Project N°:	**FP7-610582**
Project Acronym:	**ENVISAGE**
Project Title:	**Engineering Virtualized Services**
Instrument:	**Collaborative Project**
Scheme:	**Information & Communication Technologies**

Deliverable D1.4.2
Simulation Demonstrator 2

Date of document: T36

Start date of the project: **1st October 2013** Duration: **36 months**

Organisation name of lead contractor for this deliverable: **UIO**

Final version

STREP Project supported by the 7th Framework Programme of the EC		
Dissemination level		
PU	Public	✓
PP	Restricted to other programme participants (including Commission Services)	
RE	Restricted to a group specified by the consortium (including Commission Services)	
CO	Confidential, only for members of the consortium (including Commission Services)	

Executive Summary:
Simulation Demonstrator 2

This document summarises deliverable D1.4.2 of project FP7-610582 (Envisage), a Collaborative Project supported by the 7th Framework Programme of the EC. within the Information & Communication Technologies scheme. Full information on this project is available online at `http://www.envisage-project.eu`.

This deliverable reports on the second demonstrator of the simulation engine for ABS, as part of the activities of Task 1.4. Task T1.4 develops a simulation environment for ABS and addresses work package objective O1.4.

List of Authors

Rudolf Schlatte (UIO)

Contents

Chapter 1

Introduction and New Developments

This deliverable reports on the second demonstrator of the simulation engine for ABS, as part of the activities of Task 1.4. This task develops a simulation environment for ABS and addresses work package objective O1.4. The simulation environment envisaged will combine system level descriptions with resource and deployment models to allow rapid prototyping. The simulation environment closely reflects the formal semantics of the modeling language, yet allows the rapid prototyping of models in different deployment scenarios and with different load balancing strategies.

In this document, we report on new developments since D1.4.1. For basic information about the tool chain, we refer back to D1.4.1. The following new developments have taken place since Deliverable D1.4.1:

- The source code of the simulator is now hosted on github (`https://github.com/abstools/abstools`)
- The simulator has been integrated in the Online Collaboratory (see Deliverable D5.2.2)
- The Erlang backend is on feature parity with the Maude reference backend
- Model simulation time has been reduced by orders of magnitude via the Erlang backend
- Resource simulation has been implemented
- Query functionality for running models has been added, using developer-friendly web-based protocols and data types
- Based on the query functionality, visualization tools have been implemented, both locally and in the Online Collaboratory

The new features in the simulation demonstrator have been evaluated in the case studies of the Envisage project, as reported in Deliverable D4.5.

Installation instructions for the simulator are available at `http://abs-models.org/installation/`. An online version of the simulator is accessible within the Online Collaboratory at `http://abs-models.org/laboratory/`.

Chapter 2

Visualization

2.1 Basic Visualization

Basic visualization support is built into the Erlang backend. To switch on visualization, a model is started with a `-p <port>` argument that defines the local port where the model listens for requests.

When accessing `http://localhost:<port>` (where `port` is the port number given when starting the model), all deployment components and their resource usage over time are depicted. The user can interact with the graph, zoom into selected parts of the timeline and toggle the visibility of selected deployment components.

Figure 2.1 shows the visualization of the resource usage of two deployment components in a basic load balancing scheme.

2.2 Advanced Visualization

More involved and model-specific visualizations can be implemented using the model API, leveraging industry-standard visualization tools.

Figure 2.2 shows the same data as Figure 2.1, but using InfluxDB (`https://www.influxdata.com/time-series-platform/influxdb/`) and Grafana (`https://grafana.org`). These tools support advanced querying and visualization over collected model data. As an example, the two graphs in Figure 2.2 show the same data, once separated into two graphs, once integrated into one graph.

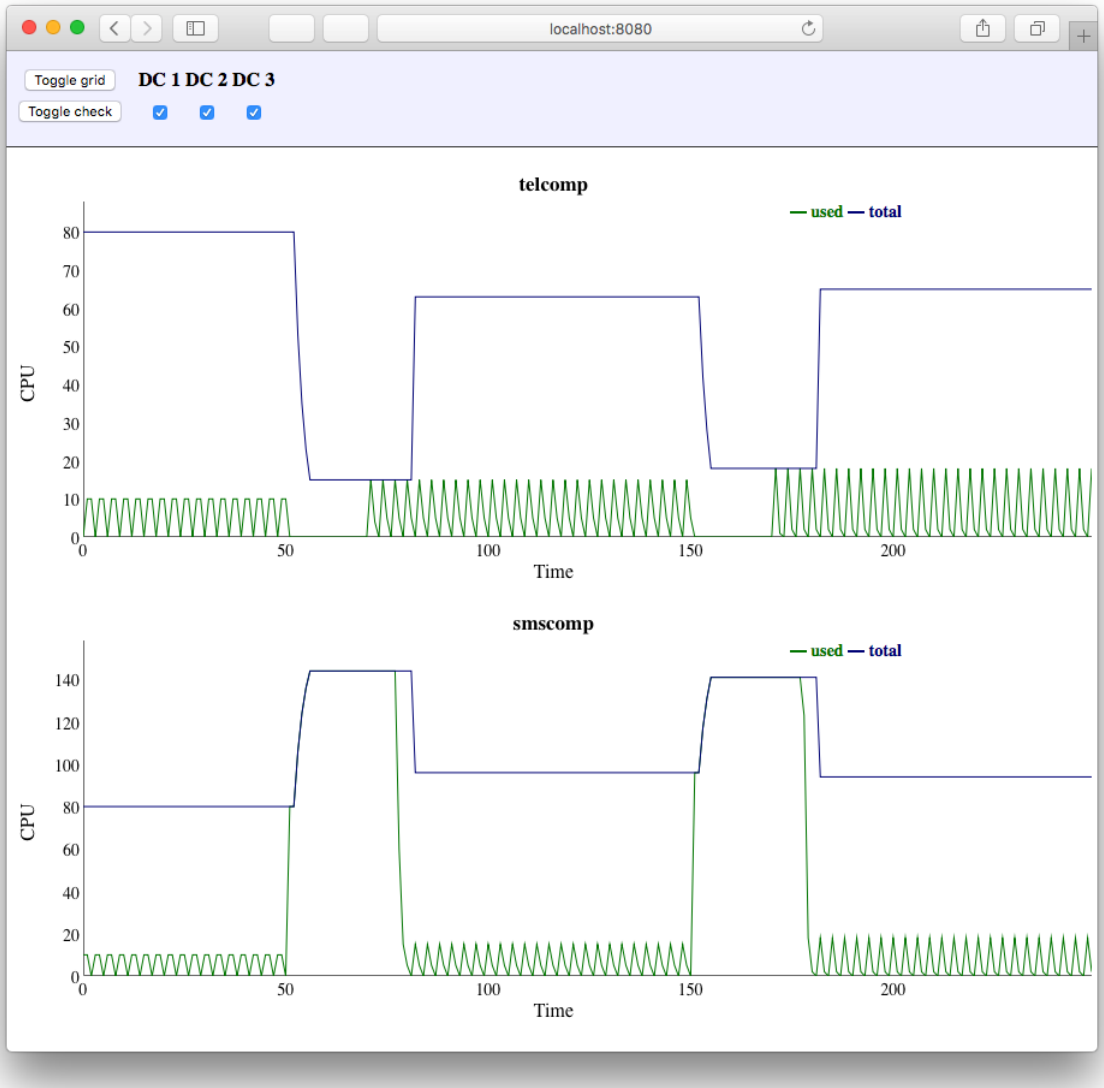

Figure 2.1: Built-in resource visualization

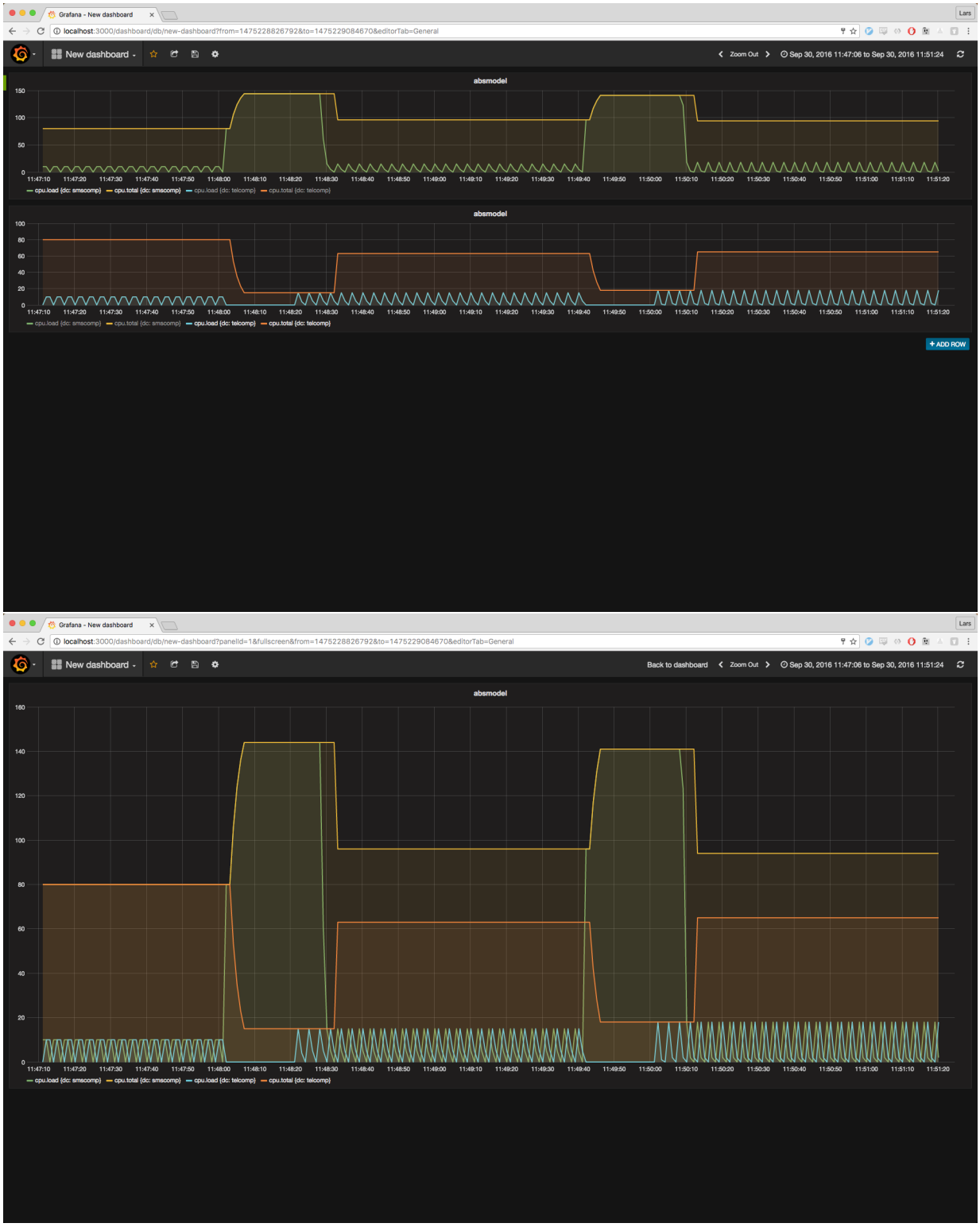

Figure 2.2: Resource visualization using InfluxDB and Grafana. The two screen shots show different views onto the same data.

Chapter 3

Querying Models with the HTTP API

This chapter summarizes the newly-added querying API. To access the API, the model is started with a `-p <port>` argument that defines the local port where the model listens for requests. The `-p` parameter also causes the model not to terminate after the simulation has finished; it instead continues to answer query requests.

The HTTP API provides a means of querying and accessing a simulated model from the outside via HTTP requests. HTTP was chosen for its ubiquity and ease of use. The following features are implemented:

- Objects are registered via an annotation on `new` expressions;
- Methods are made callable via an annotation in an interface definition;
- Objects are protected from garbage collection as long as they are registered;
- Field values of registered objects can be accessed (read-only);
- Callable methods of registered objects can be called via the API, and the return value can be read;
- Registered objects and their fields and callable methods can be enumerated via the API.

This Appendix presents an interim version of the HTTP API; changes in supported parameter and return values and other aspects will be documented in the ABS manual.

3.1 Registering Objects

The annotation

```
[HTTPName: "name"] new C();
```

registers the new object as `name`. In case `name` is already in use, the previous object is unregistered and will not be accessible via the API anymore.

3.2 Making Methods Callable

In an interface declaration, the following

```
[HTTPCallable] String method(Int parameter);
```

declares the method `method` to be callable via the API on registered objects.

- It is a compile-time error if the method takes parameters whose types are not supported. Currently supported types are the ABS types `Int`, `String` and `Bool`.
- The method can have an arbitrary return type. Integers, strings and boolean values will be returned as the equivalent JSON types, other values will be returned as strings via the ABS `toString()` function.

3.3 Querying for Objects

The HTTP request

```
GET http://localhost:8080/o
```

returns a list of registered object names.

3.4 Reading Object State

The HTTP request

```
GET http://localhost:8080/o/name
```

returns a JSON map of state of the object registered as `foo`, as `{ "field": "value", "field2": "value2"}`. Field values are converted in the same way as method return values.

The HTTP request

```
GET http://localhost:8080/o/foo/field
```

returns a JSON map with a singleton entry `{ "field" : "value" }`. In addition:

- Unknown object requests result in a 404 response code
- Unknown field names result in a 404 response code

3.5 Calling Methods

For a registered name `name`, the HTTP request

```
GET http://localhost:8080/call/name
```

returns a, JSON array with metadata about callable functions, with each element of the list being a map with the following entries:

`name` method name

`parameters` array with one object per parameter, each with the following entries:

- `name` name of the parameter
- `type` type of the parameter

`return` return type of the method

The HTTP request

```
GET http://localhost:8080/call/name/method?param1=value&param2=50
```

calls `method` on the object registered as `name`, which is equivalent to performing the ABS method call `foo.method("value", 50)`. The following datatypes can be used for parameters:

`Bool` given as literal upper- or lowercase "true" / "false", i.e. `?p=True`, `?p=true`, `?p=False`, `?p=false`

`String` URLEncoded text, e.g., `?p=Hello%20World!`

`Int` Integer, e.g., `?p=42`

- Successful calls produce a 200 response code and a JSON map with a single entry `result` mapping to the result value.
- Unknown object requests produce 404 response code
- Unknown method name produces 404 response code
- Invalid parameters produce a 400 response code
- Errors during method invocation produce 500 code