Project Deliverable D7.3

Demonstrator Results Documentation – initial version

Project name: Q-ImPrESS
Contract number: FP7-215013
Project deliverable: D7.3 Demonstrator Results Documentation
Author(s): Heiko Koziolek, Jens Doppelhamer, Roland Weiss, Carlos Bilich, Bastian Schlich, Ivan Skuliber, Marijan Zemljić, Saša Dešić, Darko Huljenić

Work package: WP7
Work package leader: ABB
Planned delivery date: M28
Delivery date: 20/07/2010
Last change: 09/07/2010
Version number: 0.32

Abstract
This document describes the results of the industrial demonstrators for the Q-ImPrESS project. Two industrial demonstrators are provided by ABB and Ericsson, while results of the evaluation of the Q-ImPrESS Enterprise SOA showcase are provided by Itemis. The ABB demonstrator comes from the industrial automation domain and evaluates the reverse engineering, performance prediction, and reliability prediction methods and tools of Q-ImPrESS. The Ericsson demonstrator comes from the telecommunication domain and evaluates the performance and reliability prediction tools of Q-ImPrESS. Finally, the Enterprise SOA showcase focuses on performance prediction. This document reports on the prediction accuracy of the methods and the efforts for data collection and modelling.
Revision history

<table>
<thead>
<tr>
<th>Version</th>
<th>Change date</th>
<th>Author(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>10/02/2010</td>
<td>H.Koziolek</td>
<td>Initial Version.</td>
</tr>
<tr>
<td>0.2</td>
<td>28/05/2010</td>
<td>I. Skuliber</td>
<td>Initial ENT validation information</td>
</tr>
<tr>
<td>0.3</td>
<td>31/05/2010</td>
<td>H.Koziolek</td>
<td>Intermediate version for partner review.</td>
</tr>
<tr>
<td>0.3</td>
<td>07/07/2010</td>
<td>M. Trifu</td>
<td>Review</td>
</tr>
<tr>
<td>0.31</td>
<td>09/07/2010</td>
<td>M. Zemljic, I. Skuliber</td>
<td>Corrections according to M. Trifu's review comments</td>
</tr>
<tr>
<td>0.32</td>
<td>09/07/2010</td>
<td>H.Koziolek</td>
<td>Incorporated M. Trifu’s review comments.</td>
</tr>
</tbody>
</table>
Table of contents

1 Introduction .........................................................................................................................4
   1.1 Purpose ..........................................................................................................................4
   1.2 Scope ................................................................................................................................4
   1.3 Structure ..........................................................................................................................4

2 Terminology & Glossary ....................................................................................................5

3 General Findings ................................................................................................................6

4 ABB Evaluation ..................................................................................................................7
   4.1 Introduction and Setting .................................................................................................7
   4.2 Reverse Engineering the SAM ......................................................................................8
      4.2.1 The Q-ImPrESS Toolchain: SISSy and SoMoX ......................................................8
      4.2.2 Browsing Large SAM Repositories .......................................................................9
      4.2.3 Reverse Engineering Results ..............................................................................12
      4.2.4 From Reverse Engineered SAM Instance to Prediction Model ............................13
      4.2.5 Effort estimations .................................................................................................14
      4.2.6 Reverse Engineering Metrics Validation ............................................................15
   4.3 Performance Prediction ................................................................................................16
      4.3.1 Data Collection ....................................................................................................16
      4.3.2 Prediction Results ...............................................................................................18
      4.3.3 Performance Prediction Metrics Validation ........................................................19
   4.4 Reliability Prediction ....................................................................................................20
      4.4.1 Overview ...............................................................................................................20
      4.4.2 Subsystem Failure Rate Estimation .....................................................................21
      4.4.3 Subsystem Transition Probabilities .....................................................................29
      4.4.4 Reliability Prediction Results and Sensitivity Analysis .......................................32
      4.4.5 Reliability Metrics Validation .............................................................................34

5 ENT evaluation .................................................................................................................36
   5.1 Introduction and setting ...............................................................................................36
      5.1.1 Brief introduction to the demonstrator ................................................................36
      5.1.2 Validation setting ..................................................................................................37
   5.2 Manual creation of software architecture models .......................................................39
      5.2.1 Black-box component measurements ...................................................................40
      5.2.2 Black-box component modelling ........................................................................41
   5.3 Performance prediction ...............................................................................................43
      5.3.1 Modelling and performance prediction of evolution scenarios ............................43
   5.4 Reliability prediction ..................................................................................................43

6 Conclusion .........................................................................................................................62

7 Bibliography .....................................................................................................................63
1 Introduction

1.1 Purpose
This report documents the experiment execution and results obtained by applying the Q-ImPrESS method and tools to the industrial demonstrators. These results include the lessons learned, an analysis of the quality of predictions and the impact on design decisions, problems and difficulties in applying the Q-ImPrESS method in a real development project.

1.2 Scope
This document focuses on the evaluation of the Q-ImPreSS method. It describes the experiments execution and results from the demonstrator systems. It neither includes details about the Q-ImPrESS method (cf. D6.1) nor do the demonstrator systems (cf. D7.1).

1.3 Structure
The remainder of this document is organized as follows: Section 2 contains a glossary, defining the most important terms and their meanings in this document. Section 3 describes general findings which are not specific for one of the domains of the industrial partners. Section 4 describes the results of the ABB demonstrator. Section 5 describes the results of the ENT demonstrator. Section 6 describes the results of the ITE showcase. Section 7 concludes the document.
## Terminology & Glossary

The following terms will be used throughout the document:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black-box component</td>
<td>A component having only interface information. Black-box components cannot be analysed using static analysis tools, because they have no information about their internals, however, black-box components can be monitored. In order to apply static analysis, a black-box component needs to be turned into a grey-box component.</td>
</tr>
<tr>
<td>Component</td>
<td>A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition. (Szyperski)</td>
</tr>
<tr>
<td>Connector</td>
<td>Connectors are architectural building blocks used to model interactions among components and rules that govern those interactions. (Medvidovic)</td>
</tr>
<tr>
<td>G-AST</td>
<td>Generalized Abstract Syntax Tree. An abstract syntax tree with resolved dependencies (e.g., pointers at each occurrence of a variable to its definition)</td>
</tr>
<tr>
<td>Grey-box component</td>
<td>A component having interface information and additional model information needed for analysis (QoS annotations + behavioural aspects in case of composite components)</td>
</tr>
<tr>
<td>Non-functional requirement</td>
<td>A requirement that specifies criteria that can be used to judge the quality of the operation of a system, rather than specific behaviour.</td>
</tr>
<tr>
<td>SAM</td>
<td>Service Architecture Model, an instance of SAMM</td>
</tr>
<tr>
<td>SAMM</td>
<td>Common meta-model which contains everything to describe the information needed for quality prediction analysis. Serves as shared data-repository for all quality analysis methods.</td>
</tr>
<tr>
<td>Service</td>
<td>A deployed component, i.e. a runtime entity providing functionality to users or other services.</td>
</tr>
<tr>
<td>Service architecture</td>
<td>A set of services connected to each other via connectors. Subject to analysis.</td>
</tr>
<tr>
<td>SISSy</td>
<td>Structural Investigation of Software Systems. SISSy is a platform for problem pattern identification in OO source code written in Java, C++, or Delphi. SISSy produces a G-AST.</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>Static Structure</td>
<td>Description of component and connector composition that captures the architecture of a service at the time of deployment</td>
</tr>
<tr>
<td>White-box component</td>
<td>A component with available source code</td>
</tr>
</tbody>
</table>
3 General Findings

This section will be completed in the final version of this document.
4 ABB Evaluation
This section contains the results from applying the Q-ImPrESS method on the ABB demonstrator. Section 4.1 briefly describes the ABB demonstrator and gives an introduction into the evaluation of the method. Section 4.2 describes the reverse engineering activities performed on the demonstrator. Section 4.3 shows how the performance predictions for different evolution scenarios were executed. Section 4.4 details on the reliability prediction carried out on the ABB demonstrator.

4.1 Introduction and Setting
This section will contain a brief summary of D7.1, some specifics on the demonstrator implementation and set-up.
4.2 Reverse Engineering the SAM

The reverse engineering applied on the ABB demonstrator is embedded into the overall Q-ImPrESS process as depicted in Figure 1 (please note the reverse engineering activities highlighted with red frames in the figure). The reverse engineering tools SISSy and SoMoX process the ABB C++ code and produce a SAM repository, which needs to be manually enhanced with dynamic behaviour and quality annotations to enable automatic analysis and simulation.

This section describes the application of the reverse engineering method and tools of Q-ImPrESS on the ABB demonstrator. Section 4.2.1 recalls the tool chain and explains the configuration of SoMoX. Section describes a browser plugin for large repositories implemented by ABB. Section 4.2.3 details on the results of the reverse engineering method applied on successively more complex components. Section 4.2.4 sketches the process from the reverse engineering output to an analyzable model. Section 4.2.5 discusses the effort for applying the method and executing the tools. Finally, Section 4.2.6 concludes this section by presenting obtained data for the metrics defined in D7.1.

4.2.1 The Q-ImPrESS Toolchain: SISSy and SoMoX

The construction of the internal architecture of any given source code is accomplished by a double-staged process realized by the tools SISSy\(^1\) and SoMoX\(^2\). SISSy derives a generalized-abstract-syntax-tree (GAST) from the sources, which in the next step is used by SoMoX to identify the service components. SISSy requires an installation of a

\(^1\) Structural Investigation of Software Systems
\(^2\) Software Model Extractor
Postgres database\(^3\) to store data extracted from the source code before generating the GAST. The further configuration parameters of SISSy are described in D6.2. The tool can be launched from the Q-ImPrESS workbench using the respective launch configuration. SoMoX analyses the architecture of the source code based on the GAST and produces an instance of the SAMM (Service Architecture Meta-Model) stored in three files:

- `samm_repository`: contains the actual architecture
- `sourcecodedecorator`: maps architecture components to the GAST
- `gastbehavior`: describes the dynamic behaviour of the SAM

These files are produced first by mapping every class to a primitive component; in the next step these primitive components are clustered into a more abstract structure by merging into composite components.

![SoMoX run configuration from the Eclipse IDE showing the Metrics Configuration tab](image1)

**Figure 2:** SoMoX run configuration from the Eclipse IDE showing the Metrics Configuration tab

SoMoX incorporates several metrics and weights for these metrics (see Figure 2) for determining whether to merge two classes or components into a higher level component. Details on the metrics are provided in the SoMoX tool manual (D6.1). The weights for the metrics need to be tuned manually to let SoMoX produce a desired high-level architectural structure. They are system- and technology-specific. The weights determined for the ABB code are described in Section 4.2.3).

### 4.2.2 Browsing Large SAM Repositories

As a prerequisite to understand the diagrams used in the later sections, this section describes the SAM Repository browser implemented at ABB.

\(^3\) http://www.postgresql.org/download
The need for the browser evolved from the daily usage of the tools, from which it became clear that the resulting SAM repositories of the reverse engineering are very large (containing up to 1000 components). Even though Q-ImPrESS provides several (graphical and text-based) editors and viewers to display models, viewing and browsing of large models is uncomfortable.

Due to these shortcomings the Repository Browser was implemented based on a publicly available Java library (TreeViz⁴) that allows an interactive browsing of large structures. The browser is embedded into Eclipse as an editor and can be accessed from the Q-ImPrESS perspective through the context menu (see Figure 3).

The browser expects a `samm_repository` resource as input. Furthermore, it also tries to load the `sourcecodedecorator` resource and the primary GAST file; assuming the names and locations as created by SoMoX. If one of the resources cannot be found, a resource chooser dialog is provided; this enables the browser to be independent from the SoMoX output and to be used for any browser of `samm_repository` resources. The `sourcecodedecorator` as well as the GAST file are necessary to change the naming of components. If any of these resources cannot be loaded, only the data of the `samm_repository` is used for the browser.

The TreeViz library supports different kind of visualizations. From these, the Repository Browser provides Hyperbolic Trees and Rectangular Maps; both are described in the following.

The Hyperbolic tree visualization (see Figure 4) displays primitive components as light blue rectangles, composite components as blue rounded rectangles and the root node as a dark blue rectangle. For navigation purposes each node can be simply moved by clicking and dragging. Dragging of nodes to the center magnifies them as well as their most related neighbors. Further distant nodes are scaled down to achieve clarity.

All nodes can be given new names by double clicking. To check the location of any single primitive component the root node can be double clicked to activate a search dialog with auto-completion assistance. Further information regarding single components can be accessed

⁴ http://www.randelshofer.ch/treecviz/
via tooltip texts. These contain information about implemented and used interfaces as well as the number of subcomponents.

The **Rectangular Map Visualization** offers a rectangular view in which all components are nested (see Figure 4). The size of the rectangles depends on the lines of code of each class contained in the corresponding components. To dive into the structure, composite components (indicated with a red frame) can be clicked. Analogously to the Hyperbolic Tree Visualization, every component offers further information about interfaces and number of children in their tooltip texts.

![Figure 4: Hyperbolic Tree Visualization showing composite components, primitive components and the repository](image)

The **Rectangular Map Visualization** offers a rectangular view in which all components are nested (see Figure 4). The size of the rectangles depends on the lines of code of each class contained in the corresponding components. To dive into the structure, composite components (indicated with a red frame) can be clicked. Analogously to the Hyperbolic Tree Visualization, every component offers further information about interfaces and number of children in their tooltip texts.

![Figure 5: Rectangular Map Visualization showing Primitive Components, Composite Components and Repository](image)
4.2.3 Reverse Engineering Results

This section describes the results of the reverse engineering activities performed on the ABB PCS demonstrator. It characterizes the code under analysis and presents the results for different parts of the system.

Prerequisites

The ABB PCS described in D7.1 is implemented in unmanaged C++ code currently using Visual Studio 2005. The code makes use of Microsoft’s Component Object Model (COM) as well as Microsoft’s Active Template Library (ATL). The code base is structured into 9 subsystems and more than 100 components and consists of more than 3 million lines of code. Some parts of the code are currently being migrated to .NET technologies, but these are not in the scope of the analysis described here. The code provides some challenges both for SISSy and SoMoX as described in the following two paragraphs.

For the SISSy tool extracting the GAST, the code base has to be prepared. The required include headers need to be gathered and their paths need to be supplied to SISSy as an input file. Furthermore, the code contains some Microsoft specific macro definitions, which are not supported by SISSy out of the box and thus need to be collected and specified in an additional defines file as key/value pairs.

For the SoMoX tool suited weights for the different SoMoX metrics need to be found. There is architectural documentation of the ABB PCS describing the dependencies between the high level components, which are categorized into 9 subsystems. This documentation can be used as a blueprint for the SoMoX clustering approach.

OPC DA Code

For testing the SISSy capabilities to cope with Microsoft COM code, we chose to analyze the source code of a freely available OPC DA server\(^5\), which has similar features as the ABB code. This code was chosen instead of ABB code, because it could be exchanged between ABB and FZI for testing purposes, whereas the ABB code remained internal.

Characteristics of the code are shown in Table 1.

<table>
<thead>
<tr>
<th>Files</th>
<th>127</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of Code</td>
<td>49,966</td>
</tr>
<tr>
<td>Statements</td>
<td>17,887</td>
</tr>
<tr>
<td>Class Definitions</td>
<td>61</td>
</tr>
<tr>
<td>Functions</td>
<td>241</td>
</tr>
<tr>
<td>Sum of File Sizes</td>
<td>1.2 MB</td>
</tr>
</tbody>
</table>

Table 1: Statistics for the OPC DA Server code

- Processing with SISSy: 196 seconds for 61 cpp files
- Resulting GAST: 11 MB
- Processing with SoMoX: 14 seconds (task cluster components takes 0.178 seconds, while creating the SEFFs takes about 10 seconds)
- 30 primitive component, 13 composite components
- Resulting Hyperbolic Tree:

4.2.4 From Reverse Engineered SAM Instance to Prediction Model

This section describes the additional configurations and models, which need to be created to enable analysis and simulation of the reverse engineered models. Notice, that this manual overhead is an integral part of the reverse engineering process in Q-ImPrESS as SoMoX only produces a static structure and a behaviour model of the architecture, but no dynamic information or QoS annotations.

SAM Repository

The SAM repository needs to be configured to enable the other models to reference the needed elements correctly. First, the extracted SAM must become a Q-ImPrESS Alternate and needs be placed in the alternative browser. Secondly, the needed interfaces for the use case entry points need to be identified and located in the repository. These interfaces must be accessible as delegation ports from the highest enclosing composite component. It might be necessary to create delegation ports in the inner components until the implementing components are accessed. Furthermore, all operations of these interfaces need to be equipped with an SEFF behavior stub to connect the SEFFs.

Hardware Repository

Analogously to the SAM repository the hardware repository is a collection of available hardware. Every hardware element is described with its own characteristics. All properties are stated without any units, so the basic units need be used (i.e., Aggregate Bandwidth 100,000,000 Bytes/s).
Target Environment
Using the hardware repository the server hardware can be deployed based on this information. When setting up the Target Environment the hardware resource must be loaded to enable the linking. Each Target Environment can have several nodes, with again several nodes. The containers are the execution environment of the software and have limited access to provided server hardware.

Service Architecture Model
All available software components are described and contained in the SAM repository, but the actual system is built in the Service Architecture Model. This model consists of at least one Subcomponent Instance and one Interface port, which is used later in the Usage Model. As the model is service oriented, it must also contain at least one Service, which is directly bound to the target environment by running in a Container.

SEFF Repository
SEFF diagrams model software behaviour (i.e., internal actions, branches, loops etc.). The samm_seff resource contains all these diagrams and connects them to the SAM repository using the SEFF behavior stubs.

QoS Annotations
QoS annotations comprise resource demands or probability values, which are used to annotate SEFF diagrams. They are stored in the samm_qosannotation resource. The data of these resources needs to be measured in prototypes if no experience data exists.

Usage Model
Finally a usage model must be created. It consists of at least one scenario, which contains a System Call. This system call triggers the system through the given interface of the System Architecture.

4.2.5 Effort estimations
The following section covers the effort estimation (Table 2) for users to make use of the Q-ImPrESS reverse engineering. It is assumed that there are sufficient sources (e.g. tutorials, screen casts, demos etc.) available to prepare and aid the user. Furthermore it is assumed that all tools and editors work as expected.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Effort (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning of Q-ImPrESS</td>
<td>8</td>
</tr>
<tr>
<td>Preparing the source code</td>
<td>4</td>
</tr>
<tr>
<td>Understanding its architecture</td>
<td>8</td>
</tr>
<tr>
<td>Extract GAST (SISSy)</td>
<td>4</td>
</tr>
<tr>
<td>Extract SAM (SoMoX)</td>
<td>8</td>
</tr>
<tr>
<td>Complete SAM</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>36h</td>
</tr>
</tbody>
</table>

Table 2: Effort estimation for developers using the Q-ImPrESS reverse engineering
4.2.6 Reverse Engineering Metrics Validation

This section will be completed in the final version of the deliverable.

<table>
<thead>
<tr>
<th>Q#</th>
<th>M#</th>
<th>Description</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>M1.1</td>
<td>Percentage of correctly reconstructed composite components on the two highest abstraction levels of the architecture in comparison with existing architecture documentation</td>
<td>0...100</td>
</tr>
<tr>
<td></td>
<td>M1.2</td>
<td>Percentage of correctly reconstructed component connections (‘uses’-relationships) on the two highest abstraction levels of the architecture in comparison with existing architecture documentation</td>
<td>0...100</td>
</tr>
<tr>
<td></td>
<td>M1.3</td>
<td>Qualitative evaluation of the reconstruction by three experienced architects</td>
<td>++,+,-,--</td>
</tr>
<tr>
<td></td>
<td>M1.4</td>
<td>Time to manually fix and extend the reconstructed models, so that they are ready for quality attribute predictions</td>
<td>Time in person hours</td>
</tr>
<tr>
<td>Q2</td>
<td>M2.1</td>
<td>Time from the installation of the tool, its configuration and execution, to the extracted SAMM model (including manually modelled hardware, deployment, behaviour, and usage, but excluding quality annotations, which are part of the performance and reliability prediction)</td>
<td>Time in person hours</td>
</tr>
<tr>
<td></td>
<td>M2.2</td>
<td>Time for preparing the source code to be processible by the Q-ImPrESS reverse engineering tools</td>
<td>Time in person hours</td>
</tr>
<tr>
<td></td>
<td>M2.3</td>
<td>Time for getting to know the Q-ImPrESS reverse engineering method</td>
<td>Time in person hours</td>
</tr>
</tbody>
</table>
4.3 Performance Prediction

The performance prediction applied on the ABB demonstrator is embedded into the overall Q-ImPrESS process as depicted in Figure 1. After manually enhancing the model produced by the reverse engineering tools with additional hardware and usage models, QoS annotations have to be supplied to enable performance predictions.

This section is structured as follows. Section 4.3.1 describes how the data collection for the Q-ImPrESS models was performed on the ABB demonstrator system. Section 4.3.2 discusses some prediction results for different evolution scenarios. Section 4.3.3 evaluates the achieved accuracy and the required efforts for this approach.

4.3.1 Data Collection

As the reverse engineering tools from the academic partners are still under development, we opted to create a demonstrator model manually in a first iteration of the validation phase. This helps us to evaluate the involved meta-models for their modeling expressiveness and to initially assess the possibilities for getting the necessary data from a running system.
In the following we describe the performance modelling of the ‘Log Aggregator’ service corresponding to the “History Retrieve” scenario described in D7.1. This service runs in the so-called ‘Data Integration Layer’ and manages logs of recorded signal data from an industrial production process. Clients (e.g. HMIs for plant operators) can request an arbitrary number of data values (items) from the data integration server, process the logged data, and generate trend charts via the service. Figure 8 (left) shows part of the static structure model of the Log Aggregator. The data integration server is connected via a standardized protocol (OPC DA) to a number of controllers, which may send thousands of items per second to the server. Figure 8 (right) shows part of the behaviour model of clients of Log Aggregator. The clients first perform marshalling, and then acquire a synchronization lock, before sending their requests to the server. After receiving the response from the server, they release the lock and unmarshal the data.

We instrumented the code of the ‘Log Aggregator’ and inserted numerous measurement points into the implementation. As a behavioural model of the service was not available, we first had to reconstruct it. We used high precision counters from Windows for generating timestamps. After some tests, we found that the execution time of the system can be abstracted into three distinctive resource demands: client CPU demand before sending a request, server CPU demand, and client CPU demand after getting a response.

Fig. 3 shows the measurement results for the three different resource demands in relation to the number of requested items. We repeated the test cases multiple times to derive reliable results for the average execution times. The results indicate that the dependency between the number of requested items and the execution time is linear. We used this data to construct a performance model from the SAMM.
4.3.2 Prediction Results
For the performance predictions, we relied on the Palladio component model (PCM) and its built-in extended queuing network simulator. The PCM is connected to the SAMM of Q-ImPrESS via a model transformation. We simulated the performance model with the same number of clients and number of requested items as in our measurement study. Then, we compared the mean response times predicted by the simulation and the mean response times measured (Table 1). The deviation is below 10 percent in all cases and therefore we deem the model sufficiently accurate for extrapolation.

Table 3: Prediction vs. Measurement for the service ReadRaw

<table>
<thead>
<tr>
<th>Number of Items</th>
<th>Predicted Mean Response Time</th>
<th>Actual Mean Response Time</th>
<th>Error</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.0089</td>
<td>0.009441</td>
<td>0.000541</td>
<td>5.73</td>
</tr>
<tr>
<td>2000</td>
<td>0.0137</td>
<td>0.014676</td>
<td>0.000976</td>
<td>6.64</td>
</tr>
<tr>
<td>4000</td>
<td>0.0233</td>
<td>0.024475</td>
<td>0.001175</td>
<td>4.80</td>
</tr>
</tbody>
</table>

In the following, we report on quality impact predictions for different evolution scenarios for this service:

Usage Profile Evolution
Because of the evolution of the underlying industrial system in terms of sensors and throughput, it is expected that the number of requested items will increase beyond 8000 items per requests. Using our model, we found that the mean client response time will be 0.079 seconds for 16000 requested items and 0.154 seconds for 32000 requested items.
Additionally, it is expected to have clients with multithreaded access to the LogAggregator in the future to update multiple trend charts in parallel. Our model predicts that clients initiating two threads at a time for 8000 items will receive a mean response time of 0.060 seconds, while clients initiating four threads at a time would receive their responses after 0.130 seconds.

**Resource Environment Evolution**

Both the Operator Workplace Client and the Log Aggregator run on Intel Core 2 Duo PCs with 2.66 GHz initially. Our model can predict the performance impact of using faster hardware when evolving the system. If the server PC is changed to an Intel Core 2 Duo PC with 3.33 GHz, the overall mean response time for requesting 8000 items will decrease from 0.042 seconds to 0.040 seconds. If the client PC is changed in the same way the response time will decrease to 0.036 seconds, because of its higher processing load. If both PC are changed to the faster processors, the response time will be 0.034 seconds. Additionally the impact for saving hardware can be predicted. If both the Operator Workplace Client and the Log Aggregator are deployed on the same PC (2.66 GHz) the response time stays the same for single-threaded access, but increases to 0.090 seconds for 2 parallel threads or 0.170 seconds for 4 parallel threads.

**Service Implementation Evolution**

After analyzing the code of the LogAggregator it is expected that the processing overhead per item can be reduced by 50 percent from 0.0022 seconds to 0.0011 seconds by optimizing the involved algorithms. Using our performance model, we can predict the impact of this change to the overall response time perceived by the clients before actually implementing the change. After adapting the model to the new resource demand for the server component, the predicted mean response time for requesting 8000 items is 0.040 seconds instead of 0.042 seconds in the original system. The result indicates that this optimization yields only a small overall impact on the performance.

### 4.3.3 Performance Prediction Metrics Validation

<table>
<thead>
<tr>
<th>Q#</th>
<th>M#</th>
<th>Description</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3</td>
<td>M3.1</td>
<td>Mean percentaged deviation between predicted and measured response times for all evolution scenarios and design alternatives.</td>
<td>0…100</td>
</tr>
<tr>
<td></td>
<td>M3.2</td>
<td>Mean percentaged deviation between predicted and measured resource utilizations for all evolution scenarios and design alternatives.</td>
<td>0…100</td>
</tr>
<tr>
<td>Q4</td>
<td>M4.1</td>
<td>Time for setting up a system for performance measurements (installations, tools, configuration, load drives, etc.)</td>
<td>Time in person hours</td>
</tr>
<tr>
<td></td>
<td>M4.2</td>
<td>Time for performance measurement and data collection for a given performance scenario of an architecture.</td>
<td>Time in person hours</td>
</tr>
<tr>
<td></td>
<td>M4.3</td>
<td>Time for data analysis of the performance measurements of a performance scenario to derive resource demands for the Q-ImPrESS models.</td>
<td>Time in person hours</td>
</tr>
</tbody>
</table>
4.4 Reliability Prediction

The reliability prediction applied to the ABB demonstrator is embedded into the overall Q-ImPrESS process as depicted in Figure 10. After manually enhancing the model produced by the reverse engineering tools, component failure rates and transition probabilities have to be supplied to enable a reliability prediction.

In the following, Section 4.4.1 gives an overview of the system and introduces the approach applied to conduct reliability prediction. Then, Section 4.4.2 describes how we obtained failure rates for the subsystems, and Section 4.4.3 details how we obtained probabilities of the transitions between the subsystems. The actual reliability prediction and a sensitivity analysis are presented in Section 4.4.4. Finally, Section 4.4.5 evaluates the efficiency of the reliability analysis approach.

4.4.1 Overview

For reliability analysis, the project's method requires constructing a discrete time Markov chain (DTMC) similar to the Cheung model (Cheung, 1980). Here, we created discrete time Markov chains for the process control system (PCS). The core of the PCS consists of more than 3,000,000 lines of C++ code and is structured into 9 subsystems, which are decomposed into more than 100 components. As we wanted to start with a cost-effective model and because subsystem failure reports were available, we decided to construct a coarse-grained model of the subsystems, which can be refined later. Such a model reflects the subsystems of...
the PCS and their interactions via DCOM. As an input the model requires component failure rates and component transition probabilities.

Figure 11: Data collection steps for reliability prediction

Figure 12 depicts our method for collecting the necessary input data and conducting the analysis. To derive component failure rates (steps 1-5), we exploited the system's bug tracker database and fitted a software reliability growth model (SRGM) using failure report data for each subsystem. This activity incorporated existing architectural documentation (step 0). To derive component transition probabilities (steps 6-11), we executed the system using two representative workloads and exploited internal logging facilities of the system to detect control flow among the subsystems. The actual analysis is then conducted in step 12. These three steps are detailed in the following three sections.

4.4.2 Subsystem Failure Rate Estimation

To determine the failure rates of the subsystems, we executed the following four steps.

Failure classification in the bug tracker database (Step 1)

Before proceeding further, it is important to revisit some definitions in order to set a common ground for the lexicon. The IEEE standard 1633-2008 on Recommended Practice on SW Reliability defines failure as:

a) The inability of a system or system component to perform a required function within specified limits.

b) The termination of the ability of a functional unit to perform its required function.

c) A departure of program operation from program requirements.
In our context, developer and end user failure reports are recorded in a bug tracker database, which is our only source to look for failures. Consequently, from our point of view, a failure occurs when the developer and/or the end user reports a failure.

The IEEE standard also defines failure rate as follows:

a) The ratio of the number of failures of a given category or severity to a given period of time; for example, failures per second of execution time, failures per month. Synonymous with failure intensity.

b) The ratio of the number of failures to a given unit of measure, such as failures per unit of time, failures per number of transactions, failures per number of computer runs.

The failures recorded in the bug tracker are time-stamped, and therefore in our context, the first definition applies.

Finally, the IEEE standard rates the failures according to the following severities:

- Severity #1: Loss of life or system
- Severity #2: Affects ability to complete mission objectives
- Severity #3: Workaround available, therefore minimal effects on procedures (mission objectives met)
- Severity #4: Insignificant violation of requirements or recommended practices, not visible to the user in operational use
- Severity #5: Cosmetic issue that should be addressed or tracked for future action, but not necessarily a present problem.

ABB has a similar classification but groups them into Project Stopper, Critical, High, Medium and Low. There’s almost a 1 to 1 match between the standard definitions and ABB categories, which facilitates the selection of failures according to the standard suggested best practices.

Filtering the data (Step 2)

When mining the huge amount of information available in the bug tracker databases, there is a need to define some data selection criteria.

The first aspect to address is what kind of failures to select with respect to the severities described before. We intend to model system availability, which includes both software and hardware errors. Therefore, we will be looking at Critical and High severity categories because these are the failures that most probably lead to system downtime.

Secondly, we also look forward to comply with IEEE 1633-2008 Quality of Assumptions, in particular, to the one assuming that “Faults are immediately removed when failures are observed” (IEEE, 2008). The records of the bug tracker database have a field to indicate whether:

a) change was applied to remove the fault
b) the report is a duplicate
c) the reported issue is forwarded
d) the reported issue is not a problem or it is working as designed
e) the issue is not repeatable
f) the reported issue will not change

Therefore, in order to comply with what was stated before, only reports of issues where a change was applied are selected.

Another assumption is that the component being modelled is somehow “Stable”, that is, it can run for some time before failing. This means that compilation and crude execution errors have been already eliminated during testing, e.g. “Project Stopper” failures. To ensure that, the
time origin for starting the failure count is taken to be the public release date of the system, and therefore, only records where the failure submit date is greater than or equal to the system release date are selected. This leaves us mostly with field, i.e. customer reported failures. Consequently, the model will predict the customer perceived failure rate and system stability, which is a desirable thing for ABB business units.

Table 6 summarizes the collection criteria described above.

<table>
<thead>
<tr>
<th>Failure Aspect</th>
<th>Collection Criteria</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Severity</td>
<td>Critical and High</td>
<td>Cause downtime that affects overall availability</td>
</tr>
<tr>
<td>Disposition</td>
<td>Change applied</td>
<td>Remove faults once they are discovered</td>
</tr>
<tr>
<td>System status</td>
<td>Released</td>
<td>Stable components, i.e. coarse faults have been already eliminated. Current failures come mainly from field, i.e. customers</td>
</tr>
</tbody>
</table>

Selecting a SRGM (Step 3)

Once proper data have been collected, a rough approximation to get a preliminary failure rate estimate would be to compute the following average:

\[
\frac{-1}{\lambda} \left[ \frac{\text{Failures}}{\text{Unit time}} \right] = \frac{\text{# of failures since release}}{\text{Elapsed time between release and the last submitted failure}}
\]  

(1)

However, if we plot the elapsed time since release in days vs. the total number of reported failures, we get a dependency that is not linear as shown in .
Then instead of (1), the following expression shall be used:

\[
\bar{\lambda} = \frac{1}{\text{Total elapsed time since release}} \int_0^{\text{Total elapsed time since release}} \lambda(x) dx
\]  

(2)

In order to evaluate (2), we need to determine the expression of the instantaneous failure rate \( \lambda(t) \). An expression for \( \lambda(t) \) can be derived using failure data from the bug tracker plus statistical inference and regression analysis. By fitting a function to the failure data, one can compute more accurate averages, instantaneous values as well as inter-/extrapolations, that is, estimations of past and current failure rates as well as future predictions.

The next step is to select a function to fit the failure data. Among many possibilities, there are mainly two approaches depending on whether the software component exhibit reliability growth or not. If the reliability grows one can try one of the many Software Reliability Growth Models that have been proposed through the years in particular those suggested by the IEEE Std. 1633 and IEC 61164 standards. If reliability does not seem to grow, effort shall be put in trying to determine a fitting curve using statistical inference and regression analysis.

An analysis of failure reports was performed for several components belonging to several different system version releases. We found that the evolution of the time between failures (TBF) vs. failure count can be summarized in three exemplary cases plotted in Figure 13.
Figure 13: Three exemplary cases summarizing the reliability characteristics encountered in most of the subsystem of our study: (a) Monotonic growth from beginning; (b) Alternating growth and decrease periods; and (c) Growth after monotonic decrease.

As shown by the three exemplary cases depicted in , reliability ends up finally growing for all three examples. As the cases reflect, it was noticed that for some software components the reliability growth from the beginning, like in example 1. For others however, the reliability seems to diminish at the beginning and then after reaching a minimum or after some fluctuations it finally starts to grow as shown in examples 3 and 2 respectively.

It was concluded that the reliability of the components grow, and thus the usage of SRGM is applicable.

Having decided to use SRGM to estimate the failure rate of software components, we were confronted basically with two complementary selection approaches one formal and another which is more qualitative. The formal approach uses well-known mathematical tools like Chi-Square, Kolmogorov-Smirnov and Prequential likelihood tests to compute the fitness of each model. Strictly speaking, one shall select the model that best fit the data.

What happened is that different components ended up modelled with different models just for small differences in the results of the fitness tests. We argue that having to deal with many reliability models within a single system architecture introduces a level of complexity that could hinder the practical usability of the methodology. Therefore, we decided to complement the results of the tests with a qualitative selection approach using industry and complexity affinity to select a model that, although good, may not be the best fit for some components but can be unique for all the components that make up the architecture.

The IEEE Std. 1633-2008 was used as source of models. The Standard divides models into initial and alternatives, the latter being recommended when the former are discarded. Table 7 lists the models and provides an overview of their origins.

<table>
<thead>
<tr>
<th>Model</th>
<th>Origins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td></td>
</tr>
<tr>
<td>Schneidewind</td>
<td>NASA. Aerospace. Warfare</td>
</tr>
<tr>
<td>Generalized Exponential (e.g. Jelinski &amp; Moranda)</td>
<td>U.S. Navy</td>
</tr>
<tr>
<td>Logarithmic Poisson (Musa/Okumoto)</td>
<td>Bell Labs. Telecommunications</td>
</tr>
<tr>
<td>Alternative</td>
<td></td>
</tr>
<tr>
<td>Duane</td>
<td>General Electric. Mixed HW &amp; SW Aircraft equipment (generator, jet engine, hydro-mechanical devices, etc.)</td>
</tr>
</tbody>
</table>

Table 5: Overview of the models described in IEEE Std. 1633-2008
The models whose origins are related to the system we are trying to model are the Yamada et al S-shaped model and the Littlewood/Verrall (LV) model. A closer look at their origins reveals that the Yamada model was based on an application of around 3000 LOC, which is very small compared to the 5MLOC’s system we are trying to model. It was mostly oriented at testing phase. We instead intend to model a system after release. Furthermore, it requires that the number of errors for the observed data must follow an S-shape, but when plotting failure count vs. elapsed time since release, most of the components exhibit an evolution containing many “S”s such as the one sample component shown in Figure 15.

![Figure 14: Failure count vs. time for a sample component](image)

Last but not least, it assumes that no other errors are introduced when faults are removed, which most probably does not hold in our case. We argue that the fact that in some cases faults are introduced when others are removed is one of the causes of the peaks and lows exhibited by the exemplary components depicted in .

Given that, we turned our attention to the LV model. Although the original publication does not disclose the LOC of the system (Littlewood & Verral, 1973), it mentions that it took around 17 man years of programmer effort alone to develop the system, which indicates a SCADA/DCS system of a complexity comparable with the one we intend to model. Contrary to the S-shaped model, it accounts for the fact that repair actions may introduce new faults and/or errors. It does that by assuming that when eliminating the fault that causes the failure, it is probable that the failure rate is diminished but not certain, i.e. the failure rate $\lambda$ is also a random variable. In other words, instead of $\lambda(i) < \lambda(i-1)$, they propose $P[\lambda(i)<l] \geq P[\lambda(i-1)<l]$ allowing for the probability that the software components could become less reliable than before. Briefly, almost all software reliability models assume that reliability will increase as
testing progresses and failures are found and removed. The Littlewood-Verrall model does not make this assumption. Furthermore, the model exhibited good fit to our data during the preliminary exploratory analysis. For these reasons, the LV model was chosen to estimate the failure rates of the components.

Table 8 summarizes the characteristics of the environment described in the original paper.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industry</td>
<td>Power generation</td>
</tr>
<tr>
<td># of sensing devices</td>
<td>~10,000</td>
</tr>
<tr>
<td>Input variables</td>
<td>temperatures, pressures, flow rates, etc.</td>
</tr>
<tr>
<td>Services</td>
<td>Trending, logging, alarm analysis, turbine control, efficiency tracking, fuel consumption, etc.</td>
</tr>
<tr>
<td>Development effort</td>
<td>~17 man-years</td>
</tr>
</tbody>
</table>

**Estimating using the Selected SRGM (Step 4)**

We chose the Computer Aided Software Reliability Estimation (CASRE) tool to perform our failure rate estimations. In the following, we describe exemplary how we applied the methodology for a given subsystem.

Figure 15 shows a CASRE plot of its critical and high failure history. Notice that the unit of the time between failures has been intentionally obfuscated for confidentiality reasons.

![Casre plot](image)

**Figure 15: Time between failures for a given software subsystem.**

The plot reveals reliability growth towards the end. But data is too noisy to devise a trend at the beginning. We followed the recommendations of the CASRE User's Guide and smoothed the data to reduce scattering. The results are shown in .

---

http://www.openchannelsoftware.com/projects/CASRE_3.0
Figure 16: Time between failures for a given software subsystem after smoothing.

The evolution of TBFs corresponds to case b in Figure 13. We were able to fit the whole dataset without filtering data at 5% significance level with the quadratic Littlewood/Verrall model (LV-Q). shows a plot of the resulting failure intensity.

Figure 17: Failure intensity of software subsystem under study showing the original points and fitting curve

Finally, the current failure intensity estimated by the model is the value used to annotate the failure rate of software subsystem under study. We applied the same procedure to all subsystems of the ABB PCS in order to estimate their failure rates.
4.4.3 Subsystem Transition Probabilities

For our demonstrator, we decided to use the internal logging facilities of the ABB PCS, which can be seen as a special case of profiling. The advantage of using the internal logging facilities is that they are capable of only logging subsystem interactions. The disadvantage, however, is that internal logging is not available for all components. We did not use design documents because the existing ones only detail static dependencies. Manual code instrumentation was deemed too expensive because the code base is large and the subsystem transitions cannot be automatically identified. The following describes steps 6-11 shown in Figure 12.

Installing the System (Step 6)

First, we installed the PCS, two additional servers for providing data, and three client applications to access the PCS. Because of the size of the system and its distributed nature, this is a non-trivial step that can take several hours or even days. Besides installing the core system, this step also involves setting up simulation components for the different devices (e.g., sensors) usually present in an industrial control system.

Defining the Workload (Step 7)

To run the system, we had to define a load profile. The failure rates from the bug tracker database obtained in step 1-5 are based on bug reports from all customers, therefore their underlying load profile is the union of all customer load profiles. These customer load profiles are not available to us and can therefore not be used to determine the transition probabilities in our case. Notice that this a generic issue when basing the component failure rates on SRGMs.

We decided to use two realistic load profiles for single customers based on two typical use cases. They provide an approximation of different customer load profiles. One of these profile focuses on the engineering of the industrial process while the other one concentrates on the steady state of the continuously running process. These load profiles have been identified with the help of domain experts and are aligned with test profiles for the system. Setting up the system for the load profiles involves configuring a number of data subscriptions and client applications in our case. For other systems, this step might involve implementing load drivers or scripts mimicking user behaviour.

Configuring System Logging (Step 8)

Before running the application, we configured an ABB development tool for logging PCS system calls to record only interactions between subsystems. This involves identifying the components responsible for subsystem transitions and adjusting the log detail level for these components. Each subsystem transition represents a DCOM call between two processes. Each log entry contains information about which subsystem called which other subsystem. The tool writes all log entries into a file.

Running the System and Collecting the Data (Step 9)

We executed each profile for two days. Running a load profile includes starting all applications (i.e., the server, the logging application, and all clients), performing the initial setup, and operating the system. In the first profile, operating comprises engineering of the system as well as observing data and interacting with the system. In the second profile
operating of the system only consists of observing data and interacting with the system. The two log files generated in this step had a size of 2 GB each.

**Processing the Data (Step 10)**

The created log files were then passed to a self-coded script, which generates the list of subsystems involved, the transitions between these subsystems, and the probabilities of these transitions. Additionally, it adds an initial and a final state to indicate the beginning and the end of the execution. The subsystems that are connected to the initial state and to the final state respectively are determined by examining their inner components. A subsystem can have more than one entry point and accordingly more than one exit point. Determining these points on the level of subsystems would potentially mask some of them. Therefore, for each component, it is determined whether it is an entry or an exit point by taking the difference between incoming and outgoing transitions. A component is an entry point if it has more outgoing than incoming transitions. It is an exit point if the number of incoming transitions is larger than the number of outgoing transitions.

For each of the transitions involving the initial and the final state, the number of occurrences is stored. Then, for each subsystem, the number of transitions from the initial and to the final state is determined by summing up the corresponding transitions of their components. The distribution between the transitions of the initial state and the transitions of the final state respectively is determined by the proportion of the corresponding transitions. and show the result of script for the first and second load profile. The second load profile is similar, but contains some additional transitions and different transitions probabilities.
Figure 18: Subsystem transition probabilities for the first load profile
Validating the Data (Step 11)
We validated the data by examining the different paths through the model and matching these paths to the operations that we actually executed. During this process, we were supported by two PCS experts.

4.4.4  Reliability Prediction Results and Sensitivity Analysis
The last step in our approach after determining the failure rates and the transition probabilities is the generation of the DTMC and the execution of the sensitivity analysis. This is done in step 12 shown in Figure 12.

Analysing the DTMC (Step 12)
To construct the DTMC, we applied another self-coded script, which takes as input the failure rates of the subsystems and the probabilities of the transitions between the subsystems. The subsystems get states in the DTMC. Each of these states denotes that control currently resides within the corresponding subsystem. Additionally, the script adds a failure state.
To account for failures, the script adds a failure state and modifies the transition probability matrix $P$ of the DTMC as follows. The former transition probability $p_{ij}$ between subsystems $i$ and $j$ is adjusted to $(1-f_i) \times p_{ij}$, where $f_i$ is the failure rate of subsystem $i$. For state $i$ representing subsystem $i$, a transition to the failure state with the probability $f_i$ is introduced. The system reliability can then be calculated by the probability of not reaching the failure state (Cheung, 1980). To be able to use PRISM for the sensitivity analysis, the script does not use the actual failure rates but variables that represent the failure rates. By instantiating these

---

**Figure 19: Subsystem transition probabilities for the second load profile**
variables to different values or intervals respectively, PRISM can be used to conduct a sensitivity analysis.

For the sensitivity analysis, we created 8 different PRISM models. For each subsystem, we created a model where the failure rate for this subsystem was provided as a range around the actual failure rate and the failure rates of the other subsystems were the actual failure rates.

With our DTMC model of the ABB PCS architecture, we analyse the overall system reliability and perform sensitivity analyses. We do not provide the actual system reliability value here for confidentiality reasons. Because of our way of estimating failure rates using SRGM, the system reliability predicted from the model does not reflect the reliability a customer of the system can expect. The failure rates are based on a union of all customer profiles. A single customer can thus expect a higher system reliability.

The validity of the predicted value is inherently difficult to assess, as we cannot compare it with reliability measurements because of the missing usage profile for all customers. Comparing the value predicted from the DTMC with the value predicted by the LV-Q model for the overall system yields a deviation of 1.94% (first profile).

The results of the sensitivity analyses are depicted in . The figure shows the system failure rate over the individual subsystem failure rates. We did not check the sensitivity of system reliability to changing transition probabilities because they cannot be changed. Each small 'X' at the centre of each line represents the failure rate from the SRGM, while the line around the 'X' represents the system reliability from the sensitivity analysis by varying the failure rate by +/- 10 points. The numbers located at the end of each line indicate the number of the subsystem.

Subsystem 8 and 6 have the highest failure rates, while subsystem 5 has the lowest failure rate. Notice, that the x and y-axis do not exhibit the same resolutions as the figure has been enhanced for best visibility of the slopes of the curves for the different subsystems.
The slopes of the curves are a measure for the sensitivity of the system failure rate to the subsystem failure rate. Table 7 summarizes these slopes for each subsystem and for the two load profiles. It contains the linear prediction model and the LV-Q model.

Table 7: Slopes for the system failure rate changes based on subsystem failure rate changes

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile 1 (average)</td>
<td>0.6326</td>
<td>0.4161</td>
<td>0.0266</td>
<td>0.3743</td>
<td>0.0302</td>
<td>0.0244</td>
<td>0.0157</td>
<td>0.1242</td>
</tr>
<tr>
<td>Profile 1 (LV-Q)</td>
<td>0.6485</td>
<td>0.4188</td>
<td>0.0266</td>
<td>0.3823</td>
<td>0.0311</td>
<td>0.0247</td>
<td>0.0161</td>
<td>0.1269</td>
</tr>
<tr>
<td>Profile 2 (average)</td>
<td>0.6334</td>
<td>0.4170</td>
<td>0.0264</td>
<td>0.3741</td>
<td>0.0295</td>
<td>0.0224</td>
<td>0.0143</td>
<td>0.1252</td>
</tr>
<tr>
<td>Profile 2 (LV-Q)</td>
<td>0.6492</td>
<td>0.4197</td>
<td>0.0264</td>
<td>0.3821</td>
<td>0.0304</td>
<td>0.0225</td>
<td>0.0147</td>
<td>0.1279</td>
</tr>
</tbody>
</table>

Subsystem 1 is most sensitive for the system reliability (slope ~0.65), which appears plausible because it is responsible for processing most of the data in the PCS and is called most often. Subsystem 2 and 4 also exhibit high sensitivities while the system reliability is relatively robust against the other subsystem failure rates. Subsystem 6, which is used by many subsystems does not contribute much to the overall system reliability. Compared to other subsystems, this subsystem is called only a limited number of times and therefore has a limited impact on system reliability. For subsystem 8, we had estimated the highest failure rate, but it is in fact also only a minor driver for system reliability.

The evaluation of different design alternatives for the next version of the ABB PCS is difficult. In the last few versions of the system, there have hardly been any changes on the subsystem level. A more refined model would be needed to evaluate the impact of different component topologies on the system reliability. Many technical design alternatives are difficult to incorporate into the model by expressing their impact on the failure rates or transition probabilities.

### 4.4.5 Reliability Metrics Validation

To assess the quality of the reliability metrics prediction and the efforts needed, we formulated two hypotheses, which are shown in Table 8. To evaluate the accuracy of the reliability metrics prediction, we used hypothesis Q5.

Table 8: Hypotheses for Validation of Reliability Metrics

<table>
<thead>
<tr>
<th>Q#</th>
<th>M#</th>
<th>Description</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q5</td>
<td>M5.1</td>
<td>Mean percentage deviation between predicted and measured reliability metrics (failure rates, probability of failure on demand, etc.) for one given evolution scenario</td>
<td>&lt; 10%</td>
</tr>
<tr>
<td>Q6</td>
<td>M6.1</td>
<td>Time for gathering and analysing the necessary data to derive reliability annotations for the Q-ImPrESS models (i.e., failure rates from bug tracker databases, component transition probabilities from the usage scenarios)</td>
<td>&lt; 40 person hours</td>
</tr>
</tbody>
</table>

However, the time needed for gathering the information needed to derive the quality annotations for the Q-ImPrESS models is known. Table 9 shows the time needed for the different steps of our approach. The table provides two effort estimations: the actual one and a potential one for future applications of the approach.
Table 9: Effort Estimations for the Different Activities

<table>
<thead>
<tr>
<th>#</th>
<th>Activity Name</th>
<th>Actual</th>
<th>Potential</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Obtain Bugtracker Access</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Analyse Literature</td>
<td>56</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Filter Data</td>
<td>24</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Apply SRGM (CASRE)</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Validate Data</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>Obtain Documentation</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>Install System</td>
<td>40</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>Define Workload</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>Configure System Logging</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>Run System Record Logs</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>Process Data</td>
<td>100</td>
<td>40</td>
</tr>
<tr>
<td>12</td>
<td>Validate Data</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>Analyse DTMC (PRISM)</td>
<td>40</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Sum in person hours</td>
<td>382</td>
<td>122</td>
</tr>
</tbody>
</table>

The actual effort estimation is 382 person hours, which is significantly higher than the estimated result of 40 person hours. Included in the 382 hours are initial activities, which only have to be done once, and iterations that had to be done due to problems that occurred. The step processing the data, for example, included the time needed for coding a script that conducted the processing. The installation of the system, on the other hand, was done several times due to some performance issues.

To remove these influences, we revised the effort estimation. The revised effort estimation is shown in the column called potential. The potential effort estimation is 122 hours, which is still higher than the estimated result of 40 hours, but it is significantly lower than the actual estimation. This shows that the efforts needed for providing the data are significantly influenced by factors, which are out of the scope of the Q-ImPrESS tools.

To judge whether these efforts pay off, we would have to quantify the actual savings achieved by applying the Q-ImPrESS tools. This is, however, not possible because some of the savings cannot be quantified and the time frame of the Q-ImPrESS is too short to get feedback from business units due to the round-trip time of development projects.
5 ENT evaluation

This chapter presents evaluation of the Q-ImPrESS project results performed on the Ericsson Nikola Tesla's demonstrator from the telecommunications domain. More precisely, the demonstrator assesses the applicability and usefulness of the Q-ImPrESS method and tools in a typical industrial scenario from the telecommunications domain. The assessment consists of the validation of requirements imposed on Q-ImPrESS method and tools, as defined in the final version of D1.1 ("Requirements document - final version") from April 2009, and the validation of software engineering methods and tools created in the project. Validation plans and expectations are described in the final version of D1.1 ("Requirements document – Final Version") from April 2010. The Q-ImPrESS process workflow used during the validation is described in D6.1 ("Method and Abstract Workflow"). More information on the Ericsson Nikola Tesla's (ENT) demonstrator is to be found in D7.1 ("Demonstrator description"), D7.2 ("Demonstrator implementation") and the official project website. Thus, herewithin, just a brief description of validation plans and demonstrator information is given.

The structure of this chapter is organized as follows. Section 5.1 describes the use of ENT demonstrator in the validation of Q-ImPrESS project's results. Section 5.2 presents in detail how we manually model the software architecture of the demonstrator in Q-ImPrESS tools. The following sections present validation of methods and tools for specific Quality Attributes groups. Section 5.3 shows performance prediction validation. Other Sections for reliability, maintainability and trade-off analysis are planned but not present in this initial version of the document.

5.1 Introduction and setting

This section briefly summarizes ENT demonstrator information and information how the demonstrator is used for validating the Q-ImPrESS project results. Brief introduction to the demonstrator is given in Section 5.1.1, and validation setting is presented in Section 5.1.2.

5.1.1 Brief introduction to the demonstrator

ENT demonstrator represents a typical research and development challenge found in current telecommunication networks. The challenge is to shift from the vertical industry model toward the horizontal one - besides own expertise and resources, to use 3rd party commercial off-the-shelf software (COTS) and hardware building blocks in creating new products and solutions. ENT demonstrator explores this challenge by creating a prototype system that extends ENT's own existing, so called legacy systems with a subsystem comprised of 3rd party COTS. Such extension of the legacy system brings an added value to the legacy system. More precisely, ENT demonstrator consists of a call control system that governs all operations regarding call handling, routing etc. The call control system is the legacy system used in the demonstrator and represents a system that has been evolving over a large number of years. The call control system is extended by a DIAMETER subsystem which offers standardized authentication, authorization and accounting functionality according to the widely accepted DIAMETER standard. The DIAMETER subsystem is built from 3rd party COTS and additional code that integrates the 3rd party COTS into a functional whole and connects it to the call control system. The 3rd party COTS includes Pen TCP-level load
balancer, OpenDIAMETER software implementation of the DIAMETER standard and OpenSAF software middleware that facilitates creation of highly-available software solutions. ENT demonstrator and all of its constituent parts are graphically presented by conceptual software architecture in Figure 20. The legacy system of call control is presented on the bottom of the figure while the extension occupies the rest of the figure. DIAMETER extension is organized into two clusters, client cluster and server cluster. The clusters consist of active and passive (redundant) nodes that utilize OpenDIAMETER software for delivering DIAMETER functionality. The clusters additionally utilize OpenSAF middleware that regulates availability of nodes within a cluster (e.g. if an active node fails, it is substituted with a certain node that was passive till that moment).

Detailed information on the demonstrator and its implementation is to be found in the final versions of D7.1 ("Demonstrator description"), D7.2 ("Demonstrator implementation") and the official project website.

5.1.2 Validation setting

Q-ImPrESS project results are validated by ENT demonstrator in two phases. In the first phase, the demonstrator operates in controlled environment where ENT's legacy systems are emulated by proprietary software. DIAMETER extension is unaware of legacy system's emulation, i.e. its functionality is the same irrelevant to its connection to the emulated legacy systems or the real-world legacy systems. In the second phase, the demonstrator operates in controlled telecommunications network testbed environment, i.e. DIAMETER extension is connected to the real-world legacy systems.
The demonstrator environment for the first validation phase is shown in Figure 20. ENT's legacy system of call control is emulated on Sun UltraSPARC workstations. One workstation emulates SIP (Session Initiation Protocol) call generation process and an access network while the other represents a softswitch node with call control software. The call control software on the workstation is configured to request AAA processing of an incoming call. This AAA processing must be done by the DIAMETER extension that is connected to the workstation emulating a softswitch node with call control software. DIAMETER extension operates on a number of standard PC computers running CentOS 5 Linux-based operating system. These standard PC computers are organized according to the conceptual software architecture presented in Figure 20. Figure 20 shows two columns of standard PCs in desktop casing. The column on the left is the implementation of the server cluster (including cluster's load balancer and point-of-presence), and the column on the right is the implementation of the client cluster (including cluster's load balancer and point-of-presence). Hardware configuration of used standard PCs consists of a Intel Core2Duo 3.0 GHz dual-core microprocessor, 2 GB DDR2-800 MHz and 160GB hard drive. All computers (both standard PCs and Sun UltraSPACE workstations) are interconnected according to conceptual software architecture from Figure 20. Network links used for interconnecting computers have the network throughput of 100 MBit/s.

The demonstrator environment for the second validation phase is shown in Figure 20. In this phase the demonstrator operates with real-world legacy systems that comprise telecommunications network testbed residing within ENT's premises. These legacy systems represent an actual platform for providing telephony and Internet services typically found in telephony service providers and Internet service providers. For the purposes of the second phase of validation, legacy systems are configured in a manner equivalent to the emulated
environment of the first phase of validation. Hence, one part of the legacy systems is configured for SIP call generation. This part also serves as an access network. The second part of legacy systems is configured as a telecommunications backbone network consisting of a number of routing nodes and specific service nodes. Routing nodes are configured to ask additional DIAMETER-based AAA processing of each SIP call incoming from the access network. AAA processing is performed on the DIAMETER extension that is connected to the legacy systems. The DIAMETER extension implementation is identical to the one used in the first phase of validation. The difference is that the DIAMETER extension is connected directly to the real-world legacy systems in the second phase of validation. This connection is achieved via dedicated network link configured to have the throughput of 100 MBit/s (equivalent to the network throughputs used in the first phase of validation).

During both phases all evolution scenarios defined in the final version of D7.1 are used for validating Q-ImPrESS project results. However, the difference between phases lies in the load imposed on each evolution scenario. The reason for this is that proprietary software, which emulates telecom legacy systems, cannot generate higher levels of load (in the case of the demonstrator - the number of calls needed to be AAA processed). Thus, this emulated software environment is used for testing functionality of implemented demonstrator with smaller loads. After these functionality tests are passed (meaning all bugs are corrected), the demonstrator is connected to the real-world legacy systems and is stressed with different amounts of load. In reality, we use the emulated software environment for measuring the characteristics of the demonstrator up to 500 SIP calls per second, and we use the real-world legacy systems for measuring the characteristics of the demonstrator for more than 500 SIP calls per second.

The users participating in the validation comprise of software architects and software engineers (the roles of these users are described in the final version of D1.1, Sections 4.2.2 and 4.2.3). In practice, software architects are represented by two senior employees, who have in-depth understanding of the demonstrator, and software engineers are represented by several employees (developers, testers) and scholarship students that have partial or no knowledge of the implemented demonstrator. Software engineers perform different tasks as asked by software architects. These tasks include scenarios setups, black-box measurements, creating demonstrator models according to black-box measurements and executing simulation of created models. Software architects govern the entire validation process and finalize validation results according to comparison of experimental (real-world) and model (simulation) data.

5.2 Manual creation of software architecture models

Before any predictions can be made by Q-ImPrESS tools, first some models must be created. This section presents the techniques for manual creation of software architecture models used in modelling the ENT demonstrator. In the overall Q-ImPrESS project workflow this manual creation of software architecture models is shown as a red, dotted and bolded arrow on Figure 20.

As stated in previous deliverables, namely D1.1 and D7.1, we cannot use the reverse engineering tools for (semi) automatic creation of Q-ImPrESS models from source code. The reason to this, simply put, is that a significant part of our code is proprietary (and written in non-mainstream languages) and also a significant part of code is completely unstructured and written in simple procedural languages like plain C. Reverse engineering such source code cannot yield meaningful results. Thus, we decided for the approach of manually creating
demonstrator software architecture models from scratch. In this approach all parts of the ENT demonstrator are treated as black-box components (software components having only interface information).

In order to use black-box components for model predictions, they must be additionally described with behavioural information (resource consumption, timings, etc.). In this turn, black-box components become gray-box components (software components having interface and behavioural information). In order to get information relevant for such additional black-box descriptions, black-box components must be monitored, i.e. black-box'es characteristics relevant for modelling are measured. Such measurements can be achieved with different techniques. The techniques we used for modelling black-box components of the ENT demonstrator are described in the following Section, Section 5.2.1. After obtaining the data relevant for modelling of black-box components, these data are used in creation of black-box components models. Section 5.2.2 briefly discusses the techniques used for these tasks during modelling of the ENT demonstrator.

5.2.1 Black-box component measurements

In the ENT demonstrator each constituting element is treated as a single black-box component. Such, coarse-grained system abstraction level is possible to use because each node in the demonstrator (see conceptual software architecture shown in Figure 20 and D7.1) serves only one, specific purpose. E.g. clients cluster Point-of-Presence (Pen load balancer) only load balances requests coming from the legacy system (call control) and routes back answers from the cluster to the legacy system. Equivalent reasoning is applicable to all other constituent elements of the demonstrator: DIAMETER clients, servers cluster Point-of-Presence...
Presence (DIAMETER relay) and DIAMETER servers. Besides each element in the demonstrator serving one specific purpose, the entire DIAMETER extension also serves only one purpose: processing AAA requests issued from the legacy systems. Due to such characteristics it is possible to use coarse-grained system abstraction level for modelling the demonstrator.

Described coarse-grained system abstraction level has additional advantage. It is possible to measure relevant characteristics of computer nodes (black-box components) without inserting instrumentation code into demonstrator source code residing on any node. In fact, we have compared measurements of node response time obtained by such non-obtrusive instrumentation (e.g. with Wireshark) with obtrusive instrumentation (measurements performed by special code injected into original code). The comparison shows difference of up to 10% between results obtained by non-obtrusive instrumentation and obtrusive instrumentation. For the purposes of our demonstrator, such difference is irrelevant. An example comparison of response times obtained by non-obtrusive and obtrusive instrumentation for DIAMETER client node is shown in Table 10.

Table 10: Comparison of DIAMETER client node response time (in milliseconds) obtained with non-obtrusive and obtrusive instrumentation

<table>
<thead>
<tr>
<th>Number of AAA requests</th>
<th>Non-obtrusive instrumentation</th>
<th>Obtrusive instrumentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AR</td>
<td>AA</td>
</tr>
<tr>
<td>500</td>
<td>132.1</td>
<td>518.5</td>
</tr>
<tr>
<td>1000</td>
<td>268.3</td>
<td>1080.3</td>
</tr>
<tr>
<td>3000</td>
<td>820.4</td>
<td>3193.8</td>
</tr>
</tbody>
</table>

It is important to note that there are two types of AAA processing performed by all nodes in the DIAMETER extension: authentication and termination. Authentication processing consists of authentication request (AR) followed by authentication answer (AA). Termination processing consists of termination request (TR) followed by termination answer (TA). In order to describe relevant characteristics of each node in the model, response times for AR, AA, TR and TA processing are measured for each node in the DIAMETER extension. Authorization and Termination message exchange and processing on each node of the demonstrator is shown in Figure 20.

5.2.2 Black-box component modelling

Coarse-grain measurements of node's response times have significant advantage when it comes to modelling systems. Measured values are entered into a model after slight mathematical transformation.

For the demonstrator purposes, we measure response times of each of the nodes for all four possible processings (AR, AA, TR, TA). We measure the response time of at least 500 processings because one processing lasts very small amount of time and is difficult to measure precisely. After the measurement, we divide the measured response time with the number of processings. This way we get the average value of AR, AA, TR and TA processing on each node. This average value of processing is entered directly into the model of the black-box component behavioural description (the SEFF model in the Q-ImPrESS IDE).
Figure 20: Authorization and termination AAA message exchange in ENT demonstrator. a), b) AAA protocol authorization requests; c), d) DIAMETER authorization requests; e), f) DIAMETER authorization answers; g), h) AAA protocol authorization answers; i), j) AAA protocol termination requests; k), l) DIAMETER termination requests; m), n) DIAMETER termination answers; o), p) AAA protocol termination answers

Figure 20: Performance predictions of ENT demonstrator in overall Q-ImPrESS process
5.3 Performance prediction

This Section presents in detail performance modelling of the ENT demonstrator for each of the evolution scenarios described in D7.1. Also, for each evolution scenario, demonstrator's response times obtained by experimental measurements and obtained by Q-ImPrESS'es performance prediction are shown and compared. In the overall Q-ImPrESS process workflow shown in Figure 20, performance prediction is designated with red rectangle. Section 5.3.1 presents in detail modelling, measurements and performance prediction results for each evolution scenario of the ENT demonstrator.

5.3.1 Modelling and performance prediction of evolution scenarios

This section describes measurement and modelling procedures, as well as the performance prediction analysis for each evolution scenario defined in the D7.1. Taking into account the fact that all scenarios derive from the basic demonstrator setup, the modelling process is adapted to facilitate creation of the common model which is, with respect to specifics, applicable for all evolution scenarios. While such a procedure is not mandatory it greatly simplifies the modelling process and excludes the need for repetitive modelling of the same components for different scenarios. Therefore, this common model is described first while specifics, such as measurements, configuration settings and prediction results, are explained later for each evolution scenario.

SAMM Repository

The first step in model creation is the definition of components. Each component is defined by its interface ports (provided or required) and SEFF behaviour stubs which must be defined for each operation in the provided interfaces. For the reasons explained in Section 5.2.1 coarse-grain component view (system abstraction level) is chosen for modelling. Hence, physical components in the demonstrator are described by primitive components in the Repository model (except legacy systems which are modelled in Usage model since they impose load on the DIAMETER extension connected to legacy systems). Although the DIAMETER extension specifies four different physical components (PEN load balancer and DIAMETER client, server and relay nodes), the simplification of the modelling procedure and the desire to create a common model applicable to all evolution scenarios requires a somehow different approach during modelling. Since the evolution scenario Changes to hardware resources requires a multiplication of DIAMETER client and server components (DIAMETER clusters), as well as the traffic distribution among cluster nodes, this has to be addressed in the actual model. At the moment of writing this document the only possible solution for simulating the load distribution in cluster environments is through the behavioural model, namely the SEFF diagrams, where calls to remote services are modelled with External Call Actions. Since every External Call Action is described with two basic features, required interface and a remote method, in order to distinguish calls to a component services within the cluster every component must implement an interface with a unique name. This means that in order to model load balancing behaviour each cluster component must be defined as a different entity with a separate interface (although in reality these are identical physical components implementing the same interface). Therefore, each physical DIAMETER client and server node is defined in the model as a separate component implementing different interface. This is illustrated in Figure 20.
Figure 20: Repository model – interface and component definitions

Figure 20: Repository model – component associations
The primitive components $GqManager1...4$ are model representation of DIAMETER client nodes implementing four different interfaces $IGqManager1...4$. The name $GqManager$ comes from the fact that we have implemented the so called $Gq$ Prime interface of the DIAMETER standard in our demonstrator. The same applies for DIAMETER servers which are modelled as $GqServer1...4$ components implementing $IGqServer1...4$ interfaces. In such setup load distribution feature is simulated with adequate component associations (provided and required interface port definitions for each component) and probability branch transitions defined in the SEFF model which is explained later in the document. Figure 20 illustrates associations between $LoadBalancer$ (PEN) and $GqManager1...4$ components. The same principles in creating associations are applied on $Relay$ and $GqServer1...4$ components. Also, it is clearly visible how all components define a SEFF behaviour stubs for all operations in provided interface ports. These stubs are used as connections to SEFF diagrams defined in a SEFF repository, explained below.

**SEFF Repository**

In order to capture behaviour of software components, SEFF diagram must be specified for each SEFF behaviour stub defined in a Repository Model. When looking from a system perspective, the demonstrator interface provides two calls ($Authorization$ and $Termination$ requests) which are, from behaviour modelling approach, identical. This means that SEFF diagrams on particular component are identical for both authorization and termination scenario but, because of the particularities of the tool, must be modelled separately. Therefore, all further explanations done for the authorization scenario also apply to the termination scenario as well. Figure 20 illustrates an activity diagram of an authorization call across all demonstrator components (note that this is not the example of the SEFF diagram but rather a means for easier explanation of modelling principles applied in SEFF diagrams). Boxes inside each component (process request/answer) represent the particular activity, which consumes local resources while transitions represent the external interface calls. Such high level views on activities related to request processing is applied in SEFF diagram specifications as well. Figure 20, representing the SEFF diagram specification of the DIAMETER client and server components, illustrates this resemblance between the actual and the modelled behaviour.

The only real differences are transitions related to answer propagation, which are modelled without external calls since network influence is omitted from the model. Special cases are load balancing components like PEN and Relay which are modelled differently because of the simplifications in the modelling procedure explained earlier. In order to model load balancing behaviour, SEFF diagrams of these components are extended with probabilistic branch transitions. These transitions enable mutually excluding behaviours to co-exist inside one SEFF diagram, and are executed according to some probability expression (i.e. constant
number, formula, etc.) defined in QoS annotations. Figure 20 shows how probabilistic branch transitions are used to model load balancing behaviour on the PEN load balancer component. Since each probabilistic branch transition includes one possible execution path, only one of the defined external call actions, here representing different DIAMETER client interfaces, is actually executed. The rate of a particular call is defined using a probabilistic expression in the QoS annotations described later.

**Hardware Repository**

The hardware repository represents a collection of common hardware descriptors, which are used to specify actual hardware components. A minimum configuration including one single core processor and one memory module descriptors are defined as illustrated in Figure 20. Note that our model disregards consumption of both network and memory resources but definition of at least one memory resource is mandatory in the model.

**Target Environment**

This model specifies the actual execution environment for the DIAMETER based AAA service using the collection of common hardware descriptors, previously defined in a hardware repository. According to our deployment strategy each service instance is running on a dedicated node. Therefore, a total of ten nodes are modelled, one for PEN and Relay and four for each...
Figure 20. SEFF diagram of authorization request operation on PEN load balancer

Figure 20: Hardware descriptors repository
Figure 20: Target environment – setting scheduling policy for LBContainer

Figure 20: Target environment – setting clock frequency parameter of GQM1Processor
DIAMETER client and server instances. For each node, minimal resource configuration includes processor, memory and container definitions. Containers, which are in fact abstractions for the service execution environment, must have resources allocated from the pool of available node resources. Since there is a 1:1 allocation of service instances on hardware components all available node resources are allocated to each node container. Also, for each container a FCFS (First Come First Served) scheduling policy is defined, as it best suites the nature of the actual system (Figure 20).

As already said, although our model omits the memory consumption, definition of at least one memory resource per node is mandatory. Hence, all memory parameters are left on their default values since they are not used in predictions. However, the Clock Frequency processor parameter is crucial for the prediction models and must be set for all nodes (Figure 20). Actual value of this parameter depends on the specificity of each evolution scenario and will be configured for each scenario separately.

**Service Architecture Model**

Actual service and subcomponent instance definitions are specified in the Service Architecture Model. For each component in the Repository Model there is a subcomponent instance definition which represents the actual implementation of the component.

![Service Architecture Model](image)

**Figure 20: Service Architecture Model – connections between Relay and GqServer subcomponent instances**
These subcomponent instances are used to model the actual system architecture using connectors, as illustrated in Figure 20. In this case, connections between subcomponent instances follow the same logic as associations between software components in the Repository Model.

Service definitions represent the workloads for subcomponent instances. For each service definition, implementing subcomponent instance must be defined along with the allocated execution container. An example of one service definition is shown in Figure 20.

To facilitate access to DIAMETER based AAA service at least one system interface must be defined. This system interface is connected to one of the available subcomponent instances using connectors in the same way subcomponent instances are connected to each other. Since PEN load balancer represents the single point of contact for the actual system, our model follows the same logic and exposes the Load balancer interface as a system interface (Figure 20).

**QoS Annotations**

The QoS annotation model holds the repository of annotations for all internal actions and probability branch transitions defined in the respective SEFF model. As illustrated in Figure 20 all listed internal actions from the SEFF model are defined as demands on a specific CPU execution resource. Furthermore, by manipulating branch probability values it is possible to simulate different service architectures and, therefore, load balancing behaviours. The only constraint is that the sum of all probability values for a specific branch action must equal one. Again, actual values are related to evolution scenarios and will be explained in more detail there.

**Usage Model**

The Usage model describes usage scenarios for the DIAMETER based AAA service. As shown in Figure 20 each usage scenario is described by two parameters: a system call which correlates to one of the available operations in the system interface, and a workload which specifies the number and profile of the users using the system. By manipulating scenario
parameters it is possible to obtain different performance prediction models, as will be explained more thoroughly for each evolution scenario.

Figure 20: Qos annotations
**Result Model**

The Result Model is defined as the final step in the modelling process. It allows the specification of the prediction result type (performance, reliability, maintainability) for each chosen alternative evaluation. An example showing the result model for performance predictions is presented in Figure 20. Since only performance predictions are of interest in this section, the same result model is used in all evolution scenarios.

**Viewing prediction results**

After all described models are defined, a performance simulation can be performed. The result of this simulation is a performance prediction graph, such as the one shown in Figure 20. The graph shows detailed performance prediction for each job in the overall workload. For validation purposes, we are interested in the execution time of the entire workload (red circle in Figure 20). We use these values when comparing experimentally measured values with the values obtained by prediction.

**Evolution scenario "Adding a new service"**

The evolution scenario of *Adding a new service* addresses an issue typical for system evolution, where the system is being extended with additional or new functionality over time. In our case, new functionality is added by introducing two dedicated load balancing nodes into the initial demonstrator architecture. The PEN load balancer for distributing AAA requests to the DIAMETER clients and the DIAMETER relay node for distributing DIAMETER requests to the DIAMETER servers.
This scenario is chosen as a basis for the detailed measurement and modelling procedure description since it includes all demonstrator components in an architecture setup applicable to most evolution scenarios. Hence, basic measurement principles and required transformations described here also apply to other evolution scenarios.

Figure 20: Result Model

Figure 20: An example of performance prediction graph (circled value is of
For the reasons explained in Section 5.2 all demonstrator components are treated as black-box components. In accordance with the approach for performance measurements of black-boxes, each component is described using a high-level view of its basic activities. Therefore, as illustrated in Figure 20, all demonstrator components are decomposed into two basic activities: process request and process answer (with the exception of the DIAMETER server, which, as a terminating node, does not perform the process answer activity). Since there are two possible system calls (authorization and termination) we can distinguish four different activities for each component: process authorization request (AR), process authorization answer (AA), process termination request (TR), and process termination answer (TA).

The idea is to measure execution time of those four basic activities for each component and transform the results into configuration parameters suitable for the software model. This procedure implies that each activity has to be measured independently, removing the influence of other activities, as well as the influence of other components in the processing chain on the measurements. For example, processing of authorization requests and answers would be, for a large number of generated requests, done concurrently on a component in a running system. Since this is not acceptable, for measurement purposes only, an external proxy node is introduced into the system, which can be configured to block requests/answers in order to allow precise measurements on every component. Figure 20 illustrates the deployment of two proxy components for measurements on DIAMETER client.

In this configuration all incoming requests/answers are first collected on the proxy nodes and then passed, all at the same time, onto the DIAMETER client for processing. This way, decoupling the request and answer processing activities is possible and the influence of other components in the system on the DIAMETER client measurements is removed. Since processing time of a single request for some components may reach the range of a few micro seconds, all measurements are done with a sample of 500 requests and are repeated 30 times. Measurement results, expressed in milliseconds, are then converted into values more suitable for modelling. Therefore, for each activity, execution time is converted into processing speed, expressed as a number of processed requests per second, using the following formula:

\[ V = \frac{S \cdot N \cdot 1000}{\sum_{i=1}^{N} T_i} \text{ [req/sec]} \]

where S is number of requests, Ti execution time of measurement i and N is total number of measurements. The results for all components and previously listed parameters (S=500, N=30) are summarized in Table 11.

![Figure 20: Deployment of proxy nodes for measurements](image-url)
Table 11: Activity processing speeds of components (requests per second)

<table>
<thead>
<tr>
<th>Node</th>
<th>Activity</th>
<th>AR</th>
<th>AA</th>
<th>TR</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEN</td>
<td>AR</td>
<td>19084</td>
<td>69444</td>
<td>58140</td>
<td>21551</td>
</tr>
<tr>
<td>DIAMETER Client</td>
<td>3906</td>
<td>986</td>
<td>4822</td>
<td>990</td>
<td></td>
</tr>
<tr>
<td>DIAMETER Relay</td>
<td>1356</td>
<td>2479</td>
<td>3541</td>
<td>2735</td>
<td></td>
</tr>
<tr>
<td>DIAMETER Server</td>
<td>1776</td>
<td>-</td>
<td>3088</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

These values are used as inputs for the model as explained next. Processing speeds for authorization requests (AR) for all components are set as clock frequency parameters of the respective hardware nodes (Figure 20). In the QoS annotations, CPU resource consumptions are then set as 1 clock cycle for authorization request and as a relative value for other activities. For example, CPU resource consumption for AA activity of PEN component is set as 19084/69444 = 0.275 clock cycles. Also, branch probability transitions are set in the QoS annotations to correspond to the respective scenario architecture. For Adding a new service scenario only one DIAMETER client/server node per cluster is simulated by setting branch probability values of authorization/termination calls on PEN and Relay components at 1 for one transition and 0 for all others (sum of probability values must equal 1). Here, it is important to note that the transition probability values on both load balancing components must be equally set for authorization and termination calls. For example, if there is a 100% probability that the Pen load balancer will forward authorization requests to DIAMETER client 1, the same setting must be applied for termination requests as well (Figure 20). This assures that both authorization and termination calls are processed by the same DIAMETER client/server which corresponds to the real architectural setup for this scenario.

Finally, in order to obtain performance prediction results for different usage scenarios Usage model parameters are set accordingly. Performance prediction analysis, which includes comparison of measured and simulated results is performed with different workloads (500, 1000, 3000, 5000 and 10000 users) for both authorization and termination scenarios. An example of the Usage model settings for a workload of 500 users is illustrated in Figure 20. The workload type parameter set to Closed specifies that 500 users perform the specified action and then re-enter the system after a think time of 100 seconds. Since we are only interested in performance results for the first 500 calls thinking time is set to a large value to omit the influence of users re-entering the system for the second time. The same logic is applied on all other simulation cases.

Table 12 and Table 13 summarize the performance results obtained both from measurements and prediction model for the specified scenarios.

Graphical comparison of the obtained results is given in Figure 20, Figure 20 and Figure 20. It is visible how prediction results obtained from the model follow the measurement results for smaller and moderate workloads. These workloads represent typical conditions under which our systems operate, and thus, are of special interest to us. For these workloads, the difference between experimentally measured values and predicted values is at most 10% and we consider that to be a very good result.
Figure 20: Branch probability settings for *Adding new service* scenario
Table 12: Performance results in milliseconds obtained from measurements

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Workload</th>
<th>500</th>
<th>1000</th>
<th>3000</th>
<th>5000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td></td>
<td>695.7</td>
<td>1280.65</td>
<td>5065.37</td>
<td>9490.4</td>
<td>26007.2</td>
</tr>
<tr>
<td>Termination</td>
<td></td>
<td>569.53</td>
<td>1123.25</td>
<td>3764.23</td>
<td>6456.6</td>
<td>18462.2</td>
</tr>
<tr>
<td>Total response time</td>
<td></td>
<td>1265.23</td>
<td>2403.9</td>
<td>8829.6</td>
<td>15947</td>
<td>44469.4</td>
</tr>
</tbody>
</table>

Table 13: Performance results in milliseconds obtained from model

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Workload</th>
<th>500</th>
<th>1000</th>
<th>3000</th>
<th>5000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td></td>
<td>780</td>
<td>1410</td>
<td>4680</td>
<td>7820</td>
<td>15670</td>
</tr>
<tr>
<td>Termination</td>
<td></td>
<td>610</td>
<td>1220</td>
<td>3670</td>
<td>6130</td>
<td>12350</td>
</tr>
<tr>
<td>Total response time</td>
<td></td>
<td>1390</td>
<td>2630</td>
<td>8350</td>
<td>13950</td>
<td>28020</td>
</tr>
</tbody>
</table>

Figure 20: Usage Model settings for workload of 500 users
Figure 20: Comparison between measured and predicted results for authorization scenario

Figure 20: Comparison between measured and predicted results for termination scenario
However, we also examined what happens in extreme conditions when CPUs in the DIAMETER extension are saturated all the time (or almost all the time). In such conditions, with a burst of more than 5000 parallel requests, there is clear differentiation between experimental and predicted results. It is obvious that prediction techniques aren't applicable in such cases. But, it is also necessary to take into consideration that in such extreme cases even experimental measurements differentiate significantly from a measurement to a measurement. Therefore, based on presented results, we can state that the prediction results obtained from the model are satisfactory for the evolution scenario of Adding a new service.

**Evolution scenario "Service implementation exchange"**
To be written / to be assessed.

**Evolution scenario "Changes to service deployment"**
To be written / to be assessed.

**Evolution scenario "Changes to hardware resources"**
To be written / to be assessed.

**Evolution scenario "Changes to usage profile"**
To be written / to be assessed.
5.4 Reliability prediction

Here, we discuss our current effort regarding the reliability experiments on the ENT demonstrator. It is important to note that for the telecommunications domain the term reliability is strongly connected to the term availability (which is of special interest to us). Depending on the scientific (and even engineering) field, reliability can be defined in different ways. One of the possible definitions is the ability of a system or a component to perform its functions under stated conditions for a specified period of time. This definition is quite appropriate for the telecommunications domain where the term availability is defined as the proportion of time in which a system is in a functioning condition. These definitions show tight correlation between the terms reliability and availability in the telecommunications domains. Both terms are reported as a probability.

We are interested in making our demonstrator highly available, meaning it has the property of being in a functioning condition (and servicing its users) 99.999% of the time. E.g. for a period of one year, such high availability would mean a downtime of just 6.05 seconds. Thus, in the following text we present our current experiments with the availability of the demonstrator.

Service Availability

In order to gain an overview on the service availability of the ENT demonstrator it is necessary to measure failover period when a service failure occurs. Failover management is provided by the high availability middleware OpenSAF, so we turn our focus to that part of the demonstrator. We chose a simple scenario in which we simulate DIAMETER client crash by crashing the relevant process and measure the service transfer time. Measurement results are shown in Figure 20 for 20 experimental measurements sequentially grouped into groups of 5 measurements showing the average value of service outage. Measurements performed for "out-of-the-box" OpenSAF settings are shown as the "3 ARP" line, while measurements performed for custom optimized OpenSAF settings are shown as the "1 ARP" line. When using "out-of-the-box" OpenSAF settings, service outage averages on 3 seconds. One second of that time is spent on IP address migration from failed to standby node. The other two seconds are spent on ARP (Address Resolution Protocol) address announcement which informs all neighbouring nodes that a standby node has taken over the IP address of the failed node. The cause for the two-second delay is found after closer inspection of the OpenSAF source code. After installing the virtual IP on the standby node, OpenSAF uses a system call to broadcast three gratuitous ARP requests (ARP announcements) on local network to announce the IP address change. Since ARP requests are sent with one-second interval, the system call takes two seconds to complete. During that time application execution is blocked.

![Figure 20: Failover time in the ENT demonstrator for 3 ARP requests and 1 ARP request](image-url)
However, since we use dedicated LANs for the clusters, we were able to modify the OpenSAF code to send only one gratuitous ARP request. Thus, in the case of 1 gratuitous ARP request, service outage averages at only one second.

Current measurement cases show potential for achieving high availability. If noted absolute times are written as availability percentage, the results can be expressed as 99.99999% availability (or "seven-nines" availability). However, this would imply a system running during a year with just one outage of 3 seconds. Our experiments are not designed to run for such prolonged time periods. Thus, we conclude that using OpenSAF as high-availability middleware is very promising and should be further investigated on a more complex system.
6 Conclusion
This section will be completed in the final version of this document.
7 Bibliography