -i <interface> port <portNum>
  ```

- `sarReq(intervals)` is implemented by
  ```
  sar -A <intervalLen> <intervalNum>
  ```

- `pthreadReq(application)` is implemented by
  ```
  ```

The following sequence diagram describes one of Trace Analyzer usage scenario.
4.3 Installation

4.3.1 SYSTEM REQUIREMENTS
Trace Analyzer is a bundle of Eclipse plug-ins. It requires:
- JDK 5.0 or newer
- Eclipse 4.2 or newer
- Eclipse CDT
- Eclipse PTP

4.3.2 INSTALL
Download the ZIP package from the above link. Unzip its contents to the dropins folder for testing. Once everything works you can copy the contents of the plugin and features folders to their counterparts in the Eclipse installation folder.

4.4 User Guide
The full user guide is available as part of Eclipse help pages by pressing the F1 key

4.4.1 Creating inputs
There are two ways to generate input files from Trace Analyzer:
- Run the SAR, TCPdump, pthread_mon tools on the remote machine, SSH their output to the local machine, accordingly rename the outputs (SAR -> .sar, TCPDump -> .tcpdump, pthread_mon -> pthread) and open them from File --> Open in Eclipse (alternatively, add the files to an existing project and then double click them).
- Make use of the remote (or local) execution feature of Trace Analyzer as explained below.

4.4.1.1 Collecting the trace using the Eclipse "Profile As" dialog
4.4.1.1.1 Profiling a local project

1. Right-click the project in the Navigator view and choose "Profile as" and then "Profile Configurations" to open the dialog:
2. In the Profile Configurations dialog, select "Trace Analyzer for xxx (your trace extension)" from the list on the left and click the "New" button to create a profile:
3. Set up the profiling configuration on the extension tab (more details about this tab can be found in data source directory according to the extension).

4. When ready, click "Profile". The program will execute (its output will be shown in the Console view), and the collected trace will open with Trace Analyzer.

4.4.1.1.2 Profiling a remote project

Before profiling a project, ensure that it has been built. In addition, ensure that the extension monitoring tool is installed on the remote server.

1. Switch to the parallel runtime perspective by clicking Window > Open Perspective > Other > Parallel Runtime. Click OK.

2. In the open space in the Resource Managers tab, right-click and then click Add Resource Manager. Click Next.

3. In the Choose Resource Manager Type window, ensure that Remote Launch is selected. Click Next.

4. In the Connection configuration window, in the Remote service provider field, click the arrows and select Remote Tools.

5. In the Connection name field, click the arrows to select a connection to the remote server, or click New to create a connection. If creating a connection, use your ssh credentials instead of public key.
6. Click Next or Finish. The resource manager you just created should now appear on the Resource Managers tab.

7. On the Resource Managers tab, right-click the resource manager. Click Start Resource. The resource manager name should now be green, and the Machine tab should now have a server displayed.

8. Switch to the remote C/C++ perspective by clicking Window > Open Perspective > Other > Remote C/C++. Click OK. Existing remote projects are displayed under Project Explorer view. To create a new remote project right-click Project Explorer, then click New > Remote C/C++ Project and follow the wizard.

9. Right-click the remote project in the Navigator view and choose "Profile as" and then "Profile Configurations" to open the dialog:

![Profile As dialog](image)

10. In the Profile Configurations dialog, select "Trace Analyzer for xxx (your trace extension) Remote" from the list on the left and click the "New" button to create a profile:
11. Customize the new configuration:

- On the Resources tab, choose the resource manager where the project was previously built from the drop down list.

- On the Application tab, select the application program by clicking Browse to locate the binary file on the remote server file system.

- On the extension tab – please read more details about this tab in the data source directory according to the extension.

- When ready, click "Profile". The program will execute (its output will be shown in the Console view), and the collected trace will open with Trace Analyzer.

4.4.2 Navigating Main View

Notice that new buttons appeared on the toolbar:

These buttons allow to navigate through the graphical view.

- The zoom-in button allows to zoom into a region of the trace by clicking on it.
This tool zooms in by a fixed amount. To fit a specific region into the view use the Zoom On Area tool.

Once you zoom in, the scrollbar at the top of the graphical view becomes active, allowing to scroll the view horizontally.

Alternatively, you can scroll using to drag the view. Use to zoom out, and to fit the entire trace to the width of the view.

The leftmost button is the selection tool. Use it to select a record in the graphical view. The record is colored in orange in the graphical view, indicating that it is selected.

Please note that the record is also selected in the Trace Table View, and that the Record View and Properties View display the contents of the record.

4.5 Examples

4.5.1 SAR

This view shows the CPU utilization data: time spent in user- and system-level code, idle time, idle time waiting for I/O, time executing with nice priority, involuntary wait time due to hypervisor servicing another request. The view shows a row per CPU; the top row shows the average for all CPUs.
Run queue statistics

This view shows the run queue statistics. The top row shows one-, five, and fifteen-minute load averages. The second row shows the run queue length, and the third row shows number of processes and threads in the process list.

Context switches

D.9.4.3 Application Testing and Deployment Support Tools
This view shows the amount of context switches performed by the application in a 'time period' (defined when running the tool). The x axis shows the time and the y axis shows the amount of context switches. Combined with the information about throughput displayed in the top row, the user can decide whether the behavior is correct or not. At a first glance, the information displayed in the screen-shot looks like a problem. However, if the application has no data to work on at that time, then this is the expected behavior. Thus the user should compare this with the I/O view data, for example.

4.5.2 TCPDump

This view shows the amount of data being passed in time; the x coordinate is time, and the y coordinate the amount of data. The amount is positive if data is being sent and negative if data is being received. Each series in the graph corresponds to a single session, identified by the local and remote ports and the remote host name.
Sessions example explained

The above image shows a zoom-in into the sessions view. The $x$ coordinate shows the time in usec and the $y$ coordinates shows the number of sessions running concurrently. The color of the sessions depends on its length relative to the other sessions; blue represents a short session and red a long one.

Record

This shows information about the selected record/event.
4.5.3 PThread Monitor

The above screenshot shows the pthread_mon tool in 'action'. The invocation may be done from the command line, like above, or through Trace Analyzer.

The above screenshot shows the thread holding each mutex at any moment as well as the length of the waiting queue on the resource and the thread waiting the longest. This can be used to easily identify threads hogging a resource - threads that hold the mutex for too long and do not allow other threads to perform their job. Fairness problems can also be located easily from this view by looking at the wait queue - if the length of the queue changes and the longest waiting thread remains the same this may indicate a problem (even if priorities are associated with the threads).
Waits by thread (showing a potential fairness problem)

The above screenshot is an example of how waits per thread are displayed. This is a very good way to analyze when threads are working and when they are waiting on a resource. Fairness problems can be readily detected here by focusing on the longest rectangles. Interaction between different threads can be analyzed here as well. In the above picture, thread 4998 may be waiting an unreasonable amount of time. Also, thread 4969 never attempt to lock any mutex which may indicate a problem.

A potential thread starvation

Another example of a problem that readily be detected by the thread monitoring plug-in is displayed in the screenshot above. The image shows that the same thread is waiting to lock the mutex although many other threads get priority over it.
5 SoPeCo

You can find the content of this chapter as well in the wiki of FI-WARE.

5.1 Introduction

The Software Performance Cockpit (SoPeCo) is a framework for systematic performance evaluations of software systems, based on systematic measurements, statistical methods, and machine learning. It enables developers to do systematic goal-oriented measurement scenario during early stages of the creation of their artifacts and thereby reduces costs and increases time to market. Being improved and adapted for FI-WARE, SoPeCo plays an essential role in FI-WARE by enabling the performance-aware development of GEs and FI Applications including performance-based comparison and selection of GE implementations.

5.1.1 Information

Name: Software Performance Cockpit
Version: 5.0.2
License: BSD
Website: http://www.sopeco.org/
Catalogue: http://catalogue.fi-ware.eu/enablers/sopeco
Download: http://forge.fi-ware.eu/FRS/?group_id=15#title_sopeco
Scope: Measurement-based Performance Testing and Analysis

5.2 Architecture

A SoPeCo setup consists of the following main components shown in the figure below.

SoPeCo components (high level)
At the bottom, we have the service-or system under test (SUT), which represents the artifact a developer wants to analyze. From SoPeCo's point of view, this artifact is considered to be a black box. That might run on the developer's local machine or some remotely accessible location.

In order to observe and interact with the SUT, the developer has to implement a SoPeCo Measurement Controller (MEController). This MEController acts as an adapter connecting the SUT with the SoPeCo Engine. It can act as a workload driver or coordinate an external workload driver. Also the measurements and observations are done by the MEController itself or coordinated in case of using external tools (like SAR etc.). It is important to note, that a MEController only has to be implemented once, so there is a good chance that existing ones can be reused or at least adopted for new types of SUTs. Examples of existing MEControllers are the one for the Repository GE or external ones like SoPeCoYCSB by Tom Zhang which implements an MEController for the Yahoo Cloud Serving Benchmark.

On top of the MEController, the SoPeCo Engine is responsible for coordinating actual experiment run and to analyze the results. It steers the MEController through the defined series of experiments using the defined exploration strategies for varying the test parameters and dimensions. As such, it acts as the core of the SoPeCo framework.

In order to increase ease of use, the SoPeCo Engine is accompanied by the SoPeCo Web UI, which provides a browser based interaction with the SoPeCo Engine for defining experiments, scheduling their execution or visualizing the results.

5.3 Installation

5.3.1 Setup Eclipse environment

In order to implement the MEController, you need a Java development environment like Eclipse. So please install a regular Eclipse IDE package and download the SoPeCo core jar file from the tools section of the FI-WARE forge at http://forge.fi-ware.eu/frs/?group_id=15#title_sopeco

5.3.2 Setup SoPeCo instance

SoPeCo is offering a publicly available instance at http://app.sopeco.org/ where it can be directly used. It consists of the SoPeCo Web UI and the SoPeCo Engine and provides an ready to go environment for running experiments.

If you however decide to setup your own SoPeCo environment, then you can learn more at the SoPeCo website at http://www.sopeco.org/tutorials/host-your-own-sopeco-instance

5.4 User Guide

In the following we will describe the application of SoPeCo to the Repository GE provided by FI-WARE. We will cover the implementation of a simplistic MEController, the definition and execution of an experiment as well as the analysis of the resulting data.

5.4.1 Write the MEController

In order to implement the MEController, you have to first setup a corresponding Eclipse project:

1. Create the project via File -> New -> Project ... -> Java Project
2. Create the lib folder by selecting the created project and clicking New -> Folder
3. Copy the SoPeCo core jar file from http://forge.fi-ware.eu/frs/?group_id=15#title_sopeco and place it in the lib folder.

4. Select Build Path -> Configure Build Path ... in the context menu, click the Libraries tab, press the Add JARs... button and select the SoPeCo core jar file.

Now we can start the actual development by creating a new class for our MEController.

```java
import org.sopeco.engine.measurementenvironment.AbstractMEController;
import org.sopeco.engine.measurementenvironment.InputParameter;
import org.sopeco.engine.measurementenvironment.ObservationParameter;
import org.sopeco.engine.measurementenvironment.app.MECApplication;
import org.sopeco.persistence.dataset.ParameterValueList;
import org.sopeco.persistence.entities.exceptions.ExperimentFailedException;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;

public class FiwareRepositoryController extends AbstractMEController {
    @InputParameter(namespace="input")
    String acceptHeader = "application/xml";

    @InputParameter(namespace="input")
    String collection = "testCollection";

    @InputParameter(namespace="input")
    String repositoryUrl = "http://10.55.149.20:8080/FiwareRepository/v1";

    String strategy = "iterate";

    @ObservationParameter(namespace="output")
    ParameterValueList<Double> responseTime;

    private Client c;

    @Override
    protected void defineResultSet() {
        addParameterObservationsToResult(responseTime);
    }

    @Override
    protected void finalizeExperimentSeries() {
        // TODO Auto-generated method stub
    }
}
```
The code above provides the implementation of a basic MEController for testing the Repository GE provided by FI-WARE. It implements three input parameters:

- acceptHeader = The output format to be requested from the repository
- collection = The ID of the collection to be addressed in the repository
- repositoryUrl = The URL of the repository

In addition one observation parameter (aka output) is defined for collecting response times. Next, the code implements the processing logic for the experiment in the runExperiment() method and finally adds the startup code in the main() method.
5.4.2 Create the Experiment Definition

The experiment definition describes what should be done. It describes the input and observation parameters, the input value ranges, parameter variation strategies and the number of repetitions for the experiments.

First access the public instance of the Web UI at http://app.sopeco.org/ and create an account.

Next step is to add a scenario in which the experiment definition will be stored.
In the following screen, the connection with the MEController is done. So first you have to get back to Eclipse and start the MEController. Then please enter the hostname or IP address of the machine where the MEController is running. Press the Refresh button and check that the status changed to Controller is online. Move on pressing the Add scenario button.

Next we are in the screen of the experiment specification. Select ContentHeader Negotiation Test (on the left side below Specification), the experiment series that was just created.
Experiment definition - pick experiment series

Keep *Full Exploration Strategy* as exploration strategy, select the *Number of Repetitions* checkbox and enter 50 as the number of repetitions.

Experiment definition - experiment series configuration

Move over to the *Parameter Assignments* tab. Check the second column checkboxes to pick the input parameters. Then choose the variation types and values as shown in the screen shot. In this case example we only vary the acceptHeader variable in order to test the different output formats the repository GE provides.

Experiment definition - parameter assignment

5.4.3 Run the Experiment

For executing the experiment series we have defined in the step before, you have to click on the *Execute* topic on the left hand side of the screen. Pressing the *Execute Experiment* button will start the process.
Experiment execution - start

The experiment series will be run through all the specified experiment variations. During the run, the status bar provides an overview on how many test cases have already been processed. The console provides additional information on the status. Once all experiments are done, the console will state *Measurement finished*.

Experiment execution - finish

5.4.4 Download results

One the experiment series has been successfully executed, the results can be downloaded. For this you click on *Result* topic on the left hand side of the screen. Then you pick the relevant experiment series and time of the run and press one of the icons trailing it. The available options are CSV file download, R code and charts.

Result download

Given the charts option does not yet support charting of results where one of the input values is a text (acceptedHeader in this case), we can't chart the results of this specific test within SoPeCo. So instead we download the data as CSV file by pressing the download button. As a result, a dialog is shown which allows you to specify the different CSV file options. Once you press the *Export* button, the download is triggered.
Result format specification

This CSV file then can be opened in Excel or post processed in tools like gnuplot etc.

Result in Excel
6 ETICS
You can find the content of this chapter as well in the wiki of FI-WARE.

6.1 Introduction

ETICS stands for "eInfrastructure for Testing, Integration and Configuration of Software". It provides a service to help software developers, managers and users to better manage complexity and improve the quality of their software. The service allows to fully automate the way a software is built and tested. It provides "out-of-the-box" build and test system, powered with a product repository.

ETICS is a solution for managing the complexity of software development with the aim of improving the quality of the software produced. ETICS can execute its activities (build and test) not only on users' computers but also in a remote and multi-platform system.

The main aspects to highlight are:

- to describe the components of the software and its dependencies in order to properly run operations such as compile, build, package;
- to store the configurations on a versioning system so that it's possible to maintain the history of changes;
- to create a repository of components/libraries in order to build different configuration conditions;
- to take advantage from running in a distribute infrastructure (compute and storage);
- to store the results of test execution.

ETICS is the result of the homonymous FP7 project.

6.1.1 Information

Name: ETICS
Version: 3.6.17 (SaaS version)
License: Apache 2
Documentation: http://etics.eng.it/doc
SaaS: http://etics.eng.it
Source Code: http://cloud.eng.it/svn/main/etics/
Scope: Compile, Build, Deploy, Test (Continuous Integration)

6.2 Architecture

ETICS architecture is based on Master-Agent communication paradigm.

ETICS Master is a machine containing all the configurations for different hosts. It will act as a node controller for all its agents.

ETICS Agent is the daemon that will run on all the hosts that are (to be) managed by ETICS. Every agent is called "worker node".

When a build and test or a Continuous Integration plan is started, the ETICS Agent receives a test-plan job from the ETICS Master server.

The communication between ETICS Agents and Masters uses a HTCondor protocol and follows these main steps:

1. ETICS User creates a test-plan by using the ETICS Portal (which is exposed by the ETICS Master node).
2. The ETICS Master creates a job related to that test-plan.
3. The ETICS Master reaches all its agents and decides to assign the job to one of the agents that are “free”.
4. The ETICS Agent gathers the parameters from the job, including the test-plan that has to be executed.
5. After the execution of the test-plan, the Agent creates a report and stores it in a specific repository.
6. The Agent refreshes its status as “free” again to the Master.

The pool of ETICS worker nodes is defined by the ETICS Master node (Web Server), basing on the available hosts at network level. If the Master node is able to create a socket (communication channel) towards a given host, on a specific list of ports, than the Master deduces that an ETICS Agent is installed on that host. As a result, that specific host acquires the right grants in order to belong to the pool of machines that can be used by the Master node.

The schema shows that the infrastructure used by ETICS could be composed by physical nodes or managed by a Cloud Manager (as in this specific case).

ETICS Architecture is composed by a Web Server, which includes the ETICS Portal Component, the Web Interface, used to configure the ETICS test-plan, and its Worker Nodes.
Update web server table
ETICS sequentially performs specific actions that involve all the components of the architecture.

ETICS Process
The main step is the communication between the ETICS Master (web server) and its worker nodes. The Web Server is responsible of the configuration of the test-planning, which is defined through the ETICS Portal component included in the Master. The worker nodes are the performers of the test-planning which has been previously configured.
6.3 Installation

The installation steps reported in this chapter are taken from the official guide at: https://twiki.cern.ch/twiki/bin/view/ETICS/SA1InstallationStepByStep#Service_Installation_Step_by_Step.

6.3.1 Base platform

The ETICS Services deployment has been proven to work on SLC4 32-bits platform. Other RH-based distributions might be suitable as well, however they were not fully tested.

6.3.2 MySQL database

It is assumed that no MySQL database operational instance is running at the target server. The MySQL server is meant to be installed during the described here installation process. In case the MySQL server is already operational a respective user and passwords need to be provided to guarantee correct database access.

In order to ensure the correct operation of MySQL server, a proper security policy has to be applied to the Operating System.

6.3.3 Security certificates (etics server)

The host security certificates needs to be present on target etics server under: /etc/grid-security. The certificates should be accessible with passwords.

- hostcert.pem
- hostkey.pem
- hostcert.pkcs12 (same as above in pkcs12 format)
- hostcert.chain (moving to certchain.pem) - Certification Authority Chain

6.3.4 Conflicting packages

The installation will not succeed if the following packages are already installed in the system:

1. jpackage-utils - conflicting with tomcat5
2. mysql - conflicting with MySQL-server v5.x
3. in case there is a jdk vs. java conflict one might need to disable the jpackage repository (enabled=0 inside jpackage.repo file)

6.3.5 Basic Installation Procedure

- fetch the repository for etics software


or for pre-production


- install the ETICS server or repository together with its dependencies
yum install etics-deployment-server / yum install etics-deployment-repository

- Edit parameters in the section 'common' and 'server' or 'repository' depending on which service you are deploying. Configuration files: etics.server.conf and etics.server_defaults.conf are located under: /opt/etics/deployment/. Parameters in the 'common' section must be defined. For the server deployment one must also specify repository server DN as extracted from repository server certificate.
- You may also want to change the passwords for different services like: database, keystore.
- from within the /opt/etics/deployment directory run the configuration script which will configure various services and register them in the system
  
```
  cd /opt/etics/deployment
  sh configure.sh server | repository
```

6.3.6 Verifying the service presence

At this point you should be able to verify the success of you installation by opening the ETICS Server or repository page at address:

- ETICS Repository Server: https://hostname:8443/repository/download/

6.4 User Guide

6.4.1 Continuous Integration with ETICS

The ETICS Platform allows to configure a Build and Test plan and, afterwards, execute it repeatedly, so many times as needed. This is because ETICS stores the parameters and configuration set-up the first time (by the ETICS user) in order to perform a given test-plan (for an application and/or a source code). Thus, in order to implement a Continuous Integration process, a FI-App developer should define only once the needed criteria in ETICS to perform the test-planning.

The schema below depicts the execution of a given test-plan within a Continuous Integration process in ETICS.

![Test-plan within a CI process with ETICS](image)
6.5 Use Case

6.5.1 Context
Implementation of a Continuous Integration process by using ETICS tool.

6.5.2 Scenario
The process of Continuous Integration for a FI-Application on a FI-WARE infrastructure is automatically managed by ETICS tool. The FI-App developer has to configure only once (the first time) the test-planning and the (possible) dependencies to test the FI-Application (Use Case Diagram #1). ETICS stores the configuration of this Continuous Integration process, so that the FI-App developer can later on execute the same test-planning (each time is needed) on the following versions of the same FI-Application that is being developed. After having executed the test-planning configured for the Continuous Integration process, ETICS creates and return a results report (Use Case Diagram #2).

6.5.3 ETICS configuration of a Continuous Integration process on a FI-WARE infrastructure

6.5.3.1 Description
To implement a Continuous Integration process a FI-App developer configures a test-planning on an ETICS Master node by using the ETICS Platform. This test-plan has to be configured only once (the first time) and has to define and include all needed parameters, all test scripts to be executed and all (possible) dependencies on the worker-nodes needed for testing that FI-Application. After that, the developer starts the defined Continuous Integration process and ETICS executes the test-planning as previously configured.

6.5.3.2 Use Case Diagram #1

<table>
<thead>
<tr>
<th>Use Case</th>
<th>ETICS configuration for a Continuous Integration process on a FI-WARE infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>FI-App developer</td>
</tr>
</tbody>
</table>
| Actions      | • Configure the dependencies among worker nodes needed for testing the FI-Application  
              | • Configure test scripts for each node                                              
              | • Start the Continuous Integration process                                          |
| Subject      | ETICS                                                                               |
| Precondition | • Define the worker nodes involved in the test-planning                             
              | • The FI-App developer is registered to ETICS                                       
              | • The FI-App developer can access the process/module created for testing that FI-Application |
### 6.5.4 ETICS result report of a Continuous Integration process

#### 6.5.4.1 Description

When the Continuous Integration process is stated by the FI-App developer, ETICS creates a job to execute the defined test-planning related (configured) to that Continuous Integration process. After the execution of the test-planning, ETICS returns the result report.

#### 6.5.4.2 Use Case Diagram #2

<table>
<thead>
<tr>
<th>Use Case</th>
<th>ETICS result report of a Continuous Integration process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>ETICS</td>
</tr>
<tr>
<td>Actions</td>
<td>• Create the job to execute the test-planning configured for the Continuous Integration process</td>
</tr>
<tr>
<td></td>
<td>• Execute the (configured) Continuous Integration process</td>
</tr>
<tr>
<td></td>
<td>• Generate the result report</td>
</tr>
<tr>
<td>Subject</td>
<td>FI-WARE</td>
</tr>
<tr>
<td>Precondition</td>
<td>ETICS is already configured for executing the CI process</td>
</tr>
<tr>
<td>Post condition</td>
<td>Report listing the results of the execution of the CI process</td>
</tr>
</tbody>
</table>
6.6 Improvements

There are some features in ETICS that could be improved, in particular the Graphical User Interface (which is not user friendly) and the multi-node testing section.
7 UFT Framework

You can find the content of this chapter as well in the wiki of FI-WARE.

7.1 Introduction

The Unit Functional Testing Framework (aka UFT Framework) is a framework to test REST services. These tests can be executed directly from the Eclipse IDE or as part of a continuous integration process. The framework is based on two main components:

- REST-Assured
- TestNG

REST-Assured is used to execute REST calls (e.g. GET, POST, PUT, DELETE) toward the service under test. TestNG is used for managing the execution of the tests, group them, and create test reports. Binaries and source code are available from the official download site reported below.

For the list of known issues or to provide feedback on this plug-in, use the on-line Bug tracker and select "UFT Framework" as Asset option.

7.1.1 Information

Name: Unit Functional Testing Framework
Version: 1.0.0
License: EPL v1.0
Download: http://forge.fi-ware.eu/frs/?group_id=15#title_uft-framework (Binaries and Source Code)
Scope: Framework to test REST services

7.2 Architecture

This chapter describes how the code of the UFT Framework is organised and for which purpose.

The framework is structured as a maven project and it is divided into four main sections:

- **API**
  In the API section contains all the core classes which call the REST services to test.

- **Unit Test**
  In the Unit Test section contains all the classes which test the REST services individually.

- **Scenario**
  In the Scenario section contains all the classes which test a sequence of REST service calls, so that outputs of a service can be used as inputs for the next services.

- **Utility**
  In the Utility section contains all the utility classes which can be used in the other sections.

7.2.1 The API section

The API section contains the classes to call the REST services to test. Every class is dedicated to a single (REST) resource to call.
The API section

These classes have to satisfy two constraints, they have to extend the AbstractApi class and implement the execute method. Into the execute method it's possible to define the calls to the REST service. The previous figure represents a POST call to the notifyContext service exposed by the Context Broker GE.

7.2.2 The Unit Test section

The Unit Test section contains classes to test the services. The method of the Unit Test class that implements the test is annotated as Test and the same annotation also specifies the data source (dataProvider) to use as test data. It is proposed to follow the naming convention to use the "Test" postfix for class names; the Unit Test class for the notifyContext service is named NotifyContextTest, and therefore the execution of NotifyContextTest calls the execute method of the NotifyContext class.
The Unit Test section

Anyway, before to call the execute method you can add some parameters. Parameters are represented by a hashmap in which you can store key value pairs. These parameters can be added; both manually, via code; or through a XLS file using the "DataProvider" feature of TestNG. The "DataProvider" reads parameters from XLS file and associates them to the map by key and value.

7.2.3 The Scenario section

The Scenario section contains all classes to test a scenario. The method of the Scenario Test class that implements the test is annotated as Test and the same annotation also specifies the data source (dataProvider) to use as test data. Every scenario class calls, sequentially, a number of APIs.
The Scenario section

As for the Unit Test section, parameters are represented by a hashmap in which it's possible to store key-value pairs. For example, the parameters can be retrieved from XLS files using the appropriate "DataProvider", or they can be extracted directly from previous responses and inserted in the parameter hashmap in order to make them available to the next calls. In the example reported in the previous figure, the first step retrieves the query context using the \texttt{queryContext} call to get an attribute, then the second step updates this attribute.

7.3 Installation

The UFT Framework is available as .ZIP archive from the FI-WARE forge web site at link \url{http://forge.fi-ware.eu/frs/?group_id=15#title_uft-framework}. Before starting to use the framework in Eclipse IDE, some dependencies have to be resolved:

- TestNG plug-in.
- Maven Integration for Eclipse (m2e);

7.3.1 Install TestNG plug-in

To install the plug-in from the update site, open the Eclipse IDE, select the menu item: "Help" - "Install New Software...".
Press the "Add" button and type the "name" and "location" for TestNG plug-in which is http://beust.com/eclipse. Press the "OK" button. When the TestNG version is shown, select it and press the "Next" button. Restart the Eclipse IDE.

7.3.2 Install Maven Integration for Eclipse plug-in
To install the plug-in from the update site, open the Eclipse IDE, select the menu item: "Help" - "Install New Software...".
**Maven Integration for Eclipse**

Press the "Add" button and type the "name" and "location" for m2e plug-in which is [http://download.eclipse.org/technology/m2e/releases](http://download.eclipse.org/technology/m2e/releases). Press the "OK" button. When the m2e version is shown, select it and press the "Next" button. Restart the Eclipse IDE.

**7.3.3 Install plug-ins from Eclipse Marketplace**

It's also possible to install the plug-ins directly from the Eclipse Marketplace instead of using the update sites. From the Eclipse IDE select the menu item: "Help" - "Eclipse Marketplace", type the name of the plug-in in the "find" field and press the "Go" button. The Eclipse IDE retrieves all plug-ins that satisfy your request.
Eclipse Marketplace

Select the right one from the list and press the "Install" button. Restart the eclipse IDE.

7.3.4 Install UFT Framework

The UFT Framework is available as .ZIP archive from the FI-WARE forge web site. Download and unzip the package. Import the unzipped content as a project into the Eclipse IDE using the menu item: "File" - "Import", then select "Existing Projects into Workspace" and follow the wizard.
7.4 User Guide

7.4.1 Main Features

7.4.1.1 Annotation
The UFT Framework makes use of an important feature provided by TestNG (the annotation) to organize the execution of the tests. It's possible to create simple atomic tests and combine them in more complex sequences including activities pre and post test execution.

A useful usage of this feature is also the definition of a set of tests to be executed as "smoke test" before to proceed with more detailed and time/resources consuming testing scenarios.

7.4.1.2 Test Data Separation
Once the test logic is implemented (but also during the it’s implementation) it might happen that the test developer is not enough aware of the business context of the service to test. A business/functional analyst can complement the work done by the developer by providing the most proper test data (inputs and outputs) to use for the tests execution. In order to facilitate the work of the functional analyst, the UTF Framework allows do report the test data externally from the test.
logic and in a way user friendly approach, or from an already existing source (e.g. xls file, XML, database).

From the developer side this means that a proper "data provider" has to be implemented and configured (via annotation).

```java
@Test(dataProvider = "testData")
public void test(...) {
    ...
}

@DataProvider(name = "testData")
public Iterator<Object[]> data() {
    ...
}
```

TestNG data provider scenario

### 7.4.1.3 Dot Notation

The integration of the rest-assured library, into the UTF Framework, provides an easy to define notation for service invocation and assertion. Using a simple dot notation it's possible to concatenate a sequence of operations that at the end remain well understandable. Here below a couple of examples (refer to the official web site for the complete user manual).

Simple GET with parameters:

```java
given().param("param1", "value1").param("param2", "value2").when().get("/something");
```

Validate the compliance of a resource (/products) with the schema:

```java
get("/products").then().assertThat().body(matchesJsonSchemaInClasspath("products-schema.json"));
```

### 7.4.2 Run the Framework

Before running the framework it has to be compiled with Maven; use the pom.xml file for this. The next two sections describe how to run the UFT Framework in the Eclipse IDE using the TestNG plug-in.
7.4.2.1 **Run the Unit Test**

To run the Unit Test, click the right mouse button on the Unit Test class and select "Run As" - "TestNG Test" to start the execution of the unit test. The "Debug As" option allows to run the test in debug mode. Note that TestNG runs for that class all the methods annotated as Test respecting the dependencies.

![Screenshot of running unit test](image)

```
[TestNG] Running:
/tmp/testng-eclipse--359773445/testng-customsuite.xml

PASSED: test{{contextAttributeType=cell, contextMetaType=all, contextAttributeName

===============================================================================

Default test
Tests run: 1, Failures: 0, Skips: 0

===============================================================================
```

**Run Unit Test**

The previous figure presents an example on how to run the `NotifyContextTest`, where the test method with `Test` annotation is executed.

7.4.2.2 **Run the Scenario**

To run the Scenario, click the right mouse button on the Scenario Test class and select "Run As" - "TestNG Test" to start the execution of the scenario test. The "Debug As" option allows to run the test in debug mode. Note that TestNG runs for that class all the methods annotated as Test respecting the dependencies.
7.5 Use Case

7.5.1 Context

Execution of a functional test for REST services by using UFT Framework

7.5.2 Scenario

Test the REST services by using the UFT Framework that combines the features made available by the TestNG plug-in and the REST-assured library.

7.5.3 Description

A FI-App developer that wants to test a REST service uses the REST Client Generator plug-in (being installed in the Eclipse IDE) to generate the Java client of the REST service on the basis of the WADL url of the service. After that, the developer implements the logic of the functional test-plan by means of the Java client created. The developer can start the functional test by using TestNG and REST-assured plug-ins, in order to test and validate the REST service.

The developer can run the test-plan within the Eclipse IDE or within a Continuous Integration process. This means that the developer can create and run a new test scenario being the
composition of different test-plans that can be run in sequence so to simulate a real interaction among different REST services.

7.5.4 Use Case Diagram

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Test of REST services with UFT Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>FI-App developer, Functional Analyst</td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td></td>
</tr>
<tr>
<td>FI-App developer</td>
<td>create the REST Java client by using REST Client Generator plug-in</td>
</tr>
<tr>
<td>Functional Analyst</td>
<td>data needed for the functional test are made available by the Functional Analyst (e.g.: through a .xls or .xml file)</td>
</tr>
<tr>
<td>FI-App developer</td>
<td>implement the logic for the test</td>
</tr>
<tr>
<td></td>
<td>execute the test</td>
</tr>
<tr>
<td></td>
<td>execute scenario</td>
</tr>
<tr>
<td><strong>Subject</strong></td>
<td>Eclipse + TestNG + REST-assured + REST Client Generator plug-ins</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>WADL url is available for the REST service(s) that has to be tested</td>
</tr>
<tr>
<td><strong>Post condition</strong></td>
<td>REST service(s) tested through the UFT Framework</td>
</tr>
</tbody>
</table>

Functional test for REST services by using UFT Framework
8  NGSITestServer

You can find the content of this chapter as well in the wiki of FI-WARE.

8.1  Introduction

The NGSI10 TestServer Tool is meant as a test tool for programmers. The tool allows to test the interaction of your software with FI-WARE Generic Enablers (GEs) that make use of the FI-WARE NGSI binding (see below for details). Examples of such GEs are the IoT Broker GE and the Template Handler GE from the IoT Chapter.

Instead of directly accessing the IoT GEs during development, you may choose to interact with the NGSI10 Test Server instead. When accessing the NGSI10 Test Server, it will then provide lots of useful information in the status field of the reply (which is human readable). The information provided by the NGSI10 Test Server is embedded in valid replies to server invocations (though it does not implement the fully-fledged NGSI functionality). You can also test if your client is able to fully understand and handle those valid replies. When you are done testing the NGSI10-based interaction with the NGSI10 Test Server, then you can swap out the test server and swap in the actual GE.

8.1.1  The FI-WARE NGSI binding

The FI-WARE NGSI binding is FI-WAREs own binding of the OMA standard known as Next Generation Service Interface Context Enabler (NGSI 9 / NGSI 10). The OMA standard specifies the entities and generic datatypes, but does not specify any transport technology. The FI-WARE NGSI binding then exactly specifies that: how the NGSI standard is used in FI-WARE, namely based on RESTful principles and using HTTP and XML as transport technologies.

OMA Next Generation Service Interface Context Enabler (NGSI 9 / 10) specification models objects of the real world, being it sensor/actuator devices or arbitrary objects (like tables, rooms, cars, ...). They are represented as so-called Context Entities, while information about these objects is expressed in the form of attributes. For more information about the OMA NGSI 9/10 information model and the related interfaces, please refer to the Open API Specification.

8.1.2  Information

Name: NGSI10 Test Server  
Version: 1.0.0  
License: BSD  
Website: (has no dedicated Website)  
Download: http://forge.fi-ware.eu/frs/?group_id=15#title_ngsitestserver  
Scope: Testing the interaction with services based on the FI-WARE NGSI10 binding

8.2  Architecture

The NGSI10 Test Server test server is meant to test clients to FI-WARE GEs that expose an API based on the FI-WARE NGSI10 binding. Such GEs can be found in the Internet of Things (IoT) Services Enablement Architecture. The figure below depicts your application as a client to one or more such GEs.
Instead of directly accessing the IoT GEs during development, you may choose to interact with the NGSI10 Test Server instead. The test server exposes the same REST interface as the actual GEs do. So in order to use the test server instead of the actual GE it is sufficient to replace the hostname in your client: from the host of the actual GE to that of the test server.

To the user the NGSI10 Test Server is a black box, just as the actual GEs are (in the figure below it's actually a yellowish box :) If you are interested in the internal architecture of the NGSI Test Server, please see this page.

When accessing the NGSI10 Test Server, it will then provide lots of useful information in the status field of the reply (which is human readable). The information provided by the NGSI10 Test Server is embedded in valid replies to server invocations (though it does not implement a fully-fledged server functionality). You can also test if your client is able to fully understand and handle those
valid replies. When you are done testing the NGSI10-based interaction with the NGSI10 Test Server, then you can swap out the test server and swap back in in the actual GE.

8.3 Installation
The NGSI10 Test Server runs as a Web application on Apache Tomcat 7. The main component for the installation of the server is a *.war file that is to be deployed on Apache Tomcat as a server module. Therefore you only need to install Apache Tomcat 7 (if you haven't already done so) and deploy the *.war file to it.

8.3.1 Non Installation
In fact, you don't need to install the NGSI10 Test Server in order to use it. It also runs as a hosted version on http://130.206.81.82:8080/NGSI10/. To check if the hosted version is up and running, please navigate to http://130.206.81.82:8080/NGSI10/contextEntities/my_house. If it is up you get an XML document as reply, otherwise an error message. If you are happy with the hosted version, you can skip the rest of this section.

8.3.2 Apache Tomcat 7
8.3.2.1 Download
If you don't have Tomcat 7 already running, you can download it here: Apache Tomcat 7.x

8.3.2.2 Installation
Apache Tomcat 7 is available on several operating systems. The installation instructions given below are for Windows but work accordingly for other platforms.

- Tomcat can be simply installed by unpackaging the download file in a folder of your choice. In the following we will refer to this folder as CATALINA_HOME.
- To start Tomcat, execute

  CATALINA_HOME\bin\startup.bat

- To test if Tomcat is working, open a browser and navigate to http://localhost:8080 (on the machine you installed Tomcat on). If Tomcat is working, you'll see a Congratulations page.
- To shut down Tomcat, execute

  CATALINA_HOME\bin\shutdown.bat

8.3.3 NGSI10 Test Server
8.3.3.1 Download
The NGSI10 Test Server is available here as a *.war file here: NGSI10 Test Server

8.3.3.2 Installation
To deploy the NGSI10 Test Server, copy the Web Application Archive file "NGSI10.war" to

  CATALINA_HOME\webapps\
To test if the NGSI10 TestServer tool is working, open a browser and navigate to http://localhost:8080/NGSI10/contextEntities/my_house (on the machine you installed Tomcat on). If everything worked out you will get a response that looks like this:

```
<contextElementResponse>
<contextElement>
<entityId><id>my_house</id></entityId>
...
</contextElement>
</contextElementResponse>
```

Congratulations, you have successfully installed the FIWARE NGSI10 TestServer.

8.4 User Guide

8.4.1 How it Works

The NGSI10 TestServer provides textual output meant for human interpretation. It does not provide a graphical user interface. Rather, the test server behaves like a normal NGSI10 server in that it sends a valid response to every NGSI10 request sent to it. But unlike normal NGSI10 servers, the NGSI10 Test Server appends extensive debugging information to its responses, namely in the field statusCode.

Every reply to a NGSI10 request has a statusCode element, which in turn has a details element (among others). The NGSI10 TestServer uses this element to return error and debug information to the caller. This way the caller does not need to have access to logfiles on the TestSever. The statusCode element looks like this (for a full example see below):

```
<statusCode>
<code>200</code>
<reasonPhrase>Ok</reasonPhrase>
<details xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema">
... lots of debug information here!
</details>
</statusCode>
```

8.4.2 First Steps

Before you have your software sending NSGI10 requests to the TestSever (or to actual IoT GEs talking NGSI10), you can (and in fact should) send NGSI10 request manually to get the hang of it. You can do this from any browser using a REST client addon (see below). Such addons format the output in a human readable way and allow you to use all HTTP operations (i.e., GET, PUT, POST, DELETE), which you need to access the full functionality of the FIWARE NGSI binding.
Response from the NGSI10 Test Server displayed in the REST Client Firefox plugin

Start your favorite addon and, within the addon(!), navigate to the URL given above. Make sure to specify GET as the request method. You should get a response with status code 200 (OK) and a response body that tells you all the context information available for “my_house”, like this:

```
<contextElementResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <contextElement>
    <entityId>
      <id>my_house</id>
    </entityId>
    <attributeDomainName>in-house</attributeDomainName>
    <contextAttributeList>
      <contextAttribute>
        <name>temperature</name>
        <contextValue>21</contextValue>
        <metadata>
          <name>ID</name>
          <value>temperature</value>
        </metadata>
      </contextAttribute>
      <contextAttribute>
        <name>humidity</name>
        <contextValue>75</contextValue>
        <metadata>
          <name>ID</name>
          <value>humidity</value>
        </metadata>
      </contextAttribute>
    </contextAttributeList>
  </contextElement>
</contextElementResponse>
```
<name>temperature</name>
<contextValue>21</contextValue>
<metadata>
    <contextMetadata>
        <name>ID</name>
        <type>xsd:string</type>
        <value>-622616692</value>
    </contextMetadata>
</metadata>
</contextAttribute>
<contextAttribute>
    <name>humidity</name>
    <contextValue>75</contextValue>
    <metadata>
        <contextMetadata>
            <name>ID</name>
            <type>xsd:string</type>
            <value>-622616691</value>
        </contextMetadata>
    </metadata>
</contextAttribute>
<contextAttribute>
    <name>temperature</name>
    <contextValue>21</contextValue>
    <metadata>
        <contextMetadata>
            <name>ID</name>
            <type>xsd:string</type>
            <value>-622616690</value>
        </contextMetadata>
    </metadata>
</contextAttribute>
<contextAttributeList>
</contextElement>
<statusCode>
    <code>200</code>
    <reasonPhrase>Ok</reasonPhrase>
    <details xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema">====================================
    ============================
    *** Errors and Warnings ***
</details>
</statusCode>
Number of errors: 0

*** Request details ***
GET http://modeler.iot4bpm.de/NGSI10/contextEntities/my_house
Absolute path: http://modeler.iot4bpm.de/NGSI10/contextEntities/my_house
Base Uri: http://modeler.iot4bpm.de/NGSI10/
Path segments:
  segment 1 : contextEntities
  segment 2 : my_house
Query parameters:
  This request has no QueryParameters.

*** Headers ***
Request headers
  header "Accept"
    text/html
    application/xhtml+xml
    application/xml;q=0.9
    */*;q=0.8
  header "accept-encoding"
    gzip
deflate
  header "accept-language"
    null
  header "cache-control"
    max-stale=0
  header "connection"
    Keep-Alive
  header "Content-Type"
  header "dnt"
    1
  header "host"
    modeler.iot4bpm.de
  header "user-agent"
    Mozilla/5.0 (Windows NT 6.1; WOW64; rv:25.0) Gecko/20100101 Firefox/25.0
  header "x-bluecoat-via"
    e1b0f7ae10cc3386
MediaType is null
Acceptable languages

*** Payload ***
payload is empty (i.e., length = 0).

The statusCode element contains a lot of information about your request that can be very handy when debugging the request sent by you (manually or by your software), such as the headers you sent (e.g., the MediaType and the query parameters). Now change the request method to POST and fire the request again. You should get a status code of 400 (Bad Request) with lots of information about what went wrong.

Now you can start developing your own NGSI10 client.

8.4.3 Implementing your own NGSI10 client

In order to access FI-WARE GEs based on the FI-WARE NGSI10 binding, your client needs to be capable of sending XML-encoded RESTful requests to the URL of the GE, which is typically structured like this: http://<hostname>:8080/NGSI10/<service>.

An example URL is http://130.206.81.82:8080/NGSI10/contextEntities/my_house

There are many libraries available for creating such requests in many different programming languages. In order to support you to quickly create a client capable of sending NGSI10 requests, we provide an example Eclipse project for Java. It shows how to create a request using the provided NGSI10 Java library and how to send the request to a NGSI10 TestServer instance.

You can download the Eclipse project from http://catalogue.fi-ware.eu/sites/default/files/storage/enablers/ExampleClient.zip. In order to use the example client you need the Eclipse IDE and a Java Development Kit (JDK) installed on your development machine:


8.4.4 RESTful browser addons

(Incomplete list of) browser addons to invoke operations on RESTful services

- Google Chrome: "Simple REST Client" addon
- Mozilla Firefox: "RESTClient addon"
- Safari: "RESTClient" addon