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

# Deliverable D4.5
# Overall Assessment

Date of document: T36

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

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

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:
## Overall Assessment

This document summarizes deliverable D4.5 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 contains the overall assessment of the methodology, modeling artifacts, and tool-supported analysis techniques developed in Envisage. The assessment is in terms of the list of expected project outcomes (Description of Work, Part B, Figure 10 and Figure 26), the initial user requirements (D4.1 [3]), and the association of the case studies to the project objectives (described in D4.x.1 [1,2,6]). This deliverable forms the basis for verification of Milestone M5 (Project assessment).

## List of Authors

Einar Broch Johnsen (UIO)
Stijn de Gouw (FRH)
David Costa (FRH)
Keven Kearney (ENG)
Vegard Havdal (ATB)
Crystal Din (UIO)
Samir Genaim (UCM)
Rudolf Schlatte (UIO)
Jacopo Mauro (UIO)
Richard Bubel (TUD)
Antonio Flores-Montoya (TUD)
Elena Giachino (BOL)
Gianluigi Zavattaro (BOL)
Guillermo Roman-Diez (UCM)
Miguel Gomez-Zamalloa (UCM)
Vlad Serbanescu (CWI)
Nikolaos Bezirgiannis (CWI)

# Contents

# Chapter 1

# Introduction

This is Envisage deliverable D4.5. The task description in the DoW is as follows:

> In this task we evaluate the methodology, modeling artifacts, and tools developed in WP1–3 to support the development of services in virtualized environments with respect to their application to the case studies. These tools may include, among others, model simulators, static analyzers, debuggers, log/trace analyzers and visualizers, and so on. The (semi-)automation of the tool-supported design process for services in virtualized environments requires all the theoretical methods, application analyses, modeling and implementation to have reached certain level of maturity. Therefore, this task contributes to the final evaluation of the project results against the initial academic and industrial expectations, and against the additional understanding of the area developed during the course of the project. This evaluation will involve all industry partners and forms the basis for the overall assessment of the potential use and practical impact of formal specification and verification technology in the area of highly adaptable software. It will be the basis for verification of Milestone M5.

**Outline** Chapter 2 assesses how and the extent to which the initial industrial and academic expectations are satisfied. Chapter 3 discusses how the three industrial cases have realized coverage of the overall high-level project objectives. Chapter 4 evaluates how the detailed feedback after Year 2 from the industrial partners given in D4.x.2 [9–11] to specific ongoing technical tasks is addressed. Chapter 5 details the Envisage technology (specifically, the ABS language and the analysis tools) that is used in the case studies. The deliverable concludes with Chapter 6.

# Chapter 2

# Expectations

This chapter assesses the extent to which the initial industrial (Section 2.1) and academic (Section 2.2) expectations have been satisfied by the Envisage framework.

## 2.1 Initial Industrial Requirements

This section assesses how the initial industrial requirements identified in D4.1 [3] are covered by the Envisage framework. For each requirement in the "Evaluation" segment, we discuss whether the requirement remains relevant and update it if needed based on the additional understanding gained throughout the course of the project, and assess if and how the requirement is satisfied. Table 2.1 shows a legend for the scoring system.

| Score | Meaning |
|---|---|
| MF | Modeled and fully supported by Envisage framework |
| PF | Partially modeled, but fully supported by Envisage framework |
| NF | Not modeled, but could, in principle, be fully supported |
| ML | Modeled and supported by Envisage framework with minor limitations |
| MP | Modeled and partially supported by Envisage framework |

Table 2.1: Meaning of Requirement Evaluation Scores

### 2.1.1 Requirements from ATB

Requirements ATB-R001, ATB-R016, ATB-R017, ATB-R018, ATB-R020, ATB-R021, ATB-R025, ATB-R026 and ATB-R027 are deprecated as effort has been refocused on other parts of the modeling, as well as supporting Envisage tool authoring work. All other requirements are discussed below.

| | |
|---|---|
| **Identifier** | ATB-R002 |
| **Name** | Specification and Simulation of Crawl Activation Time for New Sites |
| **Description** | The Envisage framework must be able to model and simulate the effect of the time between content site addition and active crawling ($T_{included}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | In the Memkite model (in Crawling.abs), content publishing is naturally modeled by the addition of an object to a crawling queue, the most common way for crawlers to work. ABS supports the modeling of such a queue. The model expresses the time between addition to this queue and crawling as an asynchronous call to the target web-server. |
| **Score** | MF |

Table 2.2: ATB-002 Specification and Simulation of Crawl Activation Time for New Sites

| | |
|---|---|
| **Identifier** | ATB-R003 |
| **Name** | Specification and Simulation of Content Publish to Crawler Detection Time |
| **Description** | The Envisage framework must be able to model and simulate the effect of the time between content publishing and crawler discovery ($T_{discover}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | The action of getting and storing a web document is naturally expressed in the ABS model, including possible processing, such as new link discovery, of the document. |
| **Score** | PF |

Table 2.3: ATB-003 Specification and Simulation of Content Discover to Crawl Time

| | |
|---|---|
| **Identifier** | ATB-R004 |
| **Name** | Specification and Simulation of Content Update to Crawl Time |
| **Description** | The Envisage framework must be able to model and simulate the effect of the time between content updates (of old objects) and crawling ($T_{update}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | In the ATB model, the crawling queue is modeled as a ring queue, so updates and content site addition share the same code path in the model. See ATB-R002. |
| **Score** | PF |

Table 2.4: ATB-004 Specification and Simulation of Content Update to Crawl Time

| | |
|---|---|
| **Identifier** | ATB-R005 |
| **Name** | Specification and Simulation of Discovery to Crawl Time |
| **Description** | The Envisage framework must be able to model and simulate the effect of the time between new object discovery and crawling ($T_{crawl}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | It is easy to express in ABS that time elapses between the time when a new object enters the crawling queue and the crawling itself. |
| **Score** | MF |

Table 2.5: ATB-005 Specification and Simulation of Discovery to Crawl Time

| | |
|---|---|
| **Identifier** | ATB-R006 |
| **Name** | Specification and Simulation of Re-crawl Scheduling |
| **Description** | The Envisage framework must be able to model and simulate the effect of how recrawls are scheduled ($T_{reschedule}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | Re-adding an object to the crawling queue was covered in the model in a natural way. Different strategies for priorities of adding new content as opposed to re-crawled content have been left for future work, but we are not aware of any limitations of ABS for modeling such strategies. |
| **Score** | PF |

Table 2.6: ATB-006 Specification and Simulation of Re-crawl Scheduling

| | |
|---|---|
| **Identifier** | ATB-R007 |
| **Name** | Specification and Simulation of Priority Classes for Content Sites |
| **Description** | The Envisage framework must be able to model and simulate different aggregate crawl priority strategy distributions ($P_{site}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | Crawling priorities have not been directly incorporated in the model, but it is relatively straightforward future work and ABS should allow modeling of a priority queue without any problems. That queue would then replace the current crawling queue without affecting any other parts of the model.. |
| **Score** | NF |

Table 2.7: ATB-007 Specification and Simulation of Priority Classes for Content Sites

| | |
|---|---|
| **Identifier** | ATB-R008 |
| **Name** | Specification and Simulation of Different types of Content Sites |
| **Description** | The Envisage framework must be able to model and simulate different content site/type distributions ($S_{class}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Extend model for different types of content and run the ABS simulation tool. |
| **Evaluation** | The are no restrictions in ABS with respect to modeling data and operations on data. Although the current model does not directly classify content site types, the `WebServer` class could be easily augmented to cover this aspect. |
| **Score** | NF |

Table 2.8: ATB-008 Specification and Simulation of Different types of Content Sites

| | |
|---|---|
| **Identifier** | ATB-R009 |
| **Name** | Specification and Simulation of Different Classes of Content Object Sizes |
| **Description** | The Envisage framework must be able to model and simulate different content object size distributions ($C_{size}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of content sizes using the ABS simulation tool. |
| **Evaluation** | The crawling model keeps track of content objects in internal data structure. This is easy to model in ABS. |
| **Score** | PF |

Table 2.9: ATB-009 Specification and Simulation of Different Classes of Content Object Sizes

| | |
|---|---|
| **Identifier** | ATB-R010 |
| **Name** | Specification and Simulation of the Effect of Bandwidth Usage |
| **Description** | The Envisage framework must be able to model and simulate the effect of bandwidth usage, including peak bandwidth ($B_{peakbandwidth}$) and aggregate bandwidth ($B_{aggregatedailybandwidth}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Compare against measured value from crawler in cloud system production. |
| **Evaluation** | To evaluate this requirement, we focused on the handset serving subsystem rather than the crawling system, since this was the most interesting. Bandwidth resource analysis was carried out and compared against measured serving bandwidth. |
| **Score** | MF |

Table 2.10: ATB-010 Specification and Simulation of the Effect of Bandwidth Usage

| | |
|---|---|
| **Identifier** | ATB-R011 |
| **Name** | Specification and Simulation of the effect of CPU Usage |
| **Description** | The Envisage framework must be able to model and simulate the effect of CPU usage, including peak usage ($C_{peakcpuresources}$) and average usage ($C_{averagedailycpuresources}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of CPU usage using the ABS simulation tool. |
| **Evaluation** | The ATB model supports CPU speed on the handset, the load balancer, and the back-end deployment components. This is directly supported using cost annotations in ABS. Cost annotations based on real CPU capacity have not been included due to time constraints. |
| **Score** | PF |

Table 2.11: ATB-011 Specification and Simulation of the effect of CPU Usage

| | |
|---|---|
| **Identifier** | ATB-R012 |
| **Name** | Specification and Simulation of Mapreduce Mapper function |
| **Description** | The Envisage framework must be able to support writing and simulating the performance of Mapreduce Mapper functions and generate tests for these |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Compare against the resource consumption of corresponding real Hadoop/MapReduce mapper jobs, e.g. running at scale on AWS Elastic MapReduce or Windows Azure HDInsight Service. |
| **Evaluation** | The Mapreduce model supports CPU performance modeling with parameterized cost. SYCO systematic testing has been carried out for this model. A detailed model of Hadoop was additionally developed by UIO and used to compare model-based predictions with Hadoop YARN for standard Hadoop YARN benchmarks [24]. |
| **Score** | PF |

Table 2.12: ATB-R012 Specification and Simulation of MapReduce Mapper function

| | |
|---|---|
| **Identifier** | ATB-R013 |
| **Name** | Specification and Simulation of MapReducer Reducer function |
| **Description** | The Envisage framework must be able to support writing and simulating the performance of MapReduce Reducer functions and generate tests for these |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Compare against the resource consumption of corresponding real Hadoop/MapReduce Reducer functions, e.g. running at scale on AWS Elastic MapReduce or Windows Azure HDInsight Service. |
| **Evaluation** | Analogous evaluation to ATB-R012. |
| **Score** | PF |

Table 2.13: ATB-R013 Specification and Simulation of MapReduce Reducer function

| | |
|---|---|
| **Identifier** | ATB-R014 |
| **Name** | Specification and Simulation of Processing Time |
| **Description** | The Envisage framework must be able to model and simulate the effect of the time between content object crawl and processing ($T_{processed}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | The coupling between crawling and processing is currently not covered in the model, the two subsystems are separated. Connecting them and simulating the time is future work for ATB. The evaluation of this requirement was given low priority by ATB because it is similar to previously evaluated requirements; there are no limitations of ABS with respect to addressing this requirement. |
| **Score** | NF |

Table 2.14: ATB-014 Specification and Simulation of Processing Time

| | |
|---|---|
| **Identifier** | ATB-R015 |
| **Name** | Specification and Simulation of Indexing Time |
| **Description** | The Envisage framework must be able to model and simulate the effect of the time between content object crawl and indexing ($T_{indexed}$). |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | The coupling between crawling and indexing is currently not covered in the model, the two subsystems are separated. Connecting them and simulating the time is future work. The evaluation of this requirement was given low priority by ATB because it is similar to previously evaluated requirements; there are no limitations of ABS with respect to addressing this requirement. |
| **Score** | NF |

Table 2.15: ATB-015 Specification and Simulation of Indexing Time

| | |
|---|---|
| **Identifier** | ATB-R019 |
| **Name** | Specification and Simulation of Dynamic Cloud Resource Pricing |
| **Description** | The Envisage framework must be able to model and simulate the effect of dynamic cloud resource pricing on general resource usage. |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Evaluate using the ABS simulation tool. |
| **Evaluation** | The Mapreduce model features parameterized cost annotations. The relation between used resource and spending can be defined using the functional layer of ABS, it is straightforward to accumulate the incurred cost for the previous pricing intervals whenever the pricing model changes. |
| **Score** | PF |

Table 2.16: ATB-019 Specification and Simulation of Dynamic Cloud Resource Pricing

| | |
|---|---|
| **Identifier** | ATB-R022 |
| **Name** | Specification and Simulation of Indexing to Distribution Time |
| **Description** | The Envisage framework must be able to model and simulate the effect of the time between content object processing/indexing and push to mobile apps. |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | The processing subsystem is not currently connected to the serving system in the model. The latter uses a static data structure in lieu of results from processing. There are no limitations that we are aware of for modeling this in ABS, both the modeling and evaluation would be similar to previous requirements. |
| **Score** | PF |

Table 2.17: ATB-022 Specification and Simulation of Indexing to Distribution Time

| | |
|---|---|
| **Identifier** | ATB-R023 |
| **Name** | Specification and Simulation of Click to Index Bundle Download Time |
| **Description** | The Envisage framework must be able to model and simulate the effect of the time between an app link click and object download to the mobile app as part of an index bundle. |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Observe effect of time using the ABS simulation tool. |
| **Evaluation** | The handset model keeps track of link click events and object download times. This is captured naturally in the model. |
| **Score** | MF |

Table 2.18: ATB-023 Specification and Simulation of Click to Index Bundle Download Time

| | |
|---|---|
| **Identifier** | ATB-R024 |
| **Name** | Specification and Simulation of Distribution Outbound Bandwidth |
| **Description** | The Envisage framework must be able to support modeling and simulation of bandwidth, in particular the (out)bound bandwidth from the cloud service to a large number of mobile app customers that have data plans and potentially periodic download behavior. |
| **Objectives** | O1.1, O1.2, O1.3, O1.4, O3.2 |
| **Evaluation criteria** | Compare against measured value from production. |
| **Evaluation** | One of the main efforts in year 3. Measured bandwidth numbers from real cloud back-end has been aligned with model. Some discrepancies and Erlang back-end stability issues have come into play. |
| **Score** | ML |

Table 2.19: ATB-024 Specification and Simulation of Distribution Outbound Bandwidth

### 2.1.2 Requirements from ENG

The requirements with the following Identifiers all pertain to the original, rejected ENG case study and have thus been deprecated: ENG-R001, ENG-R002, ENG-R003, ENG-R004, ENG-R005, ENG-R006, ENG-R007, ENG-R008, ENG-R009, ENG-R010, ENG-R011, ENG-R012, ENG-R013, ENG-R014, ENG-R015, ENG-R016 and ENG-R017.

The following requirements have been added and pertain to the revised ENG case study—which is described in the resubmitted deliverable D4.4.1 [6], and in more detail in D4.4.2 [10]. In brief: the case study concerns an on-line code build and test service, ETICS, for software developers, that is able to dynamically exploit on-demand, distributed computing resources (virtual machines, VMs) to 'optimally' satisfy service requests—taking into account existing workloads, as well as cost and penalty terms specified in both consumer- and provider-facing SLAs. The elastic pool of VMs is managed by an automated Resource Pool Manager (RPM), which employs a distributed genetic algorithm (DGA) to determine the best allocation of request processing tasks to VMs. The goal of the case study is to model the RPM & DGA using ABS, and then use the Envisage tools to test this model for possible deadlocks, to analyze its scalability w.r.t. resource consumption, and finally to automatically generate a Java version of the model for live deployment.

| | |
|---|---|
| **Identifier** | ENG-R018 |
| **Name** | ETICS - ABS Model Coverage |
| **Description** | It must be possible to model a significant proportion of RPM and DGA computational operations in detail in ABS. |
| **Objectives** | O1 |
| **Evaluation criteria** | The degree to which the relevant computational operations can be modeled. |
| **Evaluation** | It was possible to fully model all relevant operations at the required level of detail in ABS. |
| **Score** | MF |

Table 2.20: ENG-R018 ETICS - ABS Model Coverage

| | |
|---|---|
| **Identifier** | ENG-R019 |
| **Name** | ETICS - ABS Modeling Effort |
| **Description** | The effort required for ABS modeling (see ENG-R018) must be comparable to the effort required for the standard programming languages used by ENG (e.g. Java and Swift). |
| **Objectives** | O5.3 |
| **Evaluation criteria** | Open list of comments based on user experience. |
| **Evaluation** | • ABS is somewhat verbose compared to modern programming languages. <br> • As an OO language, ABS only supports inheritance at the interface (as opposed to class) level, hence it is not possible to inherit class method implementations - leading either to code repetition/redundancy, or the added complication of using deltas. <br> • The current ABS implementation offers only minimal debugging facilities - namely syntax & type checking. |
| **Score** | MP |

Table 2.21: ENG-R019 ETICS - ABS Modeling Effort

| | |
|---|---|
| **Identifier** | ENG-R020 |
| **Name** | ETICS - Deadlock Analysis |
| **Description** | The Envisage framework must provide tools capable of demonstrating (to a high degree of confidence) that the ABS model of the RPM is either free from deadlocks, or if it is not, exactly where and under what conditions deadlocks may occur. |
| **Objectives** | O3.3 |
| **Evaluation criteria** | Degree to which the requirement is satisfied. |
| **Evaluation** | • Deadlock analysis tools, with provable efficacy, were provided that reported the model to be deadlock free. • In addition, it turns out that ABS models can be constructed in such a way that they are *a priori* free of deadlocks. |
| **Score** | MF |

Table 2.22: ENG-R020 ETICS - Deadlock Analysis

| | |
|---|---|
| **Identifier** | ENG-R021 |
| **Name** | ETICS - Resource Analysis |
| **Description** | The Envisage framework must provide resource analysis tools capable of determining (with good confidence) the computational cost (or bounds on the cost) of invoking specific methods in the ABS model for ETICS (under differing input assumptions). |
| **Objectives** | O3.3 |
| **Evaluation criteria** | Degree to which the requirement is satisfied. |
| **Evaluation** | Appropriate resource tools were provided, and succeeded in giving useful cost expressions for all the required ABS methods. |
| **Score** | MF |

Table 2.23: ENG-R021 ETICS - Resource Analysis

| | |
|---|---|
| **Identifier** | ENG-R022 |
| **Name** | ETICS - Java Code Generation |
| **Description** | The Envisage framework must provide a tool capable of automatically generating complete, working, reliable & efficient Java code from the ABS model for ETICS. |
| **Objectives** | O3.1 |
| **Evaluation criteria** | Degree to which the requirement is satisfied. |
| **Evaluation** | At the time of writing, this tool is still under development and is not incorporated into the standard ABS tool chain. In its current version the tool does not support the complete ABS syntax, and we have not, as yet, been able to generate Java code for the full ETICS model. We have successfully, however, used the tool to generate Java from small test case ABS models. |
| **Score** | MP |

Table 2.24: ENG-R022 ETICS - Java Code Generation

| | |
|---|---|
| **Identifier** | ENG-R023 |
| **Name** | ETICS - Industrial Context |
| **Description** | It must be possible to deploy and use the Envisage framework within an industrial context. For Engineering this includes the use of Windows PCs, tight control over software installation (only approved applications), and high network security. |
| **Objectives** | O5.3 |
| **Evaluation criteria** | Degree to which the requirement is satisfied. |
| **Evaluation** | • The basic Envisage framework (excluding the Java code generation tool) is bundled in a single 'jar' file, and all the tools can be invoked from the command line (with Erlang/Haskell engines preinstalled).<br>• The Collaboratory (a graphical web interface for the tools) is available on-line.<br>• The Collaboratory may also be deployed locally (using Vagrant, Docker, and VirtualBox visualization software) - but doing so on Windows PCs from behind the company proxy is highly problematic (and to date we have not succeeded). |
| **Score** | ML |

Table 2.25: ENG-R023 ETICS - Industrial Context

### 2.1.3 Requirements from FRH

The requirements FRH-R008, FRH-R009, FRH-R016 and FRH-R017 became obsolete during the evolution of the case study. They relate to the impact of code changes as specified using code change specifications (Delta modeling), which was the focus of the previous FRH case study in the HATS project. Delta's have not been used for this purpose in the FRH Envisage case study as the impact of code changes in the production system can be communicated *in real time* with the new HTTP API (delivered as part of D2.3.2 [16]).

| | |
|---|---|
| **Identifier** | FRH-R001 |
| **Name** | Resource usage |
| **Description** | The Envisage framework must not generate monitors that would consume more resources than those using the current monitoring system |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Compare against the resource consumption of the current monitoring system |
| **Evaluation** | We successfully investigated resource consumption of the new monitors with SACO and CoFloCo: the monitors are efficient (linear in the size of the event trace) in both CPU time and memory consumption. Memory consumption is linear since the history of the metric values are stored. This allows scaling decisions based on the full history. Memory usage can be improved to constant usage (!) by restricting to store the metric values for a more limited, fixed time. The history of metric values for visualization purposes is stored in the same platform (Grafana based) as the "legacy" monitoring system, and (thus) has the same resource consumption. |
| **Score** | MF |

Table 2.26: FRH-R001 Resource usage

| | |
|---|---|
| **Identifier** | FRH-R002 |
| **Name** | Executable monitor |
| **Description** | The Envisage framework must generate executable monitors |
| **Objectives** | O2.3 |
| **Evaluation criteria** | The generated monitor must be executable |
| **Evaluation** | Executable monitors are generated in ABS from declarative descriptions. Delivered as part of D2.3.2 [16]. |
| **Score** | MF |

Table 2.27: FRH-R002 Executable monitor

| | |
|---|---|
| **Identifier** | FRH-R003 |
| **Name** | Automatic monitor generation |
| **Description** | The Envisage framework must be able to generate monitors automatically based on a service contract |
| **Objectives** | O2.3 |
| **Evaluation criteria** | The Envisage framework must be able to generate monitors automatically based on a service contract |
| **Evaluation** | Monitors are generated fully automatically from declarative descriptions. Delivered as part of D2.2.2 [14] and D2.3.2 [16]. |
| **Score** | MF |

Table 2.28: FRH-R003 Automatic monitor generation

| | |
|---|---|
| **Identifier** | FRH-R004 |
| **Name** | Correct monitor generation |
| **Description** | The Envisage framework must generate correct monitors with respect to the service contract |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Verify monitor implementations against the service contract. |
| **Evaluation** | Initial verification of monitor correctness was investigated in D2.3.1 [8], based on a translation of ABS to UPPAAL timed automata. This initial work has been improved in two ways:<br>1. We now take the full, final SLAs and service contracts into account (delivered in D2.2.2 [14] and D2.3.2 [16], which were not available yet at the time)<br>2. We leverage KeY ABS to verify monitor correctness, to allow modular, more scalable verification and user interaction for complex properties. |
| **Score** | MF |

Table 2.29: FRH-R004 Correct monitor generation

| | |
|---|---|
| **Identifier** | FRH-R005 |
| **Name** | Efficient monitor generation |
| **Description** | The Envisage framework must generate monitors based on a service contract more efficiently than the manual configuration of the current monitoring system |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Compare the time that the Envisage framework takes to generate monitors against the manual configuration of the current monitoring system |
| **Evaluation** | Monitors are fully automatically generated in a few seconds from a grammar formalizing an SLA metric. |
| **Score** | MF |

Table 2.30: FRH-R005 Efficient monitor generation

| | |
|---|---|
| **Identifier** | FRH-R006 |
| **Name** | Unified monitors management |
| **Description** | The Envisage framework must generate monitors that can be managed under a unified management interface |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Generated monitors can be managed under a unified management interface |
| **Evaluation** | An API to interact with ABS models over HTTP was delivered. This API was used to plug in user-defined monitors as data sources for the Grafana visualization framework (delivered in D2.3.2 [16]).<br>With the HTTP API, one can plug-in external monitors, implemented in an arbitrary language. This allows all monitors to be managed under the unified interface offered by Grafana, and enables the definition of high-level monitors that combine multiple lower-level monitors from different external systems. |
| **Score** | MF |

Table 2.31: FRH-R006 Unified monitors management

| | |
|---|---|
| **Identifier** | FRH-R007 |
| **Name** | Non-invasive monitor management |
| **Description** | The Envisage framework must be able to generate and deploy monitors without decreasing the uptime of FRH services |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Check that the uptime of FRH services does not decrease during monitor deployment |
| **Evaluation** | Monitors are fully decoupled (separately running asynchronously, and deployed on dedicated VM's) from the service instances, and thus cannot affect the uptime of the service instances. |
| **Score** | MF |

Table 2.32: FRH-R007 Non-invasive monitor management

| | |
|---|---|
| **Identifier** | FRH-R010 |
| **Name** | Automatic monitor adaptation |
| **Description** | Given the analysis result of code changes provided by the Envisage framework, the Envisage framework must be able to adapt automatically the monitoring system at runtime to ensure the relevant properties are monitored |
| **Objectives** | O2.1, O2.3 |
| **Evaluation criteria** | Check that this facility is provided by the Envisage framework |
| **Evaluation** | During the evolution of the case study, the requirement with respect to code change specifications became obsolete. Nonetheless, the delivered HTTP API for ABS allows plugging in external monitors over HTTP (for example, from the live production environment), which enables immediate, real-time, propagation to the ABS model of changes to the in-production system (as a simple example: we can take into account current infrastructure problems in an Availability Zone in Amazon). |
| **Score** | NF |

Table 2.33: FRH-R010 Automatic monitor adaptation

| | |
|---|---|
| **Identifier** | FRH-R011 |
| **Name** | Non-invasive monitor adaptation |
| **Description** | The Envisage framework must be able to adapt the monitoring system at runtime without decreasing the uptime of FRH services |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Check that the uptime of FRH services do not decrease during adaption |
| **Evaluation** | The monitors themselves are decoupled (separately running asynchronously, and deployed on dedicated VM's) from the service instances. However, further tests are needed to see if/how the corrective actions taken by monitors at run-time (such as auto-scaling) affect the uptime. |
| **Score** | ML |

Table 2.34: FRH-R011 Non-invasive monitor adaptation

| | |
|---|---|
| **Identifier** | FRH-R012 |
| **Name** | Unified visualization of monitored data |
| **Description** | The Envisage framework must provide a unified interface to visualize and manage monitored data |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Monitored data can be visualized and managed under a unified interface |
| **Evaluation** | All monitors are visualized using the unified interface offered by Grafana. See D2.3.2 [16] for more information. |
| **Score** | MF |

Table 2.35: FRH-R012 Unified visualization of monitored data

| | |
|---|---|
| **Identifier** | FRH-R013 |
| **Name** | Automatic visualization generation |
| **Description** | The Envisage framework must be able to generate real-time visualization for monitored data. |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Check that this facility is provided by the Envisage framework |
| **Evaluation** | Monitored data is forwarded and stored in InfluxDB, and is then exposed as a HTTP endpoint for the Grafana visualization framework. The delay (in seconds) to propagate new monitored data to InfluxDB can be configured, but is close to real-time if needed. |
| **Score** | ML |

Table 2.36: FRH-R013 Automatic visualization generation

| | |
|---|---|
| **Identifier** | FRH-R014 |
| **Name** | Non-invasive visualization |
| **Description** | The Envisage framework must be able to generate real-time visualization for monitored data without affecting the monitoring process and the uptime of FRH services |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Check that both the monitoring process and the uptime of FRH services are not affected by the visualization generated by the Envisage framework |
| **Evaluation** | Each monitor exposes its history of metric values as a time-value series, as a HTTP endpoint. The time series is retrieved for visualization periodically by invoking an HTTP request to that endpoint.<br>Invoking the endpoint generates a new asynchronous task in the monitor, which means that when this task is processed (scheduled), other tasks in the process queue of the monitor wait for it to finish. Thus, it may be possible to negatively affect other processes inside the monitors by polling the endpoint at extremely short delays. In practice however, we observed no measurable effects even when polling with intervals of much lower than 20s (the rate at which the existing monitoring system updates). |
| **Score** | ML |

Table 2.37: FRH-R014 Non-invasive visualization

| | |
|---|---|
| **Identifier** | FRH-R015 |
| **Name** | Efficient visualization |
| **Description** | The Envisage framework must be able to generate real-time visualization for monitored data efficiently |
| **Objectives** | O2.3 |
| **Evaluation criteria** | Compare the resource usage of the Envisage framework when generating visualization of monitored data against existing techniques. Compare any time delay in the visualization generated by the Envisage framework against existing techniques. |
| **Evaluation** | The same visualization tooling (Grafana) is used as in the in-production monitoring system. This means the new monitors have the same resource consumption as the existing monitors. Monitored data from the new monitors is propagated for visualization with a configurable time delay. Preliminary tests were successful even with delays shorter than 20s (the delay of the existing monitors). |
| **Score** | MF |

Table 2.38: FRH-R015 Efficient visualization

| | |
|---|---|
| **Identifier** | FRH-R018 |
| **Name** | Efficient test case execution |
| **Description** | The Envisage framework must generate and execute test cases that have better test coverage than the existing testing methodologies. Moreover, The Envisage framework must generate test cases that can be executed on existing resources |
| **Objectives** | O3.2, O3.3, O3.4 |
| **Evaluation criteria** | Compare the test coverage generated by the Envisage framework against existing test cases |
| **Evaluation** | Existing test cases focus on functional unit tests of service implementations. The automated tests generated by the Envisage framework extend the test coverage of the existing test suite by exploring all different task interleavings to expose race conditions. |
| **Score** | ML |

Table 2.39: FRH-R018 Efficient test case execution

## 2.2 Envisage KPIs

Table 2.40 summarizes the Envisage KPIs and their realization by the end of the funding period. Further details concerning these KPIs are given in the Year 3 periodic progress report.

Table 2.40: Quantitative Key Performance Indicators of Envisage.

| KPI title | KPI target value | Realized |
|---|---|---|
| ♯ Modeling Tools | **4** (Computations, Resources, Deployment, SLA) | **4** (These have been realized) |
| ♯ Analysis Tools | **4** (Resource analysis, Deadlock analysis, Performance, SLA compliance) | **4** (These have been realized) |
| ♯ Variability of the Technological Experimentations | **3** (Search Technology, e-commerce, cloud provisioning) | **3** (These have been realized) |
| ♯ Scientific Dissemination | **50** peer-reviewed scientific publications (in Conferences and Journals with $\geq$ **20** average field rating according to MS academic research)<br>$\geq$ **15** joint publications<br>**2** scientific workshops | **51** high-tier publications (**97** publications in total)<br>**30** joint publications<br>**2** scientific workshops |
| ♯ Industry Dissemination | **5** members of the Advisory Board<br>$\geq$ **15** members of the Industry Follow Group<br>$\geq$ **5** industrial on-site presentations<br>$\geq$ **4** blog posts with more than 8.000 views<br>**2** Industry Days | **5** Industrial Advisory Board members<br>**40** Industrial Follow Group members<br>**12** industrial on-site presentations<br>**6** blog posts with more than 8000 views<br>**2** Industry Days |
| ♯ Technological Dissemination | **3** (Eclipse plugin, Open-Source Collaboratory, invited keynote lectures) | **3** (all three points have been addressed) |
| ♯ Societal Dissemination | $\geq$ **8** (public releases, press releases)<br>**3** white papers | **8** public releases<br>**3** white papers |

# Chapter 3

# Coverage of Project Objectives by Industrial Cases

This chapter discusses how the case studies have covered the (verification of the) overall project objectives. Deliverables D4.x.1 [1, 2, 6] outlined an initial planning of the association between the case studies and the project objectives. Here we discuss how the association was actually realized.

## 3.1 Objective O1: Foundations of Computation with Virtualized Resources

**ATB.** We have used the modeling constructs covering deployment architectures as well as compute and bandwidth resources associated with these architectures to capture virtualization aspects of the crawling, Mapreduce, serving and handset subsystems of the Memkite system.

**ENG.** The ENG case study employs the Envisage methodology to develop an automated Resource Pool Manager (RPM) for the elastic management of the ETICS virtualized resources. A detailed account of what (from the perspective of industry) such elastic management entails is reported in D4.4.1 [6], where ENG also outlined a probabilistic model of service demand, from which estimates of workload (and in particular, of dynamic variation in workload) can be derived.

**FRH.** The outcome of this objective is a semantic framework for scalable architectures, infrastructures, and virtualized resources. The framework provides the means to model and to specify resource-related non-functional requirements that arise in the context of virtualized resources. Deliverable D4.3.1 [2] forms the verification of milestone M1. Specifically, using the ABS language, we were able to model the Fredhopper Cloud Services. This initial model focuses on the component structure of the Fredhopper Cloud Services and the functionalities of individual components, while abstracting away from the implementation of the services offered.

## 3.2 Objective O2: Behavioral Specification Language for Virtualized Resources

**ATB.** The Memkite model uses the Envisage ABS language. Resource modeling, specifically bandwidth modeling, has been successfully deployed in year 3. The work is detailed in D4.2.3 [12].

**ENG.** The automated Resource Pool Manager, and all the virtual resources (machines) that it manages, are all implemented (as executable models) entirely in the abstract behavioral specification language (ABS). This ABS model critically relies on the resource modeling capabilities of ABS (cloud provider and deployment component constructs). The development of the model also relied critically on the (Erlang) prototype simulator for debugging, tracing the execution, and testing.

**FRH.** The initial model of the Fredhopper Cloud Services (reported in D4.3.1 [2]) was successfully extended with *resource-awareness* based on the resource and deployment modeling in ABS developed in T1.2 and T1.3. We modeled the different kinds of virtual machines used in the Fredhopper Cloud Services as deployment components (allocated through the infrastructure service as offered by the CloudProvider API), we added cost annotations based on measurements from real-world log files, and were able to capture real-world deployment scenarios. This work was reported in detail in D4.3.2 [11].

## 3.3 Objective O3: Design-by-Contract Methodology for Service Contracts

**ATB.** As discussed in D4.2.2 [9], the primary SLA for ATB is how fast data that has arrived in the Cloud Backend becomes searchable on a customer's mobile device. This is an end-to-end property which distributes over the different components of the ATB case study; i.e., the individual components have been modeled with respect to their expected available bandwidth. The bandwidth modeling carried out supports this objective: Using the ABS model, we can assess the end-to-end processing time with different numbers of handsets and back-end topologies.

**ENG.** The ENG case study is defined in terms of the relation between functional service components (in particular the RPM) and their required non-functional properties (derived from consumer & cloud provider facing SLAs). D4.4.1 [6] provides detailed definitions of the relevant QoS terms, and explains how these terms impact:

- the functional properties of the service (in particular, the internal calculations of the RPM);
- the non-functional properties of the service - i.e. the actual completion-times & failure-rates (as opposed to the normative values specified in SLAs); and
- the overarching business-level concerns of the service (for simplicity, just profit).

**FRH.** Using the technical contributions developed in T2.2 and T2.3, we were able to bridge the gap from the FRH SLAs formulated in natural-language, to formal SLAs and Service Contracts amenable to static and dynamic analysis.

Specifically, SLAs and Service Contracts are modeled as properties of service metric functions. A service metric function maps a trace of events from the Service APIs to a value that indicates a QoS property or the value of a Service Level Objective. The service metric functions themselves have been formalized in a declarative manner with attribute grammars (which are synthesized to ordinary ABS code, amenable to the Envisage analyses). This work is reported in D2.2.2 [14], D2.3.2 [16] and D4.3.3 [13], and in the papers on which these deliverables are based.

## 3.4 Objective O4: Model Conformance Demonstrator

**ATB.** A comparison has been carried out with respect to model conformance for network bandwidth of the serving system. Model conformance has been gauged for this subsystem.

**ENG.** The actual ETICS service is implemented in Java. Using the Envisage Java Code Generation and verification tools we expect to be able to automatically generate working Java code from the ABS model, which also verifiably conforms to the formal semantics of the source ABS model. The goal of employing Java code generation in the ENG case study is just to provide a proof of concept assessment of the feasibility of adopting ABS in Engineering's service production life-cycle.

**FRH.** We took various steps to ensure a close relation between the in-production Fredhopper Cloud Services and its ABS model. The resources (i.e. kinds of virtual machines, and their properties) and resource consumption of Service instances modeled in ABS were based directly on data from the in-production Fredhopper Cloud Services. Furthermore, at the API level, the ABS model is almost identical to the in-production system. This allowed us to:

1. Faithfully formalize the FRH SLAs, as the SLAs between FRH and its customers concern QoS properties of the Service APIs exposed to the customers.
2. Plug in data from the in-production system in real-time (such as externally defined monitors), through the ABS HTTP API.

More details are reported in D4.3.1 [2], D4.3.2 [11], D4.3.3 [13] and D2.3.2 [16].

## 3.5 Objective O5: Model Analysis Demonstrator

**ATB.** As D4.2.3 shows in detail, we have utilized the following analysis tools:

- Deadlock analysis
- SYCO systematic testing
- Erlang resource simulation

The D4.2.3 [12] report details the purpose of each.
**ENG.** The ENG case study used the tools developed in WP3 to verify the ABS model of the RPM against non-functional requirements – specifically: to ensure the efficacy, scalability and cost-effectiveness of the RPM (as described in D4.4.1 [6]). The main objectives of this activity were:

- To employ Envisage deadlock analysis tools to ensure (if possible) that the Distributed Genetic Algorithm (DGA), employed by the RPM to determine the best resource assignment option, is deadlock free;
- To employ Envisage resource analysis tools (specifically for calculating the number of computational steps required for critical code segments) to determine both the scalability of the DGA, and the values of key constants in the ABS model which directly impact overall profitability of the service.

**FRH.** The outcome of this objective is the runtime support for the resource analysis and for the validation with the SLA. This was successfully accomplished through the application of a combination of Envisage analyses on the Fredhopper Cloud Services.

- Dynamic analysis: We have used the monitoring framework to automatically synthesize executable ABS monitors (which can be fed with real-time information from the in-production system, through the HTTP API) that directly calculate the QoS attributes in the FRH SLAs and Service Level Objectives. We integrated the SmartDeployer (see D1.3.2 [15]) into the monitoring framework: this allows to autoscale the deployment configuration, while respecting high-level deployment requirements, and does so on a rigorous basis (i.e. based on the QoS attribute or SLOs measured by the monitors).
- Static Analysis: We have also used several static analyses on the ABS model to statically show several properties required for compliance with SLAs. For instance we have verified deadlock freedom, proved termination of all methods expected to terminate, statically derived bounds on resource consumption of services (SACO) and verified monitor correctness (KeY ABS).

More details on the above work are reported in D4.3.3 [13], D1.3.2 [15] and D2.3.2 [16].

Chapter 4

# Feedback to Technical Tasks in Year 2

Based on the usage of the Envisage framework in the case studies, the industrial partners provided detailed feedback after year 2 in deliverables D4.x.2 [9–11], pointing out desired extensions and refinements to increase accuracy and effectiveness.

This feedback formed input for the final reports of T1.2, T1.3, T1.4, T2.2 and T2.3. In Section 4.1 we assess how this feedback was addressed. Each item shows the affected **Task**, the industry **Partner** that gave the feedback, the **Feedback** itself, and a **Discussion**, detailing how the feedback is addressed. Section 4.2 shows some of the other feedback given by the industrial partners.

## 4.1 Formal Feedback

| | |
|---|---|
| **Task** | T1.2 |
| **Partner** | ATB |
| **Feedback** | The support for modeling cloud resources with ABS is overall good, but given the increasing importance of mobile becoming the primary point of contact between people and the cloud, abstractions for modeling mobile resources could be an advantage to add to ABS. |
| **Discussion** | Bandwidth resource simulation was added at the behest of the FRH and ATB case studies. For the ATB case study in particular, other mobile resources can be added easily once their semantics are fully specified. |

| | |
|---|---|
| **Task** | T1.2 |
| **Partner** | ENG |
| **Feedback** | In D4.4.2 [10] ENG provided a detailed model of the distributed implementation of a real life industrial service – employing an elastic pool of virtual machines (VMs), with VMs dynamically added/removed according to the changing needs of the service consumers. |
| **Discussion** | The needs of the ENG case study were successfully covered by the resource modeling task. |

| | |
|---|---|
| **Task** | T1.2 |
| **Partner** | ENG |
| **Feedback** | In D4.4.2 [10] ENG provided a detailed model of the resources (virtual machines, CPUs, memory, etc.) relevant to Engineering's case study - including a precise account of how this resource information is used to determine which VMs to deploy and how to assign/schedule requests to VMs. |
| **Discussion** | The needs of the ENG case study were successfully covered by the resource modeling task. |

| | |
|---|---|
| **Task** | T1.2 |
| **Partner** | FRH |
| **Feedback** | The accuracy of the modeling of the resource types can be enhanced by supporting multiple metrics as the capacity of the resource. More specifically, for memory, in addition to the total size of the memory, its speed can be considered. For the CPU resource type, the number of cores could be distinguished from the speed of the cores. At the moment, the capacity of each resource type present is given by a *single* number, forcing the user to choose between memory capacity *or* memory speed (and similarly for the other resource types). |
| **Discussion** | Based on the feedback from the case studies, we have extended the modeling of resource types to accommodate the number of cores, the memory size, as well as a pricing model (i.e., how resource usage translates into a pricing scheme) |

| | |
|---|---|
| **Task** | T1.2 |
| **Partner** | FRH |
| **Feedback** | A fundamental additional resource type not currently supported is *storage*. The speed (rather than the storage capacity) of the storage device has been a bottleneck in the in-production system at FRH, and has affected deployment decisions, in the sense that I/O-optimized machines are used for certain kinds of services. |
| **Discussion** | Based on this feedback and the related feedback from ATB, we have extended the modeling of resources with support for bandwidth modeling, which expresses the data transmission delay between two ABS objects. |

| **Task** | T1.2 |
|---|---|
| **Partner** | FRH |
| **Feedback** | In practice, virtual machines and resources can fail (more on this below, in T1.3), and machines can be terminated. This should not change only the state of the machines (or `DeploymentComponents`), it also affects the objects running on these machines: in reality, these stop executing / fail. To capture this behavior accurately, the ABS semantics for objects should take failures of the underlying virtual machine on which it runs into account. |
| **Discussion** | Appropriate failure models for virtualized systems in ABS were discussed extensively in Task T1.1. A failure model in this setting needs to address asynchronous communication and delegation in an efficient way. Based on the experiences from these discussions and the feedback from the case studies, a failure model was developed and integrated with the ABS semantics. |

| **Task** | T1.3 |
|---|---|
| **Partner** | ATB |
| **Feedback** | In actual cloud deployment there is typically a separation between 1) provisioning of resources and 2) performing software/data deployment on provisioned resources, where the former might be specific for a cloud vendor (e.g. AWS has tools like Elastic Beanstalk2 or third party libraries like Boto3 for doing this—ATB uses Boto), and the second is usually more generic and is similar for most IAAS clouds (e.g. using tools like Chef4 or Puppet5—ATB uses Chef). The provisioning support in ABS is good (see also T1.2), but support for modeling of (automated) software/data deployment could be extended. Typical issues that needs to be handled in software/data deployment is version conflicts (e.g. on library dependencies), configuration file handling, security (e.g. creation and distribution of encryption keys and SSL certificates) and tight integration with continuous deployment system. |
| **Discussion** | Issues like library dependencies, creation and distribution of encryption keys are outside the Envisage objectives; they are considered orthogonal w.r.t. the typical ABS behavioral specifications mainly considered in Envisage. Nevertheless, even if this is not central for Envisage, ABS includes delta modeling facilities that can be used to appropriately specify these orthogonal aspects in a modular way.<br>Concerning the tight integration with continuous deployment systems, the SmartDeployer enables a such tight integration with its fully automated and optimized deployment synthesis. It does not commit to a particular continuous deployment system, but the SmartDeployer could be adapted to such a system in a relatively straightforward way. |

| | |
|---|---|
| **Task** | T1.3 |
| **Partner** | FRH |
| **Feedback** | Virtual machines can fail, for instance due to an operating system kernel error, or a hardware failure, but currently, failures are not taken into account. Support for modeling such failures opens the door for reasoning about failures (both static *and* dynamic analyses, through monitoring and simulation), such as how failures *propagate*, and design of fault-tolerant systems. |
| **Discussion** | Failure of machines can be modeled as *abrupt shutdown* and lead, in ABS semantics, to objects becoming unresponsive. Unresponsive objects can be used to model both network and machine failures. The failure model of ABS can be used to model all of these failures. |

| | |
|---|---|
| **Task** | T1.3 |
| **Partner** | FRH |
| **Feedback** | Virtual machines can be started, stopped, fail, and so on: they have a well-defined *state*. Throughout its lifetime, the instance moves from state to state. When an instance is started, it enters an initialization state. After it initializes successfully, it enters a “running” state. Instance states can also be triggered involuntarily, due to failures.<br>Adding states to virtual machines, and making it possible to retrieve this state for a specified machine, is important for load balancing, monitoring, cost management (i.e. charge only for machines in running state), etc. and allows implementing services with increased robustness. |
| **Discussion** | The Cloud API and, specifically, the `CloudProvider` interface, implement this deployment component life-cycle. Additionally, a mechanism was introduced to facilitate exclusive access to deployment components among cooperative actors; this proved useful for high-level load balancing and worker pool objects. |

| | |
|---|---|
| **Task** | T1.3 |
| **Partner** | FRH |
| **Feedback** | The expressiveness of the cost annotations that the Model-Driven Deployment Engine (MODDE) (see D1.3.1 [7], Chapter 2) supports could be enhanced: currently costs are constants. In practice, the exact cost is not necessarily fixed over time, and can depend on certain parameters, for instance: execution of a query by the FRH query service typically increases the larger the product catalog is. |
| **Discussion** | MODDE (and its successor, SmartDeployer) supports defining multiple deployment scenarios. This can be used to specify the cost of an object in different situations (i.e. with a different constant for each scenario).<br>As future work, we are investigating invoking the solver and deployment synthesizer at run-time through a foreign language interface or HTTP API for ABS. This would allow arbitrary cost expressions (the solver at run-time would simply receive the value the expression evaluates to), while avoiding to complicate the solver. |

| **Task** | T1.3 |
|---|---|
| **Partner** | FRH |
| **Feedback** | The Cloud API (D1.3.1 [7], Chapter 3) currently does not take into account which kind of virtual machines are offered by the infrastructure provider: it is possible to acquire a machine with *any* resource capacity, include resource configurations not offered by any existing machine. This could be addressed by making the API parametric with respect to the available virtual machine types. |
| **Discussion** | We have adapted the Cloud API such that the modeler can define machine configurations and create instances by name. The possibility of creating arbitrary deployment components still exists but is optional. |

| **Task** | T1.4 |
|---|---|
| **Partner** | ATB |
| **Feedback** | The model supports simulation end-to-end, from the data arrives in the cloud service until it has become searchable on the mobile devices. It also supports simulation with varying amounts of load (e.g. amounts of data to index and number of mobile devices) and varying resources to handle the load (e.g. number and capacity of virtual machines used for indexing and serving of data in the cloud backend).<br>Since individual latency—from when the data arrives in the cloud until it arrives on the mobile device—is of key importance for the SLA, a useful addition to the model could be to add “tracer bullet” style logging that follows the data through the system, i.e. annotate data with timestamps in every step of the model in order to see where it uses time. |
| **Discussion** | This feature can be implemented by adding a small amount of model code at the inspection points, and defining the data in a suitable way. Current developments integrating the ABS simulator with a time series database will make the results visualizable and query-able easier. |

| **Task** | T1.4 |
|---|---|
| **Partner** | ENG |
| **Feedback** | The Erlang simulator successfully executes Engineering’s ABS model – but only when setting the number of generated requests to 4 or less. At 5 or more requests the simulator stalls. We would like to be able to simulate results for thousands of requests. |
| **Discussion** | When moving from Maude to Erlang from the simulation back-end, we got a significant performance gain. The Erlang back-end has been tested with models consisting of 10000 requests and 100 virtual machines, but not for the ENG case study. The Erlang simulator scales well for many other models. (We are working on it. Remark that this issue was reported at M30, not at M24.) |

| | |
|---|---|
| **Task** | T1.4 |
| **Partner** | FRH |
| **Feedback** | Simulation to observe the effect on the system while, for example, varying the number of received requests can be done by modeling (various kinds of) “the user” in ABS code: the user triggers the system by invoking requests to an end-point at a certain rate. Nonetheless, modeling the user in this manner is cumbersome. A “log-replay” tool that fires queries to the system according to a specified time and duration given in a log file would be a very useful addition for simulation purposes. |
| **Discussion** | This is a very good feedback and such support is now under development. As part of the work in T1.4, the ABS simulation back-end has been opened up to allow interaction with external programs through an API. A prototype log replay tool using this API has been implemented and tested on the FRH case study. |

| | |
|---|---|
| **Task** | T1.4 |
| **Partner** | FRH |
| **Feedback** | An important enhancement to the Eclipse plug-in is support for navigation, such as “browse to declaration”, “display type hierarchy” and “display call hierarchy”. |
| **Discussion** | This would be nice to have, but has not been given high priority in the Envisage project. In the project, we have given priority to the collaboratory rather than the Eclipse plug-in. |

| | |
|---|---|
| **Task** | T1.4 |
| **Partner** | FRH |
| **Feedback** | The Maude back-end supports inspection of the complete state of the system, which allows exploring detailed run-time information. However, this state can be unwieldy; sometimes a more abstract version of the information is already sufficient and allows faster identification of relevant data. In particular, support for visualizing the object graph, visualization of resource usage over time, and a visualization of the trace of messages between distributed COGs would be useful. |
| **Discussion** | This is very good feedback. The simulation back-end now directly generates visualizations of the resource usage over time through a generic format for sharing time series with a visualization back-end in the collaboratory. Further support for visualization is under development. |

| | |
|---|---|
| **Task** | T2.2 |
| **Partner** | ATB |
| **Feedback** | The primary SLA for ATB is how fast data that has arrived in the Cloud Backend becomes searchable on a customer's mobile device. The model, which allows simulations of that time, is given various scenarios relevant for the SLA, e.g. mobile bandwidth capacity for customers. Since latency is a common metric in SLAs, we expect that the support for SLAs can be combined with the suggested "tracer bullet" approach for simulation in T1.4. |
| **Discussion** | Talking about SLA verification, latency is one of those metrics we didn't expect to verify statically. On the contrary, we left this for the runtime verification framework. Simulation may indeed be used to provide auxiliary information to the static analysis. |

| | |
|---|---|
| **Task** | T2.2 |
| **Partner** | ENG |
| **Feedback** | In D4.4.2 [10] ENG provided a detailed model of the SLAs and QoS terms/guarantees relevant to Engineering's case study – including a precise account of how this SLA information is used to determine which VMs to deploy and how to assign/schedule requests to VMs. |
| **Discussion** | The needs of the ENG case study were successfully covered by the task on service contracts and SLAs. |

| | |
|---|---|
| **Task** | T2.2, T2.3 |
| **Partner** | FRH |
| **Feedback** | Behavioral interfaces (D2.2.1 [5], Chapter 3) naturally capture properties of the behavior of a *single* instance of a given type, such as the response time guarantee example. Support for defining *aggregated* properties that involve multiple objects (or computation / communication traces) executing on different (distributed) virtualized resources would be useful. This would allow to capture properties of statistics—aggregations of a metric—such as "Total Number of Fredhopper Query Requests", or the "Maximum response size" (in a given time window). |
| **Discussion** | Behavioral type systems express information useful for a particular property or measurement of the code. They support the definition of properties that involve multiple distributed objects, as long as we know either the code of those objects (in order to be able to extract their behavior), or explicitly their behavioral type. Furthermore, in D2.2.2 [14] and D2.3.2 [16] we have finalized the formalization of service metric functions, which allow mapping a stream of heterogeneous events from distributed sources to an aggregated value. |

| | |
|---|---|
| **Task** | T2.3 |
| **Partner** | FRH |
| **Feedback** | Service metric functions, or statistics, aggregate a sequence of basic measurements (of a certain metric) to allow determining QoS levels. The question arises how such statistics can be defined in a systematic manner. A possible option to investigate is using attributes defined in an attribute grammar for this purpose. Informally, an attribute is a function that assigns an aggregated value to a list of symbols (in our context, basic measurements). The definition of attributes can exploit structure present when different kinds of measurements should be aggregated. |
| **Discussion** | This suggestion has been taken up. Service metric functions are formalized as attributes of a regular attribute grammar, as reported in Deliverable D2.3.2 [16]. |

| | |
|---|---|
| **Task** | T2.3 |
| **Partner** | FRH |
| **Feedback** | The system is initially executed in a declaratively specified deployment configuration using the techniques developed in T1.3. The monitors generated in T2.3 adapt the deployment configuration dynamically, for example due to usage peaks of a Service. Thus it is interesting to investigate the relationship between the monitoring service and possible dynamic re-deployment actions. In particular, do the monitors ensure that the evolved system preserves (a certain subset of) the deployment requirements specified initially? Could a monitor be generated automatically that checks at run-time whether the current deployment configuration respects the requirements? |
| **Discussion** | The initial report on deployment (D1.3.1 [7]) focused on static synthesis of the initial deployment configuration. To ensure preservation of deployment requirements at run-time, we follow a correct-by-construction approach based on the new Smartdeployer for dynamic deployment (D1.3.2 [15]).<br>In particular, we annotate the dynamic deployment actions (which are taken based on monitored SLA metrics, SLOs and KPIs) with the relevant high-level deployment requirements and use the Smartdeployer to carry out the actions. This ensures that whenever the system scales, the scaling is done in such a way that the requirements are satisfied. This work is reported in D2.3.2 [16]. |

## 4.2 Other Feedback (Year 3)

The following list collects other feedback from the use of the ABS tool suite which is not part of the formal feedback loop planned in Year 2 (addressed in the previous section). It concerns general aspects of ABS and its tool suite which have not been the research focus of the Envisage project.

- The requirement to use different syntactic forms for blocking method calls dependent on whether sender and receiver are located in the same or different COGs (see D4.4.1 [6]) is a major potential source of run-time errors.
  *This has been done.*

- Earlier versions of ABS required that the results of all function calls should be assigned to variables, which prevented the use of nested function calls.
  *This is now supported.*

- Numerals in ABS are limited to just integer (**Int**) and rational (**Rat**) types. It would be very useful to also include floating point numbers, even if only at the syntactic level – e.g. the floating point syntax "0.012" could easily be (pre-)compiled into the rational form "12/1000".
  *Floating point numbers introduce complications for analysis that go against the focus of Envisage.*

- It would be useful to add syntactic sugar for "for" loops and add list and map subscripts to ABS (e.g. "a[i]" to access the 'i'th element of array 'a') to ABS.
  *Although nice to have, adding this kind of syntactic sugar has not been a priority of the Envisage project to address our research focus.*

- ABS *per se* has only limited support for modularity – in particular: there is no class inheritance (and hence no calls to code in "super" methods). Inherited methods thus need to be re-implemented separately in every class that inherits those methods – leading to code redundancy. The add-on 'delta' language allows such redundant code to be inserted automatically – but at the cost of developing and maintaining the additional delta code. Overall, this can easily give rise to ABS models that are bigger (more verbose) than comparable code written in other OO languages (e.g. Java or Swift).
  *This has not been a main focus of Envisage, but code reuse for ABS has recently been fully implemented and documented by TUD in a collaboration with the University of Torino in the form of traits.*

- The Java Code Generation tools are not distributed as part of the ABS tool chain, and do not work with the latest version of the ABS syntax.
  *Priority has been given to the Haskell back-end.*

- The online (http://ei.abs-models.org:8082/clients/web/) and local (virtual machine) versions of the Collaboratory are not synchronized, and often display functional differences, e.g. giving different results for the same tools, or missing the ABS examples (content) or Erlang simulator statistics.
  *This has been dealt with. The Collaboratory is distributed as Vagrant and Docker images, and we use the same (latest) image for the on-line version. This should keep all instances synchronized.*

- In general, throughout the project, there has been a lack of comprehensive and up-to-date documentation. The documentation that does exist has failed to keep pace with the development of the ABS language and code-base, such that many of the documented details are out-of-date.
  *This is the nature of a research project about developing and evaluating ongoing developed analysis tools. For this reason, the tool developers have encouraged direct communication to support the application of the tools to the industrial cases.*

- In the ENG office environment (Windows PCs with proxy mediated internet access) the deployment of the Collaboratory on a local virtual (Vagrant/VirtualBox) machine proved to be difficult. Indeed, despite a great deal of effort (over several months), we were unable to install the tools in the office.

Installation was only possible working from home (without a proxy) and on employees personal (non-Windows) machines.
*This has been dealt with. To avoid spending time on compatibility issues between operating systems and library versions the tool set has been distributed using Vagrant and Docker images.*

- Overall there is no planned release schedule for the ABS language and code-base, or for any of the Envisage tools. Updates are ad-hoc and there are no formal "version" numbers, hence no common references for benchmarking or test/bug reporting.
  *For version control we have used Github and open source tool development, which allow referencing unique versions of the platform. During the project lifetime, the codebase for the Envisage tool chain has been evolving continuously. There are currently no plans to move towards a formal roadmap, but release numbers have been addressed since the tool integration meeting in Amsterdam (August 2016).*

Chapter 5

# Language Features and Analysis Tools

## 5.1 Language Features

We briefly survey key features of ABS from the perspective of modeling services deployed on the cloud. For more details of the ABS modeling language, we refer to the reference publications (e.g., [20, 22]) and to the language manual (`http://abs-models.org/documentation/manual/`). The description below discusses the main feature categories used in the table.

- *Functional Layer.* ABS is built around a first-order functional language which supports user-defined data-types and functions. The functional language allows abstraction from low-level implementation-oriented data structures in the models. For the sake of modeling time and costs, ABS has been extended with rational numbers as a built-in data-type.
- *Imperative Layer.* The imperative layer of ABS is used for synchronization and communication between concurrent objects executing in parallel. A particular feature of ABS is concurrent object groups (COGs); i.e., the unit of deployment is not a single object but a group of objects. Groups are created by creating local objects inside COGs and allows more complex local data models than isolated concurrent objects.
- *Delta Layer.* ABS supports the modeling of software product lines by linking a feature model to deltas, which are flexible program modifiers. Deltas, originally developed in the HATS project, have to a limited extent been combined with deployment modeling [21] in Envisage, but have to some extent been used in the case studies.
- *Deployment Modeling.* To model the deployment of services on the cloud, ABS supports the modeling abstraction of deployment components which make resources available to COGs. Static deployment models are obtained by creating all deployment components in the main block of the ABS model, whereas dynamic (elastic) deployments are expressed by creating deployment components during the execution of the models. In Envisage, we developed a "CloudProvider API" as an abstraction over basic deployment components which allowed users to configure virtual machine instances in a high-level way (using resource profiles) as well as to account for the accumulated cost during the execution of a deployed model, the start-up time for virtual machines, etc.
- *Resource Modeling.* Several resources were integrated in the semantics of ABS to support the modeling of cost of computations and the capacity of virtual machine instances, including the number of CPU cores, the speed (corresponding to Amazon's notion of Elastic Compute Unit), and the bandwidth. The API to the deployment layer allowed models to query the deployment components with respect their current resource capacity as well as their average load over a given window (a number of time intervals).

Table 5.1 shows the coverage of the ABS features by the industrial case studies.

| **Feature** | ATB | ENG | FRH |
|---|---|---|---|
| *Functional layer* | | ✓ | ✓ |
| User-defined data-types | | ✓ | ✓ |
| Rational Numbers | | ✓ | ✓ |

| **Feature** | ATB | ENG | FRH |
|---|---|---|---|
| *Imperative Layer* | ✓ | ✓ | ✓ |
| Local objects in COGs | | ✓ | ✓ |

| **Feature** | ATB | ENG | FRH |
|---|---|---|---|
| *Delta layer* | | | ✓ |
| Deltas | | | ✓ |
| More than one product in product line | | | |
| Non-trivial feature model | | | |

| **Feature** | ATB | ENG | FRH |
|---|---|---|---|
| *Deployment modeling* | ✓ | ✓ | ✓ |
| Creating deployment components in main block using “new DeploymentComponent” | ✓ | | |
| Creating deployment components in main block using the CloudProvider class | ✓ | | ✓ |
| Creating deployment components dynamically using “new DeploymentComponent” | ✓ | ✓ | |
| Creating deployment components dynamically using the CloudProvider class | ✓ | ✓ | ✓ |

| **Feature** | ATB | ENG | FRH |
|---|---|---|---|
| *Resource modeling* | ✓ | ✓ | ✓ |
| Using static resources (one or more of Cores, Memory) | | ✓ | ✓ |
| Using dynamic resources (one or more of Bandwidth, Speed) | ✓ | ✓ | ✓ |
| Using cost accounting (PaymentInterval, CostPerInterval; or StartupDuration, ShutdownDuration) | | | ✓ |
| “Cost: ” (CPU) annotations using constants | ✓ | ✓ | ✓ |
| “Cost: ” (CPU) annotations using non-constant expressions | ✓ | ✓ | ✓ |
| “DataSize: ” (Bandwidth) annotations using constants | ✓ | | |
| “DataSize: ” (Bandwidth) annotations using non-constant expressions | ✓ | | |
| Using load monitoring (e.g., using the result of “dc.load(...)”) | | | ✓ |

Table 5.1: Usage of ABS language features in the industrial case studies

## 5.2 Tools

This section provides an overview of the tools developed and delivered during the course of Envisage. Table 5.2 shows on which industrial case studies they have been applied. The section concludes with a more detailed discussion of each tool.

| **Tool** | ATB | ENG | FRH |
|---|---|---|---|
| Erlang Simulator | ✓ | ✓ | ✓ |
| Deadlock Analysis - SDA | ✓ | ✓ | ✓ |
| Deadlock Analysis - SACO | ✓ | ✓ | ✓ |
| Resource Analysis - SRA | ⚠ | ⚠ | ⚠ |
| Resource Analysis - SACO |  | ✓ | ✓ |
| Resource Analysis - CoFloCo |  | ✓ | ✓ |
| Monitoring - SAGA |  |  | ✓ |
| Systematic Testing - SYCO | ✓ |  |  |
| Verification - KeY ABS |  |  | ✓ |
| Deployment - Smart Deployer |  |  | ✓ |
| Haskell Backend | ⚠ |  | ✓ |
| Java Backend |  | ⚠ |  |

Table 5.2: Tool usage on industrial cases

### 5.2.1 Erlang Simulator

#### Fundamental Approach

The Erlang simulator is the main tool for simulating and visualizing all aspects of an ABS model, including resource usage and deployment configuration. It relies on the common compiler front-end and type checker, and generates code in the Erlang language which is subsequently compiled and run on the Erlang BEAM VM.

#### Tool Description

The Erlang simulator comprises a compiler back-end to generate Erlang code, compile it and generate an OTP-compliant Erlang application. Included is the embedded cowboy web server[1], which enables data access to a running model. This is used for implementing visualization and the HTTP API, including inspection of objects and deployment components and invoking ABS methods from outside the model.

#### Case Study

The Erlang simulator has been used in all case studies. It supports all features of the ABS language, and is the only back-end that supports all of resource simulation, visualization and a HTTP API. The case studies have served to uncover bugs in the Erlang back-end, and to generate and formulate requirements for further development that has taken place or will take place.

[1]`https://github.com/ninenines/cowboy`

#### Evaluation

The Erlang simulator has surpassed the Maude back-end in usefulness by providing a significant runtime speed-up when simulating ABS, better output facilities, visualization support, and a Model API for input. It has been deployed in all case studies and served for validating functional correctness of the models, gaining insight into model behavior, and as a stepping stone for other tools.

### 5.2.2 Deadlock Analysis - SDA

#### Fundamental Approach

Deadlocks may be particularly insidious to detect in systems where the basic communication operation is asynchronous (e.g., ABS asynchronous method invocation) and the synchronization explicitly occurs when the value is strictly needed (e.g., ABS get operation). In this context, when a thread running within the object group $x$ performs a `get` operation on a thread within object group $y$, then it blocks every other thread that is competing for the lock on $x$. This blocking situation corresponds to a dependency pair $(x, y)$, meaning that the progress on $x$ is possible provided the progress of threads on $y$. A *deadlock* then corresponds to a circular dependency in some configuration, such as a collection of pairs of the form $(x, x_1), (x_1, x_2), ..., (x_n, x)$. The goal of the tool [19] is to infer these dependency sets in a fully automatic way, and then to analyze these sets and detect dependency cycles.

#### Tool Description

There are three key components in this tool. (i) A type system that abstracts concurrency patterns in the instructions, (ii) an inference algorithm that allows to automatically extract the type of the program, (iii) a decision algorithm that analyses the program type and detects deadlock presences. Since deadlock detection is an undecidable problem, some restrictions apply to the tool. In particular, the inference algorithm is the component to which restrictions apply. This means that for every *well-typed* program the tool output is precise, however not every program can be typed. This happens, for example, in presence of recursive data types or boolean `await` operations, in these cases the analysis process can be improved by manually annotating the program.

#### Case Study

The tool has been tested against the case studies from the three major partners. In all three cases there were no deadlocks present. In the **ATB** case study, the tool did not find any of the necessary conditions for a dependency cycle, namely blocking `get` operations nor pure `await` cycles. In the case of the **ENG** case study, the tool was able to run after resolving some minor bugs related to the presence of futures in generic arguments. The **FRH** was thoroughly analyzed in collaboration with the industrial partner, we finally concluded that the tool was able to correctly infer the program behavior in regards to concurrent operations and to successfully verify the deadlock freedom.

#### Evaluation

The tool has been evaluated against a large number of programs from different sources. In addition to the industrial case studies, the tool has been evaluated against several didactic examples created to test specific patterns and against programs inspired in classical problems like variations of the `Philosophers` problem. It is possible to test these examples as well as any other `ABS` program through the **Collaboratory suite**. On top of the practical testing, the correctness of the major theoretical basis of the tool like the inference system and the detection algorithm have been formally demonstrated.

### 5.2.3 Deadlock Analysis - SACO

#### Fundamental Approach

SACO is a static analysis framework that can automatically infer a wide range of properties for ABS models. The deadlock analysis of SACO can automatically certify the absence of deadlocks in an ABS model. The analysis builds a dependency graph such that the absence of cycles in the graph ensures the absence of deadlock in the original ABS model. The analysis uses information from other powerful analyses, such as the *may-happen-in-parallel* analysis, which infers the program points of the model that might execute in parallel, and the *points-to* analysis, which infers the abstract objects that could be created by model and the objects that might be referenced by each variable in the model.

#### Tool Description

The deadlock analysis is part of SACO, an open source static analysis framework written in Prolog. The deadlock analysis takes an ABS model and a series of options as input. It uses the ABS compiler to generate an intermediate representation that will be used in the analysis. The options that the deadlock analysis can receive include: the precision of the points-to analysis and whether the analysis should use the may-happen-in-parallel information or not. As a result, the analysis outputs a message certifying that the ABS model has no deadlocks or a list of possible deadlock cycles that could not be discarded. The deadlock cycles include the objects, methods, and synchronizations points involved in the potential deadlock.

#### Case Study

The deadlock analysis has been applied to the FRH and ENG case studies.

#### Evaluation

SACO has been successfully applied to the FRH and ENG case studies and, for both case studies, it has been able to ensure the absence of deadlocks in the models.

### 5.2.4 Resource Analysis - SRA

#### Fundamental Approach

Cloud computing introduces the concept of elasticity, namely the possibility for virtual machines to scale according to the software needs. In order to support elasticity, cloud providers, including Amazon, Google, and Microsoft Azure, have pricing models that allow one to hire on demand virtual machine instances and paying them for the time they are in use, and have APIs that include instructions for requesting and releasing virtual machine instances dynamically. The theory behind this tool [18] targets precisely this scenario in an attempt to statically estimate the upper bound of the virtual machines necessary to fulfill the requirements of an elastic cloud program, by extracting and analyzing behavioral types for the ABS models.

#### Tool Description

We propose a static analysis technique that computes upper bounds of virtual machine usage in a concurrent language with explicit acquire and release operations of virtual machines. In our language (a dialect of **ABS**) it is possible to delegate other (ad-hoc or third party) concurrent code to release virtual machines (by passing them as arguments of invocations). Our technique is modular and consists of (i) a type system associating programs with behavioral types that records relevant information for resource usage (creations, releases, and concurrent operations), (ii) a translation function that takes behavioral types and return cost equations, and (iii) an automatic out-the-shelf solver for the cost equations (in our case the solvers used, `CoFloCo` and `PUBS`, are also part of the **Collaboratory suite**).

#### Case Study

Given the current state of the tool, it is not possible yet to fully analyze **ABS** programs. For the moment it has been only possible to manually test some patterns present in the industrial case studies. It is important to remark that current case studies do not take full advantage of the dynamic releasing of virtual machines, making the programs have cumulative resource consumption which is a less difficult problem that can be targeted almost directly by cost equations solvers.

#### Evaluation

This tool is the first (to the best of our knowledge) static analysis technique that computes upper bound of virtual machines usages in *concurrent* programs that may *acquire* and, more importantly, may *release* such machines. We have compared the results of the tool against others that performs similar analysis either for concurrent programs or for programs with dynamic releasing obtaining in both cases very good results. More details on the evaluation as well as on the tool description may be found in the tool paper [18] and in the **Collaboratory suite**.

### 5.2.5 Resource Analysis - SACO

#### Fundamental Approach

SACO is a static analysis framework that can automatically infer a wide range of properties for ABS models. The resource analysis of SACO can obtain sound upper bounds on the resource consumption of functions, methods or complete ABS models. Time (evaluation steps), memory, bandwidth are some of the resources that can be considered by SACO. SACO also implements more advanced notions of cost such as parallel cost i.e. the time required to execute an ABS model taking into account the parallelism or cost centers i.e. consider the cost of each distributed component separately.

#### Tool Description

SACO is an open source framework written in Prolog. SACO takes an ABS model and a series of options as input. It uses the ABS compiler to generate an intermediate representation that will be used in the analysis. These options that SACO can receive include: the names of the functions or methods to be analyzed, the type of resource that will be considered (time, memory, etc.) and the specific analysis that will be performed (regular cost analysis, peak cost, parallel cost, etc.). Once selected these options, the analysis is completely automatic. As a result, SACO generates symbolic expressions in terms of the input parameters that represent the upper bounds of the analyzed functions or methods. Optionally, users can add annotations to the ABS model to help the analysis.

#### Case Study

SACO has been used in the FRH and ENG case studies. It has been applied to obtain the complexity of the main methods of the corresponding ABS models. We have also applied SACO to the FRH case study to get all possible states of the queues of tasks for each object. We also used SACO (in combination with CoFloCo) to obtain upper bounds on the time and memory resource consumption of the generated monitors for the FRH case study.

#### Evaluation

SACO has been successfully applied to the FRH and ENG case studies. In combination with CoFloCo it has been possible to obtain upper bounds for the main methods in the genetic algorithm that comprises the ENG case study. We have also successfully used SACO to bound the cost of some relevant methods of the FRH case study and, additionally, we have also obtained the cost of some relevant program points, and the abstract objects responsible of executing these program points.

### 5.2.6 Resource Analysis - CoFloCo

#### Fundamental Approach

CoFloCo is a static analysis tool to infer automatically symbolic complexity upper and lower bounds of programs. CoFloCo's analysis is not bound to any specific programming language, instead it takes an abstract representation of programs as an input. The abstract representation is a set of cost relations (similar to recurrence relations) that can be generated from ABS models using SACO. CoFloCo uses polyhedral analysis and linear programming to obtain sound and precise bounds of the cost relations.

#### Tool Description

CoFloCo is an open source publicly available tool written in Prolog. It has been designed to be used as a back-end for other resource analysis frameworks and it has been integrated in SACO. SACO provides an option to use CoFloCo as a back-end solver. This enables the users to apply all the SACO analyses with additional precision and inference power provided by CoFloCo.

#### Case Study

CoFloCo has been used together with SACO to obtain time upper bounds of the ENG case study. This case study makes use of complex data structures. These data structures can be precisely abstracted by SACO but the resulting cost relations are too complex to be solved by the default SACO cost relation solver (PUBS). With the help of CoFloCo, we succeed at analyzing most of the challenging methods in the ENG case study model. Furthermore, we successfully analyzed the resource consumption of generated monitors for the FRH case study.

#### Evaluation

CoFloCo has been successfully used to obtain upper bounds in the ENG case study. Its power has proved to be essential to analyze some of the most challenging methods in the ENG case study. CoFloCo represents a good alternative to PUBS (SACO's default cost relation solver) for complex pieces of code.

### 5.2.7 Monitoring - SAGA

#### Fundamental Approach

SAGA is a tool that generates executable ABS monitoring add-ons for SLA, SLO and KPI metrics. The approach is event-based: the events are interactions with endpoints from Service APIs (a common example is a call or return of a resource method in a REST API). A service metric function is formalized as an attribute from a declarative attribute grammar, which maps an event traces to a value that indicates the QoS level.

#### Tool Description

SAGA requires two inputs: a communication view that names the events relevant for the metric to be formalized, and an attribute grammar where the value of the metric is defined. SAGA then fully automatically generates ABS code for a parser (an automaton) corresponding to the grammar. An Eclipse-based syntax highlighting module for views and grammars is available. SAGA is implemented as a meta-program in the language Rascal [23].

#### Case Study

Using SAGA we formalized metrics such as Availability and Service Degradation (the percentage of queries with response time between 200-500ms, and the percentage slower than 500ms, minus the 2% slowest queries) from the Fredhopper Cloud Services. The corresponding monitors are executable with all back-ends and simulators supporting core ABS. We also investigated correctness of the generated monitors with KeY-ABS.

#### Evaluation

SAGA supports *regular* attribute grammars (all productions have the form $S ::= a\ T$, where $a$ is a terminal and $S, T$ are non-terminals). This allowed generation of resource-efficient, fast monitors: memory usage is constant and the computation time to calculate the value of the metric is linear in the length of the event trace. Since the generated monitoring code by SAGA is ordinary ABS, the monitors are amenable to all Envisage analyses. This enabled for example to successfully verify monitor correctness with KeY ABS.

### 5.2.8 Systematic Testing - SYCO

#### Fundamental Approach

The SYCO tool is a dynamic testing tool for ABS concurrent objects which systematically explores all relevant combinations of task interleavings. It includes state-of-the-art partial-order-reduction (POR) techniques in order to detect redundant interleaving combinations dynamically during the execution avoiding a considerable number of redundant explorations.

#### Tool Description

The user interacts with SYCO through its web interface which is integrated within the ABS collaboratory. Essentially, it receives an ABS program, and, as a result, it outputs a set of executions. For each one, it shows the output state and the sequence of tasks/interleavings and concrete instructions of the execution (highlighting the source code). SYCO also generates sequence diagrams for each execution which provide graphical and more comprehensive representations of execution traces.

#### Case Study

SYCO has been successfully applied to perform systematic testing of the core map-reduce algorithm of the ATB case-study. The application of SYCO's systematic testing on the case-study with a concrete input allow observing the output state and trace for each different combination of assignments of map and reduce tasks to worker objects.

#### Evaluation

We have experimentally evaluated the impact of the POR techniques available in SYCO using the ATB case-study as benchmark. The reductions both in the number of explored executions and exploration time due to the use of POR techniques are huge (up to three orders of magnitude). This demonstrates the effectiveness of the POR techniques in SYCO and specially that of the most recent and accurate technique based on exact dependencies.

### 5.2.9 Verification - KeY ABS

#### Fundamental Approach

KeY ABS is an interactive (semi-automatic) deductive verification tool that enables one to verify functional correctness properties for concurrent and distributed ABS models. The specification is provided in terms of history-based class invariants. It is then proven that the ABS model adheres to its specification. The approach is symbolic and models the ABS language faithfully, i.e., it does not apply any abstraction and remains fully precise. The specification language is first-order (dynamic) logic and thus highly expressive.

#### Tool Description

KeY ABS is an open source software written in Java. It is a variant of the KeY verification system for Java. As input it requires the ABS model to be verified as well as its specification, or alternatively, a problem file with the formula to be proven. The specification is provided as a separate file located in the same directory as the ABS files. The user can then start the verification for each method separately. The proof-obligation to be proven is then loaded into the theorem prover of KeY ABS. The verification process is semi-automated, i.e., KeY ABS provides powerful strategies for automation but requires sometimes user interaction.

#### Case Study

KeY ABS was used on the FRH case study to verify the correctness of generated monitors. This means it could be proven that the monitors react on critical situations *and* that they act to divert them by allocating (or deallocating) computing resources. In addition KeY ABS was used to verify Network-on-Chip (NoC). This case study presents an approach to scalable verification of unbounded concurrent and distributed systems. Formal proofs of global properties such as "no packets are lost" and "a packet is never sent in a circle" are obtained by KeY ABS regardless of the number of routers and packets on a NoC.

#### Evaluation

KeY ABS was successfully applied to the case studies mentioned above and the desired properties could be proven. The achieved guarantees are strong and require non-trivial reasoning about state properties as well as protocols. Non-trivial interaction was required for the NoC case study.

### 5.2.10 Deployment - Smart Deployer

#### Fundamental Approach

In order to take into account deployment issues already at the early stages of development, deployment specific primitives can be added as a first-class citizen in ABS. Smart Deployer allow this by following a declarative approach: programmers can specify deployment constraints and a solver synthesizes ABS classes exposing methods like deploy (or undeploy) that execute (or cancel) configuration actions changing the current deployment towards a new one satisfying the programmer's desiderata.

#### Tool Description

SmartDepl is an open source tool written in Python. SmartDepl first processes the original ABS program to retrieve relevant cost annotation and deploy annotations defined in an ad-hoc domain specific declarative language. For each annotations, Smart Deployer relies on the Zephyrus2 configuration optimizer to concretely compute the objects that need to be deploy and then it generates a new ABS class that specifies the deployment steps to reach the desired target. This class can be used to trigger the execution of the deployment, and to undo it in case the system needs to downscale directly from the ABS code.

#### Case Study

The ABS annotations introduced by Smart Deployer have been driven by Fredhopper Cloud Services. The Fredhopper Cloud Services offer search and targeting facilities on a large product database to e-Commerce companies. Depending on the specific profile of an e-Commerce company, FRH has to decide the most appropriate customized deployment of the service that can be automatically devise by using Smart Deployer.

#### Evaluation

We evaluated Smart Deployer on the Fredhopper Cloud Services. By adopting Smart Deployer, we have been able to realize a new modeling of the Fredhopper Cloud Services in which both the initial deployment and the subsequent up- and down-scale is expected to be executed automatically. This also allow us to validate that the current manually defined solution used in production by FRH is optimal (i.e., the cheapest possible one given the user desiderata).

### 5.2.11 Haskell Backend

#### Fundamental Approach

The Haskell back-end is an ABS code generation tool, which focuses on execution speed and support for all standard ABS features. The choice of Haskell was made, firstly, because the languages (ABS and Haskell) share the same purely-functional programming paradigm at their core, and secondly because of the Haskell's offerings in concurrency, parallelism and distributed computing.

#### Tool Description

The tool translates ABS code to Haskell and subsequently to native code, which is then linked to our custom-developed Haskell library to provide the concurrency of an ABS runtime. Compared to other back-ends (Erlang, Maude) the Haskell back-end relies on an external Haskell type-checker and Garbage Collector. Besides the standard ABS language, the tool implements the Cloud extension, for real deployment and distributed execution of ABS models in the Cloud, as well as the HTTP-API extension, for interacting with running ABS code from the outside. The Haskell back-end is open-source and utilizes BNFC, a Haskell parser generator, later also adopted by the Java back-end.

#### Case Study

The Haskell back-end has been applied to large parts of the FRH case study. Compared to the Erlang simulator, the notion of time in the Haskell back-end refers not to the abstract simulated time but instead to the real (wall-clock) time. A future consideration for the FRH case study is to utilize also the Cloud extension for deploying actual Cloud machines so as to gain better insight/confidence of the ABS model's behavior. The Haskell back-end has been used for the implementation of a distributed Preferential-Attachment algorithm, which is a mechanism for generating large social-network-like graphs.

#### Evaluation

The Haskell back-end has been tested and benchmarked on smaller ABS models. A recent benchmark comparison[2] of ABS back-ends shows that the Haskell back-end generates the fastest overall code in terms of execution performance and memory.

### 5.2.12 Java Backend

#### Fundamental Approach

The ABS Java Code Generation Back-end is developed for running ABS programs in a local or distributed environment by compiling ABS source into the Java language. The compiled code uses a Java library called the ABS-API library that provides support for the actor programming model, cooperative scheduling, distributed communication and remote deployment. The tool is tailored towards case studies that require real computing resources, memory and bandwidth.

#### Tool Description

The Java Code Generation Back-end is an open source application written in Java with using an grammar description written in BNFC. It is made up of three separate components:

- BNFC grammar parser.
- The ABS to Java compiler.
- The ABS-API execution library.

[2]`http://abstools.github.io/abs-bench`

The tool receives as input one or more ABS source files and generates separate Java packages for each of them containing the corresponding classes translated from ABS to Java including a Main class corresponding to the main ABS block. These packages then need to then be integrated together with the ABS-API library in a new Java program and run using the generated main method.

### Case Study

The tool is currently tested on the well-known Mapreduce application that is modeled in ABS which tests the cooperative scheduling feature, as well as parallel and distributed computing. The tool is being applied to the Engineering Case Study to further test its support for Real-Time applications and remote deployment and scalability. It is not yet able to handle the full complexity of the Engineering case study, but we expect to address this complexity in the near future.

### Evaluation

We evaluated the tool especially performance-wise, in terms of execution time, memory usage, and possible overheads introduced with the ABS-API library's support for ABS features. The evaluation focused on minimizing these overheads that are introduced by the following:

- Large number of heavyweight Java Threads created by cooperative scheduling.
- Data structures used of distributed future control and propagation.
- Using Java reflection for asynchronous method invocation on remote objects.

# Chapter 6

# Conclusion

## 6.1 Overall Coverage of Overall Project Objectives

Chapter 3 has explained how all case studies successfully covered aspects of the overall project objectives.

- **Objective O1: Foundations of Computation with Virtualized Resources.**
  The case studies have systematically explored different aspects of virtualized architectures, including both static and dynamic resource provisioning on virtual machine instances with different resources such as, e.g., cores, processing speed, and bandwidth.

- **Objective O2: Behavioral Specification Language for Virtualized Resources.**
  All case studies have been developed using the ABS modeling language, and have covered all aspects of the modeling language that were the focus of the Envisage project, including deployment modeling with different kinds of resources. Section 6.2 below discusses the usage of the modeling constructs in more detail. All case studies made use of the executable models in ABS and the simulation tool. Overall, the case studies covered all the main modeling abstractions and demonstrated their usefulness.

- **Objective O3: Design-by-Contract Methodology for Service Contracts.**
  The analysis of Service Contracts was crucial for all three case studies, albeit in complementary ways. For ATB, the main concern was related to throughput: the time from data arrived to the system until it could be leveraged to the handsets. For ENG, the main concern was to efficiently make acceptable SLAs and the business-level concerns of a cloud provider. For FRH, the main concern was the relationship between low-level (cloud-side) service contracts and the high-level (customer-side) service contracts.

- **Objective O4: Model Conformance Demonstrator**. The case studies used different approaches to ensure conformance between the ABS models and the actual, modeled systems. Whereas ENG experimented with code generation as a way to bridge the gap between models and code, FRH focused on recreating the APIs of their production system in the ABS model, together with the resource profiles of the virtual machines and Services used in production. Furthermore, FRH plugged in real-world data from the production system through the HTTP API. ATB carried out a careful validation of the throughput in the model by comparisons with the real system.

- **Objective O5: Model Analysis Demonstrator.** All case studies made comprehensive use of analysis tools to assess the non-functional properties of the ABS models. Section 6.3 below discusses the usage of the modeling constructs in more detail. Overall, the case studies covered all the main analysis techniques and demonstrated their usefulness.

## 6.2 Modeling Capabilities

For the modeling of virtualized services, ABS provides essential features which are not currently available in other modeling languages. Of particular importance is the support for *deployment modeling* and for *resource* *modeling*. The relevance and usefulness of these features were clearly demonstrated in all case studies. The non-functional aspects of the case studies could not, to the best of our knowledge, be modeled in any other modeling language available today.

In addition, the separation between a functional layer, an imperative layer, and a deployment layer proved very useful. For the development of the case studies, more conventional support for structuring could have been useful. ABS proposes delta-modeling as a way to define a family of models, but case study developers would have liked support for reuse inside the models. This was not given priority within the scope of Envisage, but we observe that support for code reuse in the form of traits has been integrated in the ABS tool chain recently in a collaboration between TUD and the University of Torino.

Table 5.1 (page 37) provides a detailed overview of the language features explored in the three case studies. As we see, all important features of the resource modeling and the deployment modeling parts of ABS were explored in the case studies.

## 6.3 Analysis Capabilities

Exploring the foundations of virtualized computation (Objective O1), ABS has a formally defined semantics amenable to a range of analysis techniques. The carefully selected feature combinations of ABS enable compositional analysis based on the concurrent object abstraction, which was crucial for many of the analysis techniques developed in the project. The suite of available analysis tools for ABS is unique, and particularly well-suited for virtualized services as targeted by Envisage, including the smart deployer, deadlock analysis, cost analysis, and simulation with visualization of deployment component loads and service metrics.

The coverage of the analysis techniques by the case studies is shown in Table 5.2 (page 38). Unsurprisingly, we see that whereas the fully automated tools for simulation and deadlock analysis were successfully used in all case studies, the tools which required more user interaction (or were developed late in the project lifetime) were not fully covered by all case studies. The selection of tools for each particular case study also reflects the identified needs of the stakeholders for that case study. As the table shows, all tools were successfully applied to at least one case study, guaranteeing that they were evaluated. Section 5.2 summarizes the evaluation process for all the tools.

## 6.4 Integration in Working Practices

We investigated in D5.3 [4] how the approach pursued in Envisage relates to, and can be integrated in industrial working practices. One of the main software development processes used in Cloud computing is the DevOps methodology. Figure 6.1 shows a diagram for a DevOps work-flow. DevOps emphasizes

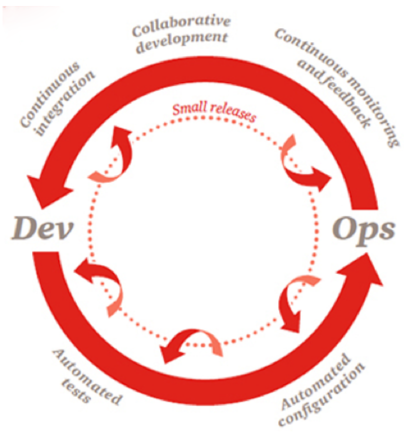

Figure 6.1: A DevOps work-flow connects development and operations in a continuous iterative process (illustration source: Gene Kim, HP, and PwC, 2013).

collaboration between software developers, operations personnel and quality assurance teams, recognizing interdependencies between software design, quality of service (QoS) and quality assurance. This is achieved by a recurring flow of rapid releases facilitated by automated configuration and continuous monitoring and formal analyses (for example, testing). Envisage puts the above DevOps work-flow on a rigorous basis as follows:

- **Automated Configuration**: Envisage allows configuration and deployment choices already at *the modeling level* for even very abstract models, and can automatically synthesize deployment configurations from high-level requirements. This allows early exploration and analysis of different alternative deployments, thereby supporting operations and development teams.
- **Continuous monitoring and feedback** is lifted by Envisage from low-level metrics to high-level metrics directly related to fully formalized SLAs and KPIs. Feedback is given in two directions: non-intrusively by visualizing and querying monitored data, or more intrusive, by automatic execution of corrective actions (such as auto-scaling while respecting high-level deployment requirements) based on the monitored data.
- The Envisage approach was shown to be amenable to **Continuous Integration** by the adoption of Jenkins, a widely used continuous integration system. This approach was successfully applied on the ATB case study.
- **Collaborative development** is supported by Envisage through the novel ABS collaboratory, and integration into widely used existing IDEs (Eclipse and Emacs plug-ins).
- **Formal Analyses**. Envisage offers a wide range of powerful tool-supported (semi-) automated analyses, including automated tests: in addition to testing, there is support for functional verification, resource analysis and deadlock freedom.

## 6.5 Follow-up on Industrial Case Studies

The industry partners of Envisage will make use of the lessons learned in the project after the project period. Whereas the concrete exploitation planes are discussed in D5.8 [17], we here briefly summarize the main messages retained by the industry partners from their participation in the Envisage project and the work on their case studies.

- **ATB.** Bugs we experience are typically related to resource overuse (e.g. out-of-memory) and concurrency (in particular within mobile apps). The use of model-based simulation, deadlock analysis, and resource analysis in ABS directly addresses these issues. Using ABS for modeling critical concurrency-related code will make us more productive and provide higher quality to our customers. ATB is already testing out ABS to develop scalable cloud solutions in collaboration with UIO.
- **FRH.** At FRH, we will continue working with a model-based approach using ABS and the SAGA monitoring system developed in Envisage to further automate the deployment of Fredhopper Cloud Services and in particular to the automated generation of monitors for service metrics. We hope to gradually move this work from research to production.
- **ENG.** Improving SLA management through formalization, analysis and test techniques as done in the Envisage project, will enable ENG to better satisfy customers' demand and to mitigate the risk of SLA violation. This directly impacts on reducing penalty costs from ENG cloud services. Indirectly it also impacts the quality of service perceived by ENG customers, empowering ENG to attract more customers.

# Bibliography

[1] Initial Modeling of the ATB Case Study, July 2014. Deliverable D4.2.1 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[2] Initial Modeling of the FRH Case Study, July 2014. Deliverable D4.3.1 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[3] Initial User Requirements, January 2014. Deliverable D4.1 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[4] Envisage Work Flow, September 2015. Deliverable D5.3 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[5] Formalization of Service Contracts and SLAs (Initial Report), March 2015. Deliverable D2.2.1 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[6] Initial Modeling of the ENG Case Study (Resubmitted), January 2015. Deliverable D4.4.1 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[7] Modeling of Deployment (Initial Report), March 2015. Deliverable D1.3.1 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[8] Monitoring Add-Ons and Visualization (Initial Report), September 2015. Deliverable D2.3.1 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[9] Resource-aware Modeling of the ATB Case Study, July 2015. Deliverable D4.2.2 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[10] Resource-aware Modeling of the ENG Case Study, July 2015. Deliverable D4.4.2 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[11] Resource-aware Modeling of the FRH Case Study, July 2015. Deliverable D4.3.2 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[12] Assurance of the ATB Case Study, September 2016. Deliverable D4.2.3 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[13] Assurance of the FRH Case Study, September 2016. Deliverable D4.3.3 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[14] Formalization of Service Contracts and SLAs (Final Report), March 2016. Deliverable D2.2.2 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[15] Modeling of Deployment (Final Report), March 2016. Deliverable D1.3.2 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[16] Monitoring Add-Ons and Visualization (Final Report), September 2016. Deliverable D2.3.2 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[17] Standardization Activities & Final Exploitation Plan, September 2016. Deliverable D5.8 of project FP7-610582 (ENVISAGE), available at `http://www.envisage-project.eu`.

[18] Abel Garcia, Cosimo Laneve, and Michael Lienhardt. Static analysis of cloud elasticity. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming, Siena, Italy, July 14-16, 2015*, pages 125–136. ACM, 2015.

[19] Elena Giachino, Cosimo Laneve, and Michael Lienhardt. A framework for deadlock detection in core ABS. *Software and System Modeling*, 15(4):1013–1048, 2016.

[20] Einar Broch Johnsen, Reiner Hähnle, Jan Schäfer, Rudolf Schlatte, and Martin Steffen. ABS: A core language for abstract behavioral specification. In Bernhard Aichernig, Frank S. de Boer, and Marcello M. Bonsangue, editors, *Proc. 9th International Symposium on Formal Methods for Components and Objects (FMCO 2010)*, volume 6957 of *Lecture Notes in Computer Science*, pages 142–164. Springer-Verlag, 2011.

[21] Einar Broch Johnsen, Rudolf Schlatte, and S. Lizeth Tapia Tarifa. Deployment variability in delta-oriented models. In Tiziana Margaria and Bernhard Steffen, editors, *6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'14)*, volume 8803 of *Lecture Notes in Computer Science*, pages 286–301. Springer-Verlag, 2014.

[22] Einar Broch Johnsen, Rudolf Schlatte, and S. Lizeth Tapia Tarifa. Integrating deployment architectures and resource consumption in timed object-oriented models. *Journal of Logical and Algebraic Methods in Programming*, 84(1):67–91, 2015.

[23] Paul Klint, Tijs van der Storm, and Jurgen J. Vinju. EASY meta-programming with rascal. In *Generative and Transformational Techniques in Software Engineering III - International Summer School, GTTSE 2009, Braga, Portugal, July 6-11, 2009. Revised Papers*, pages 222–289, 2009.

[24] Jia-Chun Lin, Ingrid Chieh Yu, Einar Broch Johnsen, and Ming-Chang Lee. ABS-YARN: A formal framework for modeling Hadoop YARN clusters. In Perdita Stevens and Andrzej Wasowski, editors, *19th International Conference on Fundamental Approaches to Software Engineering (FASE 2016)*, volume 9633 of *Lecture Notes in Computer Science*, pages 49–65. Springer-Verlag, 2016.

# Glossary

## Terms and Abbreviations

**Atbrox Cloud and App Services** A set of services and a mobile app managed by ATB through cloud computing that allows offline search for mobile customers through a subscription service.

**Continuous Deployment** is a software engineering practice that enables software products to be released to production at any time. This practice extends continuous integration with the constant feedback loop from production environments.

**Continuous Integration** is a software engineering practice of merging all developer working copies with a shared mainline several times a day. This practice is usually adopted in combination with automated testing.

**DevOps** DevOps is a software development method that stresses communication, collaboration and integration between software developers and IT professionals. DevOps is a response to the interdependence of software development and IT operations. It aims to help an organization rapidly produce software products and services.

**Enterprise Resource Planning** A business management software - usually a suite of applications - that a company can use to store and manage data from every stage of business including: product planning, cost & development; manufacturing; marketing & sales; inventory management; shipping & payment.

**ERP** Enterprise Resource Planning

**Fredhopper Cloud Services** A set of services managed by FRH through cloud computing that allows the offering of search and targeting facilities on a large product database to e-Commerce companies.

**Hadoop** Apache Hadoop is an open-source software framework for storage and large scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users. It is licensed under the Apache License 2.0. Source: Wikipedia, `http://en.wikipedia.org/wiki/Hadoop`.

**IaaS** Infrastructure as a Service

**Information Technology** The application of computers and telecommunications equipment to store, retrieve, transmit and manipulate data, often in the context of a business or other enterprise. The term is commonly used as a synonym for computers and computer networks, but it also encompasses other information distribution techniques such as television and telephones.

**Infrastructure as a Service** A provision model in which an organisation outsources the equipment used to support IT operations, including storage, hardware, servers and networking components. The service provider owns the equipment and is responsible for housing, running and maintaining it. The client typically pays on a per-use basis.

**IT** Information Technology

**IT Outsourcing** The practice of seeking resources, or subcontracting, outside of an organisational structure for all or part of an IT function (including: infrastructure, software development, maintenance & support).

**Key Performance Indicator** A performance measurement to assess the success of an organization or of a particular activity.

**KPI** Key Performance Indicator

**MapReduce** MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.

**QoS** Quality of Service

**Quality of Service** Generic term encapsulating all the non-functional aspects of a service delivery.

**PaaS** Platform as a Service

**Platform as a Service** A category of cloud service offerings that facilitates the deployment of applications without the cost and complexity of buying and managing the underlying hardware and software and provisioning hosting capabilities.

**Private Cloud** A cloud infrastructure operated solely for a single organisation, whether managed internally or by a third-party and hosted internally or externally.

**REST** Representational state transfer

**Representational state transfer** An architectural style for distributed hypermedia systems.

**SaaS** Software as a Service

**Service Level Agreement** A legal contract between a service provider and his customer. It records a common understanding about services, priorities, responsibilities, guarantees, and warranties.

**Service Level Objective** A measurable characteristics of an SLA, such as Availability or Response Time.

**Service Provisioning** The process of preparing, equipping and/or configuring a service delivery system in order to make it ready to deliver services.

**SLA** Service Level Agreement

**SLO** Service Level Objective

**Software as a Service** A software delivery model in which software and associated data are centrally hosted on the cloud. SaaS is typically accessed by users using a thin client via a web browser.

**Sprite Sheet** Sprite Sheet is a way of combining several images as a matrix of images into one file. The purpose is to reduce the number of files, and in some cases also to improve the compression of images. See `http://en.wikipedia.org/wiki/Sprite_(computer_graphics)` for more about sprite sheets.