

Project Deliverable D1.1


Requirements document - final version

Project name:	Q-ImPrESS
Contract number:	FP7-215013
Project deliverable:	D1.1: Requirements document - final version
Author(s):	Steffen Becker, Saša Dešić, Jens Doppelhamer, Darko Huljenić, Heiko Koziolok, Eckhard Kruse, Marco Masetti, Wladimir Safonov, Ivan Skuliber, Johannes Stammel, Mircea Trifu, Johannes Tysiak, Roland Weiss
Work package:	WP1
Work package leader:	ENT
Planned delivery date:	M12
Delivery date:	M12
Last change:	2009-April-30
Version number:	1.31

Abstract

This document defines high-level needs and features of methods and tools of the Q-ImPrESS project. It focuses on quality impact prediction needs required by the stakeholders and the target users, and why these needs exist. The details of how Q-ImPrESS tools and their use could fulfill these needs are detailed in descriptions within this document.

Keywords: requirements specification, stakeholders, users, user stories

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

Revision history

Version	Change date	Author(s)	Description
0.1	2008-03-07	WP1	Initial document designed at Prague meeting.
0.2	2008-03-31	WP1	Initial Introduction, Stakeholder and User descriptions.
0.3	2008-04-18	WP1	Corrections applied to Introduction, newly written Business opportunity, follow-ups to stakeholder descriptions, user descriptions, functional requirements and non-functional requirements.
0.4	2008-05-06	WP1	Corrections applied to Business opportunity, follow-ups to stakeholder descriptions, user descriptions, user stories.
0.5	2008-05-09	WP1	Follow-ups to user stories, revised requirements, minor corrections to other chapters.
0.51	2008-05-14	WP1	Minor Revision of user descriptions and stories, minor revision of requirements, inclusion of separate user story into the main document.
0.6	2008-05-21	WP1	Revision according to comments from WP2 representatives.
0.7	2008-05-26	WP1	Additional user story by ABB, minor changes to user stories, revision of functional and non-functional requirements.
0.8	2008-05-29	WP1	Revision according to comments from WP2.
0.85	2008-06-02	WP1	Revision according to comments with regard to reasons for deliverable deadline extension
0.9	2008-06-06	WP1	Updates to functional requirements and product overview. Minor revision of the rest of the document.
0.95	2008-06-06	WP1	Added research partners user story
0.99	2008-06-11	WP1	Minor revisions of entire document
1.00	2008-06-13	WP1	Final revision of entire document
1.01	2008-11-27	WP1	Changes according to meeting at Vasteras, additions to user stories
1.02	2008-12-01	WP1	Evolution scenarios, initial non-functional requirements, initial constraints, enumeration of requirements
1.1	2008-12-19	WP1	Final revision according to comments from all Q-ImPrESS partners
1.2	2009-04-03	WP1	Reviewers recommendations - initial update
1.25	2009-04-09	WP1	Reviewers recommendations - update
1.29	2009-04-20	WP1	Reviewers recommendations - finalization
1.3	2009-04-27	WP1	Update according to consortium comments
1.31	2009-04-30	WP1	Minor changes to Chapter 9


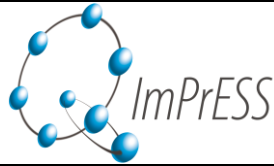
	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

Table of contents

Project Deliverable D1.1	1
Requirements document - final version	1
1 Introduction	5
1.1 Purpose	5
1.2 Scope	5
1.3 Definitions, Acronyms and Abbreviations	5
1.4 The process used for creating this document	6
1.5 References	8
2 Business Opportunity	8
2.1 Enterprise	9
2.2 Automation	9
2.3 Telecommunication	9
3 Q-ImPrESS method and tools overview	10
3.1 Service architecture modeling	10
3.2 Service architecture extraction	10
3.3 Quality impact prediction	10
3.3.1 Performance	10
3.3.2 Reliability	10
3.3.3 Maintainability	11
3.4 Trade-off analysis	11
3.5 Demonstrators	11
3.6 Method documentation	11
4 Stakeholder and User Descriptions	11
4.1 Stakeholder Summary	12
4.1.1 Automation system provider	12
4.1.2 Telecommunication system developer	12
4.1.3 Enterprise software developer	13
4.1.4 Research organization	13
4.2 User Summary	13
4.2.1 Product manager (optional user)	13
4.2.2 System architect	14
4.2.3 System engineer	14
4.3 User environment	15
4.4 User stories	15
4.4.1 General overview	15
4.4.2 Detailed description	17
4.4.3 Possible Evolution Scenarios	22
4.4.4 Additional information on automation systems	23
4.4.5 Telecommunications system developer additional information	29



4.4.6	Enterprise domain additional information	37
4.4.7	Research organization specific details.....	39
5	Functional requirements	41
5.1	Service Architecture Modeling	41
5.2	Service Architecture Model Extraction.....	43
5.3	Quality Impact Prediction	43
5.4	Trade-off Analysis	45
5.5	Tool Support	45
5.6	Demonstrators.....	45
5.7	Method Documentation	46
6	Non-functional requirements.....	46
6.1	Service Architecture Modelling.....	46
6.2	Service Architecture Extraction	47
6.3	Quality Impact Prediction	47
7	Constraints	47
7.1	Service Architecture Modeling	47
7.2	Service Architecture Extraction	48
7.3	Quality Impact Prediction	48
7.4	Trade-off Analysis	48
7.5	Tools Support	48
7.6	Scalability	48
8	Documentation.....	49
9	Requirements validation	49
9.1	Planned requirements validation per demonstrator.....	52
9.2	Demonstrator focus.....	57

1 Introduction

1.1 Purpose

The purpose of this document is to collect, analyze, and define high-level needs and features of software architecture extraction tools and quality prediction tools of the Q-ImPRESS project.

1.2 Scope

This document applies to the Q-ImPRESS FP7 project where it serves as requirements specification for the project. This document is prepared by WP1 as an input which will be used by WP2 to WP7 for design, implementation and validation of service architecture model extraction tools and quality prediction tools. The tools will support different approaches to architecture model extraction, quality prediction and automated quality assessment. Overall Q-ImPRESS project interaction with designated work packages is shown on Figure 1.

1.3 Definitions, Acronyms and Abbreviations

The Q-ImPRESS project aims at bringing service orientation to critical software systems like

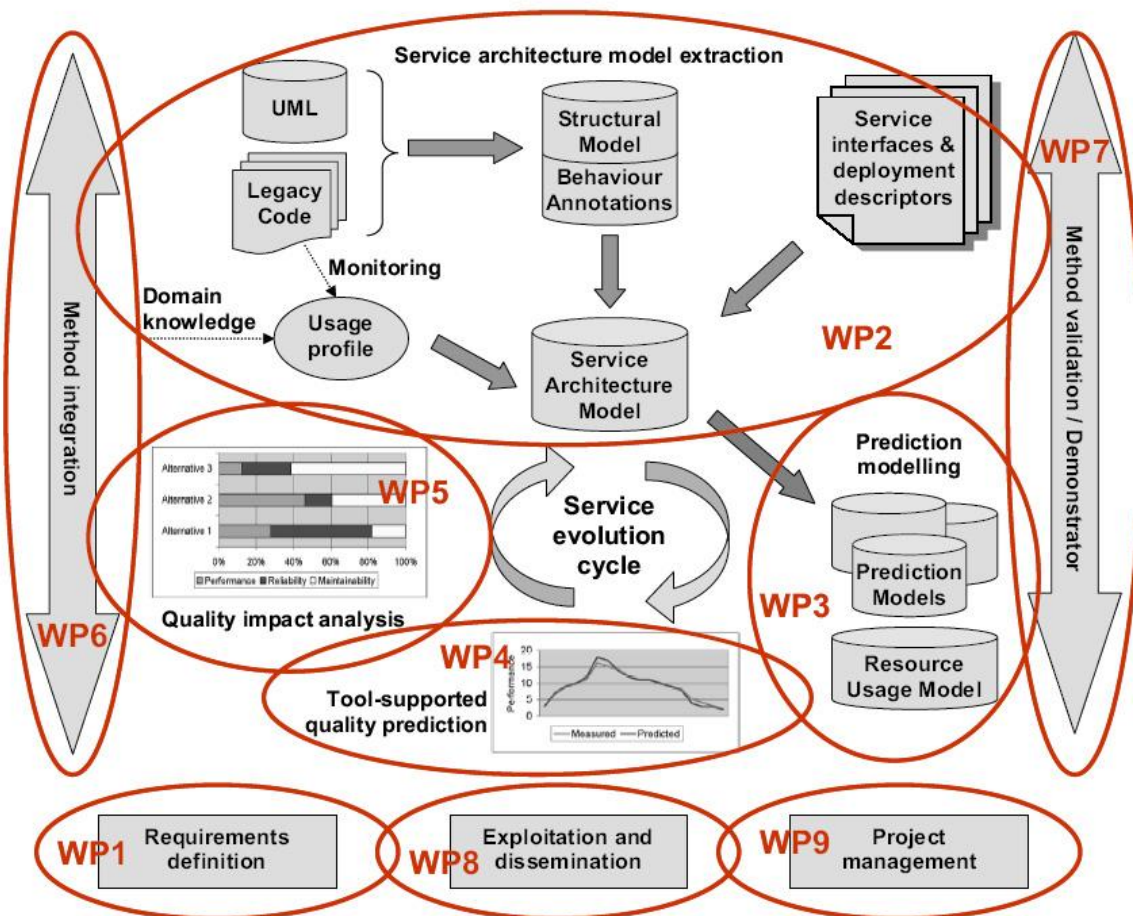
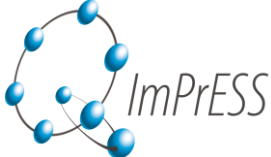


Figure 1. The Q-ImPRESS work package structure

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

industrial process control, enterprise systems, and telecommunication. Each of these three domains uses the same terminology for different concepts, architectures and implementations. This section explains common terminology used in the document.

"Black box" component - **a)** An old, proven component for which there is no information available, meaning its internal structure and behavior is unknown. "Black box" component's external behavior and performance can be measured in order to provide information needed by Q-ImPRESS prediction tools. Q-ImPRESS prediction tools use this information for analyzing "black box" components as if they were modeled from source code by Q-ImPRESS architecture extracting tools. **b)** Completely new, not yet implemented, component which has to adhere to certain performance requirements. Because this component doesn't exist there is no available information about it, meaning its internal structure and real-world behavior is unknown. However, there are performance requirements that this new component has to meet, so certain behavioral information can be provided. Also, there can be some higher-level design of the component, so certain manual modeling of component structure can be done.

"White box" component - a component whose internal structure and behavior is known and whose source code is available.

Legacy system - An existing system consisting mostly of old, proven "black box" components and, none or some, relatively newer "white box" components.

Non-legacy system - Either an existing system consisting entirely of "white box" components, or a completely new, not implemented system consisting entirely of "black box" components.

Service - A service is a deployed component, ready to serve requests.

1.4 The process used for creating this document

The process used for identifying and prioritizing requirements, as well as creating this document, suits the diversity of partners involved in Q-ImPRESS project. On the one hand, industrial partners that drive requirements specification, have proven requirements engineering processes in their organizations. These processes are relatively complex and very structured in regards to requirements engineering tools and methods. However, these processes differ among industrial partners. On the other hand, academic partners are familiar with different requirements engineering processes, but their use is limited with scope of specific research projects. Because of this, during the first WP1 meeting in Prague, we decided to use a lightweight process for requirements collection and prioritization. However, although lightweight, the process is based on experiences and knowledge of industrial partners from previous projects. We also decided to use the Vision document, introduced by the Rational Unified Process (RUP), as a structural basis for this Requirements document. Previous positive experience with the Vision document assured us with this decision. The Vision document, according to the RUP, contains a general vision of the project's core requirements, key features, and main constraints. The purpose of the document is to collect, analyze, and define high-level needs and features. It focuses on the capabilities needed by the stakeholders, and the target users, and **why** these needs exist. Thus, using the general guidelines and statements in the RUP Vision document, the structure of the Requirements document was created.

A simplified and conceptual form of the process used for identifying and prioritizing requirements in WP1 is presented in Figure 2. At the start of WP1, two branches of activities

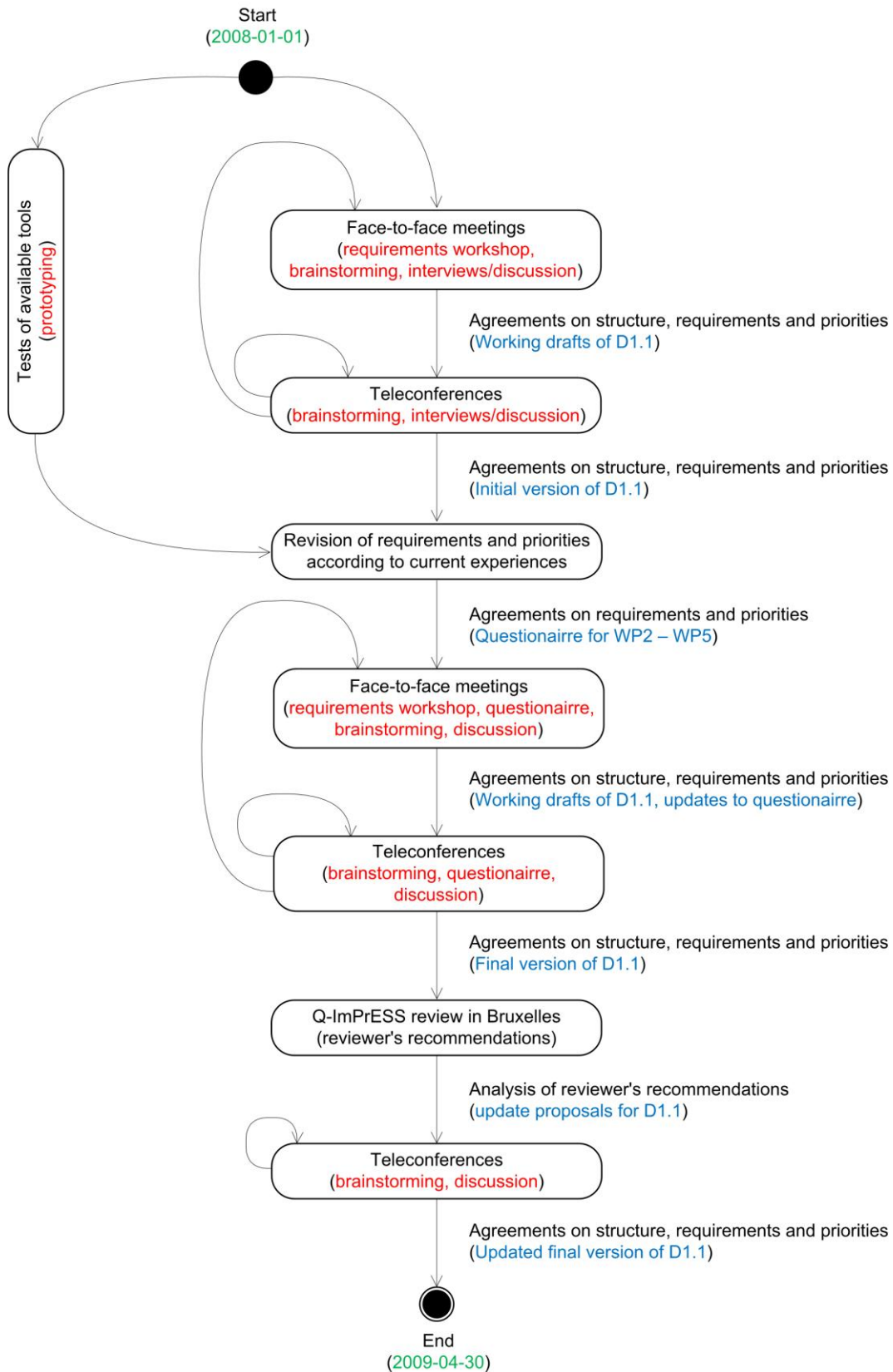
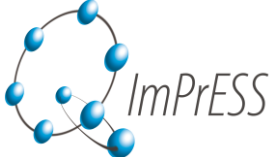


Figure 2. Conceptual overview of the process used for requirements elicitation

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

started. The right branch activities had the goal of producing initial version of D1.1. Typical requirements elicitation techniques like requirements workshop, brainstorming, interviews and discussion were used in face-to-face meetings and teleconference meetings. The goal was to identify and prioritize requirements that Q-ImPRESS methods and tools should fulfill, and that can be realized within the time-frame and scope of the Q-ImPRESS project.

The left branch activities used prototyping techniques in order to get familiarized with existing modelling and quality prediction tools. The knowledge and experiences gained from these activities helped with revision of requirements presented in initial version of this document toward requirements that are more feasible.

After the initial version of D1.1 was done, the work on the final version of D1.1 began. The established set of requirements and the priorities were modified according to experiences from prototyping techniques. Additionally, a questionnaire for WP2 to WP5 participants was formed, in order to get structured and detailed information regarding feasibility and effort needed to fulfill each of the requirements. Questionnaire contained all of the requirements from the initial version of D1.1. WP2 to WP5 members were asked to confirm if particular requirement is achievable, if its priority level is reasonable, if there are some constraints related to the requirement, and to specify coverage of requirements throughout the project and relationship with other deliveries in the project. The structure and information contained in the questionnaire enable us to also use it as a means for tracking of requirements fulfilment. Basic structure of the questionnaire, with omitted unnecessary information, is also used as a basis for the table that describes planned requirements validation per demonstrator (see chapter 9.1).

Different iterations of this questionnaire, alongside previously used techniques in face-to-face meetings and teleconferences, were used to refine requirements and their's priorities. These activities led to the final version of D1.1, which was submitted on December the 19th, 2009.

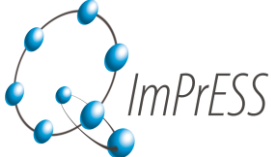
After the Q-ImPRESS project's first formal review, held in Bruxelles in February 2009, we received reviewer's recommendations on further updates of the document. Upon analysis of recommendations, we organized several teleconferences with common brainstorming and discussion techniques in order to update and complete the D1.1 according to the reviewer's recommendations.

1.5 References

- [1] "Q-ImPRESS - Quality Impact Prediction for Evolving Service-Oriented Software", FP7-215013
- [2] "Service Availability Forum", <http://www.saforum.org>
- [3] "Service Availability Framework (OpenSAF)", <http://www.opensaf.org>

2 Business Opportunity

In the domains targeted by Q-ImPRESS there is a strong trend towards using service-oriented architectures, with the goal to improve the systems with regard to adaptability, extensibility and re-configurability. On the other hand, often it is not clear how the evolution of the systems under the SOA paradigm might impact the different software qualities such as performance, reliability etc. However, to predict and optimize these qualities - with potentially different emphases/trade offs depending on the domain - is critical for the development of the systems. By taking into account the requirements listed in this document, the tools and methods to be developed by Q-ImPRESS will enable service oriented architectures with predicted end-to-end quality using analysis and simulation techniques for

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

quality impact analysis. This will help to understand the consequences of service changes, in the context of evolving service-oriented systems. Thus it is expected that development costs and risks can be significantly reduced during the evolution of the software systems.

The Q-ImPrESS methods and tools shall be able to cover a broad range of different application areas without major domain-specific adaptations. The three domains addressed in the Q-ImPrESS case studies are briefly described in the following sections, highlighting their differences and specific aspects.

2.1 Enterprise

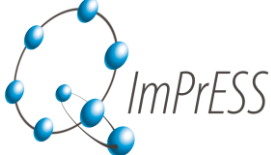
In the domain of Enterprise Applications, the introduction of SOA has already advanced and does not represent big technical innovation any more. Particularly this is the case, when the SOA paradigm is not only considered within the scope of web services, but distributed systems in general without respect to specific protocols (e.g. CORBA, SOAP). Nevertheless there is a variety of conceptions about the term SOA. Usually the introduction of a SOA is accompanied by organizational changes, e.g. to improve the responsiveness of the IT department and their provided services and applications with regard to changes in the business environment. This organizational aspect is not addressed by Q-ImPrESS, but the Q-ImPrESS prediction may support those aspects. Regarding the technical aspect, even today a variety of standards, methods and tools are provided for the introduction of a SOA. Basic and wide-spread principles are reusability (also surpassing the boundaries of departments and companies) and loose coupling. These principles have strong impacts on different quality attributes. For a concrete implementation, often there are different design alternatives, whose consequences are hard to estimate in advance. The objective for Q-ImPrESS is to eliminate these uncertainties by providing facilities for the evaluation of the impact of design decisions on the different quality attributes. In order to improve acceptance of these facilities by SOA architects and developers, an integration into the existing development process and toolchains is desirable.

2.2 Automation

Compared to other domains, in automation the adoption of SOA is comparably slow. Software systems in this domain are facing very tough requirements regarding e.g. reliability, performance, hard or soft real-time. These requirements are often perceived as contradictory to loosely coupled, service-oriented systems. Also there are potentially large legacy systems, which have to be dealt with. If the Q-ImPrESS tools and methods take these requirements into account, there is a great chance to introduce SOA approaches into automation applications and benefit from more flexibility, reduced maintenance efforts, predictable system evolution etc.

2.3 Telecommunication

In some segments of the telecommunications domain adoption of SOA is much faster than in the other parts. Service delivery segment already benefits from SOA principles and techniques. However, core network segment like routing and switching still relies on proven legacy systems. On the one hand, core network requires predictable end-to-end quality of service. On the other hand, legacy systems are proven performance-wise solution for delivering that requirement.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

In order to adopt SOA into the core network segment, Q-ImPRESS tools must successfully merge legacy systems with SOA paradigm, principles and techniques. Potential benefits of applying SOA into the core network are many. Characteristics like distribution, loose coupling and reusability would facilitate maintenance and improvement of existing systems as well as rapid development of new ones. Existing functionality could be easily extended, especially toward the IP (Internet Protocol) world.

3 Q-ImPRESS method and tools overview

3.1 Service architecture modeling

One of the central results of the project is the service architecture meta-model specification, which describes the structure and semantics of the service architecture models used to store abstract representations of the software systems under scrutiny. The meta-model description is complemented by a visual editor capable of modeling legacy systems and non-legacy systems with all the attributes required for quality impact prediction. The above mentioned editor will also be used to modify and refine the models produced by the service architecture extraction tool mentioned in section 3.2.

3.2 Service architecture extraction

Service architecture extraction will be realized as a tool that is used on existing code in legacy systems (C, C++, Java). The tool statically analyses source code to construct service architecture models. It provides a configurable abstraction from the source code to high level components and services so that large systems become manageable.

The tool supports reverse engineering of Java/C/C++ code (initially) and C#/.NET code (optionally). It is integrated with the modeling tool for seamless usability (i.e., its output format can be read by the modeling tool) and can be extended to cover other component technologies.

3.3 Quality impact prediction

The main goal of quality impact prediction is the decomposition of an abstract notion of quality down to the level of software metrics for performance, reliability and maintainability. Different quality prediction models will be described in order to provide predictions for maintainability, performance, and reliability. Using these models, different quality prediction methods perform quality impact prediction analysis for the defined service architecture models.

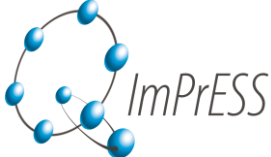
The quality impact prediction tools will implement aforementioned methods and will visually present the results of quality impact prediction analysis.

3.3.1 Performance

After examination of the initial versions of the user stories presented in this document, we assess the software metrics for performance as follows: CPU load, memory consumption, HDD load, network load and capacity (number of users supported by system without degradation of quality of experience).

3.3.2 Reliability

Similarly, we assess the software metrics for reliability as follows: mean time between failures (MTBF), mean time to failure (MTTF) and mean time to repair (MTTR).

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

3.3.3 Maintainability

Quality prediction methods using the quality prediction models, and quality prediction tools that implemented the methods, enable tracking of the evolution of a system efficiently to predict the maintenance effort required to implement the required changes.

3.4 Trade-off analysis

The method of trade-off analysis will allow specification of preferences for specific properties, such as performance or reliability (e.g., main emphasis on performance instead of reliability). This method will be implemented in quality impact prediction tools where the quality impact prediction results and trade-off analysis results will be presented in an integrated fashion.

3.5 Demonstrators

Demonstrators are based on practical industrial scenarios and present an insight into possible use of SOA-based systems as an extension of functionality of legacy systems. Demonstrators contain documented evolution scenarios and code examples that show applicability of Q-ImPrESS methods and tools in software evolution. Demonstrators are used as proof of concept and for dissemination of the Q-ImPrESS project's results.

3.6 Method documentation


Q-ImPrESS tools and methods are documented in sufficient manner to be used by the users described in section 4.2. Application domain stakeholders present domain specific guidelines and/or handbooks that facilitate adoption of Q-ImPrESS'es tools and methods into existing development processes.

4 Stakeholder and User Descriptions

Stakeholders are interested in applying and developing Q-ImPrESS tools which can be introduced into existing processes within their respective organizations. Stakeholder's intention is to reduce development costs and risks by using the tools in the evolution of the software systems under development. Through Q-ImPrESS tools, stakeholders want to facilitate adoption of SOA principles and techniques.

There are four stakeholders in the Q-ImPrESS project. The first three stakeholders are the domains where critical software systems are oriented toward services - automation system provider (represented by ABB), telecommunications system developer (represented by Ericsson Nikola Tesla d.d.) and enterprise software developer (represented by itemis). Critical software systems of these domains share a need for predictable end-to-end quality of service, but also need to evolve over their long lifetimes. The fourth stakeholder is research organization which provides its knowledge in order to develop methods and tools for the prediction of quality of service attributes of service-oriented software systems, grouped in WP2 through WP6. Differences between each stakeholder are given in subsection 4.1 (stakeholder summary). Each of these stakeholders will validate Q-ImPrESS tools and methods in their demonstrators within WP7. Each stakeholder has its own success criteria which represents general direction of how a stakeholder will evaluate Q-ImPrESS tools and methods.

The Q-ImPrESS project targets stakeholder's needs by providing a method to allow users to foresee the impact of design decisions and evolutionary changes to the system on overall

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

quality of service and different internal quality properties (e.g. performance, maintainability). The method will be incorporated in different architecture model extraction tools and quality prediction tools used by users of the Q-ImPRESS project.

Users of the tools produced as part of the Q-ImPRESS project are employees within aforementioned domains. Descriptions of users are given in subsection 4.2 (user summary).


4.1 Stakeholder Summary

4.1.1 Automation system provider

Stakeholder description	The automation system provider wants support for managers responsible for some product/system/solution, which often is a combination of hardware and software. The support is provided by Q-ImPRESS tools that enable him/her to manage development, evolution and maintenance of the product, to achieve high product quality (in terms of functional and non-functional properties, as required by the market), while keeping costs low. The tools will allow him to predict important quality attributes of typical automation software systems and to assess the consequences that system changes would have for these attributes. The tools might not be used by the managers themselves, but rather by the R&D teams. However, the managers' decisions are based on input from the R&D teams (partially) based on the information acquired by application of the Q-ImPRESS tools. The manager's primary interest is to release a high-quality product on time and on budget.
Success Criteria	The tools can be applied to the selected case study, i.e. the functional and non-functional requirements are met. The results produced by the tools can be used for making design decisions in development projects.

4.1.2 Telecommunication system developer

Stakeholder description	This stakeholder wants to deploy Q-ImPRESS tools to the organization in order to develop a quality SOA product - either through a new functionality (e.g. new protocol within the telephony exchange, extension of existing functionality within the telephony exchange, etc.) or improve existing one by applying SOA principles (e.g. reduction of maintenance cost of existing telephony exchange system, etc.). In the project the stakeholder will define requirements imposed on Q-ImPRESS tools which should facilitate development of business cases for new functionality, extension of existing functionality or improvement. Later validates developed Q-ImPRESS tools for certain business case. The stakeholder introduces telecommunications domain with specific protocols, programming languages and architectures.
Success Criteria	Q-ImPRESS tools fulfill defined requirements, or there is no significant deviation from defined requirements. Also, one of the key issues is adaptability of the tools to the existing processes and tools being used in standard development.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

4.1.3 Enterprise software developer

Stakeholder description	<p>itemis provides consulting services with expert knowledge in the areas of model driven architecture (MDA) and service oriented architecture (SOA) mainly working for corporate enterprise customers. itemis has a need for a toolset, which assists the development of service oriented software, mainly for two objectives:</p> <ul style="list-style-type: none"> • To enable the prediction of quality of service for newly developed and evolving service oriented software systems in early stages of development, thereby enabling a better production quality of such systems. • To be able to offer professional training and consulting in the use of a development toolchain which supports the development of robust and mature SOA solutions including guaranteed end-to-end quality and prediction of quality of service.
Success Criteria	The functional and non-functional requirements of the developed tools, methods and workflows are met. The tools can be applied to the Enterprise SOA showcase. The results produced by the tools assist design decisions in development projects.

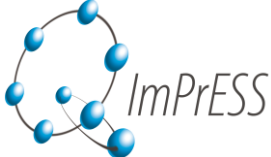
4.1.4 Research organization

Stakeholder description	<p>Stakeholder “Research organization” is used in this document to describe all partners in Q-ImPRESS project involved in the design and the implementation of the Q-ImPRESS method and tool chain. Research partners involved in the Q-ImPRESS project provide their knowledge in order to develop methods and tools for the prediction of quality of service attributes of service-oriented software systems. Existing approaches within the research community are to be adapted, extended and implemented in the project.</p> <p>The participating research partners offer long experience in the area of software quality assessment, performance and reliability prediction, maintainability evaluation and the development of reengineering methods and tools.</p> <p>This stakeholder has two major objectives:</p> <ul style="list-style-type: none"> ○ To setup a comprehensive landscape of quality evaluation techniques and trade-off analysis method. ○ To build the necessary infrastructure, models and reengineering tools needed to evaluate the methods.
Success Criteria	Development of the Q-ImPRESS method and supporting tools.

4.2 User Summary

4.2.1 Product manager (optional user)

Description	Product manager is a business level user with limited technical knowledge and thus, uses Q-ImPRESS tools in a very simple manner. In spite of limited technical knowledge, with the help of System architect
--------------------	--

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

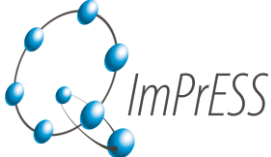
	<p>he can create a model of a simple (sketch) system that consists of one or a few very simple "black box" services. These "black box" services are described by very rough performance information that Product manager obtains from System architect. Based on this performance information, and with the help of System architect, Product manager can perform basic simulations of a simple (sketch) system. With the sketch system and its roughly predicted performance, Product manager tries to find high-level description for new SOA based functionality, extension or improvement of existing functionality with favorable cost/gain ratio. According to predictions received from Q-ImPrESS tools, Product manager develops high-level description with favorable cost/gain ratio. The description contains information which can be translated into basic technical requirements with the use of metrics such as response time, number of concurrent users, etc.</p>
Success Criteria	1. Developed high-level description is successful if Q-ImPrESS tools can provide good enough prediction upon which (re)implemented systems and services behave according to prediction.
Produced deliverables	<ol style="list-style-type: none"> 1. High-level description 2. Basic technical requirements expressed as response time, number of concurrent users, etc.

4.2.2 System architect

Description	<p>System architect is a user with deep technical knowledge about system's internal structure, inner behavior and interactions with other systems. He completely utilizes Q-ImPrESS tools in order to fulfill basic technical requirements received from Product manager. System architect defines conceptual architecture of non-legacy systems that (re)implement systems and services which fulfill basic technical requirements defined by Product manager. Conceptual architecture consists of defined "black box" services (e.g. interfaces with no implementation) and "white box" services (already implemented services that are (re)used in conceptual architecture). Information in conceptual architecture are requirements for System engineer.</p> <p>System architect validates (re)implemented non-legacy systems according to measurements information provided by System engineer.</p>
Success Criteria	1. Success if Q-ImPrESS tools provide good enough qualitative and quantitative prediction of (a possible implementation of) conceptual architecture's functional and quality properties like performance, reliability, maintainability, etc.
Produced deliverables	<ol style="list-style-type: none"> 1. Conceptual architecture to be implemented by System engineer. 2. Qualitative and quantitative predictions at the conceptual architecture level (performance, reliability, maintainability, etc.).

4.2.3 System engineer

Description	System engineer is a user who manually implements services and integrates them into a functional whole (a system) according to conceptual architecture defined by System architect. Where possible, he
--------------------	--

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

	uses Q-ImPRESS tools for analyzing and predicting functional and quality properties of services that he implements. System engineer uses different proprietary tools for performing measurements on implemented services. He puts measurements information into Q-ImPRESS tools where it is used by System architect.
Success Criteria	1. Success if it is possible to manually implement services and systems according to conceptual architecture defined by System architect
Produced deliverables	1. (Re)implemented services according to conceptual architecture. 2. Measurements information regarding services' and system's performance.

4.3 User environment

All three users use Q-ImPRESS tools on a Windows/Linux machine that is separated from modeled system. All the information needed by the tools is gathered with proprietary tools.

4.4 User stories

4.4.1 General overview

General overview of the common workflow for user stories is presented on Figure 3.

The start of the system development with Q-ImPRESS tools process is initiated by management decision (this action is governed by arrows 1 and 1'). The management decides to grant certain budget for development of some new functionality, system or application, or redevelopment of some existing functionality, system or application. Either way, the process is initiated by that management decision. Depending on management's decision, the process can be started by Product manager (arrow 1) or System architect (arrow 1'). Hereafter is described start of the process by Product manager, later by System architect (after description of actions governed by arrow 4).

After receiving the decision from management (arrow 1), Product manager starts working on high-level description of the functionality, system or application to be developed (hereafter noted only as "system"). Because Product manager is a business level user with limited technical knowledge, he uses Q-ImPRESS tools in a very simple manner. With the help of a System architect, he creates a model of a simple (sketch) system that consists of one or a few very simple "black box" services. These "black box" services are described by very rough performance information (e.g. load information, capacity, response time, etc.) that Product manager obtains from System architect. Based on this performance information, Product manager performs basic simulations of the simple (sketch) system (arrow 2). After simulation, Product manager analyses results obtained by the simulation with the emphasis on a cost/gain ratio. If the results aren't satisfactory, Product manager reiterates the high-level description of the (sketch) system (arrow 3). If the results are satisfactory, then Product manager sends high-level system description, simple description of the resources, mapping between system's services and resources, and used usage profile in the simulation to System architect (arrow 4). All these documents can be thought of as much simpler and quite incomplete version of the documents prepared by System architect. Documents prepared by Product manager serve as an (somewhat refined) initial idea description that is used by System architect in order to check its true feasibility. It should be noted that wherever black box services are part of a model, predictions for a changing usage profile assume that the characteristics of the black box services have not changed under the new profile.

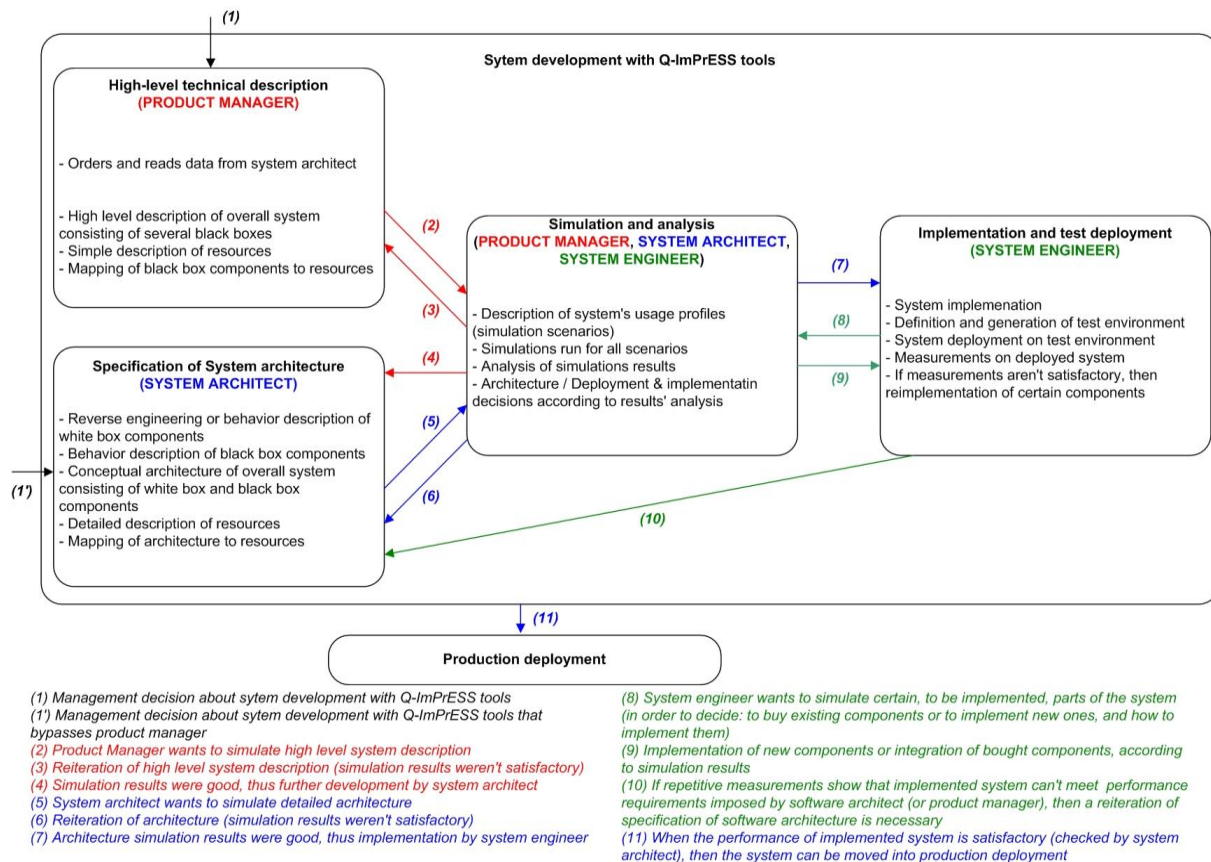
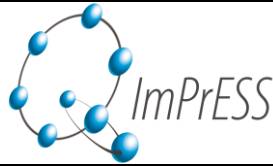
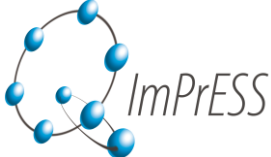


Figure 3. Common workflow

After receiving the information from Product manager (arrow 4), or after receiving the decision from management (alternative start of the process - arrow 1'), System architect starts specifying the system architecture. The architecture is specified on conceptual level and consists of "black box" services and "white box" services. System architect provides detailed description of "black box" services either by manually entering data (needed by the model to describe a service) based on his judgment or by entering data based on some previous measurements data provided by System engineer. Because "White box" services are already existing parts of code, their behavioral and structural data can be reverse engineered from the code and put into architecture description. However, if the System architect wishes so, he can describe the "white box" services the same way as "black box" services - by entering data (needed by the model to describe a service) based on his judgment or by entering data based on some previous measurements data provided by System engineer. The work on conceptual architecture is done when the architecture (static structure like components and services, connectors, etc., and dynamics like activity diagrams) is described, detailed description of resources needed to run the system is done and when the architectural description of the system is mapped to resources. Then, a simulation of conceptual architecture can proceed (arrow 5).

Before starting the simulation of conceptual system architecture, System architect describes usage profiles (how the system will be used). Then simulation starts and results are obtained from it. If the analysis of the results isn't satisfactory, then System architect reiterates the specification of conceptual system architecture (arrow 6). If the results are satisfactory, then the system can be implemented in code by System engineer (arrow 7). System architect sends conceptual system architecture description, detailed resources description, mapping between

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

resources and architecture, and expected implemented system's behavior characteristics (quality attributes predictions obtained by analyzing simulation of system architecture) to System engineer.

After receiving the information from System architect, System engineer starts manually implementing the system on code level. As well as implementing the system, System engineer defines and manually generates test environment on which the system will run. Information regarding resources and mapping between architecture and resources is used for definition and manual generation of test environment. Also, as certain parts of the system are developed, System engineer deploys them on the test environment.

System engineer implements service(s) from scratch, by using existing service(s), or even by using third-party off-the-shelf component(s). In order to choose one of these three possible approaches for implementation of each service, System engineer uses Q-ImPRESS tools for obtaining results of a simulation that predicts how a possible implementation approach will work within the overall system (arrow 8). System engineer has an in-depth knowledge of how to-be-implemented services should perform and work, so he can describe them in detail as "black box" services in Q-ImPRESS tools as an extension of a part of the conceptual architecture designed by System architect. Based on obtained and analyzed simulation results (arrow 9), System engineer chooses appropriate approach for implementation of the service (implementation from scratch, by using existing service(s), or by using third-party off-the-shelf components). When the services are implemented, System engineer performs measurements (with proprietary tools) in order to gain service performance information within the entire system. If the measurements aren't satisfactory, then System engineer reimplements services that don't conform to requirements.

It is possible that reiterative process of implementation, simulations and measurements doesn't result in implemented system that behaves to expected behavior parameters provided by System architect. Then, it is necessary to reiterate the entire process from specification of system architecture (arrow 10). This reiteration is necessary in order to get the system that behaves according to expected behavior parameters provided by System architect.

Quality of the overall system is constantly checked by System architect. When the measurements' results show that the implemented system's quality is according to expected quality parameters provided by System architect, then System architect decides that the system can be transferred to production deployment (arrow 11).

4.4.2 Detailed description

Introduction

The following vision describes the workflow associated with the toolchain and the methods that are to be developed within the scope of the Q-ImPRESS project. The typical users for these methods want to develop or evolve a service oriented software system. They expect to be assisted by the Q-ImPRESS methods and tools. They wish to be able to use the domain language, formalisms and artifacts they are already used to, while only having to adjust to new formalisms and methods where absolutely necessary.

There are three kinds of typical users:

- The **Product manager** is a business level user with limited technical knowledge. He has a set of new business level requirements, which are to be fulfilled by the system in development.

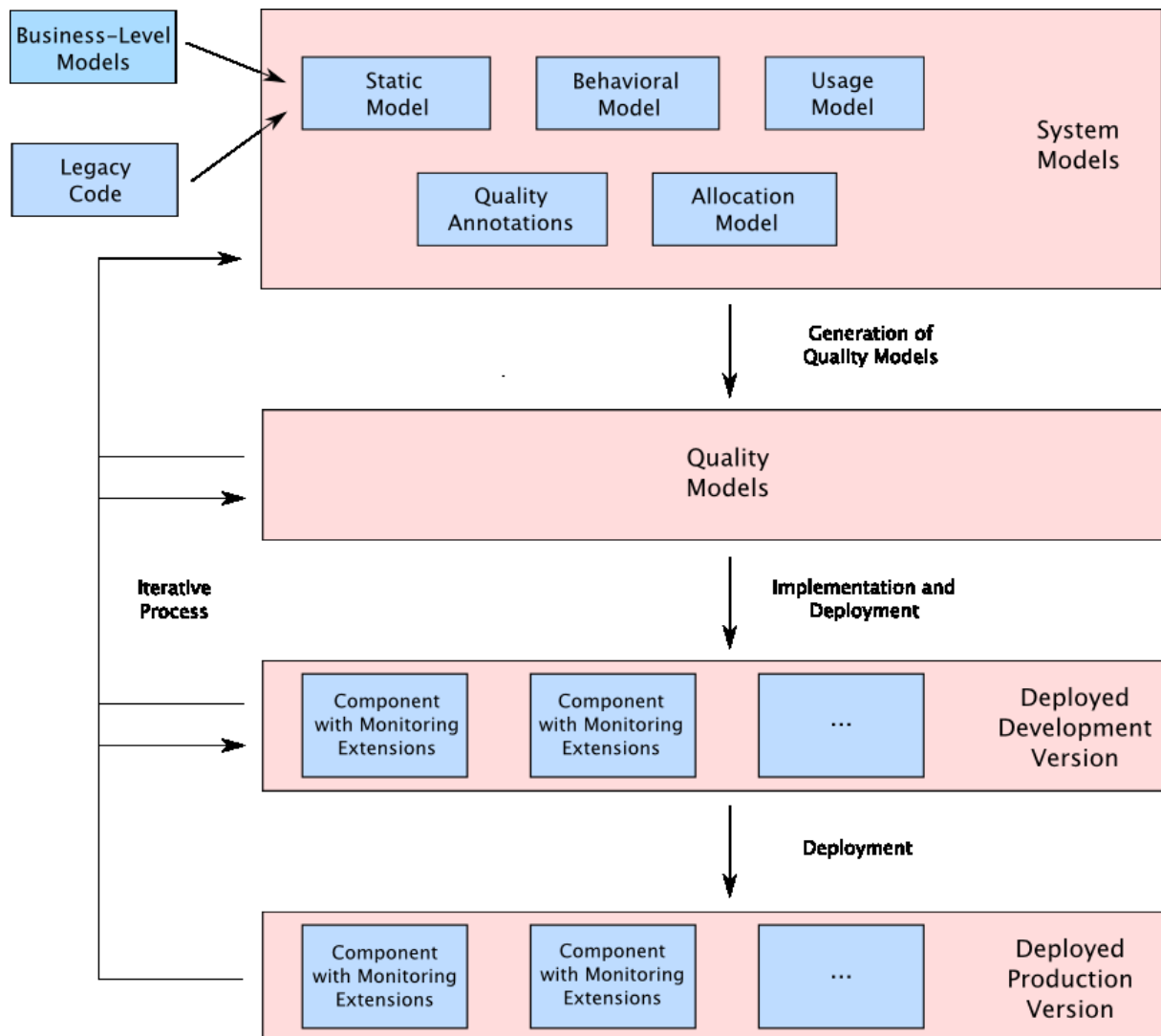


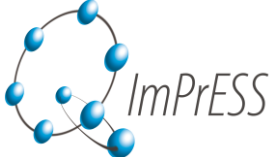
Figure 4. Artifacts in the System Development Process supported by Q-ImPRESS

- The *System architect* has deeper technical knowledge than the Product manager. Based on the Product manager's requirements he defines the overall conceptual architecture of the system in development.
- The *System engineer* implements system services and integrates them into the system under development according to the conceptual architecture defined by the system architecture.

The focus of this detailed description is the overall system development workflow, not the scope of Q-ImPRESS within this workflow. Wherever details are not within the scope of Q-ImPRESS, they will be marked accordingly.

Editing and importing system models

The Q-ImPRESS methods and toolchain follow a model-based approach. Within this approach a series of models are compiled which describe the static structure and the behavior of both the system and its infrastructure.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

The development process begins with the *Product manager*, who has a set of new requirements which need to be implemented in a new or existing system. He therefore creates the **business model**. This model contains a coarse-grained description of the static structure and the behavior of the system from the business point of view, i.e. without any focus on technical details.

The *System architect* consequently refines and extends this model, adding technical solutions and more detail. The resulting models, the so-called “**Static Model**” and “**Behavioral Model**”, are the base for a prediction of the system quality. Depending on the best fit for Q-ImPRESS, these models may be in graphical or textual form (e.g. in a domain specific language). The Static Model and Behavioral Model might also be created in a different notation and afterwards transformed into the Q-ImPRESS specific model notation (not within the scope of Q-ImPRESS).

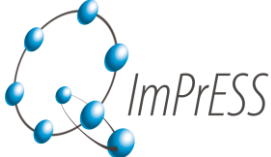
Parts of the Static Model and the Behavioral Model can be created by **reverse engineering** existing legacy code. The System architect imports the code into the tool supplied by Q-ImPRESS, which analyzes the code. The result of this analysis can be a set of Static and Behavioral models. If, after the reverse engineering process, the legacy code is changed, it is possible to repeat the reverse engineering. The System architect's manual additions to the older version of these model parts are automatically added to the new results where possible. For manual additions, which cannot be included automatically, “**ToDos**” are generated, which remind the System architect to check, if these additions still need to be implemented.

A parallel task is the description of system resources. This task is fulfilled by the Product manager and the System architect. The Product manager concentrates on a high-level description of resources from the business point of view. The System architect refines and extends this model, gathering information about the physical elements of the system environment (hardware, CPU, memory, etc.) and the logical elements involved in the system execution (queues, containers, topic, etc.). This information is notated in the system “**Allocation Model**”. Once this information is captured to a certain level, the Product manager and System architect can start mapping services of the Static Model to resources. As before, the Product manager concentrates on a high level of abstraction, while the System architect adds technical detail to the resource mapping. This mapping is also described in the Allocation Model.

For the system quality to be predicted, the Product manager and System architect also need to gather the anticipated system usage profile. They formally describe the usage estimations in the “**Usage Model**”. Besides average values this model can also contain ranges and probabilistic values. This usage model can also be used within the scope of model-driven testing.

Finally, the quality attributes are notated as “**Quality annotations**”. Business level quality attributes are defined by the Product manager, while the System architect refines and extends these, defining quality attributes on a technical level. After the simulation and measurements (see next chapters) have been completed, the simulated and measured results are annotated with the given constraints, so the Product manager and System architect are able to check whether the system in development adheres to their quality requirements.

The Product manager and System architect are free to develop alternatives for any of the aforementioned models or parts of these models. For each model, artifact subsets can be defined. Subsequently, the Product manager and System architect can create alternative models by rearranging and composing the predefined artifact subsets.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

Every time a model is changed, the model consistency is automatically checked. If a change implicates that other models need to be changed, “ToDos” are generated, reminding the editing user to implement all necessary changes.

All meta-models for the models created by the System architect are influenced by (evolving) standards such as the upcoming OMG SOA profile, SCA and Enterprise Application Integration (EAI) pattern catalogs. It should be possible to write transformations from and to standardized models to ensure that artifacts created for the workflow described in this document are reusable within other scopes. Also, the System architect can concentrate on defining the system’s static structure and behavior without having to think about quality prediction specifics at this point. He can use typical patterns and pattern expansions to specific domains in order to map domain specific technical views to the quality models (which are introduced in the next chapter).

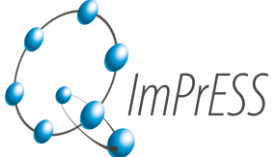
Quality of Service Prediction & Analysis

The system models created in the previous steps can now be used for the prediction of performance and reliability. For this the System architect triggers the transformation of the system models into the *quality models*. The quality models merely extend the existing system models by adding templates which enable the System architect to specify quality attributes of the system services.

Having the quality models available, the System architect now starts with the specification of quality attributes of the system services by extending the generated templates. This data includes a detailed behavior description of the services parts which would have an impact on the resource usage of the system during its execution. The System architect's input in the quality models is persistent. This means, whenever new quality models are generated from changed system models, the System architect's manual additions are automatically added to the newly generated quality models where possible. If the addition of these persistent parts is not possible in an automatic way, “ToDos” are generated assisting the System architect to fix this problem manually.

Proceeding to the next step, the System architect starts the prediction algorithm which triggers the process of system simulation. The runtime of this activity can be limited by the System architect. As soon as results of the simulation are available, they can be browsed by the System architect and visualized on the dedicated charts. The corresponding quality attributes values will be simultaneously depicted on the diagrams to simplify the analysis of the system simulation and to allow for reasoning about the anticipated system behavior. The results of simulation can be stored externally in a unified form, which is common to all tools developed within the scope of Q-ImPrESS. This enables the System architect to compare multiple simulation runs. Furthermore, the monitoring results of test runs using the real system (see next step) are saved using the same format enabling comparison of simulations with test runs.

By analyzing the simulation results, the System architect weighs the alternative design possibilities of the system and decides to go into the next iteration. During this step alternative designs of system models will be provided by the System architect and stored separately. The ability to model variants is required later for the trade-off analysis between several design alternatives to be able to switch to a particular system model. During the model transformation any existing quality model should be reused as far as possible to avoid rewriting of the resource relevant specification for the services. Now the prediction algorithm will be started again delivering simulation results for the alternative system design. This enables System architect to conduct a trade-off analysis between several design decisions.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

The simulation can be also triggered by the Product manager. He is assisted by the System architect, who enters the relevant data into the Q-ImPrESS tools. The focus of simulation runs triggered by the Product manager is not the evaluation of the detailed system architecture. A simulation triggered by him aims at evaluating the overall system performance. The results from this simulation run are presented at a higher level of abstraction, which allow the Product manager to verify that all quality business requirements are adhered to.

Implementation, Deployment and Monitoring

Once the System architect is satisfied with the predicted system quality of service, the implementation of services can start. He annotates elements of the Static Model with links to their implementation. Certain elements within this model can act as “black box” services. These elements represent services which are not yet implemented, or legacy services for which there is no model or code available. The System architect annotates these elements with a probabilistic performance and resource usage profile, so the behavior of these services can be emulated. The implementation of elements of the Static Model is performed by the *System engineer*.

As soon as all elements of the Static Model are either implemented or annotated with a behavioral description, the System architect and System engineer can automatically deploy the system to the target platform or an infrastructure equivalent to the target platform for testing purposes. (Note: The creation of the deployment scripts is out of scope of Q-ImPrESS). Black box services are implemented by dummy services, which emulate the behavior as described in the corresponding parts of the Static and Behavioral Model.

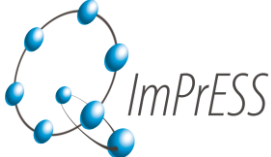
In order to provide means for monitoring the system, monitoring code is automatically integrated into the deployed system. This code facilitates the use of external live monitoring applications. It is important to note that the integration of monitoring code is not seen as a major requirement. However, certain parts of it might need to be done within the project.

The deployed system also provides means for measuring the same quality attributes as collected for the system simulation in the previous step. As before, these attributes are stored in the same standardized format. Subsequently, the System architect and System engineer compare the measured results to those obtained by the simulation. The user interface used for the analysis of measured test runs is identical to that of simulated runs. While the testing focus for the System architect lies on the overall performance of the system architecture, the focus for the System engineer lies on the adherence of services implemented by him to the defined quality attributes.

Since all models and their variants are stored in a versioned manner, the System architect proceeds by measuring the quality attributes for an alternative configuration of the system, which contains different design approaches. He thereby verifies his design decisions, which were previously only based on simulation runs. The System engineer on the other hand is able to compare different implementations for services implemented by him.

Further Iterations

The aforementioned workflow allows for iterations during the whole process. This means, any user can go back to any previous step at any time. Manual additions to artifacts are always persistent (as described in section “Quality of Service Prediction & Analysis”), so the users can reuse all of their manual work, even if parts of the model have been changed (Forward Engineering).

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

After the completion of several iterations, the Product manager, System architect and System engineer (hopefully) are satisfied with the quality of the newly developed system. The System architect can now generate a distribution version of the system. This version, compared to the test deployment version, does not contain any means for measuring and storing quality attributes, but still facilitates the use of external live monitoring applications (outside of the Q-ImPrESS scope, also see above). The Product manager and System architect can now deploy this distribution version to the target infrastructure. The facilities for the generation and deployment of a distribution version are to be developed outside of the scope of the Q-ImPrESS project.

4.4.3 Possible Evolution Scenarios

The System Architect will be able to predict the consequences on the Quality of Service resulting from the following evolutionary changes to the system:

SOA System Implementation

The implementation of a service-oriented system may be changed. This includes the following special cases:

- **New Service:** A new service is written from scratch.
- **Internal changes to services:** The implementation of one or more services may change without changes to the external interfaces provided by the affected services.
- **Service Evolution:** The implementation of services may change in a way that also affects the external interfaces provided by the changed services.
- **Service Granularity:** An atomic service (meaning a service which does not use other services to compute its results) is changed in a way that it now uses one or more auxiliary services to deliver its functionality.
- **Service Wiring:** The wiring between services is changed (e.g. introduction of a load-balancer and multiple service instances).

Evolving Deployment / Allocation

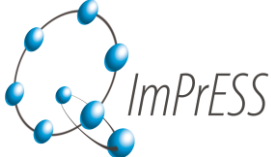
The deployment of a service-oriented system may be changed, which also may have consequences on the system's Quality of Service. These changes may concern the hardware as well as the system's underlying middleware.

- **Hardware:**
 - **Evolving hardware resources:** The characteristics of the underlying hardware resources may change (e.g. CPU, disk, memory, network throughput, etc).
 - **Evolving hardware allocation:** The allocation of services on specific hardware resources may change, e.g. two services which formerly ran on the same node now run on two separate nodes.
- **Middleware:**
 - **Evolving middleware:** The underlying middleware may evolve, e.g. a new version of the middleware is introduced.
 - **Evolving middleware deployment:** The deployment of services on specific middleware resources may change, e.g. two services which formerly ran on the same middleware container now run on two separate containers.

Evolving Usage Profile

The usage profile of the service-oriented system may change, which can also have consequences on the system quality. This includes the following evolution scenarios:

- **Number of users:** The number of users concurrently working with the system may evolve over time.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

- **Characteristics of usage scenario:** Certain characteristics of the usage scenario, which have an impact on the quality may change (e.g. average file size for a SOA-based “file upload center”).

4.4.4 Additional information on automation systems

Introduction – Process automation systems

Modern comprehensive process automation systems extend the scope of traditional control systems to include all automation functions in a single operation and engineering environment; they cover operations and configuration of continuous and batch control applications.

Figure 5 sketches some components of a typical process control system:

- Several workplace nodes, running HMI (Human Machine Interface) software that displays current process data, short term historical process data (in trend displays), current alarms and lists of recent system and process events.
- Several data integration servers that integrate data from several connected controllers store a short-term history of process data and communicate current process data, historical process data and alarms and events to clients.
- A central intelligence server (possibly redundant), running, among other things, alarm management components (e.g. for filtering of alarms) and short term alarm storage.
- Further application servers. Here, as an example, a process historian storing long-term event history.

The number of nodes (i.e. PCs) that make up a deployed system can vary from a single node system to systems deployed on many client and server nodes.

The underlying process exhibits certain I/O characteristics like the number of updating data points per second that need to be exposed to the control system and the rate at which alarms and events occur. One function of the intermediate controller servers is to adapt the update rate of the data flow to clients. For example, instead of sending one value every second to a client, the controller server could send 5 values every 5 seconds.

The connected workplace nodes, on the other end, also access the service provider components on the server nodes.

Services

Service providers are implemented as components (e.g. COM or .NET assemblies) recognizable by a special interface they implement. Clients communicate with the service providers via open standards (e.g. OPC standards) or a proprietary socket based protocol. In the latter case, clients use proxies to access service providers (These proxies are not shown in Figure 5).

A service provider together with its proxy and the communication library are called a service.

Q-ImPrESS Tool Usage

Using Q-ImPrESS tools, the System architect is able to efficiently create and edit the models of the system:

- Reengineering tools should support the creation of the needed static and dynamic information in the Service Architecture Model from source code and deployment descriptors.

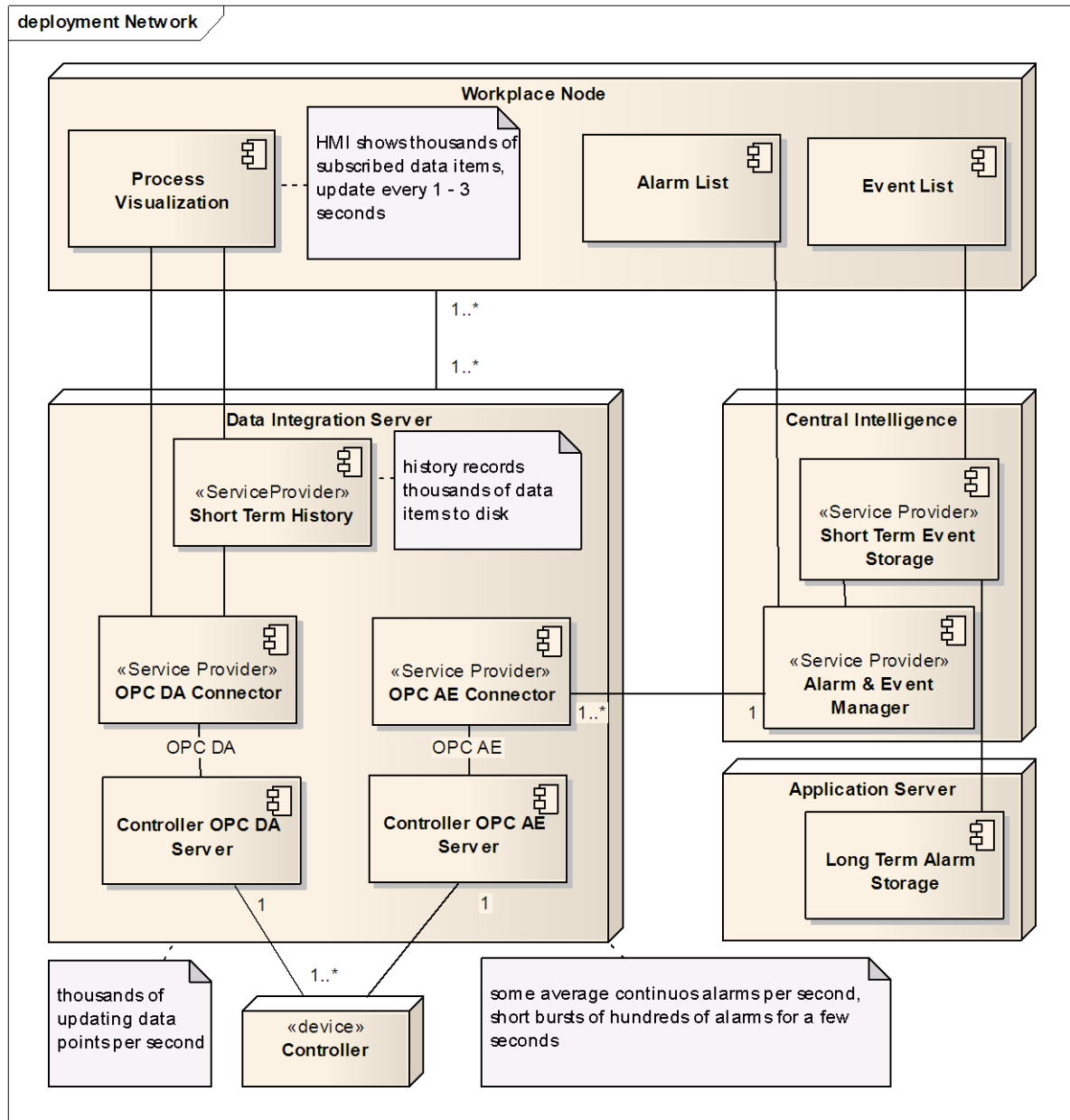
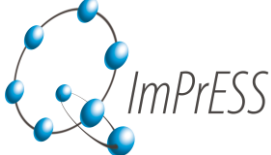


Figure 5. Simplified Sample Process Control System Architecture

- Estimating those attributes of services in the model that are needed for the prediction of runtime properties (performance, reliability) from measurements of running systems should be supported by tools.
- Graphical editors for the models should be available.

For a given Static model, Behavioral model, Allocation model and Usage model, a System architect will use Q-ImPRESS tools to predict:

- Average case and worst case resource consumption for the nodes of the system (CPU, memory, disk, and network).
- Average case and worst case response time of services.
- The reliability of the system, e.g. the MTBF (Mean Time Between Failures) for the system or parts of the system.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

- **Transient Behavior:** Investigate start-up, reconfiguration and disturbance scenarios for the system. For example when a new display is started on a workplace client, additional data subscriptions need to be set up on the controller server and device controller. The system will probably react to this changed configuration with delays in existing data displays, due to additional workload in the controllers when setting up the subscriptions.

Evolution Scenarios

The System architect will be able to predict the consequences of evolutionary changes to the system for the metrics described above. The system may evolve in the following ways:

- **Scenario 1 (Service Implementation Evolution):** The implementation of one or more services changes internally. In this case the non-functional properties of the service change, but the interfaces remain the same. For example, an OPC server may be replaced by a different version with shorter execution times leading to a higher capacity of maximally concurrently connected clients. Currently, such changes require setting up expensive hardware test environments and running stress tests. Furthermore, the process visualization service may be replaced by an alternate service provided by a customer. A server recording process history can be exchanged by one with different resource usage characteristics. An even more challenging evolution is the redesign of a whole subsystem or the system-wide upgrade of many components to a new version.

The system architect will assess the impact of these changes based on manually or semi-automatically created models for the changed services or subsystems.

- **Scenario 2 (Service Instance Evolution):** The number of instances for given service is increased. For example, the number of instantiated OPC servers allocated on the same node may increase to improve the load balancing among the connected controllers.

The system architect will be able to change the number of instances of a given service to predict their impact on the overall performance and reliability.

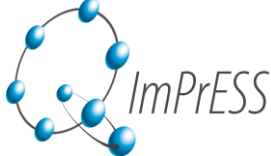
- **Scenario 3 (Functional Evolution):** Newly implemented service providers and client applications are added to the system thereby increasing functionality. For example a new OPC server might be added to the system, to allow a new kind of device to be connected.

To predict the impact of such changes, the system architect will not have to change existing parts of the service architecture, but only add models for the newly implemented parts.

- **Scenario 4 (Resource Environment Evolution):** The allocation of service providers and clients to nodes changes, e.g. two servers are combined to run on the same node. The system architect will only change the allocation model for such a change and not touch the software models or resource models. Furthermore, the used resources may change, e.g. different PCs are used with more or faster CPUs, disks, and main memory.

In this case, the system architect only edits the resource model and does not interfere with the software model or allocation model.

- **Scenario 5 (Usage Profile Evolution):** The usage changes, e.g. the number of workplace nodes changes or assumed I/O characteristics of the process change. For example, the system architect will be able to determine the required hardware

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

platform based on changed usage characteristics, such as the number of connected clients, the frequency of process log entries, and the number of external subscriptions to services.

- **Scenario 6 (Component Configuration Evolution):**

The data flow of process values in a connection between two nodes is characterized by message size (i.e. how many process values are in a message) and cyclic update rate. Larger message are more compact (less header information send), but require larger buffers on sender and receiver sides and have a larger data update delay.

The system architect will investigate optimal settings for the communication connections.

Introduction – Robotics systems

Modern Robotics systems simplify the configuration, deployment, and operation of industrial robots. The software systems cover a broad range of domains, including automotive, consumer goods, and general industries. Figure 6 shows the typical deployment view of a robotics system.

- The robot controller lies at the heart of every robot installation. It is responsible for path planning and calculation of the robot movements. The robot controller also provides communication capabilities for external devices to control and monitor the connected industrial robots (IRBs).
- An interpreter of the robot control language on the robot controller allows programmatic control of the connected IRBs. Control code can be downloaded to the robot controller from PC devices or from the connected Teach Pendant.

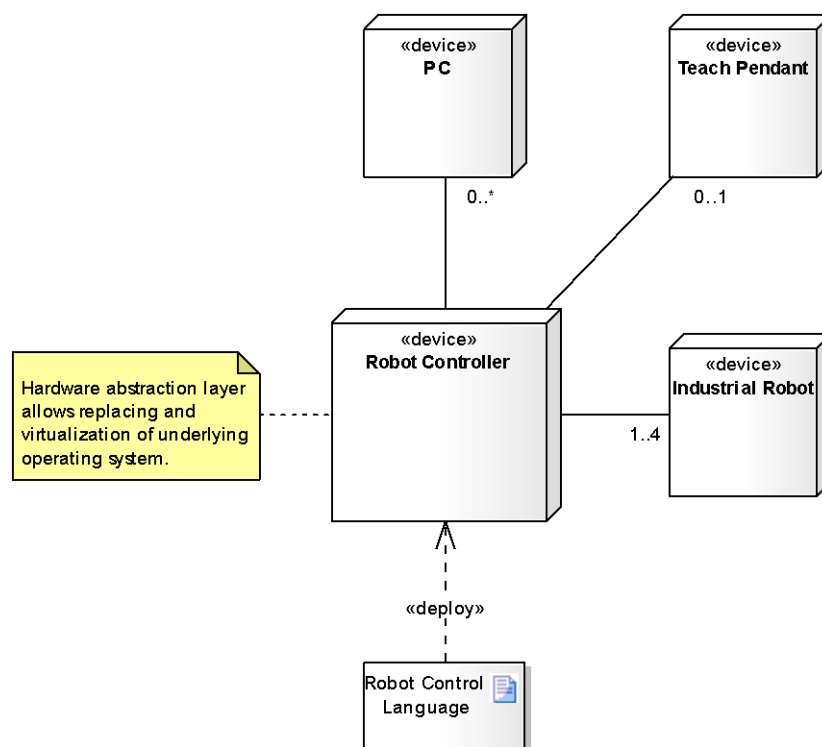


Figure 6. Deployment view of industrial robotics systems

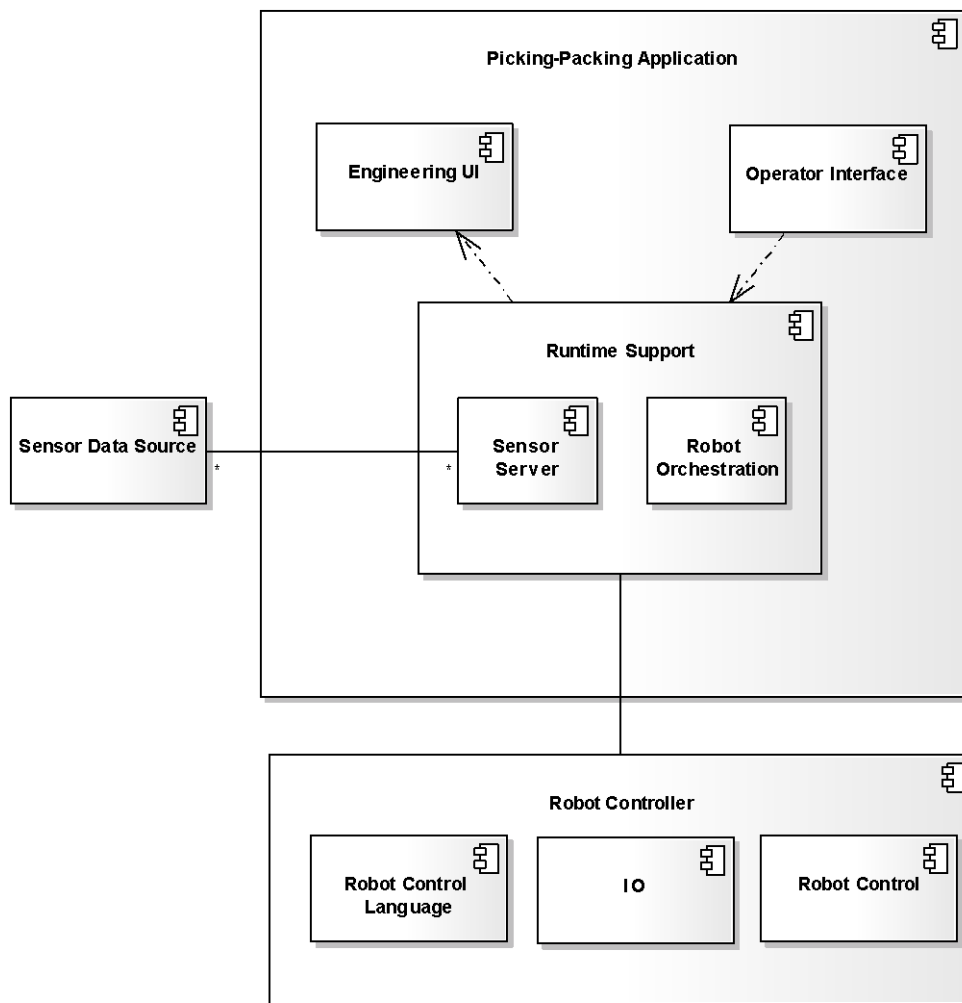


Figure 7. Component structure of a picking-packaging application

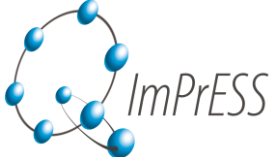
- The Teach Pendant is the operator device. It allows controlling the robot through a generic operator interface with programmable extensions.

An arbitrary number of PC devices can be connected to the robot controller. The function of these devices varies from configuration and simulation to runtime support and operation monitoring.

A complete installation ranges from single controller/robot installations up to lines with dozens of controllers/robots that communicate with each other.

The PC and Teach Pendant devices typically run Windows operating systems. The robot controller itself runs on an embedded operating system, e.g. VxWorks or Windows CE. In order to have a more detailed view of the involved components in the applications running on the PC and Teach Pendant devices, we describe the architecture of picking-packaging applications (PPApp) in Figure 7.

The PPApp application runs on an industrial PC with a Windows operating system installed. The PPApp has an Engineering UI component that allows the customers or system partners to graphically configure and engineer the pick-pack solution. Once the engineering activity is finished, the pick-pack solution can be started, i.e. based on sensor input the runtime support

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

orchestrates the connected robots to perform the defined tasks by communicating with the robot controller. The operator interface allows monitoring the execution of the tasks as well as managing the execution.

Services

In the Robotics systems, two main sources expose services. These are the robot controller, and the PPApp.

The robot controller publicly exposes services through a .NET API. The services include I/O handling, control language management, motion control, alarms and event management, file handling, and user management. The robot controller API wraps the remoting boundary from PC applications to the robot controller. In general, an arbitrary number of PC clients can connect to a robot controller. However, only one client can have write access at one point in time, i.e. the robot controller access applies the mastership concept.

The PPApp services are:

- An operation interface that allows monitoring and managing production.
- A logging service for alarms and events originating from the robot controller and the PPApp.
- A sensor configuration service, e.g. for teaching vision models.
- A sensor data analysis engine for creating target coordinate batches. By default, the service analyses images and identifies objects and their coordinates.
- A customizable service for adding extended vision models.
- A customizable service for integrating other sensor devices.
- A customizable service for manipulating generated coordinate batches, e.g. removing, adding, load balancing.

The services are either part of the PC application and used internally, or exposed as COM and/or .NET interfaces.

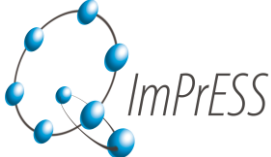
Q-ImPrESS Tool Usage

Using Q-ImPrESS tools, the system architect is able to efficiently create and edit the models of the system:

- The size of the PC application allows modeling the system architecture manually. Reverse engineering of the PC application could be very hard, because some of the services are used internally and not exposed through an identifiable interface.
- The PPApp services have to be modeled both as white-box and black-box services. For most services detailed internal information are available, but some services rely on external libraries, e.g. vision analyses.
- The system architect uses graphical editors to create, update, and review the models.

For a given Static model, Behavior model, Allocation model and Usage model, a system architect will use Q-ImPrESS tools to predict:

- Average case and worst case network utilization for the data generated by the sensor devices.
- Average case response time of sensor analysis engine, depending on the configured system.
- The minimal hardware requirements (CPU speed, RAM size) of the PC for running the PPApp, including the sensor analysis engine.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

- The reliability of the system, e.g. the MTBF (Mean Time Between Failures) for the system or parts of the system.

Evolution Scenarios

The system architect will be able to predict the consequences of changes to the system for the above metrics. The system may evolve in the following ways:

- **Scenario 1 (Service Implementation Evolution):** The implementation of one or more services changes internally. For example the sensor data analysis engine may be replaced by another service with more efficient image analysis and coordination generation algorithms. Also the operator interface service or the controller software can be replaced by updated versions.
The system architect will model the changes to the system manually and analyze their impact on the performance and reliability metrics.
- **Scenario 2 (Functional Evolution):** Newly implemented service providers are added. This refers mainly to the customizable parts of the Robotics system, e.g., the service for adding vision models, the service for integrating additional sensor devices, and the service for manipulating generated coordinate batches. Usually such extensions are implemented by customers.
The system architect will model these changes manually or reverse engineer the newly implemented code.
- **Scenario 3 (Allocation Evolution):** The allocation changes, e.g. the assignment of service providers to nodes changes, two servers are combined to run on the same node or a service is distributed over multiple nodes. The system offers multiple variants of allocating the services to different hardware nodes. The allocation may also change for specific customers.
Therefore, the system architect will be able to move service to different hardware nodes, so that the non-functional properties of different client installations can be assessed.
- **Scenario 4 (Interaction Evolution):** The wiring of services changes, e.g. from internal communication to remoting interfaces. For example, certain parts of the Robotics systems shall offer a web service for client interaction.
The system architect will predict the impact on the network utilization and overall response time of the system in these cases.

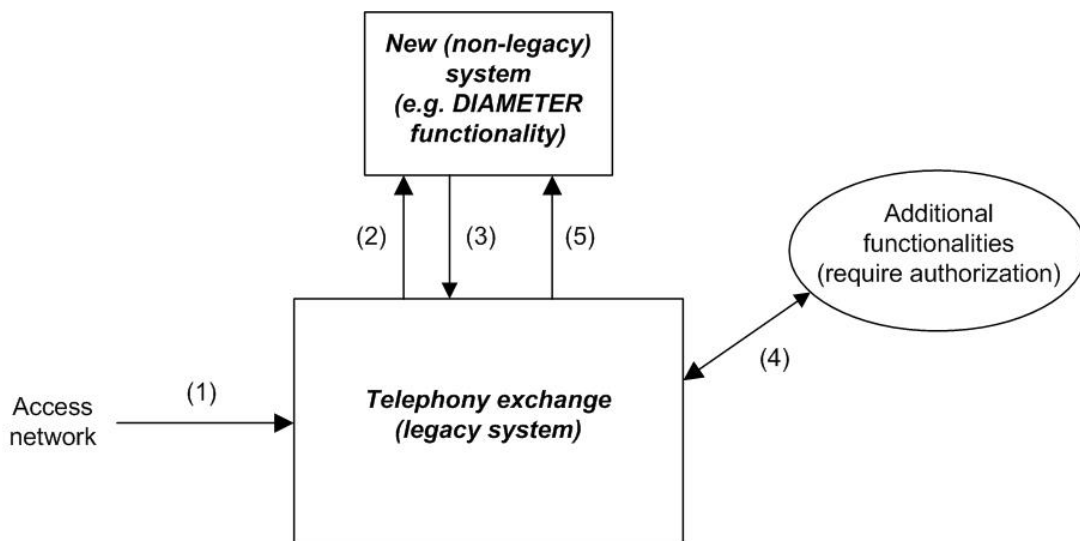
4.4.5 Telecommunications system developer additional information

ENT plans to use Q-ImPRESS tools for facilitation of development of new IP (Internet Protocol) based functionalities in telephony exchange. An example of such IP based protocol is DIAMETER protocol, a computer networking protocol for AAA (Authentication, Authorization and Accounting). Because telephony exchange systems are legacy systems based on proprietary technologies and protocols, they can't be easily extended with IP based functionalities.

Telephony exchange is a legacy system constituted of proven legacy code which provides many kinds of different functionalities. In order to extend legacy system's code with IP-based functionality some parts of legacy code must be componentized, and in the end wrapped into a service. Such service can communicate with new, non-legacy systems via common Internet protocols like UDP, TCP or SCTP. Non-legacy systems can be built anew or from some existing code like open-source components. In one of possible target application scenarios,

we plan to extend our legacy systems with adding support for DIAMETER protocols and DIAMETER based applications (Figures 8 and 9). We plan to use existing open-source DIAMETER implementations as a basis for protocol functionality and OpenSAF open-source telecommunications middleware for achieving high availability and robust performance of DIAMETER components running on OpenSAF middleware. Since telephony exchange can support very large number of concurrent users, important performance-wise metrics for us are capacity (number of users supported by system without degradation of quality of experience), response time (to user's request), network and processor load. Response time, network and processor load are tightly connected to quality of experience. Overloaded network(s) or processor(s) must be dealt with, either in hardware or software manner, because they introduce time-lag effect that degrades quality of experience. Response time is connected to network load and processor load (overloaded network and/or processor will cause larger response time), and is one of clear markers of user's quality of experience.

A service, in OpenSAF environment, is a set of deployed components. A component, in OpenSAF environment, is a set of software and hardware resources, realized by standard computer processes. In current state of OpenSAF environment, components (and thus services) can be easily implemented in C/C++ code. Components' functionality can be implemented in C/C++ code like any common C/C++ code (meaning there are no imposed limitations by OpenSAF environment). Within components' code, OpenSAF functionalities are called via C-code APIs that are defined in C-language header files (".h" files). JAVA isn't directly supported by current version of OpenSAF, but if necessary, certain parts of



- (1) User's request arrives from access network; user wants to access additional functionality provided by his service provider (e.g. some kind of advanced multimedia session)
- (2) User is being authenticated
- (3) User is successfully authenticated, user's authorization rights are returned, legacy system checks user's request against authorization rights

- (4) User has authorization rights for requested functionality; User uses the functionality
- (5) As user uses the functionality, accounting information is being inserted into non-legacy system

Figure 8. Conceptual use of DIAMETER protocol as an extension of telephony exchange functionalities

components (or even entire components themselves) can be made of JAVA code wrapped into a shell of C/C++ code.

Components communicate with OpenSAF via TIPC (Transparent Inter-Process Communication) protocol. TIPC is a network communications protocol for inter-process communication that is designed for intra-cluster communication. In regard to communication between OpenSAF and implemented components, it provides high performance messaging infrastructure. OpenSAF's internal services also use TIPC protocol for communication. Components can communicate among themselves via any protocol. In our target application scenario, components will communicate among themselves either by TIPC protocol, or DIAMETER protocol. Since DIAMETER is client-server based protocol, it is reasonable to assume that components which are going to constitute client-side of DIAMETER protocol will communicate by TIPC protocol as well as components which are going to constitute server-side of DIAMETER protocol. However, communication between client-side and server-side components of DIAMETER protocol is going to be done by DIAMETER protocol.

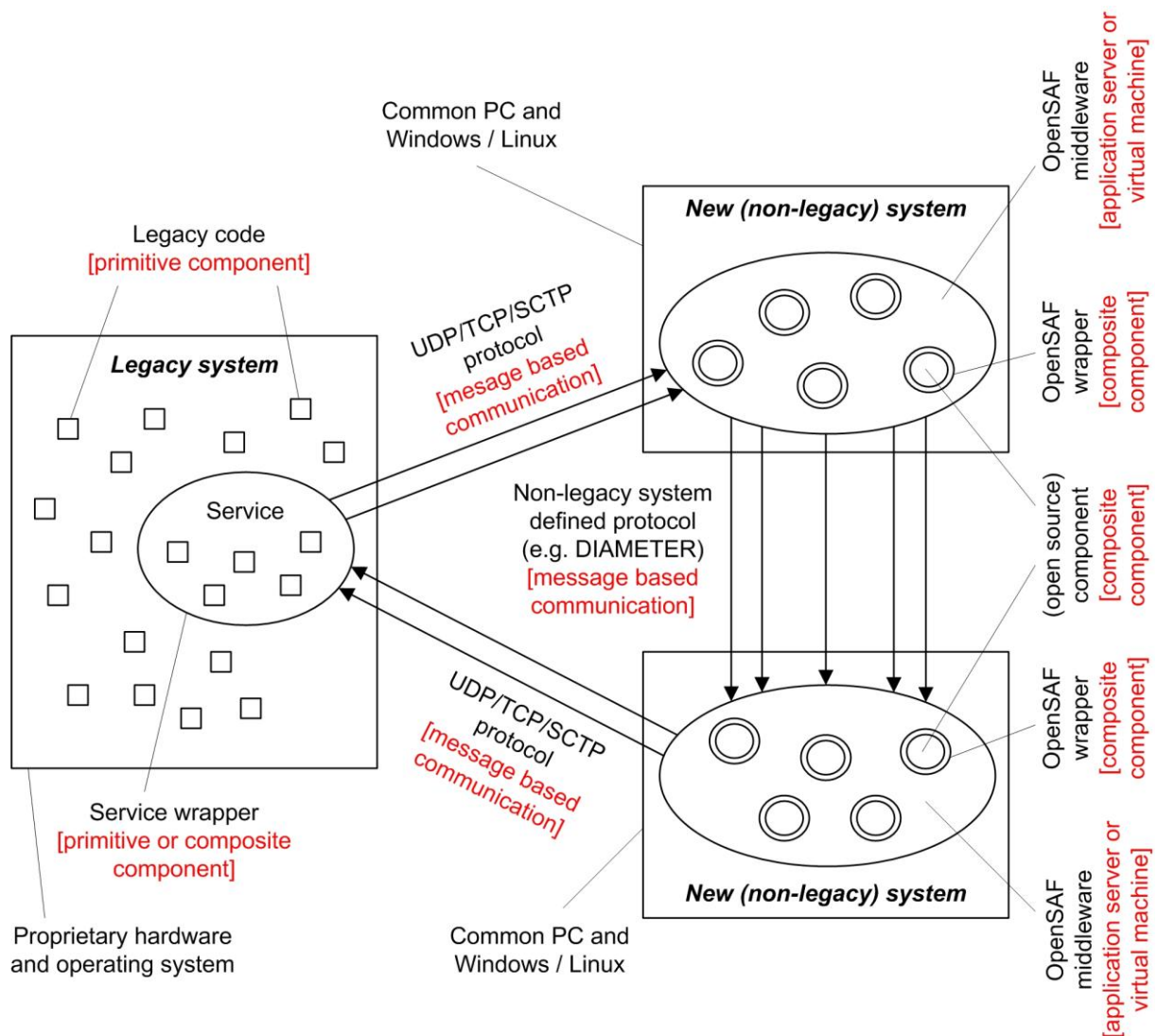
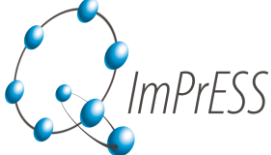


Figure 9. Concept of Ericsson Nikola Tesla's target application (terms in angular brackets correspond to terminology defined in deliverable D2.1 - Service Architecture Meta-Model)

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

Components which will constitute both client-side and server-side of DIAMETER, are going to be implemented with redundancy, availability, reliability, capacity and maintainability in mind. Most of noted quality attributes will be achieved by using functionalities provided by OpenSAF. Quality attributes that cannot be achieved through existing functionalities of OpenSAF, will be implemented as custom wrappers around DIAMETER components. Granularity of components will vary depending on level of opportunities provided by DIAMETER implementations to successfully integrate OpenSAF functionalities and, where necessary, additional wrappers. Thus, certain components can be made of smaller C/C++ files (1K lines of code), while others can consist of several larger C/C++ files (10K-100K lines of code).

Interaction ("orchestration") of client-side DIAMETER components among themselves, and server-side DIAMETER components among themselves, will be done by functionalities provided by OpenSAF. These include synchronization and messaging services like different types of semaphores, mailboxes and event channels.

In order to facilitate migration toward SOA system as the one described, we require versatile modeling tools within which we will model legacy systems and new (non-legacy) systems. Based on the information entered in the models, prediction tools should provide reasonable predictions.

Detailed description with respect to Service Architecture Meta-Model

Described telecommunications system developer scenario from Figures 7 and 8 can be further described with regards to terminology defined in deliverable D2.1, the "Service Architecture Meta-Model (SAMM)". Figure 9 further develops target application concepts described in Figures 7 and 8, and additionally, takes SAMM from D2.1 into consideration. It is important to note that Figure 9 presents one possibility of how the target application will be developed.

As in Figures 7 and 8, target application depicted in Figure 9 consists of two major parts: Legacy system and New (non-legacy) system. Legacy system can be a proprietary telephony exchange that initiates request for IP-based DIAMETER functionality, which is implemented in New (non-legacy) system. Also, during different development phases of the target application, Legacy system can be simplified to proprietary traffic generators that mimic the telephony exchange. In both cases, legacy system is built from black-boxes (primitive components). We plan to use Q-ImPrESS tools for modeling these primitive components as well as modeling connections and interactions between primitive components. This modeling includes defining static structure, a deployment/allocation model, quality annotations and an usage model. It is important to note that there can be several black-boxes (primitive components) that perform the same task but with different performance characteristics (different quality annotations). The aforementioned black-boxes (primitive components) have the same static characteristics (interfaces/ports), but have different quality annotations and can be deployed/allocated in a different manner.

New (non-legacy) system implements IP-based DIAMETER functionality that extends existing functionalities of the telephony exchange (Legacy system). Since DIAMETER is a client-server based application system, New (non-legacy) system consists of two corresponding parts: a client cluster and a server cluster.

DIAMETER client cluster is achieved by running DIAMETER client code on OpenSAF middleware. DIAMETER client code can be based on OpenDIAMETER solution or OpenBloX solution. OpenDIAMETER solution is based on C and C++ code, and uses ACE (Adaptive Communication Environment) as a network abstractor. OpenBloX solution is based

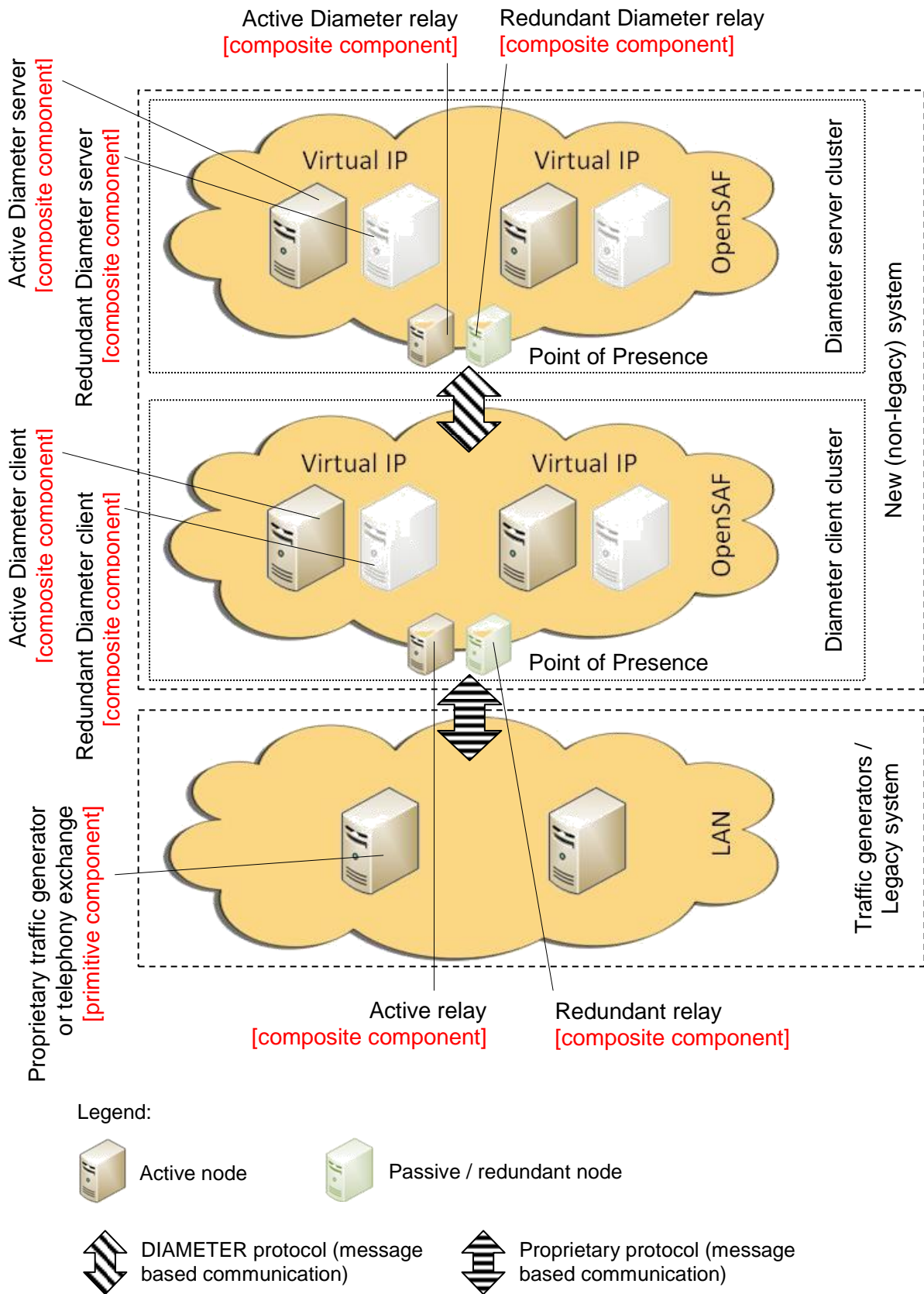


Figure 10. An example of conceptual deployment scenario example of Ericsson Nikola Tesla's target application (terms in angular brackets correspond to terminology defined in deliverable D2.1 - Service Architecture Meta-Model)

on Java code. Both solutions can be thought of as composite components which can be reverse engineered or modeled as a hierarchy of primitive components. Also, both solutions have the same interfaces/ports through which they communicate with other systems (legacy system or DIAMETER server). This means that OpenDIAMETER based client can be easily exchanged for OpenBloX based client without the need for any redefinition or reconfiguration of other components or parts of the entire system. OpenSAF middleware can be thought of as application server or virtual machine. Thus, in accordance to D2.1, DIAMETER client code can be thought of as a service consuming logical resources (OpenSAF) that run on physical resources (CPU, memory, HDD, or hardware in general). OpenSAF middleware assures reliability and availability of DIAMETER clients through redundancy of each DIAMETER client. Redundancy is achieved by Virtual IP address sharing among active and redundant DIAMETER client node.

DIAMETER client cluster has an entry point, noted as Point-of-Presence on Figure 10. The entry point is the only known entrance to the DIAMETER client cluster. External systems that use DIAMETER client cluster need only to know the IP address of the DIAMETER client cluster's Point-of-Presence. DIAMETER client cluster's Point-of-Presence can be based on Linux Virtual Server, Pen, HAProxy or other similar open-source solutions. From the aspect of Q-ImPrESS, these solutions can be thought of as primitive or composite components that have the same interface for message-based communication, but different quality annotations.

DIAMETER server cluster is achieved in similar manner as DIAMETER client cluster. DIAMETER server code runs on OpenSAF middleware. DIAMETER server code can be based on OpenDIAMETER solution or OpenBloX solution. Servers based on OpenDIAMETER and OpenBloX can easily be exchanged because both solutions have the same interfaces/ports for server code as well as client code. OpenSAF middleware is used in the same manner as in client cluster for assuring reliability and availability of DIAMETER servers.

Like DIAMETER client cluster, DIAMETER server cluster also has an entry point, noted as Point-of-Presence on Figure 10. Entry point is the only known entrance to the DIAMETER server cluster. External systems that use DIAMETER server cluster need only to know the IP address of the DIAMETER server cluster's Point-of-Presence. DIAMETER server cluster's Point-of-Presence is based on relay agent functionality of DIAMETER standard. The role of the relay agent can be thought of as the role of a proxy in common HTTP traffic.

Communication between legacy system and DIAMETER client cluster is based on proprietary protocol that spans between the telephony exchange network domain and IP-based network domain. The protocol uses message-based communication. Hence, the ports on DIAMETER clients and primitive components of legacy systems that deal with this proprietary protocol can be represented with source and sink ports.

Communication between DIAMETER client cluster and DIAMETER server cluster is based on standardized DIAMETER protocol. DIAMETER protocol uses message-based communication and client-server communication paradigm. Thus, the ports on DIAMETER clients and DIAMETER servers that communicate with DIAMETER protocol can be represented with source and sink ports

Component alternatives

Previous detailed description of target application opens the possibility of simple exchanging of component alternatives. Component alternatives comprise of components that have the same interface but have different implementation. Thus, component alternatives have the same ports in static structure description, but can have different behavioral model, allocation model and quality annotations. Such components exchange is possible in DIAMETER client cluster and DIAMETER server cluster where OpenDIAMETER and OpenBloX client or

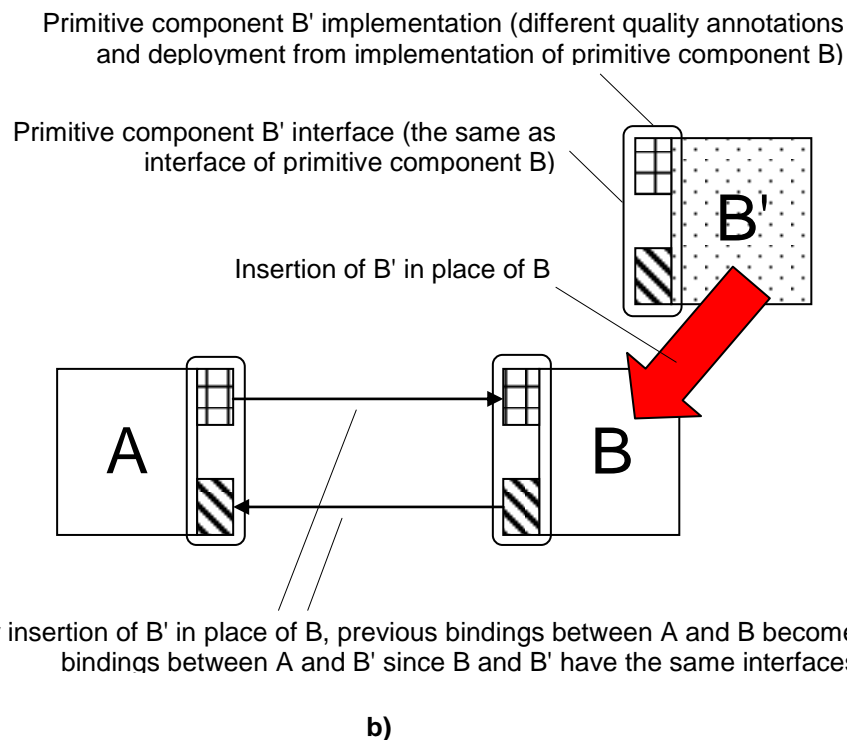
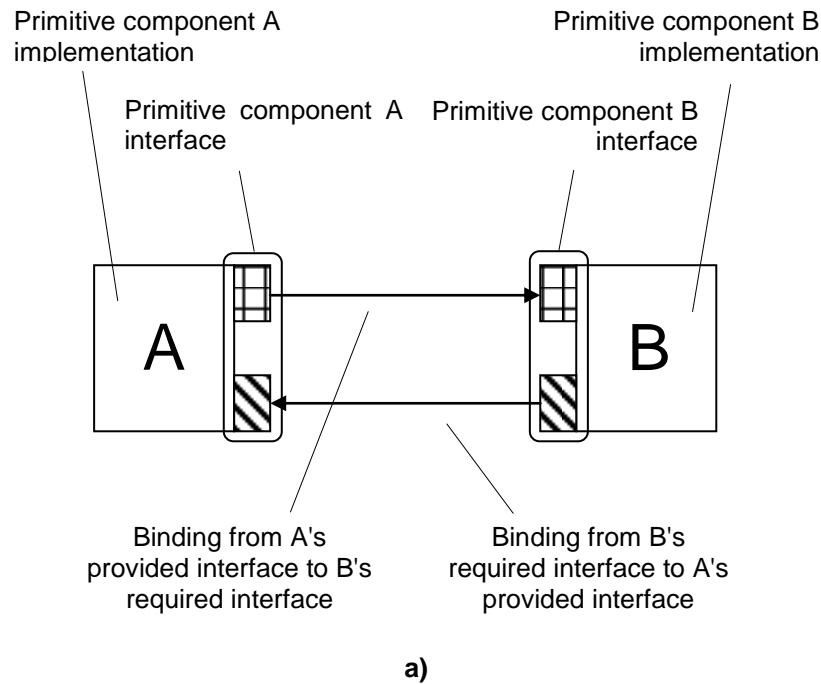
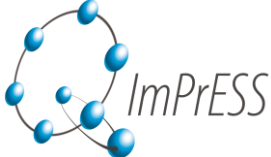


Figure 11. a) An example static structure of two components in ENT's target application, **b)** an example of insertion of primitive component B' in place of primitive component B

server code can be exchanged. Also, components exchange is possible in Point-of-Presence in DIAMETER client cluster where Linux Virtual Server, Pen and HAProxy can be used as a basis for Point-of-Presence. All mentioned exchanges are possible because corresponding components have the same interface.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

Conceptual illustration of the mentioned component exchange is shown on Figure 11. Figure 11.a shows an initial static structure of components where component A is bound to component B. Component B' has the same functionality as component B, and has the same interface used for communication with component A. Thus, a simple component exchange (see Figure 11.b) assumes a simple replacement of component B from Figure 11.a with component B'. In this simple component exchange all previous bindings between component A and component B are preserved and used as bindings between component A and component B'. This is possible because component B and component B' have the same interfaces.

Possible evolution scenarios

The system described above can evolve in the following ways:

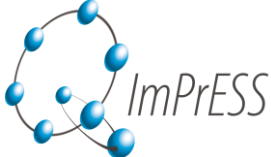
- **Scenario 1 (service implementation).** As stated in the above description, any part of the system (traffic generators / telephony exchange, client cluster, server cluster) can have changed one or more of its constituent services (changes to service implementation, while service interfaces remain the same). For example, OpenDIAMETER clients are exchanged with OpenBloX clients, or traffic generators are exchanged with telephony exchange.

System architect models these changes manually (according to description given in "component alternatives" section) and analyses impact of these changes on performance and reliability.
- **Scenario 2 (new functionality).** The above described system can be extended by addition of complex DIAMETER applications that bind to client and server cluster. Also, an entirely new IP-based functionalities can be bound to telephony exchange. Both of these possibilities present an added value to the existing telephony exchange system.

System architect manually models new functionalities and analyses their performance and reliability.
- **Scenario 3 (service deployment/allocation).** Any mentioned service in the system can be multiplied in order to keep users' Quality of Experience on the same level. Service multiplication can be done on one computer, or over a number of computers in the system. System architect manually deploys service instances over the modeled system. Afterwards, System architect analyses the impact of such service deployment on overall system performance and reliability.
- **Scenario 4 (hardware resources).** As the number of users increase, it is likely it'll be necessary to increase the number of clients in the client cluster and servers in the server cluster. Additionally, it is likely that computers running controller nodes of OpenSAF middleware will also be increased in order to achieve high level of service reliability and availability. Furthermore, any computer in the system can be upgraded with faster CPUs, HDDs, additional memory, etc.

In this scenario, system architect describes changes to Allocation model without changing any other models.
- **Scenario 5 (usage profile).** Described system can be used by a wide range of users. The number of users, amount and frequency of user's requests influence the system performance.

System architect models different usage profiles in order to predict system's performance under different conditions. By comparing and analyzing predictions for different usage

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

profiles, system architect can make conclusions regarding a proper service allocation and changes required for hardware resources.

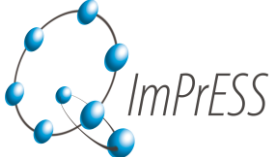
4.4.6 Enterprise domain additional information

The considered application area for Q-ImPrESS tools encompasses solutions development in the area of enterprise application integration. Old legacy systems as well as other existing systems offer a number of business services, which handle business data and enable access to the enclosed system functionality. These services can be accessed by different means and protocols, depending on the particular system design, ranging from simple file transfer over any possible proprietary access protocol to standard communication ways like JMS or SOAP-based interaction. New solutions based on Java technology will be built upon all these services and typically provide some new business functionality or effectively integrate existing functionality into a new business process. Their implementation employs the Java Business Integration (JBI) standard, which is the standardized approach in the Java world for building an Enterprise Service Bus (ESB). JBI defines an API for the services, which have to be provided by the bus itself, as well as a Service Provider Interface (SPI) for implementing components and deploying them into the bus. Here the notion of component slightly differs from the common established one in the software engineering industry. In JBI it can be any self-contained deployable unit of code, which offers some functionality or service, which fits well to the Q-ImPrESS definition of service. A JBI component is best viewed as technical gluing code for connecting business logic to the bus. For example, an XSLT transformation engine is a particular component. Other examples include POJO components for accessing POJO-based logic in the bus, BPEL components for executing BPM scenarios, scheduler components for time-based service activation and a component for integrating business rules. All these components are also called service engines, as they provide some service functionality directly in the bus. Another type of components, called binding components, is used for connecting any external services to the bus, which can be accessed over a particular protocol. This can be for instance JMS, FTP, HTTP and SOAP.

The itemis SOA showcase uses Petals as an open-source implementation of the JBI specification. Most of the above mentioned components are included in the Petals distribution. So the System engineer usually has to pick up the required component and deploy a service upon it. Although the component communication is transparent for the developer, internally Petals employs JMS for component and node interaction. From the developer view it is only important to know message exchange patterns, which represent the order of interaction between service consumer and service provider (In-Only, Robust In-Only, In-Out, In Optional-Out).

Currently the development effort for the itemis showcase doesn't follow a model-driven approach and thus, all involved system artifacts have been composed manually. This is due to the fact that so far no appropriate description language for modeling the service structure and deployment has been introduced. Although this modeling language is subject to the future development, design-time and runtime view of services in the showcase can still be described by means of established modeling concepts.

Any JBI component has to implement a certain SPI to be able to participate in the deployment life-cycle and be plugged in to the bus. Such a JBI-ready component offers some functionality either implemented in the custom code (e.g., format aggregation component) or contained in the third party libraries (e.g., JMS adapter). The entry point of this component can be viewed as a design-time service. In many cases this design-time service is expected to be provided

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

with additional information during the deployment, configuring the service to fulfill a specific task, for example, executing specific XSL transformation. Such a deployed configured instance of the design-time service can be considered as a runtime service, which becomes reachable in the JBI bus under a unique endpoint name. It is important to distinguish between these service views, as a single design-time service can have multiple corresponding runtime services, which employ its functionality. This is very similar to the fundamental relation between class and object.

To sum up, JBI services can be modeled as components with a number of properties, which need to be specified for the component deployment in order to activate a corresponding runtime service.

Evolution Scenarios

The scenarios for evolutionary changes to a system in the Enterprise domain are already covered in the generic section on Possible Evolution Scenarios (see section 4.4.3). The following section describes concrete examples for the evolution scenarios from the generic section from the perspective of the Enterprise domain. It is concluded by a set of evolution scenarios which will be included in the Q-ImPrESS E-SOA Showcase.

Evolution Scenario Examples

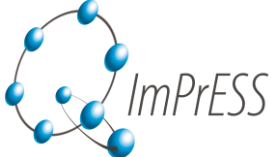
The System architect is able to predict the consequences on the Quality of Service resulting from the following evolutionary changes to the system before the he begins the actual implementation.

SOA System Implementation

The implementation of a service-oriented system may be changed. This includes the following special cases:

- **New Service:** A new service is written from scratch. This may be caused by new or changed requirements, which demand the introduction of additional services.
- **Internal changes to services:** The internal implementation of one or more existing services may change, e.g. due to changed requirements, while at the same time the external interfaces provided by the service remain unchanged. A concrete example for this scenario is the implementation of an enhanced algorithm, which improves the accuracy of service results. As for new services, the impact of design decisions can be predicted before the changes are actually implemented.
- **Service Evolution:** Certain changes may also affect the external interfaces provided by a modified service. A concrete example for such an evolution scenario might be a web service delivering national phone book functionality, which is to be internationalized. After its evolution it accepts the country code as an additional parameter.
- **Service Granularity:** A service, which formerly did not use other services to compute its results, is changed in a way that it now uses one or more auxiliary services to deliver its functionality. For example, an online banking service formerly had an integrated authentication routine, which should now be replaced by an authentication service delivered by an external service.
- **Service Wiring:** The wiring between services is changed. As a concrete example, the phone book service mentioned above, is deployed on several machines and managed by a load balancer in order to deliver its functionality to an increased user base.

Evolving Deployment / Allocation

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

The deployment of a service-oriented system may be changed, which also may have consequences on the system's Quality of Service. These changes may concern the hardware as well as the system's underlying middleware.

- **Hardware:**
 - **Evolving hardware resources:** The characteristics of the underlying hardware resources may change (e.g. CPU, disk, memory, network throughput, etc). In many concrete examples the entire machine on which a service is deployed, is replaced by a new machine in order to enhance the QoS.
 - **Evolving hardware deployment:** The deployment of services on specific hardware resources may change. For example, two services which formerly ran on the same node now run on two separate nodes.
- **Middleware:**
 - **Evolving middleware:** The underlying middleware may evolve. For example a new version of the Petals ESB implementation may be introduced.
 - **Evolving middleware deployment:** The deployment of services on specific middleware resources may change. For example, two services which formerly ran on the same Petals ESB container now run on two separate containers.

Evolving Usage Profile

The usage profile of the service-oriented system may change, which can also have consequences on the system's QoS. This includes the following evolution scenarios:

- **Number of users:** The number of users concurrently working with the system may evolve over time.
- **Characteristics of usage scenario:** Certain characteristics of the usage scenario, which have an impact on the QoS may change. For example, a SOA-based music store might begin to distribute music videos which have a significantly larger file size than plain music files (such as MP3).

Evolution Scenarios included in the E-SOA Showcase

The Open Source E-SOA Showcase developed within the Q-ImPrESS project will be able to cover the following sample evolution scenarios:

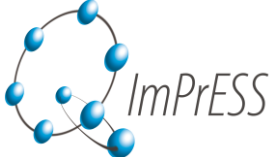
- **Different Architectures and Deployment:** Architectural alternatives will be included in the E-SOA showcase. It will be easy to set up the showcase on different machines in order to test changes in the deployment.
- **Different Usage Profiles:** Load on the E-SOA Showcase can be generated via simulators. Different usage profiles can be simulated by running the system with different parameter sets.

4.4.7 Research organization specific details

During the last years the research community gathered a lot of knowledge in the area of software quality prediction. Lots of different approaches, methods and techniques were developed to evaluate non-functional quality attributes of software systems. These consist of approaches for prediction of performance, maintainability and reliability.

One of the major challenges is the combination of all these approaches in a comprehensive integrated method landscape. The Q-ImPrESS project aims to achieve such integration.

The developed methods and tools are going to be used by the research partners in different ways and activities. Analysis Method Developers want to use the tools to *validate their methods* and compare results with other methods. This holds for *comparison of analysis*

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

methods for the same quality attribute (e.g. compare different performance prediction methods), as well as for comparison of analysis methods for different quality attributes, as well as for doing *trade-off analysis* (e. g. trade-off between performance, maintainability, performance). Therefore the tools need to be flexible enough to support different analysis methods and also to be extensible to support new analysis methods. One goal of the aforementioned validation and comparison is to detect advantages, disadvantages, limitations and open questions for individual analysis methods, analysis models and system / service models.

The *interaction between different methods* is necessary. This can be obtained by providing a common model from which method-specific models can be derived. The common model can be used as a shared data store for all participating analysis models and methods. The methods can get their input data from the common model, do their analysis and put their output in the common model. Afterwards other methods can use this data for further analysis.

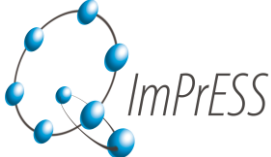
Research partners also expect to be able to *extract information from legacy software* by usage of reverse engineering techniques. Therefore different reverse engineering techniques have to be applied. Two major categories of reverse engineering techniques are static analysis methods and dynamic analysis methods. Static analysis methods parse the source code of a software system and build models like Abstract Syntax Trees (AST). Dynamic analysis methods gather information from running software by instrumenting the code and by monitoring and benchmarking during execution of the system. Examples for possible (advanced) reengineering techniques which could be considered are control-flow analysis, data-flow analysis, slicing and architecture recovering. The tool chain should be open and flexible enough to be able to integrate different reverse engineering methods when necessary. Beside static code other information sources should be considered. This might be additional meta-data information about the service architecture, for example in a standardized service description language like WSDL. On the other hand manual interaction with models might be necessary. This implies manual editing capabilities of the models, for example by using graphical or textual model editors.

Another usage scenario relevant for research partners is the application of the methods and tools provided by Q-ImPrESS in their *lectures and teaching activities*. Students can be taught, how to use quality assessment methods and trade-off analysis in their own projects. This may lead to a better knowledge transfer between university (research community) and industry, especially if students get used to method and tool application and establish their knowledge as future employees in software companies. Students can also be encouraged to provide their own refinement or extensions to the methods and tool chain in the context of master theses.

Some research partners also want to apply tools and methods for consultant purposes within collaborations with industry partners. According to this, one goal is to help companies to integrate quality assessment methods and tools in their software development process.

Since all research partners already have certain tools and methods, which they want to reuse and refine if possible, there should be some continuity between the old tools and the new tool landscape, as well as a high consistency within the new tool landscape. This might be established by using a homogenous technical environment and application of model-driven techniques like the usage of model transformation engines (e. g. QVT).

The integration of different methods and tools also enforces the communication and consensus of all participants about common terminology. This also provides for a better communication with other stakeholders of the tools. From the users perspective a homogenous representation of the problem domain is very important. On the other side the

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

research community has to ensure that used terminology has a sound foundation in literature according to the given state of the art.

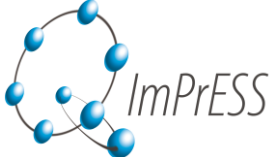
5 Functional requirements

Functional requirements described in this chapter are prioritized into three levels: low, medium and high. Requirements prioritization is presented hierarchically. Priorities at outer levels of hierarchy represent average of inner level priorities. Inner level priorities present in detail the priority for a certain requirement. High priority of the requirement means mandatory fulfillment of the requirement. Medium priority of the requirements means the requirement is important, but depending on time constraints and agreements among all partners, the requirement doesn't have to be fulfilled. It is important to note that requirements of medium priority do not hinder proof-of-concept implementation, but provide further usability of the method and the tools. Low priority of the requirements means the requirement is optional and can be fulfilled if time-constraints permit so.

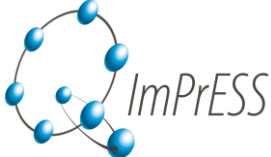
5.1 Service Architecture Modeling

The quality prediction methods that shall be developed within the Q-ImPrESS project follow a model-based approach. The following requirements concern the modeling of the system in development.

- **[R_SAM_1] Separation of System Views:** It is possible to model different parts of the system in development with separate views/models. [high]
- **[R_SAM_2] Transformations from and to Q-ImPrESS model.**
 - **[R_SAM_2.1] Transformations from Q-ImPrESS model:** SAM instances can be at least transformed into Palladio CM for performance prediction, KLAPER for reliability predictions, and to a newly created maintainability prediction meta-model specified as ECore-Instance. Other transformations may be supplied for additional analyses but are optional [high]
 - **[R_SAM_2.2] Transformations to Q-ImPrESS model:** SAM instances can be (partially) derived from other software meta-model instances like UML models or similar special software meta-models via transformations [medium]
- **[R_SAM_3] Relevant models:** The following models comprise Service Architecture Model, and are relevant for service architecture modeling: [high]
 - **[R_SAM_3.1] Static Model:** This model contains the static structure of the system in development (i.e., components and connectors). [high]
 - **[R_SAM_3.1.1] Connection of Static Model and implementation Artifacts:** Artifacts within the Static Model can be connected to their implementation. This is necessary for the test deployment (see below) and the seamless integration into the model-based software development process. [medium]
 - **[R_SAM_3.1.2] Black Box Artifacts enabling:** Artifacts within the Static Model can be modeled as black boxes, where no information is (yet) known about their internals. [high].
 - **[R_SAM_3.1.3] Common Data Store for Method-Specific models:** The model should serve as common data store from which method-specific models for all supported prediction methods can be derived. [high]
 - **[R_SAM_3.1.4] The model should cover static information:** The model should cover static information: this class of information contains details on services, their interaction and their implementation, the provided and required

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

- interfaces of the services. [high]
- **[R_SAM_3.2] Behavioural Model:** This model contains the behaviour of the system under development [high]
 - **[R_SAM_3.2.1] Model contains dynamic information:** The model contains dynamic information which characterizes the control and data flow of the service oriented architecture. [high]
 - **[R_SAM_3.2.2] Black Box described through quality annotations:** Behavior of black boxes is described through quality annotations, see [R_SAM_3.5.1] [high]
 - **[R_SAM_3.3] Allocation Model:** [high]
 - **[R_SAM_3.3.1] Model contains physical and logical resources:** This model contains the physical (e.g. CPU, Memory, etc.) and logical (e.g. queues, containers, etc.) resources which are involved in the system execution. [high]
 - **[R_SAM_3.3.2] Additional logical resource modelling:** It should be possible to model additional logical resources which are not already offered by Q-ImPRESS tools. These new logical resources should be modelled as a service. [high]
 - **[R_SAM_3.3.3] Mapping between Static and Allocation Model:** This model maps elements of the Static Model to resources described in the Allocation Model. [high]
 - **[R_SAM_3.4] Usage Model:** This model describes the anticipated usage profile (i.e., the number of users and input parameters) of the system in development. The usage profile can be provided with e.g. average values, ranges, probabilistic values. [high]
 - **[R_SAM_3.5] Quality Annotations:** This model describes the quality attributes which are imposed on elements of the Service Architecture Model. [high]
 - **[R_SAM_3.5.1] Black Box Behavior description:** Behavior of black boxes with regard to the qualities covered by Q-ImPRESS (see below) can be annotated with average values, ranges, or probabilistic information. Black box components are necessary within the scope of the static quality prediction analysis and test deployment. [high]
 - **[R_SAM_4] Model Creation:** It should be possible to create the aforementioned models from within a graphical editor. [medium]
 - **[R_SAM_5] Design Alternatives:**
 - **[R_SAM_5.1] Design alternatives as parts of models:** It should be possible to model alternatives for parts of the aforementioned models. These alternatives should be saved in a versioned manner. Amongst other reasons, this is necessary in order to perform trade-off analyses for different design alternatives. [high]
 - **[R_SAM_5.2] Incremental adaptation of the model changes:** It is possible to use the tools, methods and models during the evolution of a system efficiently. After small changes to the systems, it should not be necessary to completely regenerate models etc. but the models are incrementally adapted to the changes. A list of different alternatives is a sequence of evolution. [high]
 - **[R_SAM_6] Model Validation:** It should be possible to perform syntax checks of the model validity (e.g., no obviously wrong quality annotations, no contradictions within the model) and completeness (e.g., all required interfaces are connected, all components are deployed, all necessary quality annotations are available) with regard

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

to specific tasks such as the quality prediction simulation and tests. [medium]

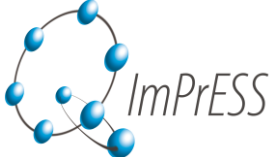
5.2 Service Architecture Model Extraction

Parts of the system in development may rely on legacy code, for which the necessary models are not available. The following requirements enable quality prediction for systems containing such code.

- **[R_SAE_1] Service Re-Engineering from Legacy Code:** Legacy code can be inspected by static and dynamic (run-time) analyses. Resulting from these analyses is a set of models, which directly enable the quality prediction for legacy parts of the system. Manual additions to these models may be necessary. [medium]
 - **[R_SAE_1.1] Targeted System Size:** A few millions of LOC [medium], or at least up to 500 kLOC [high].
 - **[R_SAE_1.2] Programming Languages:** Java [medium], C++ [high, needed now], C# or .NET IL [low, for future applications]. C++ extraction should be able to deal with Microsoft Visual C++ as used by Microsoft Platform Frameworks like ATL and MFC.
 - **[R_SAE_1.3] Service Discovery:** It should be possible to retrieve structural information such as system components and services. As far as possible, this retrieval is performed automatically with manual interaction where necessary. Services may have to be detected by the interfaces they implement, by attributes e.g. in .NET code or by additional, proprietary service descriptors. [medium]
 - **[R_SAE_1.4] Extensibility:** The architecture extraction process is extensible, i.e. there should be plug-in support for different component technologies/architectures to cover a broader range of systems. [medium]
- **[R_SAE_2] Monitoring and Measuring Legacy Code:** Should reverse engineering via static code analysis of legacy system parts not be necessary or wanted, it should be possible to monitor and measure those legacy system parts. [low]
 - **[R_SAE_2.1] Quality attribute Estimation:** Q-ImPreSS should support the prediction of necessary performance properties (i.e. execution times) [high] and reliability properties (i.e. failure rates) [low] as well as usage patterns of services [medium] from measurements of running systems by tools.

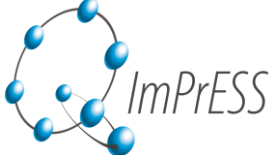
5.3 Quality Impact Prediction

- **[R_QIP_1] Quality Prediction Metrics:** The following metrics concerning service quality can be predicted by the Q-ImPreSS methods and tools.
 - **[R_QIP_1.1] Performance:** [high]
 - **[R_QIP_1.1.1] Average case and predicted maximum resource consumption:** It should be possible to predict the average case [high] and worst case [high] resource utilization (CPU, HDD, memory, network) for the elements described in the Deployment Model.
 - **[R_QIP_1.1.2] Average case and worst case response time:** It should be possible to predict the average case [high] and worst case response time [high] for the services of the system in development.
 - **[R_QIP_1.1.3] Average case and worst case throughput:** It should be possible to predict the average case [high] throughput of the methods of all services described in the Common Service Architecture Model.
 - **[R_QIP_1.1.4] Determining minimal hardware:** It should be possible to let

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

the tools analyze the model for the minimal required hardware (e.g., CPU speed, memory size, etc.) to fulfill the modeled non-functional requirements through iterative process. [medium]

- **[R_QIP_1.2] Reliability:** [high]
 - **[R_QIP_1.2.1] MTBF:** It is possible to predict the mean time between failures. [high]
 - **[R_QIP_1.2.2] MTTF:** It is possible to predict the mean time to failures. [high]
 - **[R_QIP_1.2.3] MTTR:** It is possible to predict the mean time to recover. [high]
 - **[R_QIP_1.2.4] Failure probability per demand:** It is possible to predict the failure probability per demand [high]
- **[R_QIP_1.3] Maintainability:** [medium]
 - **[R_QIP_1.3.1] Time and Cost Effort of Maintenance Task/Change:** Time and Cost Effort of Maintenance Tasks according to Change Scenarios. [medium]
 - **[R_QIP_1.3.2] Violations of Design Principles:** Number and severity of the violations of Design Principles (Best Practices, Heuristics) [medium]
- **[R_QIP_2] Simulation and Analysis:** The aforementioned quality attributes can be predicted by statical, numerical, and symbolic analyses (e.g., queuing networks, Markov models, finite state machines, SAT) as well as a simulation of the expected system behavior.
 - **[R_QIP_2.1] Subsystem Simulation and Analysis:** It should also be possible to simulate and analyze only parts of the system in development, e.g. for performance reasons. [medium]
 - **[R_QIP_2.2] Specific Metric Simulation and Analysis:** The simulation and analysis concerning a specific quality metric can be triggered, as soon as all necessary information for this specific metric is specified. The tool chain should be flexible enough to integrate and support different quality prediction methods. [high]
- **[R_QIP_3] Quality Impact prediction based on different given design alternatives:** [high]
 - **[R_QIP_3.1] Quality impact of different architecture designs:** It should be possible to compare prediction results gained from models of different architectural solutions [high]
 - **[R_QIP_3.2] Quality impact of different allocation scenarios:** It should be possible to compare prediction results gained from models of different allocation solutions [high]
- **[R_QIP_4] Test Allocation:** It should be possible to deploy the system under development into a testing environment. This test allocation allows the validation of the results from the simulation and analysis. [low]
 - **[R_QIP_4.1] Mock implementations:** Artifacts from the Static Model, which are depicted as black boxes are implemented as mock components for the test allocation. Their characteristics with respect to the quality attributes regarded by the Q-ImPrESS methods and tools is simulated based on the Quality annotations. [low]
 - **[R_QIP_4.2] Monitoring and Measuring Facilities:** It should be possible to

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

monitor and measure the allocated test system with regard to the aforementioned quality attributes. The results obtained from this test can be graphically compared to the respective results of the simulation and analysis. [low]

5.4 Trade-off Analysis

It is possible to perform a variety of trade-off analyses, which enable the comparison of design alternatives with regard to their impact on the service quality. The results of these analyses are graphically visualized. [high]

- **[R_TOA_1] Trade-off between different quality attributes:** It is possible to assign preferences for specific quality attributes (e.g. preference of performance over reliability by putting higher weight factor on performance). Based on these preferences a ranking of different design alternatives is presented. [high]
 - **[R_TOA_1.1] Storing results:** It should be possible to store results of analysis [high]
- **[R_TOA_2] User trade-off scenarios:** It should be possible for a user to create different trade-off scenarios and to make conclusions regarding these trade-off scenarios by re-running simulations. [high]

5.5 Tool Support

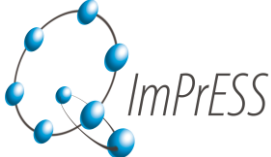
The methods developed within the Q-ImPrESS project are supported by a set of tools. The following requirements concern these tools in general.

- **[R_TS_1] Integration into the Eclipse IDE:** The tools developed within the scope of Q-ImPrESS are integrated into the Eclipse IDE in such a way that all necessary tasks can be directly triggered from this development environment. [medium]

5.6 Demonstrators

The following requirements concern the demonstrators which are to be developed within the scope of the Q-ImPrESS project. The demonstrators will be used for the dissemination of the project results amongst other purposes.

- **[R_DEM_1] Practical industrial scenario:** The demonstrators are based on a practical industrial scenario, which demonstrates the applicability of the methods and tools developed within the scope of the Q-ImPrESS project on such a scenarios. [high]
- **[R_DEM_2] Evolution scenarios:** The demonstrators will contain a set of evolution scenarios, which show the strength of the Q-ImPrESS approach with regard to software evolution. [high]
- **[R_DEM_3] Open Source Demonstrator:** [high]
 - **[R_DEM_3.1] Open Source:** The demonstrator is to be developed under a open source license. This allows the demonstrator to be easily spread amongst the scientific and industrial communities. [high]
 - **[R_DEM_3.2] Open Source middleware and frameworks:** The demonstrator is to be built on top of open source middleware and frameworks for the aforementioned reason. The demonstrator will be written in Java, based on an Enterprise Service Bus (ESB). [high]
 - **[R_DEM_3.3] Usage simulators:** It is possible to simulate the usage of the demonstrator. This allows to validate the results of the Q-ImPrESS prediction against the behavior of the implemented and deployed system. [high]
- **[R_DEM_4] Industrial Demonstrators:** [high]

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

- **[R_DEM_4.1] Legacy source:** At least one industrial demonstrator will be built on a traditional runtime platform and will be implemented in C/C++ using technologies like COM and MFC. [high]
- **[R_DEM_4.2] Medium to High complexity:** The industrial demonstrators will be of considerable size (i.e. in the range of hundred thousands lines of code) and system complexity to make sure the Q-ImPrESS tools and methods are applicable to real-life systems. [high]

5.7 Method Documentation

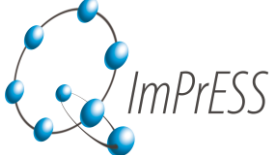
- **[R_DOC_1] Working method documentation.** [high]
- **[R_DOC_2] Domain Specific Guidelines:** For each of the targeted domains there are domain specific guidelines, which allow an easy adoption of the Q-ImPrESS methods and tools into existing development processes. [medium]
- **[R_DOC_3] Best Practices:** The documentation contains a cookbook composed of best practices for the usage of the Q-ImPrESS tools and methods. [medium]

6 Non-functional requirements

Non-functional requirements described in this chapter are prioritized into three levels: low, medium and high. Groups of non-functional requirements correspond to the same groups of functional requirements in chapter 5. High priority of the requirement means mandatory fulfillment of the requirement. Medium priority of the requirements means the requirement is important, but depending on time constraints and agreements among all partners, the requirement doesn't have to be fulfilled. It is important to note that requirements of medium priority do not hinder proof-of-concept implementation, but provide further usability of the method and the tools. Low priority of the requirements means the requirement is optional and can be fulfilled if time-constraints permit so.

6.1 Service Architecture Modelling

- **[NR_SAM_1] Usability:** The modelling tools shall provide a graphical representation of the Static model, the Behavioural model, the Allocation model, the Usage model, and the Quality annotations. They shall also allow editing the models in this representation. The graphical representation must be easy to understand and thus save time for model creation and modification in relation to textual modeling. [high]
- **[NR_SAM_2] Performance:** It should be possible to work fluidly (UI response times < 1s) with model repositories containing in the range of 1000 components (each modelled with interfaces and behavioural descriptions) using treeview or textual editors. Diagrams (i.e. views of the model) can contain in the range of 50 elements, thus, graphical diagram editors should also allow editing such diagrams fluidly (UI response time < 1s). [low]
- **[NR_SAM_3] Reliability:** The modelling tools shall be well tested and enforce error-free modelling by providing mechanisms to check models or forbid invalid operations. [low]
- **[NR_SAM_4] Maintainability / Modifiability:** Evolution of the service architecture meta-model (SAMM, as defined in D2.1) shall be possible to fix errors in the model and to extend the model for other quality attributes. Modifying the meta-model requires migrating existing models. There should be at least semi-automatic tools to support the migration of model instances. [medium]

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

6.2 Service Architecture Extraction

- **[NR_SAE_1] Performance / Scalability:** It shall be possible to reverse engineer systems consisting of less than 5 MLOC. This functionality should be performed semi automatically in less than 8 hours (1 person day). [high]
- **[NR_SAE_2] Maintainability:** The tools shall allow developers to quickly react on failures in the reverse engineering process. Therefore, there should be enough sufficient developer documentation to allow independent (i.e. for non-project members) maintenance and extension of the reverse engineering tools during project execution and after the project has ended. [medium]
- **[NR_SAE_3] Extensibility:** To broaden the scope of the reverse engineering tools, it shall be possible to extend the tools, so that they for example support additional programming languages (e.g., C#). This implies a sufficient documentation of the tool front-end. [medium]
- **[NR_SAE_4] Usability:** The tools shall be configurable and sufficiently documented, so that a non-expert can learn to use the tools in less than 1 person day. [medium]

6.3 Quality Impact Prediction

- **[NR_QIP_1] Performance / Scalability:** The simulation and analysis tools shall be able to make performance and reliability predictions for models in the range of 50 components in a reasonable amount of time. For initial, low confidence results, the tools shall provide prediction results in less than 1 hour. For detailed, high confidence results, the tools shall provide prediction results in less than 8 hours (1 person day). At any time the tools shall inform the user about the simulation or analysis process with intermediate results and progress bars. This enables users to abort predictions prematurely. [high]
- **[NR_QIP_2] Usability:** The prediction tools shall be integrated into the modeling tools, so that tool usage becomes seamless. It is furthermore important that the tools visualize the prediction results in an easy-to-understand fashion that requires limited special knowledge. [medium]

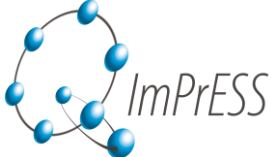
7 Constraints

After the initial version of this document was reviewed by all Q-ImPRESS partners, certain constraints were noticed in regard to feasibility or understanding of certain functional requirements. Thus, this chapter explains the reasons why certain requirements were omitted in final version of requirements document, and why certain requirements were significantly modified in comparison to the same requirements described in initial version of requirements document.

Additionally, section 7.6 explains the Q-ImPRESS view on scalability, as requested by reviewers at the first project review held in Bruxelles, February 2009.

7.1 Service Architecture Modeling

- **Models name change:** the names of service architecture models described in initial version of requirements document were changed according to names of equivalent models defined in D2.1. This change will help following WPs in easier understanding of requirements.
- **Separation of static and behavioral modeling:** "Common Service Architecture

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

Model (CSAM)" defined in initial version of requirements document was separated into Static Model and Behavioral Model according to models defined in D2.1. CSAM model combined information that is now present in Static Model and Behavioral Model.

- **One deployment model:** in accordance with D2.1. "Hardware and Middleware Deployment Model (HMDM)" and "Component Deployment Model (CDM)" were combined into Deployment Model. All information present in HMDM and CDM is included in Deployment Model.

7.2 Service Architecture Extraction

- **Targeted system size:** extraction tools support targeted system size up to a few millions of LOC. For larger systems no guarantees regarding extraction tools' functionality can be made.
- **Tool-supported instrumentation:** this requirement was omitted due to its ambiguity.

7.3 Quality Impact Prediction

- **Quality impact prediction based on different design alternatives:** Q-ImPrESS tools can't provide conclusions regarding quality impact prediction for different user-defined design alternatives. Thus, user can compare models of different solutions, and based on that, make conclusions.

7.4 Trade-off Analysis

- **User reasoning regarding trade-off analysis:** Q-ImPrESS tools can't provide conclusions regarding user defined trade-off analysis. Thus, user creates and runs different trade-off scenarios and makes conclusions based on their results.

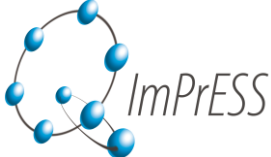
7.5 Tools Support

- **No integration with Visual Studio:** Q-ImPrESS tools will not be integrated with Visual Studio due to lack of manpower.
- **Efficient evolution:** this requirement is omitted because it is covered by requirements [R_QIP_3.1] and [R_QIP_3.2].

7.6 Scalability

In the context of Q-ImPrESS scalability is seen as a meta-attribute, i.e., it is the relationship of an input attribute to an output value. For example, we could look at the scalability of the system's response time (output) when the number of concurrent users increases (input). Or we could consider the change in the throughput (output) when the type and size of the user's parameters vary (input). This view becomes available due to the explicit usage model available in Q-ImPrESS which is commonly unavailable in related approaches.

However, in Q-ImPrESS we can compare scalability therefore only in a discrete manner meaning that we can conduct several analyses changing the input variable manually each time and compare the outputs. For example, we can perform single response time analyses and have 1, 5, 10, 20, 30, etc. concurrent users accessing the system. In so doing, we get an approximation of the system's scalability behaviour. However, Q-ImPrESS is limited if the Q-ImPrESS users want to compare the scalability of two design alternatives directly (meaning as function) as the analyses never return the overall scalability of the system's per se. We can

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

only compare the scalability for distinct values of the input variable, e.g., for distinct values of the amount of concurrent users.

8 Documentation

The documentation provided with the software tools should include:

- an installation manual for each tool of the Q-ImPrESS suite,
- an operator manual for each tool of the Q-ImPrESS suite,
- a manual of generic guidelines on how to use the suite in a generic environment,
- a set of domain specific guidelines and cookbooks for the environments targeted by the project (Industry, Telecommunication, Enterprise).

9 Requirements validation

This chapter presents initial guidelines for requirements validation. Information provided in this chapter serves as a basis for detailed description of requirements validation, to be presented in D7.1 ("Demonstrator description - final version"), as planned in Description of Work of the Q-ImPrESS project.

In order to explain how the requirements are going to be used in validation activities we use system development process supported by Q-ImPrESS (see Figure 4) that is applicable to all industrial partners which will perform requirements validation. Also, we integrate the knowledge and experience from the work on D6.1 ("Method and abstract workflows documentation - initial version") in order to describe general guidelines for requirements validation that are common for all industrial partners. As previously stated, detailed requirements validation process is going to be described by each industrial partner separately (adjusting the validation process to its demonstrator and its industrial development processes) in D7.1 ("Demonstrator description - final version").

Initial mapping between functional and non-functional requirements, and development process common to all industrial partners is presented in Figure 11. The figure presents the mapping between the requirements and the process in two parts. The first part, presented in the left side and the center of the figure, addresses the mapping between the requirements and the common process workflow. Used common process workflow, here simplified, corresponds to common workflow described in D6.1 ("Method and abstract workflows documentation - initial version"). The second part, presented in the right side of the figure, additionally addresses the mapping between the requirements and the tools that will be produced by the Q-ImPrESS project and that will be used in the common workflow. The general requirement imposed on the Q-ImPrESS tools requires the tools to integrate with Eclipse IDE (R_TS_1 in Section 5.5)

The workflow common to all industrial partners starts with determination of requirements imposed on the to-be-designed and to-be-implemented system. When the system requirements are determined, design and implementation scenarios are created. These scenarios describe potential design and implementation solutions that address the issues requested by the requirements. With the regards to Q-ImPrESS project, these scenarios encompass all the requirements imposed on the demonstrators (R_DEM in Section 5.6). These demonstrator requirements guide industrial partners in selection of development and prototype projects which are going to be used within the scope of Q-ImPrESS project. Furthermore, since there can be more than one design and implementation scenario related to one devised system, these scenarios represent design alternatives for the originally devised system. Thus, the

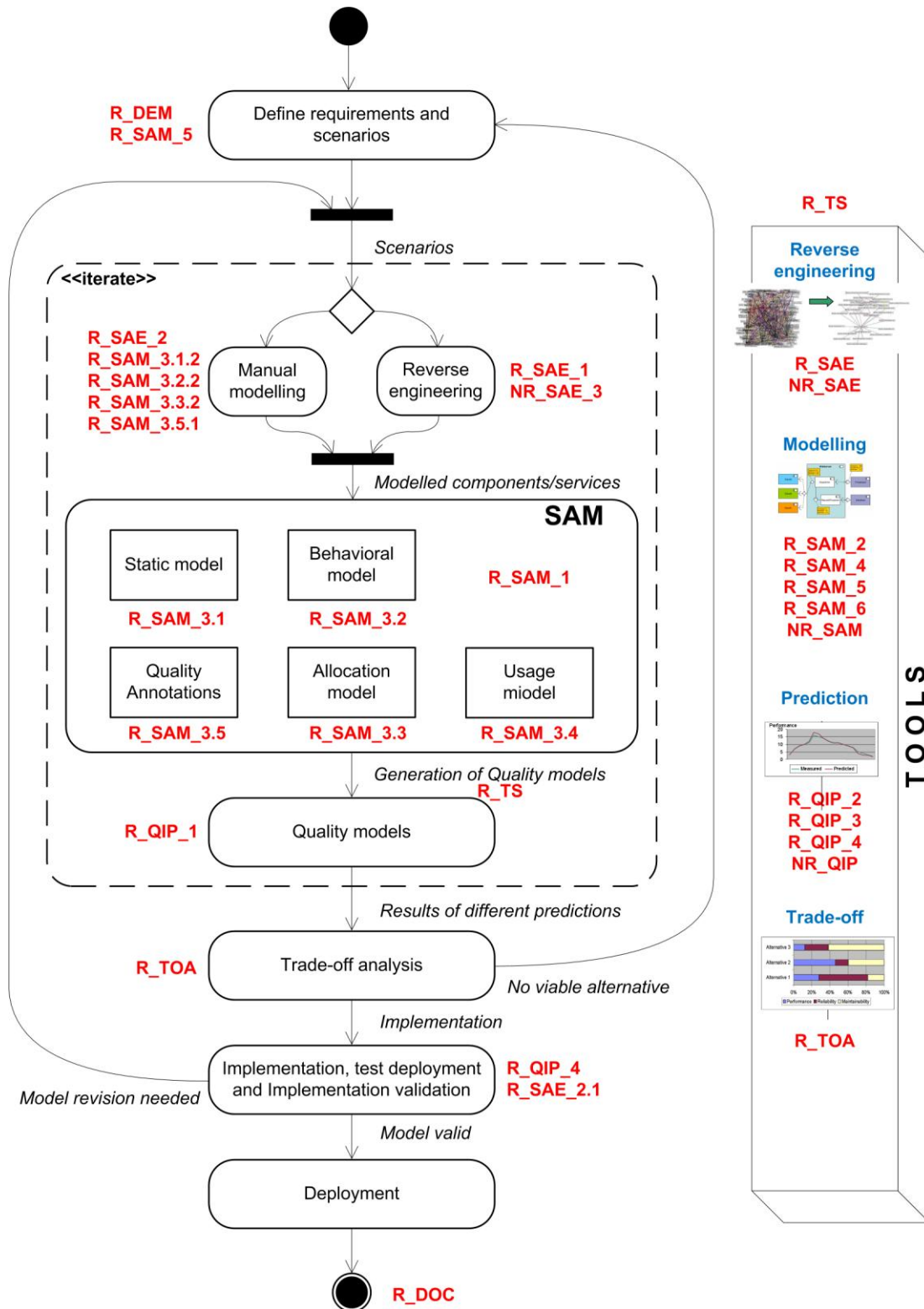
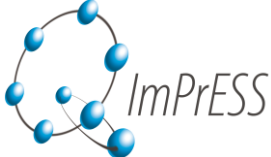


Figure 11. Mapping between requirements and system development process common to all industrial partners

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

ability of creating and using design alternatives (R_SAM_5) is also addressed in this step. It is important to note that during system lifecycle, design alternatives can represent the evolution of the system.


After finalizing the definition of the scenarios, the Software Architect starts the design of the system that addresses the requirements. System design is an iterative process consisting of choosing existing solutions and creating new ones. Existing solutions consist of old, proven legacy and proprietary systems, off-the-shelf components and solutions, as well as old code written in programming languages that are still used today (i.e. C/C++). However, the new solutions do not exist (are not implemented yet) and, thus, must be devised and modelled first. So, certain parts of the overall architecture that describes the design of the system can be modelled manually or modelled by reverse engineering.

Parts of the system that can be reverse engineered are going to be modelled by the reverse engineering tools. The tools will analyze available source code and will produce a partial model that corresponds to the analyzed parts of the system. The analysis assesses the possibilities of reverse engineering with regards to size of reverse engineered system (R_SAE_1.1), programming languages used for implementing the system (R_SAE_1.2), proper discovery of service information from available source code (R_SAE_1.3) and extensibility of the reverse engineering process (R_SAE_1.4, NR_SAE_3). Furthermore, reverse engineering tools itself will additionally be assessed through ability of monitoring existing solution's source code (R_SAE_2), ability to handle systems consisting of several MLOCs (NR_SAE_1), developers allowance to quickly react on failures in the reverse engineering process (NR_SAE_2), as well as configurability and documentation (NR_SAE_4).

New solutions that are not yet implemented (and thus cannot be reverse engineered) must be manually modelled first. This also applies to existing solutions that either have no available source code or have source code written in proprietary programming language. In regards to Q-ImPRESS project, the manual modelling of system's parts assesses possibility of monitoring and measuring legacy code (R_SAE_2) in order to get information for precise descriptions of manually modelled parts of the system, as well as the general possibility of creating descriptions and modelling of parts of the system that cannot be reverse engineered (R_SAM_3.1.2, R_SAM_3.2.2, R_SAM_3.3.2, R_SAM_3.5.1). It is our wish to have the ability to manually describe parts of the system as if these parts were perfectly reverse engineered by the reverse engineering tools produced by the Q-ImPRESS project. As for the modelling tools itself, this phase also assesses the ability of transforming models from and to Q-ImPRESS model (R_SAM_2), the ability to graphically created models (R_SAM_4), the ability to create design alternatives for any model (R_SAM_5) and the ability of performing syntax checks of model's validity (R_SAM_6). Furthermore, the tools will be assessed by their usability (NR_SAM_1), performance (NR_SAM_2), reliability of producing valid models (NR_SAM_3) and simplicity of creating extension to existing models (NR_SAM_4).

The result of both manual modelling and reverse engineering is a Service Architecture Model (SAM) of the system. SAM must comply with requirement of system view separation (R_SAM_1) which enables modelling of different parts of the system through the following models: Static Model (R_SAM_3.1), Behavioral model (R_SAM_3.2), Allocation Model (R_SAM_3.3), Usage Model (R_SAM_3.4) and Quality Annotations (R_SAM_3.5).

After the SAM is created, all the necessary information for modelled system's quality prediction is present in the Q-ImPRESS process. The quality prediction provides prediction of system's performance, reliability and maintainability (R_QIP_1). Quality prediction tools

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

themselves should have the ability to provide prediction through simulation and different types of analysis (R_QIP_2), comparison of quality impact prediction based on different design alternatives (R_QIP_3) and possibility of deploying the system into a test environment (R_QIP_4). Furthermore, prediction tools should be seamlessly integrated with modelling tools (NR_QIP_2) and should provide predictions of a relatively complex system in a reasonable amount of time (NR_QIP_1).

Results of different predictions provide a basis for performing trade-off analysis. Trade-off analysis enable comparison of design alternatives with regard to their impact on service quality. The trade-off analysis and accompanying tools must provide trade-off between different quality attributes (R_TOA_1) and enable creation of different trade-off scenarios and to make conclusions regarding these scenarios by re-running simulations (R_TOA_2).

In the case of unsatisfactory prediction results by all possible trade-off scenarios, it is necessary to redefine the original requirements imposed on the designed system, or perhaps to try alternative system designs in order to fulfil the original requirements.

However, if the prediction results of modelled system are satisfactory, the system can be implemented. Once implemented, system's quality attributes are validated against predicted values of corresponding quality attributes. If there is a good-enough correspondence between predicted values of the modelled system and the real-world measured of the implemented system, the system can be deployed. However, if there is a significant discrepancy between the predicted values and the measured values, it is necessary to re-design (re-model) the system in order to understand why such discrepancy occurred and in order to correct it.

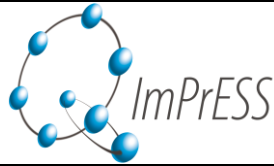
Deployed system presents the end of the common workflow for all industrial partners. The experiences of getting to this last phase of the workflow provide basis for creating Q-ImPrESS method documentation for each of the domains (R_DOC_1), as well as general cookbook and best practices (R_DOC_2).

9.1 Planned requirements validation per demonstrator

In this section we present the table containing plans for using specific demonstrator for validating certain requirements. These plans supplement information given in the introduction of this chapter and serve as an additional guideline for requirements validation. The table and the next section address the issues pointed out by the reviewers (recommendation [R1], part 3) in the Q-ImPrESS technical review report from February the 19th, 2009.

It is important to note that information presented in this section serves as a basis for detailed description of requirements validation, to be presented in D7.1 ("Demonstrator description - final version"), as planned in Description of Work of the Q-ImPrESS project.

Table 1. summarizes requirements presented in chapters 5 and 6, and shows if a demonstrator (automation demonstrator, enterprise demonstrator or telecom demonstrator) is going to validate certain requirement. The table consists of following columns: "No", "ID", "Title", "Priority", "Exec", "Automation", "Enterprise" and "Telecom". The column "No" designates ordered number to a requirement. The column "ID" corresponds to requirement ID as assigned in chapters 5 and 6. The column "Title" shows summarized requirement's title, as presented in chapters 5 and 6. The column "Priority" shows requirement's priority, as presented in chapters 5 and 6. The column "Exec" states if a requirement must be implemented. Requirements with the word "MUST" filled in the "Exec" column must be implemented. On the other hand, if the "Exec" field for certain requirement is empty, this means that the requirement doesn't hinder Q-ImPrESS proof-of-concept and doesn't have to be implemented in case of time or resource constraints. The column "Automation" designates

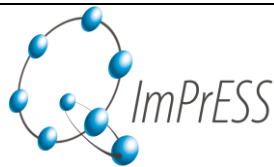


which requirements are going to be validated by the automation demonstrator. The column "Enterprise" designates which requirements are going to be validated by the enterprise demonstrator. The column "Telecom" designates which requirements are going to be validated by the telecom demonstrator. The last three columns are filled with fields containing text "YES", "NO" and "Partly". The "YES" text designates that the demonstrator is going to be used for validating the requirement. The "NO" text designates that the demonstrator is not going to be used for validating the requirement. The "Partly" text designates that the demonstrator is going to be used for validating the requirement with certain restrictions to validation. If possible, a comment below the table further explains the "NO" and "Partly" text. As stated before, detailed information regarding requirements validation per demonstrator is going to be presented in D7.1 ("Demonstrator description - final version"), as planned in Description of Work of the Q-ImPRESS project.

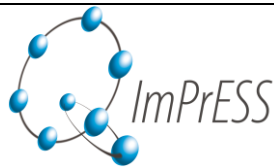
Additionally it is important to note that the Table 1. can easily be used as an requirements execution tracking tool. During validation, each industrial partner can add a column where it will be noted if a requirement was fully or partially implemented, or wasn't implemented at all.

Table 1. Planned requirements validation per demonstrator

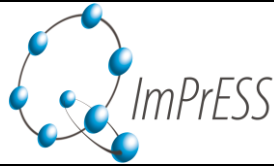
No	ID	Title	Priority	Exec	Planned validation		
					Automation	Enterprise	Telecom
		Service Architecture Modeling					
1	R_SAM_1	Separation of System Views	high	MUST	YES	YES	YES
2	R_SAM_2.1	Transformations from Q-ImPRESS model	high	MUST	YES	YES	YES
3	R_SAM_2.2	Transformations to Q-ImPRESS model	high	MUST	YES	YES	YES
4	R_SAM_3	Relevant models	high	MUST	YES	YES	YES
5	R_SAM_3.1	Static Model	high	MUST	YES	YES	YES
6	R_SAM_3.1.1	Connection of Static Model and implementation Artifacts	medium	MUST	YES	YES	YES
7	R_SAM_3.1.2	Black Box Artifacts enabling	high	MUST	YES	YES	YES
8	R_SAM_3.1.3	Common Data Store for Method-Specific models	high	MUST	YES	YES	YES
9	R_SAM_3.1.4	The model should cover static information	high	MUST	YES	YES	YES
10	R_SAM_3.2	Behavioral Model	high	MUST	YES	YES	YES
11	R_SAM_3.2.1	Model contains dynamic information	high	MUST	YES	YES	YES
12	R_SAM_3.2.2	Black Box described through quality annotations	high	MUST	YES	YES	YES
13	R_SAM_3.3	Allocation Model	high	MUST	YES	YES	YES
14	R_SAM_3.3.1	Model contains physical and logical resources	high	MUST	YES	YES	YES
15	R_SAM_3.3.2	Additional logical resource modeling	high	MUST	NO	YES	Partly
16	R_SAM_3.3.3	Mapping between Static and Allocation Model	high	MUST	YES	YES	YES
17	R_SAM_3.4	Usage Model	high	MUST	YES	YES	YES
18	R_SAM_3.5	Quality Annotations	high	MUST	YES	YES	YES



19	R_SAM_3.5.1	Black Box Behavior description	high	MUST	YES	YES	YES
20	R_SAM_4	Model Creation	medium		YES	YES	YES
21	R_SAM_5	Design Alternatives	high	MUST	YES	YES	YES
22	R_SAM_5.1	Design alternatives as parts of models	high	MUST	YES	YES	YES
23	R_SAM_5.2	Incremental adaptation of the model changes	high	MUST	YES	YES	YES
24	R_SAM_6	Model Validation	medium		YES	YES	YES
		Service Architecture Model Extraction					
25	R_SAE_1	Service Re-Engineering from Legacy Code	medium		YES	Partly (*ITE_1)	Partly (*ENT_1)
26	R_SAE_1.1	Targeted System Size					
		up to 500 kLOC	high	MUST	YES	NO	Partly
		1+ MLOC	medium		YES	NO	Partly
27	R_SAE_1.2	Programming Languages					
		Java	medium		NO	Partly (*ITE_1)	Partly
		C++	high	MUST	YES	NO	Partly
		C#	low		YES	NO	Partly
28	R_SAE_1.3	Service Discovery	medium		YES	Partly (*ITE_1)	Partly
29	R_SAE_1.4	Extensibility	medium		YES	Partly (*ITE_1)	Partly
30	R_SAE_2	Monitoring and Measuring Legacy Code	low		YES	YES	Partly (*ENT_2)
31	R_SAE_2.1	Quality Attribute Estimation					
		Performance properties	high	MUST	YES	YES	Partly
		Reliability properties	low		NO	Partly	Partly
		Usage patterns	medium		Partly	Partly	Partly
		Quality Impact Prediction					
32	R_QIP_1	Quality Prediction Metrics	high	MUST	YES	YES	YES
33	R_QIP_1.1	Performance	high	MUST	YES	YES	YES
34	R_QIP_1.1.1	Average case and predicted maximum resource consumption	high	MUST	YES	YES	YES
35	R_QIP_1.1.2	Average and worst case response time	high	MUST	YES	YES	YES
36	R_QIP_1.1.3	Average case and worst case throughput	high	MUST	YES	YES	YES
37	R_QIP_1.1.4	Determining minimal hardware	medium		YES	YES	Partly
38	R_QIP_1.2	Reliability	high	MUST	Partly (*ABB_3)	YES	YES
39	R_QIP_1.2.1	MTBF	high	MUST	Partly	YES	YES
40	R_QIP_1.2.2	MTTF	high	MUST	Partly	YES	YES
41	R_QIP_1.2.3	MTTR	medium		Partly	Partly	Partly
42	R_QIP_1.2.4	Failure probability per demand	medium		Partly	Partly	Partly
43	R_QIP_1.3	Maintainability	low		Partly (*ABB_3)	YES	YES



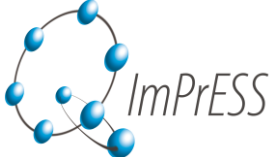
44	R_QIP_1.3.1	Time and Cost Effort of Maintenance Task/Change	low		Partly	YES	YES
45	R_QIP_1.3.2	Violations of Design Principles	low		Partly	YES	Partly
46	R_QIP_2	Simulation and Analysis	high	MUST	YES	YES	YES
47	R_QIP_2.1	Subsystem Simulation and Analysis	medium		YES	YES	YES
48	R_QIP_2.2	Specific Metric Simulation and Analysis	high	MUST	YES	YES	YES
49	R_QIP_3	Quality impact prediction based on different given design alternatives	high	MUST	YES	YES	YES
50	R_QIP_3.1	Quality impact of different architecture designs	high	MUST	YES	YES	YES
51	R_QIP_3.2	Quality impact of different allocation scenarios	high	MUST	YES	YES	YES
52	R_QIP_4	Test allocation	low		NO	Partly	Partly
53	R_QIP_4.1	Mock implementations	low		NO	Partly	Partly
54	R_QIP_4.2	Monitoring and Measuring Facilities	low		NO	Partly	Partly
		Trade-off Analysis					
55	R_TOA_1	Trade-off Between Different Quality Attributes	high	MUST	Partly	YES	YES
56	R_TOA_1.1	Storing Results	high	MUST	Partly	YES	YES
57	R_TOA_2	User trade-off scenarios	high	MUST	Partly	YES	YES
		Tool Support					
58	R_TS_1	Integration into the Eclipse IDE	medium		NO	YES	YES
		Demonstrators					
59	R_DEM_1	Practical industrial scenario	high	MUST	YES	YES	YES
60	R_DEM_2	Evolution scenario	high	MUST	YES	YES	YES
61	R_DEM_3	Open Source Demonstrator	high	MUST	NO	YES	Partly
62	R_DEM_3.1	Open Source	high	MUST	NO	YES	NO (*ENT_3)
63	R_DEM_3.2	Open Source middleware and frameworks	high	MUST	NO	YES	Partly (*ENT_4)
64	R_DEM_3.3	Usage simulators	high	MUST	YES	YES	YES
65	R_DEM_4	Industrial demonstrators	high	MUST	YES	NO	YES
66	R_DEM_4.1	Legacy source	high	MUST	YES	NO	Partly (*ENT_5)
67	R_DEM_4.2	Medium to high complexity	high	MUST	YES	NO	YES
		Method Documentation					
68	R_DOC_1	Working method documentation	high	MUST	YES	YES	YES
69	R_DOC_2	Domain Specific Guidelines	medium		YES (*GEN_1)	YES (*GEN_1)	YES (*GEN_1)
70	R_DOC_3	Best practices	medium		YES (*GEN_1)	YES (*GEN_1)	YES (*GEN_1)
		N-F Service Architecture Modeling					
71	NR_SAM_1	Usability	high	MUST	YES	YES	YES
72	NR_SAM_2	Performance	low		YES	YES	YES



73	NR_SAM_3	Reliability	low		YES	YES	YES
73	NR_SAM_4	Maintainability/Modifiability	medium		Partly (*ABB_1)	YES	YES
		N-F Service Architecture Extraction					
74	NR_SAE_1	Performance/Scalability	high	MUST	YES	Partly (*ITE_1)	Partly (*ENT_1)
75	NR_SAE_2	Maintainability	medium		Partly (*ABB_1)	Partly (*ITE_1)	YES
76	NR_SAE_3	Extensibility	medium		Partly (*ABB_2)	Partly (*ITE_1)	YES
77	NR_SAE_4	Usability	medium		YES	Partly (*ITE_1)	Partly
		N-F Quality Impact Prediction					
78	NR_QIP_1	Performance/Scalability	high	MUST	YES	YES	YES
79	NR_QIP_2	Usability	medium		YES	YES	YES

Additional comments for table 1:

Comment	Description
(*GEN_1)	This requirement is not validated by the demonstrators. However, this requirement describes one of the outputs of the validation process. "YES" means we will produce the output stated in the requirement
(*ABB_1)	To the extent needed
(*ABB_2)	i.e. support for C#
(*ABB_3)	We are interested in the prediction of reliability and maintainability, but we cannot guarantee to provide the necessary input data (e.g., failure probabilities, costs) for the prediction methods.
(*ENT_1)	As stated before, although we do not plan to use reverse engineering (due to proprietary nature of our legacy systems) we can test reverse engineering tools by trying to reverse engineer software like OpenSAF and OpenDIAMETER which we are going to use in our demonstrator
(*ENT_2)	While we will be able to use our proprietary tools for measuring and monitoring our legacy code (in order to provide descriptions for black-boxes that model the legacy code), we cannot guarantee we'll be able to apply Q-ImPRESS tools for monitoring and measuring services (deployed components). We will try, but due to proprietary nature of our systems we cannot make any guarantees
(*ENT_3)	We cannot make open-source demonstrator. Our systems are proprietary.
(*ENT_4)	The open-source solutions that are going to be used in our demonstrator are implemented in C and C++. Thus, we cannot guarantee implementation in Java and ESB
(*ENT_5)	Our "true" legacy code is written in proprietary languages. However, OpenSAF and OpenDIAMETER, which are going to be used in the demonstrator, are written in C/C++.
(*ITE_1)	We do not plan to use reverse engineering, as we already do have models of our systems. However, we can test reverse engineering tools by trying to reverse engineer parts of the demonstrator and comparing against manually created models.

	D1.1: Requirements document - final version	
	Version: 1.31	Last change: 2009-Apr-30

9.2 Demonstrator focus

In this section we briefly present the information stating demonstrators focus in regards to validation activities. This information further supplements information already presented in this chapter with the goal of providing insight in planned validation activities and the mapping between requirements and validation activities. However, once more we must emphasize that information presented in this section serves as a basis for detailed description of requirements validation, to be presented in D7.1 ("Demonstrator description - final version"), as planned in Description of Work of the Q-ImPrESS project.

The Automation demonstrator focuses on performance and reliability prediction methods of Q-ImPrESS and will also evaluate the maintainability prediction methods. It requires support for measuring performance properties and estimating failure probabilities. In addition, reverse engineering and architecture reconstruction methods and tools to support model creation and evolution are of great importance and will be validated by the Automation demonstrator.

The Enterprise SOA showcase enables the demonstration of all Quality Impact Prediction methods and tools which are developed within the scope of Q-ImPrESS, with a focus on performance prediction. In addition to the efficient creation and editing of Service Architecture Models, the reuse and transformation of existing models to the Q-ImPrESS SAMM are of great importance. Usage simulators as well as assembly and deployment alternatives enable the validation of all Quality Impact Prediction and Trade-off Analysis tools. The use of Open Source Licenses for the implementation, used middleware and frameworks as well as the integration into the Open Source Eclipse IDE ensures the availability of the E-SOA showcase for public demonstration purposes.

The Telecommunications demonstrator focuses on performance and reliability quality attributes, prediction methods and the tools resulting from the Q-ImPrESS project. The demonstrator will test and validate the efficiency of modelling, specially the ability to qualitatively and quantitatively describe black-box components. Since performance related quality attributes of telecommunications legacy systems are monitored and measured with custom made and third-party software, the demonstrator will also test the applicability of information provided by such software. The demonstrator will test and validate trade-off analysis capabilities, specially in regards to semi-automatic comparison of performance vs. reliability trade-off analysis of a single architectural model, and to manual comparison of performance vs. reliability trade-off analysis of multiple architectural models. Furthermore, although the demonstrator isn't planned to be used for reverse engineering, the reverse engineering tools will be tested by trying to reverse engineer parts of the demonstrator that can be used for such task. Telecommunications demonstrator will also evaluate the applicability of maintainability quality attributes, prediction methods and tools. However maintainability isn't in the focus of the demonstration activities.

It is important to note that validation of reliability and maintainability predictions is a non-trivial task. Thus within the scope of the demonstrators, we will do our best-effort to collect the needed input and real-world data as good as it is known today.