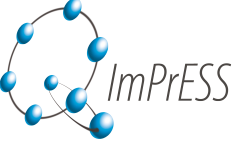


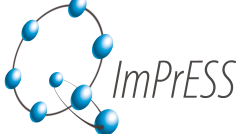
# Project Deliverable D3.1 Prediction Models Specification (revised version)

**Project name:** Q-ImPrESS  
**Contract number:** FP7-215013  
**Project deliverable:** D3.1: Prediction Models Specification (revised version)  
**Author(s):** Danilo Ardagna, Steffen Becker, Aida Causevic, Carlo Ghezzi, Vincenzo Grassi, Lucia Kapova, Klaus Krogmann, Raffaella Mirandola, Cristina Seceleanu, Johannes Stammel, Petr Tuma  
**Work package:** WP3  
**Work package leader:** PMI  
**Planned delivery date:** M21  
**Delivery date:** September 15, 2009  
**Last change:** September 15, 2009  
**Version number:** 2.0

**Keywords** Quality Models, Quality Annotations, Quality Metrics

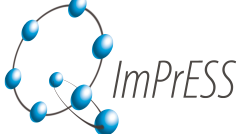
	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

**Abstract** This document is a revised version of Deliverable D3.1 and includes the description of prediction models which support the estimate of the quality attributes considered in the Q-ImPrESS project. Models are discussed within the Model Driven Development framework and their points of strength and weakness are analyzed. An example of use of the models with respect to the client-server architecture introduced in Q-ImPrESS Deliverable D2.1 is also provided and the interrelationships among the model parameters and the Q-ImPrESS Service Architecture Meta-Model is analyzed. Furthermore, the document describes the selection of prediction models used in Q-ImPrESS project for performance and reliability estimates and presents an original approach for the evaluation of maintainability property.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

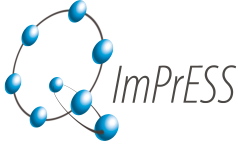
## Revision history

Version	Change Date	Author(s)	Description
0.1	01-10-2008	PMI	Initial version of the document
0.11	20-10-2008	FZI	Added LQN Section
0.12	30-10-2008	PMI	Added MDD Framework Section
0.13	05-11-2008	PMI	Improved Layout and Formatting, revised version of metrics
0.14	05-11-2008	PMI, FZI	Added running example
0.2	14-11-2008	PMI, FZI, MDU, CUNI	Added Automata Section, SAMM annotations, Resource sharing analysis, Document revision
0.3	20-11-2008	PMI, FZI	Added Maintainability metrics and analysis, Document revision
0.4	26-11-2008	PMI	Document revision
0.5	28-11-2008	PMI, FZI, MDU	Final document revision
1.0	29-11-2008	PMI	Final Version
1.1	26-07-2009	PMI	Document revision
1.2	20-08-2009	PMI, FZI	Added Model selection part
1.3	10-09-2009	FZI	Added Maintainability part
1.4	14-09-2009	PMI, FZI	Document revision
2.0	15-09-2009	PMI	Final Version

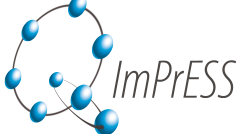
	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Quality Metric Definition</b>	<b>7</b>
2.1	Performance	7
2.2	Dependability	8
2.3	Maintainability	9
<b>3</b>	<b>SAMM Quality Annotations</b>	<b>15</b>
3.1	Quality Annotation Requirements	15
3.2	Quality Annotation Metamodel	16
3.3	Quality Annotation Integration in the Service Architecture Meta-Model	17
3.3.1	Control flow	17
3.3.2	Data flow	18
3.3.3	Usage model	18
3.3.4	Black box	18
3.3.5	Performance	19
3.3.6	Reliability	19
3.3.7	Maintainability	19
3.3.8	Remarks	20
<b>4</b>	<b>Q-ImPrESS Model Driven Development Framework for Quality Analysis</b>	<b>21</b>
<b>5</b>	<b>Performance and Reliability Models</b>	<b>24</b>
5.1	Queueing Network Models	24
5.1.1	Product-form QN	26
5.1.2	Open and Closed Models	27
5.1.3	Single and Multi-Class Models	27
5.1.4	A running example	28
5.1.5	Solution Techniques	28
5.1.6	Strength and Weakness	30
5.1.7	Model Adoption	31
5.1.8	Tools for Model Derivation	31
5.2	Layered Queueing Networks	32
5.2.1	A running example	34
5.2.2	Solution Techniques	35
5.2.3	Strength and Weakness	35
5.2.4	Model Adoption	35
5.2.5	Tools for Model Derivation	35

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

5.3	Markov Models	36
5.3.1	Discrete Time Markov Chains	36
5.3.2	Continuous Time Markov Chains	38
5.3.3	Markov Decision Processes	39
5.3.4	Stochastic Model Checking	40
5.3.5	A running example	41
5.3.6	Solution Techniques	41
5.3.7	Strength and Weakness	42
5.3.8	Model Adoption	43
5.3.9	Tools for Model Derivation	43
5.4	Automata Models	43
5.4.1	Priced Timed Automata	44
5.4.2	Multi Priced Timed Automata	45
5.4.3	Weighted CTL	45
5.4.4	A Running Example	46
5.4.5	Solution Techniques	49
5.4.6	Strength and Weakness	50
5.4.7	Model Adoption	50
5.4.8	Tools for Model Derivation	51
5.5	Simulation Models	51
5.5.1	Strength and Weakness	52
5.5.2	Model Adoption	52
5.5.3	Tools for Model Derivation	53
5.6	Control-Oriented Models	53
5.6.1	Strength and Weakness	54
5.6.2	Model Adoption	54
5.7	Model Comparison and Discussion	54
5.8	Model Selection	58
<b>6</b>	<b>Maintainability Analysis</b>	<b>60</b>
6.1	Running Example	60
6.2	Maintainability Prediction Process	60
6.2.1	High-level Process Description	61
6.2.2	Preparation Phase	62
6.2.3	Maintainability Analysis Phase	63
<b>7</b>	<b>Conclusions</b>	<b>67</b>

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

# 1 Introduction

This document represents Deliverable D3.1 of the Q-ImPrESS description of work. It discusses the prediction models that support the estimate of the quality attributes considered in the Q-ImPrESS project. Models are discussed within the Model Driven Development (MDD) [1, 2] framework and their points of strength and weakness are analyzed. The proposed mapping provides a taxonomy, which tries to capture the main facets that are needed to understand, choose, and use models appropriately in the various phases of service oriented software development.

The analyzed models will be implemented in the Q-ImPrESS project tool suite. An example of use of the models for the prediction of the quality metrics of interest of the client-server architecture introduced in Q-ImPrESS Deliverable D2.1 is discussed. Furthermore, the interrelationships among the model parameters and the Q-ImPrESS Service Architecture Meta-Model (SAMM) are also analyzed.

This document is structured as follows:

**Chapter 2:** Chapter 2 discusses the quality metrics of interest which are in the focus of the Q-ImPrESS project according to the industrial partners requirements listed in Q-ImPrESS Deliverable D1.1.

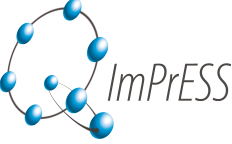
**Chapter 3:** Chapter 3 provides the quality annotation meta-model needed on various elements of the Service Architecture Meta-Model to detail the quality aspects. In particular, the requirements for the quality annotations, the metamodel for their specification, and their relationship with the SAMM are presented.

**Chapter 4:** Chapter 4 introduces the MDD framework adopted in the Q-ImPrESS project which supports the prediction of non-functional properties of high level software descriptions specified according to the Service Architecture Meta-Model.

**Chapter 5:** Chapter 5 provides a description of the models supporting the estimation of performance and reliability quality metrics. An example of use of these models according to the client server example introduced in Q-ImPrESS Deliverable D2.1, their solution techniques, their mapping to the Q-ImPrESS MDD, and their points of strength and weakness are discussed.

**Chapter 6:** Chapter 6 introduces models supporting the estimation of software maintainability and defines the maintainability analysis process.

**Chapter 7:** Chapter 7 concludes the presentation of the Q-ImPrESS models and provides an outlook to future work and extensions.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

## 2 Quality Metric Definition

The Q-ImPRESS project tackles several quality characteristics which can be classified in three broad families: *performance*, *dependability*, and *maintainability*. The classification of the quality attributes follows the general approach from [3]. Performance encompasses multiple quality attributes, with each quality attribute quantified by multiple metrics. As concern dependability, we focus on reliability and availability measures according to the Q-ImPRESS DoW and requirements definitions in Q-ImPRESS Deliverable D1.1. Maintainability is an internal software quality aspect which is not directly visible to the application user, but has a significant influence on software development costs. Maintainability quality models and metrics considered in Q-ImPRESS represent the relationship between architectural alternatives and their impact on maintenance efforts.

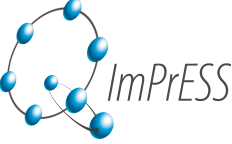
### 2.1 Performance

Performance is a fundamental concept that groups together the quality attributes related to the temporal behaviour of a system. The quality attributes of performance are: *responsivity*, *capacity* and *scalability* [4, 5, 3]. The relation among performance metrics and energy consumptions of systems will be optionally further investigated in Q-ImPRESS.

*Responsivity* is an attribute that describes temporal behaviour of a service from the point of view of a single client request. It can be quantified by the following metrics:

- *Response Time*: The metric gives the time elapsed from the start of a specific service request to the end of a service response [4, 5]. Additional metrics can be defined by considering the time elapsed between other points along the execution path, such as the start and the end of user interaction or the start and the end of request processing.
- *Mean Response Time*: A summary statistic providing the average value of multiple response time observations [5].
- *Worst Case Response Time*: A summary value providing the maximum observed response time of all client requests [5].
- *Jitter*: A summary statistics providing the variation in response time. Variation of other metrics can also be considered.
- *Response Time Percentile Distribution*: given  $\alpha \in [0, 100]$  the percentile distribution of the response time  $R_\alpha$  is defined as the value of response time such that the probability of getting a response time value  $R$  less than or equal to  $R_\alpha$  is  $\alpha/100$ . In other words,  $P(R \leq R_\alpha) = \alpha/100$ .

*Capacity* is an attribute that describes temporal behaviour of a service from the point of view of the overall architecture. It can be quantified by the following metrics:

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

- *Throughput*: The metric gives the number of service requests handled per unit of time [4, 5].
- *Utilization*: The metric gives the ratio of busy to total time for a computational resource [4, 5].

*Scalability* is an attribute that describes changes in temporal behaviour of a service depending on the workload. It can be quantified by the changes in the metrics for responsiveness and capacity, related to the changes in the metrics for the scale of the workload [6].

*Energy consumption* is another important attribute that is part of ongoing research and that could be analyzed in Q-ImPrESS. In recent years, the energy consumption associated with IT infrastructures has been steadily increasing. The reduction of energy usage is one of the primary goals of green computing, a new discipline and practice of using computing resources with a focus on the impact of IT on the environment. IT analysts predict that by 2012 up to 40% of an enterprise's technology budget will be consumed by energy costs [7]. The following metrics are considered [8]:

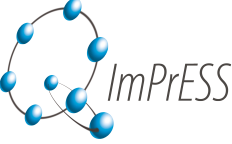
- *Energy Efficiency*: The metric provides the amount of energy consumed per service request on a given platform and for a given performance level.
- *Power Efficiency*: The metric provides the amount of energy consumed per unit of time on a given platform and for a given performance level.

An interesting point of investigation within Q-ImPrESS could be how energy and power efficiency are related to performance and dependability metrics and how they vary under different workload conditions. Note that, the energy efficiency metric focuses on the efficiency of the infrastructure to serve a single requests and could be a benchmark driver to evaluate the software system. Power efficiency, vice versa, is more related to the ability of the underlying infrastructure to serve a set of requests.

## 2.2 Dependability

Several definitions of dependable systems and dependability metrics have been provided in the literature. Dependability is a fundamental concept, defined in [3] as a justified ability of a system to deliver service. According to the Q-ImPrESS DoW, among the quality attributes of dependability, *availability* and *reliability* are considered:

- *Availability*: a measure of the delivery of correct service with respect to the alternation of correct and incorrect service;
- *Reliability*: a measure of the continuous delivery of correct service, or, equivalently, of the time to failure.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

Reliability is a specific aspect of the broader concept of dependability [9], which refers to the continuity of the service delivered by a system.

In this respect, two basic definitions of reliability can be found in the literature: (i) the probability that the system performs its required functions under stated conditions for a specified period of time [10]; (ii) the probability that the system successfully completes its task when it is invoked (also known as “reliability on demand” [11]).

The definition in [10] refers in particular to “never ending” systems that must operate correctly over all the duration time of a given mission (e.g., the on-board flight control system of an airplane, that should not fail over all the duration of a flight). The definition in [11] refers to systems offering services that, once invoked, must be successfully completed, and hence seems more suitable for service-oriented systems. Both definitions can be applied to systems at whatever level of granularity (e.g., a distributed software system, a software component, a software service, etc.), whose correctness can be unambiguously specified. A correct behaviour is intended here as a “failure-free” one, where the system produces the expected output for each input following the system specifications [9].

Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR), as discussed in the Q-ImPrESS Deliverable D1.1 (requirements document), are important measures as they provide the basis for the estimation of the system availability (expressed as the fraction of time the system is operational). MTTF is a reliability measure which appears more suitable for mission-oriented systems, where it may be difficult to identify a single service which should be delivered, but rather a set of tasks can be identified which should be collectively carried out without interruptions for a given mission time.

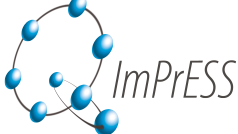
For service-oriented systems, where instead the service to be delivered can be clearly identified, the failure probability on demand seems a more suitable measure.

## 2.3 Maintainability

In this section, considered maintainability definitions and metrics are presented. In contrast to performance and reliability, maintainability is a quality aspect which can be categorized as internal software quality aspect. That means that its implications are not directly visible to software users. But from a software developer’s point of view maintainability is an important factor, because it has significant influence on the development costs.

According to [12], *maintainability* is defined as “*The capability of a software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications*”. This *capability of being modified* indirectly results in a maintenance effort which is caused when implementing changes. In Q-ImPrESS we focus on service architecture-centric maintainability analysis. Hence we concentrate on efforts triggered when implementing architectural changes.

Overall maintainability characteristics have a rather qualitative and subjective nature. In literature maintainability is usually split up into several sub-characteristics. For example in [12] the quality model divides maintainability into analysability, stability, changeability, testability and

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

maintainability compliance. Unfortunately the given definitions of these terms are rather abstract and therefore not directly applicable.

In Q-ImPrESS we use a systematic way to derive maintainability characteristics and metrics. In order to get a quality model with adequate metrics which provide consequently for specific analysis goals the Goal-Question-Metrics method (GQM) [13] is applied. In this approach, in the first step, a set of analysis goals is specified by characteristics like analysis purpose, issue, object, and viewpoint as explained here:

<b>Goal:</b>	
<i>Purpose:</i>	What should be achieved by the measurement?
<i>Issue:</i>	Which characteristics should be measured?
<i>Object:</i>	Which artefact will be assessed?
<i>Viewpoint:</i>	From which perspective is the goal defined? (e.g., the end user or the development team)

The next step is to define questions that will, when answered, provide information that will help to find a solution to the goal. To answer these questions quantitatively every question is associated with a set of metrics. It has to be considered that not only objective metrics can be collected here. Also metrics that are subjective to the viewpoint of the goal can be listed here.

In Q-ImPrESS the goal of the maintainability analysis is to analyse the impact of architectural alternatives on the maintainability. In other words we want to investigate how the maintenance effort for applying several change scenarios is influenced by a certain architectural alternative. For example, a software architect wants to know for a given software system if a pattern usage (e.g., pipe-and-filter pattern) has a positive or negative influence on the maintenance effort with respect to a change scenario (e.g., adaptation of the software system to work on another operating system).

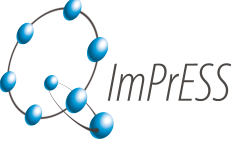
Regarding the GQM method we specify the following goal for maintainability analysis:

<b>Goal:</b>	
<i>Purpose:</i>	Comparison of Architectural Alternative $AA_i$ and $AA_j$
<i>Issue:</i>	Maintainability
<i>Object:</i>	Service and Software Architecture with respect to a specific change request $CR_k$
<i>Viewpoint:</i>	Software Architect, Software Developer

The following questions are defined according to the maintainability definitions above.

<i>Question 1:</i>	How much is the <b>maintenance effort</b> caused by the architectural alternative $AA_i$ for implementing the change request $CR_k$ ?
--------------------	---------------------------------------------------------------------------------------------------------------------------------------

This question is further refined into subquestions:

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

<i>Question 1.1:</i>	How much is the <b>maintenance workload</b> for implementing the change request $CR_k$ ?
<i>Question 1.2:</i>	How much <b>maintenance time</b> is spent for implementing the change request $CR_k$ ?
<i>Question 1.3:</i>	How much are the <b>maintenance costs</b> for implementing the change request $CR_k$ ?

We also specify a second question:

<i>Question 2:</i>	What are the <b>maintainability benefits</b> caused by the architectural alternative $AA_i$ for implementing the change request $CR_k$ ?
--------------------	------------------------------------------------------------------------------------------------------------------------------------------

While Question 1 covers the influences on the maintenance effort Question 2 has its focus on the benefits of an architectural alternative with respect to the change request. For example if an architectural alternative inserts variation points that simplify the implementation of a change request, it signifies a positive contribution to the maintainability.

Based on the questions we identify several metrics categories. Maintenance Effort Metrics cover the maintenance effort for implementing a change request  $CR_k$ . According to the GQM plan above Maintenance Effort Metrics are divided into Maintenance Workload Metrics, Maintenance Time Metrics, and Maintenance Cost Metrics. The following paragraphs describe these categories and metrics in detail. Maintenance Benefit Metrics are also discussed below.

**Maintenance Workload Metrics** Metrics in this subcategory represent the amount of work<sup>1</sup> associated with a change. To be more specific we consider several possible *work activities* and then derive count and complexity metrics according to these work activities. A work activity is composed of basic activity which is applied to an artefact of the architecture. Basic activities are *Add*, *Change*, and *Remove*. The considered architecture elements are *Component*, *Interface*, *Operation*, and *Datatype*. This list is not comprehensive, but can be extended with further architecture elements which can be found in Service Architecture Meta-Model (see also Q-ImPRESS Deliverable D2.1). Usually when we describe changes we refer to the *Implementation* of elements. In the case of *Interface* and *Operation* it is useful to distinguish *Definition* and *Implementation*, since there is usually only one definition but several implementations which cause individual work activities. Service Architecture Meta-Model uses the concept *Interface Port* to bind an interface to a component. From the perspective of work activities an *Implementation of Interface* is equal to an *Interface Port*. In Figure 2.1 there is an outline of resulting work activity types.

Using work activity types we can determine the following generic workload metrics:

- *Number of work activities of type  $WAT_i$* : This metric counts the number of work activities of type  $WAT_i$ .

<sup>1</sup>Note, the term *workload* is also used in context of performance with different semantics. In all parts of this document dealing with maintainability, the meaning of maintenance workload is implied, unless otherwise specified.

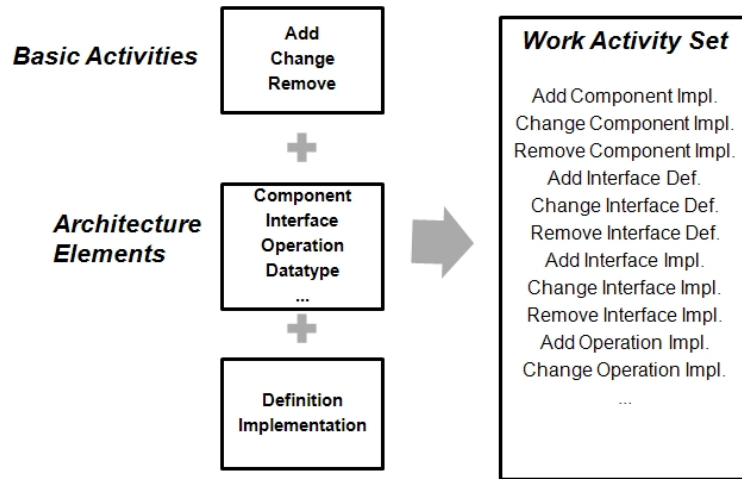


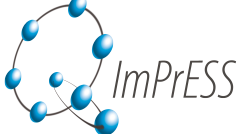
Figure 2.1: Work Activity Types

- *Complexity annotations of work activity  $AC_i$* : At this point several types of complexity annotations can be used. This can comprise architecture properties (e. g. existing architecture styles or patterns with impact on flexibility), design / code properties (e. g. how many files or classes are affected), team organization properties (e. g. how many teams and developers are involved), development properties (e. g. how many test cases are affected), or system management properties (e. g. how many redeployments of services are caused).

When evaluating the maintenance effort for an work activity a software architect determines the appropriate type of basic activity and architecture artefact and uses complexity annotations to quantify workload. Due to their diversity complexity annotations are not aggregated to a single value, but calculated and listed for each work activity. Estimations of maintenance workload metrics are an important foundation for bottom-up effort estimation of maintenance time metrics.

**Maintenance Time Metrics** These metrics describe effort spent in terms of design and development time. The following metrics are proposed in this category:

- *Working time for activity  $AC_i$* : This metric covers the total time in spent for working on activity  $AC_i$  (Unit: *PersonDays, PersonMonths*).
- *Time until completion for  $AC_i$* : This metric describes the time between start and end of activity  $AC_i$  (Unit: *Days, Months*).
- *Total working time for work plan  $WP_i$* : This metric describes the total working time spent for work plan  $WP_i$  (Unit: *PersonDays, PersonMonths*).
- *Total duration for work plan  $WP_i$* : This metric describes the time between start and end of work plan  $WP_i$  (Unit: *Days, Months*).

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

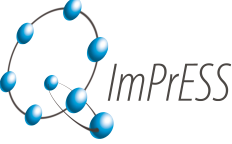
Software architects determine workload metric values first, then derive time estimates for activities. It is possible to aggregate time metrics for multiple change activities. Time metric values are highly dependent on maintenance workload, but also on other influence factors, e. g. team organization, development environment, project management decisions. Therefore a comprehensive knowledge of additional influence factors increases the precision and traceability of maintenance time estimations.

**Maintenance Cost Metrics** This subcategory represents maintenance efforts in terms of spent money.

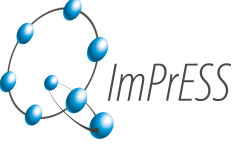
- *Development costs for activity  $AC_i$* : The money paid for development in work activity  $AC_i$ .
- *Development costs for work plan  $WP_i$* : The money paid for development of work plan  $WP_i$ .

**Maintainability Benefit Metrics** Whereas the previous categories capture the maintenance effort, this category sets its focus on the benefits resulting from maintenance activities. If some activity adds a new functionality to a system usually a benefit for the application user is visible. However, in case of refactorings or other preventive maintenance activities (which only occur within the system without behavioural changes to the outside), the achieved value is not directly visible. Metrics in this category approximate these values explicitly and are the following:

- *General maintainability contribution indicator*: This metric measures whether an activity has a negative or positive influence on the maintainability. It has three possible values: *positive, negative, no* (influence). This is an abstract metric with indicator purpose.
- *Maintainability contribution indicator for change request  $CR_i$* : This metric measures whether an activity has a negative or positive influence on the maintainability according to a specific change request. It has three possible values: *positive, negative, no* (influence). This is an abstract metric with indicator purpose.
- *Number of removed antipatterns*: This metric measures the number of antipatterns removed by a work activity.
- *Number of introduced supports for work activity type  $WAT_i$* : This metric measures the number of degrees of freedom (according to a certain work activity type) introduced. The kind of the degree of freedom is captured, as well as the work activity type which is eased by it. For example, if a *visitor pattern* for Data Type  $D_d$  is introduced this represents a support for adding new operations on Data Type  $D_d$ .
- *Time savings according to change request  $CR_i$  comparing two architectural alternatives  $AA_i$  and  $AA_j$* : The development time difference when applying change request  $CR_i$  to architectures with architectural alternative  $AA_i$  instead of  $AA_j$ .

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

- *Cost savings according to change request  $CR_i$  comparing two architectural alternatives  $AA_i$  and  $AA_j$ :* The money saved when applying change request  $CR_i$  to architecture with architectural alternative  $AA_i$  instead of  $AA_j$ .

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

## 3 SAMM Quality Annotations

The Service Architecture Meta-Model has to deal with so called quality annotations in order to specify quality properties of the modelled software system. For example, time demand for a computation step for sorting an array is (as a first approximation)  $n\log(n)$ ; this information has to be attached to the model to be able to reason about the overall complexity of a computation (i.e., a service invocation).

The remainder of this chapter is structured as follows. Section 3.1 gives requirements for the quality annotations. Based on these requirements, Section 3.2 provides a meta-model to specify quality annotations. Finally, Section 3.3 highlights how these annotations can be linked with elements of the Service Architecture Meta-Model.

### 3.1 Quality Annotation Requirements

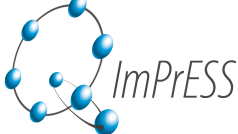
Quality annotations are needed on various elements of the Service Architecture Meta-Model to further detail on quality relevant aspects of the respective elements. The annotations depend on the quality attributes under consideration. For performance, typical annotations are the time or resource demands of particular computations. For reliability, a typical annotation is the failure rate of actions which represent computations or of hardware nodes and links. For maintainability, annotations may specify efforts needed to implement changes.

Besides these quality specific annotations, there are annotations which can be used in the analysis of different quality attributes. For example, the number of loop iterations are needed in performance and reliability predictions. Data flow annotations are another type of annotation. They contain metadata on input and output parameters of service calls, e.g., the number of elements in a collection data type.

To allow different levels of complexity within annotations (according to the needs of analysis and simulation approaches), three forms of *quantity* annotations are available:

1. Constant values (“*const*”, e.g., integer, floating point, boolean)
2. Formulas (“*formula*”, e.g.,  $n\log(n)$ )
3. Stochastic expressions (“*StoEx*“, see [14, pp. 86] for a specification; they include general distribution functions, stochastic distributions, parametric dependencies (e.g., to express a loop that is executed twice per element of an input array))

These three levels  $const \subseteq formula \subseteq StoEx$  allow automatically deriving higher levels from lower ones. For known bindings, deployment, and usage of a system lower levels can also be derived from upper ones. Using this design, analysis and simulation approaches can access annotations in a uniform way, independent of the individual capabilities of the approach. These annotation types form a hierarchy going from simple but rough estimated annotations (constants) to

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

complex but usually more relasitic annotations (stochastic expressions). Hence, these three levels  $const \subseteq formula \subseteq StoEx$  allow automatically deriving higher levels from lower ones. For known bindings, deployment, and usage of a system also lower levels can be derived from upper ones by abstractions. Using this design, analysis and simulation approaches can access annotations in an uniform way, independent of the individual capabilities of the approach.

It is a requirement for the Q-ImPrESS project to deal with hardware as an explicit parameter. This allows estimating the effects of changing hardware platforms on properties like performance and reliability. Thus, the annotation model should be able to include explicit dependencies on hardware. For example, computation durations and reliability metrics in a software model can include abstract CPU demands which changes depending on the deployment and result in faster/slower computation or more/less reliable software without changing the software model (Service Architecture Meta-Model instance).

## 3.2 Quality Annotation Metamodel

The annotation meta-model is designed as a decorator model of the Service Architecture Meta-Model and Generalized Abstract Syntax Tree (G-AST) models. Internally, it is split into multiple sub-models (see Figure 3.1):

- Quantities (see previous section)
  - Constants (*const*)
  - Formulas (*formula*)
  - Stochastic expressions (*StoEx*)
- Annotated elements
  - Control flow
  - Data flow
  - Usage model
  - Black boxes
- Quality-specific models / quality dimensions
  - Performance
  - Reliability
  - Maintainability

To ease the creation of model transformations, the annotation model does not support unstructured string annotation fields or key value pairs which can be attached to arbitrary model elements of the Service Architecture Meta-Model. Instead, fixed links between annotations and model elements, having strong semantics, are encouraged.

Annotations are themselves divided into *measured* values (e.g., timing, number of loop iterations at runtime), *calculated/estimated* values (e.g., number of loops calculated from a completely specified model), or *requirements* specifications (e.g., a response time of less than 1 seconds on average specified at design time). For all three types of values, uniform data structures are used, which are accessed again in a uniform way.

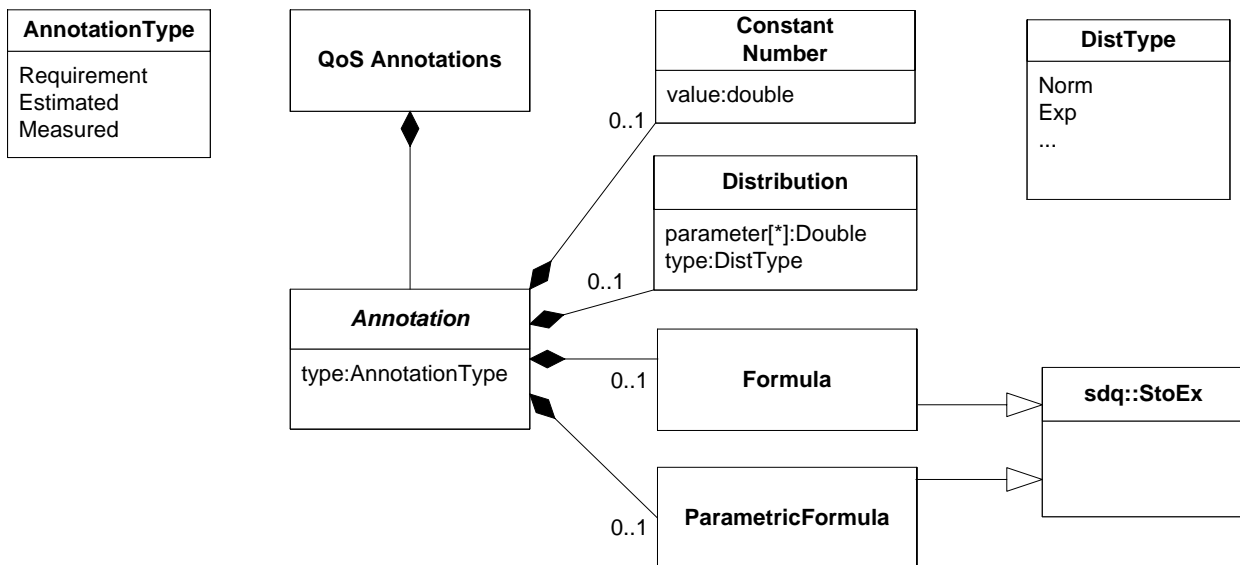


Figure 3.1: Proposal for the Annotations Meta-Model

### 3.3 Quality Annotation Integration in the Service Architecture Meta-Model

Figure 3.2 gives a short summary on how quality annotations and the Service Architecture Meta-Model are related to each other. The quality annotations meta model is a decorator for the Service Architecture Meta-Model, where specific classes from the annotations meta model are decorating specific classes of the Service Architecture Meta-Model. It is intended to not have generic (possibly abstract) superclasses of the annotations meta model annotating generic superclasses of the Service Architecture Meta-Model.

#### 3.3.1 Control flow

All control flow statements can be controlled and parametrised with the annotations model. Namely loops and branches can be assigned with quantities (e.g., probability of executing a loop four times or taking a branch depending on an input parameter of a method).

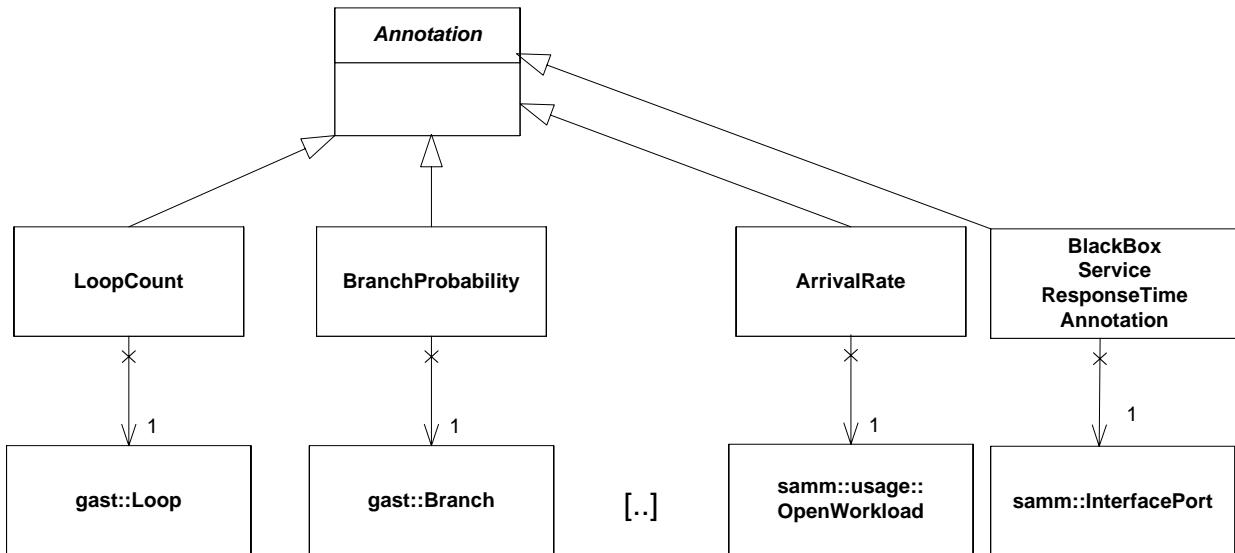


Figure 3.2: Annotations Meta Model as a Decorator for the SAMM

### 3.3.2 Data flow

Data flow can be controlled and parametrised. Input data which can be evaluated within a behavioural specification are method parameters of provided services and return values of called methods. Output data are method arguments of method calls and return values of a behaviour specification. Hence, dependencies between particular parameters can be modelled; for instance, *a parameter of a provided service x is used as an argument to a require service y*.

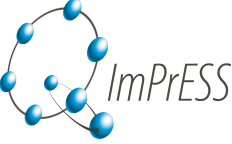
Data flow can also be used to parametrise control flow (e.g., a loop is executed for every byte of input data). To allow compositional reasoning, calls to external services can include data flow annotations which allow to specify quality relevant information of external call arguments. For example, consider a sort service having an array to sort as input parameter. One can specify the size of the array as annotation as the performance of the sort service depends on this information.

### 3.3.3 Usage model

To detail user interaction with a modelled system, or the interaction of other systems with a modelled system, frequency of user arrival and data passed to the modelled system can be annotated.

### 3.3.4 Black box

For approaches that do not support component-internals, black boxes are supported. Black boxes can, for example, carry timing or reliability values. Those values are consequently not parametrised over external service bindings, or execution environment. However, it is always possible to use different annotations to characterise the black box' behaviour in different environments.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

### 3.3.5 Performance

For performance specifications, the G-AST model is enriched with performance data. Therefore, block statements care resource consumptions, which can either be timing values or abstract resource demands. Timing values express, for example, processing time of a certain section of code and are specific for a given hardware platform. Abstract resource demands refer to standard hardware platforms (e.g., a default reference CPU) and can thus also be mapped to other hardware platforms (e.g., faster CPU).

### 3.3.6 Reliability

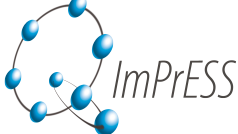
To express reliability, block statements in the G-AST model are equipped with a failure rate. The rate itself follows the quantities specification of the annotation model.

### 3.3.7 Maintainability

The Maintainability Analysis Process uses different kinds of information for various purposes which can be annotated to elements of a SAM model and a G-AST model.

#### Annotations for Derivation of Work Complexity

- Architecture Flexibility Annotations:
  - *Pattern and style roles of Architecture Artefacts*: Information about architecture styles and patterns, e. g. layering, MVC, facade, etc.
  - *Used component frameworks of Architecture Artefact*: e. g. Java Beans for components  $C_j, C_j$
- Code and Design Annotations:
  - *Mapping from Architecture Model to G-AST model*: G-AST model contains lot of information about code/design properties. This annotation can be used to define complexity metrics like number of files, number of classes, LOC etc.
  - *Programming language for Architecture Artefact*: e. g. C/C++ for component  $C_i$
- Team Organization Annotations:
  - *Team responsible for Architecture Artefact*: This information is used to calculate affected teams for an architecture change.
  - *Developer responsible for Architecture Artefact*: This information is used to calculate affected developers for an architecture change.
- Development Environment Annotations:

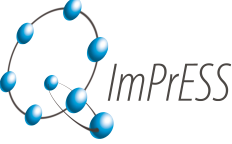
	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

- *Test cases for Architecture Artefact:* This information is used to calculate potential test-effort related with an architecture change.
- *Development tools for Architecture Artefact:* This information is used to compare tool-support for an architecture change.
- System Management Annotations:
  - *Number of Services for Component:* This information is used to calculate number of potential re-deployments in case of a component update.

**Annotations for Change Request to Architecture Mapping** In order to determine the work plan for a given change request and a given architecture the architecture parts which are affected by the change request can be annotated with mapping information to the change request. The mapping information can be attached to elements of a SAM model or a G-AST model.

### 3.3.8 Remarks

- Units for annotations (such as *sec*) are not aspired. As strong semantics for units are required, they are left as subject to future research. Later implementations could have support for units.
- In later stages, the annotations model might be extended with specification of component internal state and, consequently, with a model for component internal state transitions.
- The annotation of energy consumption is currently not covered by the Service Architecture Meta-Model as it is ongoing research and part of optional extensions of the project. We omitted annotations for it and left it open to future extensions.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

## 4 Q-ImPRESS Model Driven Development Framework for Quality Analysis

The aim of the Q-ImPRESS Model Driven Development (MDD) framework is to support early analyses of non-functional (NF) properties on high level software descriptions. In this way, the software engineer can evaluate the impact of the different design choices or candidate system architectures, before they are reified into runnable code.

Furthermore, the Q-ImPRESS MDD framework has to provide means to specify models of systems undergoing evolutions such as, for example, a change of an existing service, the introduction of new services or a different allocation of service components onto physical nodes. Finally, the Q-ImPRESS framework has to be able to support trade-off analyses which enable: (i) the comparison of design alternatives with regard to their impact on the service quality, and (ii) the ranking of multiple design alternatives according to different preferences on QoS metrics.

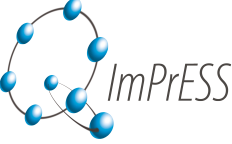
MDD has been actively investigated in the last years, and some taxonomies of model transformation approaches have been defined to help a software developer choosing the method that is best suited for his or her needs [1, 15, 16, 17].

The classical MDD framework includes three main abstraction layers [2], shown on the left-hand side of Figure 4.1:

- the Computational Independent Model (CIM), whose goal is to describe the main functionalities to be implemented by the application (e.g., in the UML 2.0 framework, a use case diagram is an example of a CIM);
- the Platform Independent Model (PIM), which describes in more detail the logic flow of the system and, possibly, the user interactions, in order to achieve the functionalities (e.g., a sequence or an activity diagram of UML 2.0);
- the Platform Specific Model (PSM), which describes how application components are mapped onto the system physical resources (e.g., a UML 2.0 deployment diagram).

In the Q-ImPRESS project, CIMs are adopted in order to: (i) support the specification of QoS goals for a given architecture, (ii) define evolution scenarios, and (iii) introduce the preferences among multiple QoS attributes for a trade-off analysis. PIMs are defined by the SAMM behavioural package, while PSMs are specified through the SAMM deployment package (see D2.1).

To reason about non-functional quality attributes of a software system, it is necessary to transform the aforementioned “design-oriented” models of the software system into “analysis-oriented” models that support the desired analysis methods [18]. In the specific case of performance and reliability attributes, the analysis-oriented models provide a probabilistic description of the system that evolves in time and space. Possible examples are queueing networks or different kinds of Markovian models. Corresponding to the three MDD layers listed above (CIM, PIM, and PSM) the following classes of analysis-oriented models have been identified [19]:

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

- CINFM (Computation Independent Non-Functional Model) represents the requirements and constraints related to a NF aspect (i.e., performance and reliability). An example of a CINFM may be a Use Case Diagram augmented with reliability requirements.
- PINFM (Platform Independent Non-Functional Model) represents the logic of the system along with estimates of NF characteristics, such as the amount of resources that the logic needs to be executed [19]. An example of a PINFM can be a Markov Model derived from the corresponding PIM annotated with non-functional aspects.
- PSNFM (Platform Specific Non-Functional Model) contains variables and parameters that represent the software structure and dynamics, as well as the platform and the environment in which the software will be deployed [19]. A characterization of the platform must include data on the underlying hardware architecture, such as the CPU speed or the failure probability of a network connection. An example of PSNFM is provided by a queueing network model derived from the corresponding PSM and including performance parameters.

The idea of non-functional analysis-oriented models associated with the corresponding design models proposed by [19] starts from the traditional view of the MDD approach [2] and embeds quantitative models to evaluate performance and reliability of the final application to be deployed. In the Q-ImPRESS project we propose a conceptual extension of this framework which is summarized in Figure 4.1. In particular, models are introduced also at run time in order to validate the quantitative analysis performed at design time. At run time, a monitor collects the results of execution of the software system in the target environment. Then, a runtime model is defined which is able to represent the behaviour of the system with high accuracy and at very fine grained time scales. In this way, the results obtained by the design-time models can be validated under multiple working scenarios.

The transformations among models described in Figure 4.1 can be classified as either *horizontal* or *vertical transformations*. Horizontal transformations (arrows from left to right) yield a target model at the same level of abstraction as the source model. The transformation adds annotations about non-functional requirements to support reasoning about performance and reliability. A detailed overview of transformation approaches is given in [16, 15, 18].

The focus of horizontal transformation is on specifying the information that is lacking in the software architecture description but is crucial for the quantitative non-functional analysis (examples of this information can be: number of invocations of a component within a certain scenario, probability of executions, etc.) [20, 21]. In particular, in the Q-ImPRESS project horizontal transformations are obtained by the SAMM *Annotation* and *Usage* packages as follows: (i) CIM to CINFM is obtained by specifying the non-functional requirements, (ii) PIM to PINFM is performed by annotating the behavioural models specifications introducing, for example, the branch probabilities of a service, (iii) PSM to PSNFM is obtained by annotating the deployment models specifying the services resource demands and, for example, the number of concurrent users.

Vertical transformations produce a target model at a different abstraction level. Vertical transformation can abstract or refine the source model (down or up arrows in Figure 4.1). A vertical

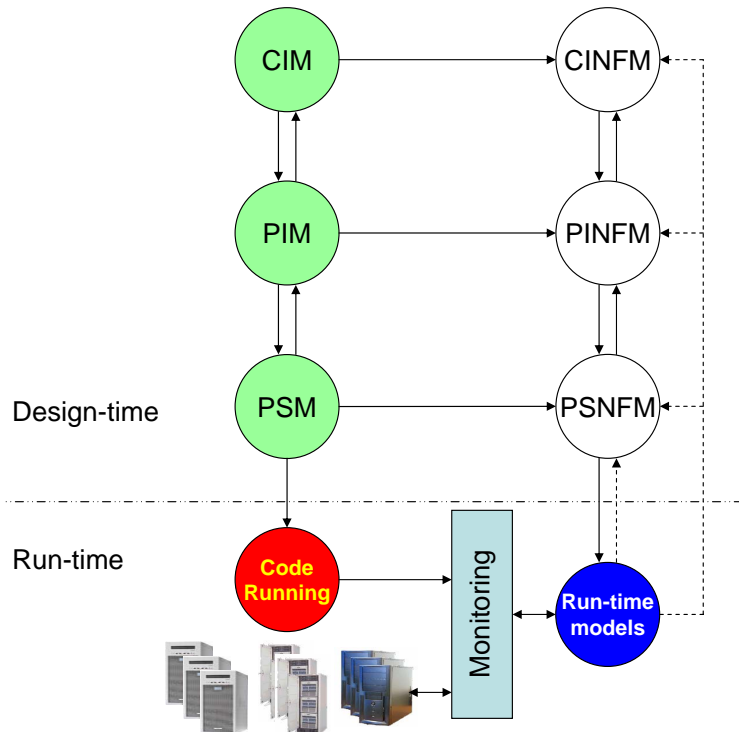
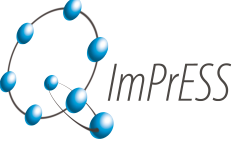


Figure 4.1: Reference Framework.

transformation of a non-functional model is used to refine a higher level model, to improve the accuracy of the performance/reliability metrics.

Both horizontal and vertical transformations are supported in the Q-ImPrESS project by the tools suite overviewed in the Q-ImPrESS *Envision* described in D2.1.

In the following, the models adopted by Q-ImPrESS to support analysis at the CINFM, PINFM, and PSNFM layers are described. Vertical and horizontal transformations are the focus of the Q-ImPrESS Deliverable D3.2.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

## 5 Performance and Reliability Models

A substantial amount of research has been devoted to devise performance and reliability prediction techniques for software systems (see for example [22, 23, 24]). The conceptual mapping of the models adopted by Q-ImPrESS according to the project requirements (see Deliverable D2.1) is summarized in Table 5.1. First of all, for each model we discuss if it can support performance or reliability analyses. Then, the aim of the prediction is considered, i.e., worst case or average case analysis (see Q-ImPrESS Deliverable D1.1). Furthermore, models will be classified in design-time and run-time. Design-time models are derived starting from an abstract system description according to the SAMM specification. The aim of design-time models is to predict QoS metrics of multiple candidate architectures. Vice versa, the aim of run-time models is the validation and calibration of design-time models parameters. Indeed, the environment abstraction used at design time may prove to be inadequate: it may reflect only partially what happens in reality. For example, certain user behaviours that were assumed as occasional during design may prove to be very common. As another example, the performance or reliability profile of certain resources used by the system in practice may differ from the figures assumed at design time. Because of the inaccuracy of the data, the initial model may be flawed, and the implementation derived from the model needs to be evolved. According to the principles of model-driven development, this in turn requires calibrating the model, and then re-generating the implementation. Run-time models are introduced in Q-ImPrESS for this purpose to validate design time predictions. The adoption of run-time models allows generalizing the results which can be obtained by running a single experiment in prototype environments. Run-time models are based on control theory [25] and are discussed in Section 5.6

From Section 5.1 to 5.6 we will provide a high level view of the main families of models adopted for performance and reliability evaluations. Maintainability models will be presented in Chapter 6.

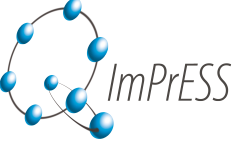
### 5.1 Queueing Network Models

Queueing network (QN) models [26, 27, 27] are a mathematical modelling approach in which a software system is represented as a collection of: (i) *service centers*, which model system resources, and (ii) *customers*, which represent system users or “requests” and move from one service center to another one. The customers’ competition to access resources corresponds to queueing into the service centers.

The simplest queueing network model includes a single service center (see Figure 5.1) which is composed of a single queue and a single server: the queue models a flow of customers or requests which enter the system, wait in the queue if the system is busy serving other requests, obtain the service, and then depart. Note that, at any time instant only one customer or request is obtaining the service. Single service centers can be described by two parameters: the requests *arrival rate*, usually denoted by  $\lambda$ , and the average *service time*  $S$ , i.e., the average time required by the server in order to execute a single request. The maximum service rate is usually indicated with  $\mu$  and is

Model Family	Model	Performance	Reliability	Prediction Aim		Design Time		Run-time Models	
				Average case	Worst case	PINFM	PSNFM	PINFM	PSNFM
Queueing Models	Bound	X			X	X			
	Product	X		X		X			
	Non-product	X		X		X			
	LQN	X		X		X			
Markov Models	DTMC	X	X	X	X				
	CTMC	X	X	X	X				
	MDP	X	X	X	X				
	SMC	X	X	X	X				
Automata Models	TA/PTA/MPTA	X	X			X			
Simulation Approaches	Simulation	X	X	X	X	X			
Control Theory	LTI			X	X				X
	LPV			X	X				X

Table 5.1: Quantitative Models Conceptual Mapping.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

defined as  $\mu = 1/S$ . Given the request arrival rate and the requests service time, QN theory allows evaluating the average value of performance metrics by solving simple equations.

In real systems, requests need to access multiple resources in order to be executed; accordingly, in the model they go through more than a single queue. A QN includes several service centers and can be modeled as a directed graph where each node represents the  $k$ -th service center, while arcs represent transitions of customers/requests from one service center to the next. The set of nodes and arcs determines the network topology.

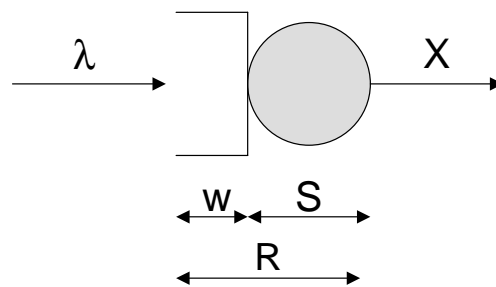


Figure 5.1: Single Service Center Model.  $\lambda$  denotes the incoming workload,  $X$  denotes the requests throughput.  $W$  indicates the requests' waiting time, i.e. the average time spent by requests in the queue.

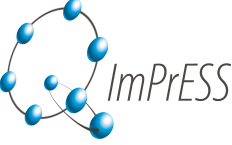
### 5.1.1 Product-form QN

One of the most important results of queueing network theory is the *BCMP theorem* (by Baskett, Chandy, Muntz, and Palacios-Gomez) which, under various assumptions (see [28] for further details), shows that performance of a software system is independent of network topology and requests routing but depends only on the requests arrival rate and on the requests *demanding time*  $D_k$ , i.e., the average overall time required to complete a request at service center  $k$ . The average number of time a request is served at the  $k$ -th service center is defined as the number of visits  $V_k$ , and it holds  $D_k = V_k \cdot S_k$ .

In time sharing operating systems, as an example, the average service time is the operating system time slice, while the demanding time is the overall average CPU time required for a request execution. The number of visits is the average number of accesses to the CPU performed by a single request.

Queueing networks satisfying the BCMP theorem assumptions are an important class of models also known as *separable queueing networks* or *product-form models*<sup>1</sup>. The name “separable” comes from the fact that each service center can be separated from the rest of the network, and its solution can be evaluated in isolation. The solution of the entire network can then be formed by combining these separate results [26, 28, 29, 30]. Such models are the only ones that can be

<sup>1</sup>The name “product-form” comes from the fact that the stationary state distribution of the queueing network can be expressed as the product of the distributions of the single queues and avoids the numerical solution of the underlying Markov chain.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

solved efficiently, while the solution time of the equations governing non-product-form queueing network grows exponentially with the size of the network. Hence, in practical situations the time required for the solution of non-product-form networks becomes prohibitive and approximate solutions have to be adopted.

### 5.1.2 Open and Closed Models

Queueing models can be classified as *open* or *closed* models. In open models customers can arrive and depart and the number of customers in the system cannot be determined a-priori (and, possibly, it can be infinite). The single service center system in Figure 5.1 is an open model. Vice versa, in closed models the number of customers in the system is a constant, i.e., no new customer can enter the system, and no customer can depart (see Figure 5.2). While open models are characterized by the requests arrival rate  $\lambda$ , a closed model can be described by the average number of users in the system  $N$  and by their *think time*  $Z$ , i.e., the average time that each customer “spends thinking” between interactions (e.g., reading a Web page before clicking the next link). Customers in closed queue models are represented as delay centers (circles in Figure 5.2).

Usually Service Oriented Architecture (SOA) based systems are modeled by means of open models [31]. Closed systems can be adopted, for example, to model an Intranet with a fixed number of users or an Internet application when the concurrent number of sessions is kept constant (e.g., as a result of an overload protection mechanism [32]).

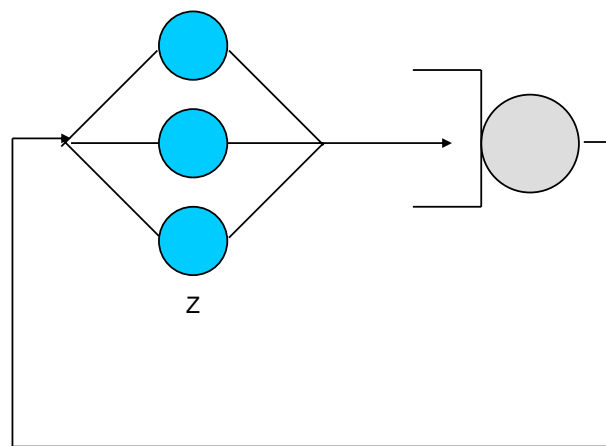
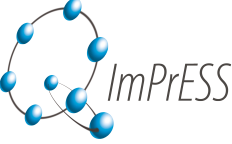


Figure 5.2: Closed Model Example.

### 5.1.3 Single and Multi-Class Models

Finally, queueing models can be classified as *single-class* and *multi-class* models. In single-class models, customers have the same behaviour; vice versa, in multi-class models, customers behave differently and are mapped into multiple-classes. Each class  $c$  can be characterized by different values of demanding time  $D_{c,k}$  at the  $k$ -th service center, different arrival rate  $\lambda_c$  in open models, or

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

number of users  $N_c$  and think time  $Z_c$  in closed models. Customers in each class are statistically indistinguishable. Queueing network theory allows determining performance metrics (i.e., response times, utilizations, etc.) on a per-class basis or on an aggregated basis for the whole system.

### 5.1.4 A running example

In the following, in order to provide a minimal example of performance and reliability models, the client-server architecture presented in the Q-ImPrESS project Deliverable D2.1 is considered. The example includes a distributed systems composed of a client, a server, and a database. The server offers a lot of services that rely on data extracted from the database. The client issues requests for services, the server component is in charge of providing a structured API to access data to the clients, and finally the database is in charge of efficiently fetching data.

The system could be modelled as an open or closed queueing network system. The former case is usually more conservative, since an infinite number of clients can access the system. The latter case is more appropriate if, for example, the SOA system includes also an admission controller which can limit the number of concurrent sessions in order to provide QoS guarantees [33]. As a first approximation, the system can be represented as a multi-class model which includes two classes: the first one represents the *Whois* users behaviour, where the client perform a simple query on the system, while the second one represents the *Whoare* behaviour where the users perform mass queries (see Q-ImPrESS Deliverable D2.1).

Figure 5.3 shows, as an example, two different open models of the system according to two different deployment scenarios. Figure 5.3(a) represents the case where the server and database components are deployed on a single node (see D2.1), while Figure 5.3(b) represents the case where the server component is replicated on three servers which work in parallel and share the load at the first tier, and the database server is deployed on a dedicated node at the second tier. In the queueing networks, each node is modelled as a service center characterized by a demanding time. In the first scenario, the demanding times  $D_1$  and  $D_2$  include the time required both by the server and database software components to serve the *Whois* and *Whoare* requests classes. In the second scenario, each class is characterized by multiple demanding times on the multiple nodes. If we assume that the physical servers at the first tier are homogeneous and the workload is evenly shared, then the demanding time of each request class at the service centers which model the first tier are equal to  $D_1^{server}$  and  $D_2^{server}$  respectively. The demanding times of the service center which models the database node will be denoted as  $D_1^{database}$  and  $D_2^{database}$ . As a first approximation we have  $D_1 \approx 3D_1^{server} + D_1^{database}$  and  $D_2 \approx 3D_2^{server} + D_2^{database}$ . Actually, if the system is deployed on a single node, the communication between server and database software components could be optimized (e.g., messages are exchanged in RAM instead of network) then the previous expressions are conservative. The demanding times can be obtained by the SAMM quality annotations.

### 5.1.5 Solution Techniques

After modelling a software system as a queueing network, the model has to be evaluated in order to determine quantitatively the performance metrics.

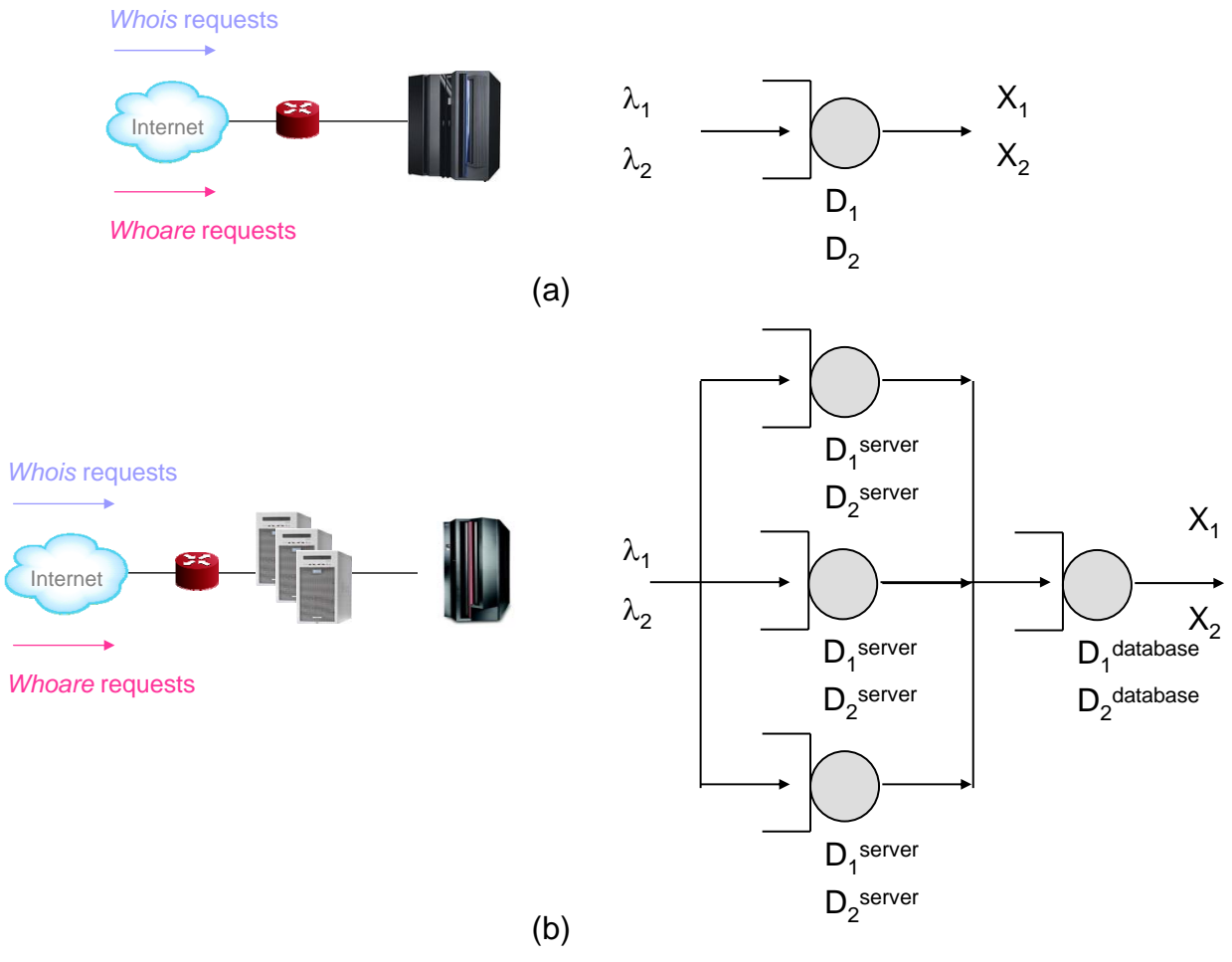
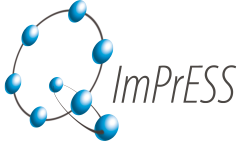


Figure 5.3: Client-server example QN modelling.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

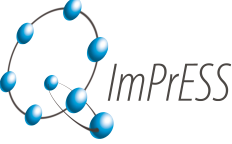
A first step in the evaluation can be achieved by determining the system bounds; specifically, upper and lower bounds on system throughput and response time can be computed as functions of the system workload intensity (number or arrival rate of customers). Bounds usually require a very little computational effort, especially for the single class case [34, 35].

More accurate results can be achieved by solving the equations which govern the QN behaviour. Solution techniques can be classified as *analytical methods* (which can be *exact* or *approximate*) and *simulation methods* (see Section 5.5). Exact analytical methods can determine functional relations between model parameters (i.e., request arrival rate  $\lambda_c$ , number of customers  $N_c$  and think time  $Z_c$ , and requests demanding times  $D_{c,k}$ ) and performance metrics. The analytical solution of open models system is very simple even for multiple class models and yields the average value of the performance metrics. The exact solution of single class closed models is known as the *Mean Value Analysis* (MVA) algorithm and has a linear time complexity with the number of customers and the number of service centers of the models. The MVA algorithm has been extended also to multiple classes, but the time complexity is non-polynomial with the number of customers or with the number of service centers and classes [26, 36]. Hence, large closed models are solved by recurring to approximate solutions, which are mainly iterative methods and can determine approximate results in a reasonable time. Approximate MVA algorithms for multi-class closed models provide results typically within a few percent of the exact analytical solution for throughput and utilization, and within 10% for queue lengths and response time [26].

Analytical solutions can determine the average values of the performance metrics (e.g., average response time, utilization, etc.) or, in some cases, also the percentile distribution of the metric of interest. Determining the percentile distribution of large systems is usually complex even for product-form networks. Indeed, while the mean value of the response time of a request that goes through multiple queues is given by the sum of the average response time obtained locally at the individual queues, the aggregated probability distribution is given by the convolution of the probability distribution of the individual queues. The analytical expression of the percentile distribution becomes complicated for large system (most of the studies provided in the literature are limited to *tandem queues*, i.e., queueing networks including two service centers [37, 27]). For this reason, some approximate formulas have been introduced. Markov's Inequality [38, 39] provides an *upper-bound* on the probability that the response time exceeds the threshold  $R_\alpha$ . This upper-bound depends only on the average response time  $E[R]$ , and can be computed as  $P(R \geq R_\alpha) \leq E[R]/R_\alpha$ . However, Markov's inequality is known for providing somewhat loose upper-bounds. Chebyshev's inequality [39] provides a tighter upper-bound based on estimates of response time variance,  $Var[R]$ , in addition to estimates of the average response time  $E[R]$ . Chebyshev's inequality is given by  $P(R \geq R_\alpha) \leq \frac{Var[R]}{(R_\alpha - E[R])^2}$

### 5.1.6 Strength and Weakness

The main reason in favor of the adoption of the queueing network models family is that their interpretation is very intuitive. Furthermore, at least for open models, the computational effort is limited (i.e., the time complexity is linear in the number of request classes and service centers). Furthermore, although the BCMP theorem assumptions almost never hold in real software systems,

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

experience has shown that the results which can be achieved by queueing network models are extremely robust with respect to violations of these assumptions [26, 22, 37]. Typically the deviations between the measured values in a real system and the ones obtained by the models come from an inaccurate estimate of parameter values for service demands or workload intensities. In terms of accuracy, a large body of experience [26, 22, 37] indicates that queueing network models can be expected to be accurate in the range 5-10% for utilization and throughput estimates and within 10 to 30% for response time. Bounding techniques are usually characterized by a worst case estimate with respect to the exact analytical solution in the range 15-35% for system throughput, while for response time usually the bounds are more inaccurate. Recently in [34], geometric bounds have been proposed, a new family of fast and accurate bounds for closed models where the bounding error for system throughput is in the worst case within 5-13%.

The weak points become evident when the limitations on the structure of the model imposed by the BCMP theorem inhibit the representation of aspects affecting performance (e.g., requests routing, blocking conditions, service time distributions, etc.). In these cases, basic product-form solutions can be extended, for example, by introducing burstiness parameters [6] or iteratively solved in order to provide more accurate results. In other words, separable models are also the basic building blocks which can be adopted for the construction of more detailed models (see Section 5.2).

### 5.1.7 Model Adoption

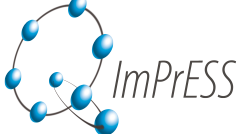
Queueing network models can be used at design time to perform performance analyses of service oriented systems. QN bounds can be used to obtain a raw estimate of performance following the method proposed in [40]. [41, 42] use product-form queueing network for the evaluation of the response time of BPEL business processes modelled by activity diagrams. Queueing models are more frequently adopted at the design time for the capacity planning of the target hardware platform [6].

### 5.1.8 Tools for Model Derivation

In the literature there exists a quite large set of methodologies which propose transformation techniques to derive QN-based models (both product and non-product) starting from software models. Some of the proposed methods are reviewed in [43, 44, 23].

Bounds are automatically derived starting from a description of the system behaviour, and the transformation is implemented with *ad-hoc algorithms* that use imperative programming languages. Always at design time, product and non-product QN can be automatically obtained by using automatic transformations. Examples can be found in [45], and in [46, 47, 48, 49, 50, 51].

As a general consideration, in these approaches the transformations are often implemented with *ad-hoc algorithms* that use imperative programming languages. However in several transformation methodologies and tools, it is possible to devise a common underlying application pattern. In this group of transformations, the source architectural model is represented by a set of UML diagrams. These diagrams are annotated with ad-hoc or standard performance annotations and

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

then exported in the underlying XMI/XML format. The transformation is then defined from the XML document of the source model to an XML model defining the target performance model. The transformation language is often JAVA or a similar imperative language. However, in some cases the transformations are defined using XSLT or graph transformation rules [48].

## 5.2 Layered Queueing Networks

The Layered Queueing Network (LQN) model is a performance model in the class of extended queueing networks [52, 51] designed to model software systems with nested resource requests frequently used in multi-tier architectures. While ordinary queueing networks model software structures only implicitly via resource demands to service centers, LQNs model a system as a layered hierarchy of interacting software entities, which produce demands for the underlying physical resources such as CPUs or hard disks. While server or resource on the lower layer process a request, all of the intermediate servers between this server and the client are blocked. This type of interaction is a form of hierarchical resource usage. Therefore, LQNs reflect the structure of distributed systems more naturally than ordinary queueing networks. In particular, they model the routing of jobs in the network more realistically [53].

A LQN model is an acyclic graph, with nodes representing software entities and hardware devices, and arcs denoting service requests. The software entities are called *tasks*. Each kind of service offered by a LQN task is modeled as a so-called *entry*, which can be further described by two alternative ways, i.e., using either *activities* or *phases*. Activities allow describing more complex behaviours, e.g., with control structures such as forks and joins and loops. Phases are used for notational convenience to specify activities that are sequences of several steps. The advantage of LQN is to introduce also software elements as resources and hence they can capture the contention to access software components.

Processors model time consumption by, e.g., actual hardware processors or hard disks. Each processor has a single queue and requests can be served according to several scheduling disciplines: first-come first-served, processor sharing, priority preemptive resume, random, head-of-line, processor sharing head-of-line, and processor sharing priority preemptive resume. A processor cannot issue requests to other entities, therefore it is only a server and cannot be a client.

Each processor may contain a number of tasks which can represent customers on a network, software servers, hardware devices, logical resources such as buffers and semaphors, or databases. Tasks can call other tasks and be invoked from other tasks. Therefore, tasks were formerly also known as “active server”.

Figure 5.4 illustrates the basic structure of a task. A task consists of a number of entries (the starting point for a service the task provides) and has a single request queue. Usually, a task serves only a single request at a time but is also possible to specify a multiplicity which represents multiple homogeneous copies of the task. Tasks which only make requests to other tasks represent clients and multiplicity specifies the number of clients. For all other tasks, the multiplicity represents the number of identical clients having a single request queue (thread pool). This enables serving multiple requests in parallel.

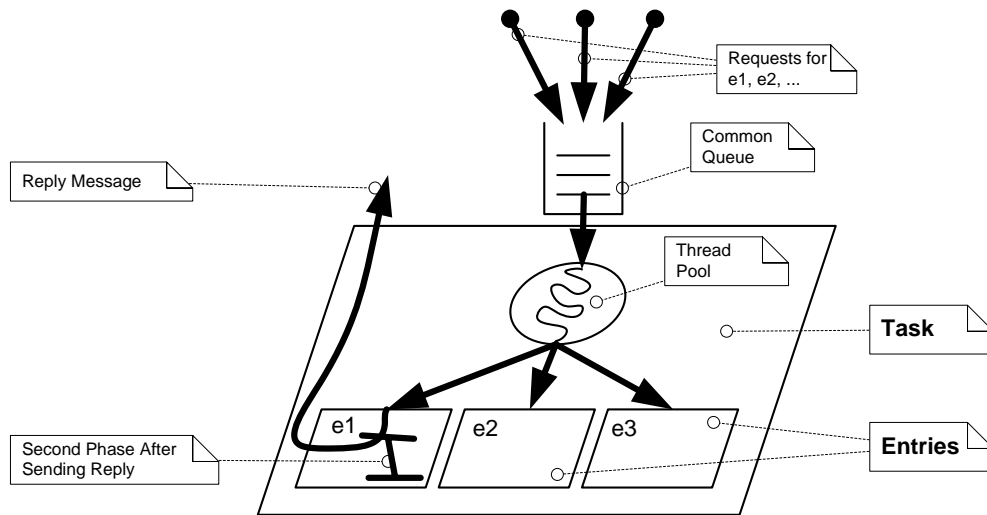


Figure 5.4: LQN-Tasks Illustration [54].

An entry serves as an interface for different classes of services provided by the task. The details of the entry are specified through phases or activities. Phases are intended to model common behaviour in distributed systems, where servers return the control back to the client as soon as possible to increase the responsiveness of the system. For example, a database commit is well modelled with phases, as it returns the control back to the user if the operation is possible, and then performs operations asynchronously from the client in the background [54] (like writing to the disc, cleaning up temporary tables, etc).

Entries accept either only synchronous or asynchronous requests, but not both. In the case of synchronous requests, an entry either generates a reply itself, which it transfers back to the calling client, or forwards the request to another task, which then in turn generates the reply for the calling client. The behaviour of an entry (or task) is specified either via activities. The activity graphs, which are either attached to entries or tasks, model the control flow using sequences, alternatives, loops, and forks.

A sequence of activities represents the execution steps of provided service and consumes time on a processor or calls other tasks. For modelling time consumption, activities either specify a mean service time or the coefficient of variation for the service time, which is the variance of the service time divided by the square of the mean.

Calls can be single or multiple and the call order may either be deterministic or stochastic. A deterministic call order implies issuing the exact number of specified requests to the underlying tasks. A stochastic call order implies issuing a random number of requests to the underlying tasks with the specified mean value. LQNs assume a geometrically distributed number of stochastic requests.

Activities can be connected to each other in sequences, branches, loops, or forks to form a control flow graph. They are split into 'pre'-precedences to join or merge activities and 'post'-precedences to branch or fork activities. Using a 'pre'-precedence, it can be specified that an

activity follows exactly one other activity in a sequence. Using a 'post'-precedence, it can be specified that an activity is followed by exactly one other activity.

### 5.2.1 A running example

Figure 5.5 shows an LQN example instance, based on the Q-ImPRESS running example, using the common concrete graphical syntax for LQNs to illustrate the concepts explained above. The figure shows tasks as large parallelograms and processors as circles. Rectangles within entries represent activities. The replication of tasks or processors is indicated via multiple parallelograms or circles on top of each other.

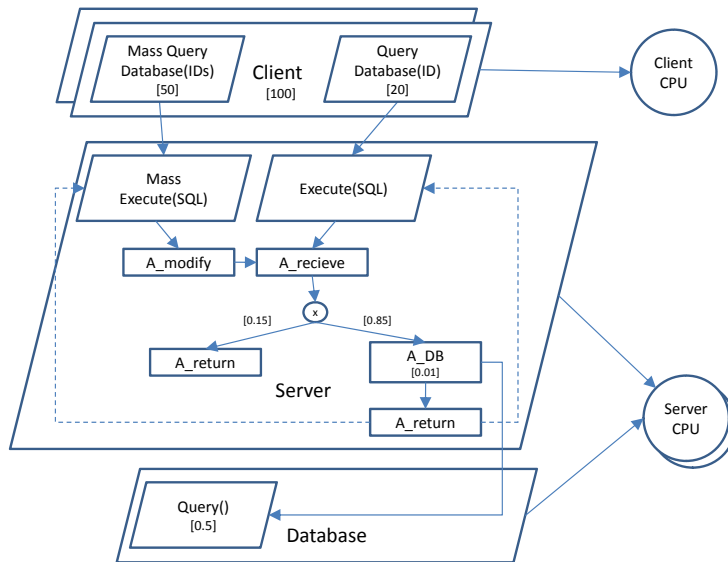
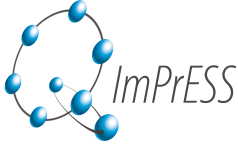


Figure 5.5: Simple LQN Example

The example models the three tier architecture with a client layer, an application server layer, and a database layer. In this example, each layer has only a single task. The client task runs on the client processor and is replicated 100 times, therefore modelling a closed workload with a user population of 100. The think time is 20 seconds for simple query and 50 seconds for mass query given in the entries in square brackets.

The entry Execute (SQL) is specified as an entry of activity graph and includes an activity graph consisting of several activities connected via OR or AND precedences, with probabilities. The activity diagram specifies the return activity of the entry and sends a reply message back to the client. The activities can both consume time on the Server Processor and issue requests to the Database task.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

Each call of the entry of the Database task takes 0.5 seconds. The structure of this LQN is strictly hierarchical, lower layers do not issue requests to upper layers.

## 5.2.2 Solution Techniques

To determine the performance metrics the model has to be solved analytically or by simulation. In many software systems, delays and congestion are related to synchronous interactions such as remote procedure calls or rendezvous. An LQN model captures hierarchical delays via incorporating the lower layer queueing and service time in the service time of upper layer service. This essence of layered queueing is a form of hierarchical resource sharing. The overall LQN model is solved by constructing a set of submodels consisting of a set of clients and a set of servers. These submodels are later solved by the tools discussed in the following section.

## 5.2.3 Strength and Weakness

LQNs model a wide range of applications due to the number of features supported by the language, e.g., open, closed, and mixed queueing networks, FIFO and priority queueing disciplines, synchronous calls, asynchronous forks and joins, the forwarding of requests into another queue, and service with two phases. The adoption of available solvers such as LQNS [55] makes the method relatively easy to use. As a disadvantage could be mentioned the lack of graphical editors to define the input models for current LQN solvers or that they restrict the number of phases to a maximum of 3.

Performance metrics include response times, throughput, and utilization information for each service. The essence and the main advantage of LQN model is a form of simultaneous resource sharing and the capability to describe the contention of software resources. In software systems delays and congestion are heavily influenced by synchronous interactions such as remote procedure calls or rendezvous. The LQN model captures these delays by incorporating the lower layer delays into the service time of the upper layer server. This “active server” feature is the key difference between layered and ordinary queueing networks.

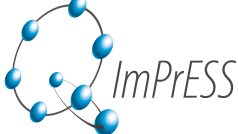
Finally, the accuracy and scalability of LQN models depends on the solution technique used (e.g., MVA and MOL) [55, 52].

## 5.2.4 Model Adoption

Similarly to QN models, LQN models can be used at design time to carry out performance analyses and to generate performance metrics. LQN models are useful for the capacity planning of the target hardware platform for distributed multi-tier architectures [6].

## 5.2.5 Tools for Model Derivation

Multiple solution techniques have been developed to derive performance metrics, such as response times, throughput, and resource utilization. Currently, the LQN tool suite contains two different

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

solvers: the analytical solver LQNS [55, 52] and the simulation tool LQSIM [56]. These solvers are the result of combining some of the formerly developed techniques. The input for the solvers is the LQN model and the output is for example an XML file containing mean service time per task or activity, service time variance per task or activity, throughput per task, and utilization per processor, task, or phase.

The analytic layered queueing network solver LQNS relies on a heuristic based on solving multiple sub-networks using the MVA algorithm. It is a popular and efficient solution algorithm for product-form queueing networks. This is a very restricted class of queueing networks, which for example requires exponentially distributed service times, a fixed user population, and service rates only dependent on the number of customers at a service center. Features such as simultaneous resource possession or fork/join interaction violate these assumptions. SRVN and MOL provide heuristic solutions to analyse queueing networks that include these advanced features by decomposing them into simpler sub-models and then performing approximate instead of exact MVA on them. MOL in particular uses the Linearizer algorithm to estimate the queue lengths in the sub-models.

The discrete-event simulation framework for LQNs (LQSIM) was formerly called ParaSRVN and uses the ParaSol simulation environment [56]. As a simulation approach, it imposes the fewest restrictions on the input models. LQSIM creates simulation objects from tasks, processors, and queues of an LQN using a library provided by ParaSol. During simulation, LQSIM collects statistics on throughput and delays of each thread.

## 5.3 Markov Models

This section illustrates the main Markov models adopted in Software Engineering for prediction of non-functional properties. In this context we consider non-functional properties (performance and reliability). Markov models are stochastic processes defined as state-transition systems augmented with probabilities. Formally, a stochastic process is a collection of random variables  $X(t), t \in T$  all defined on a common sample (probability) space. The  $X(t)$  is the state while (time)  $t$  is the index that is a member of set  $T$  (which can be discrete or continuous). In Markov models [27, 39], states represent possible configurations of the system being modelled. Transitions among states occur at discrete or continuous time-steps and the probability of making transitions is given by probability distributions. The Markov property characterizes these models: it means that, given the present state, future states are independent of the past. In other words, the description of the present state fully captures all the information that could influence the future evolution of the process.

The following sections illustrate three Markov models: Discrete Time Markov Chains (DTMC), Continuous Time Markov Chains (CTMC), and Markov Decision Processes (MDP).

### 5.3.1 Discrete Time Markov Chains

Discrete Time Markov Chains are the simplest Markovian model where transitions between states happen at discrete time steps.

Formally a DTMC is a 4-tuple  $\langle S, \bar{s}, P, L \rangle$  where:

- $S$  is the finite set of states.
- $\bar{s}$  is the initial state.
- $P : S \times S \rightarrow [0, 1]$  is the transition function, where  $P(s, s')$  is the probability of reach  $s'$  from  $s$ .
- $\sum_{s' \in S} P(s, s') = 1$  for all  $s \in S$
- $L \rightarrow 2^{AP}$  is the labeling function, it assigns at each  $s \in S$  the set  $L(s)$  of atomic propositions  $a \in AP$  holding in  $s$ . As discussed in [57],  $AP$  formally is a fixed, finite set of atomic propositions used to label states with the properties of interest which can be verified by a stochastic model checker.

A DTMC evolves starting from the initial state executing a transition at each discrete time instant. Being at time  $i$  in a state  $s$ , at time  $i + 1$  the model will be in  $s'$  with probability  $P(s, s')$ . The transition can take place only if  $P(s, s') > 0$ .

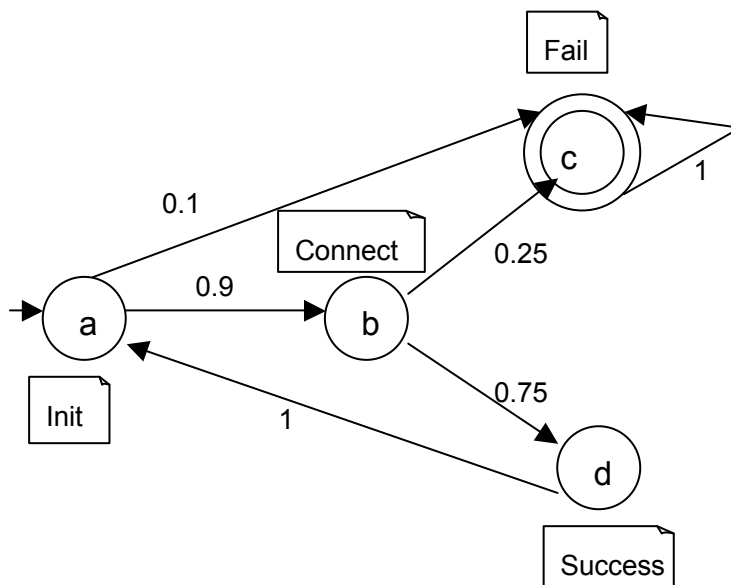
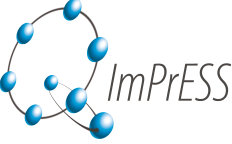


Figure 5.6: An example of DTMC,  $c$  is a final state.

If software engineers adopt DTMCs, they must specify the set of states  $S$  and transitions with an associated probability distribution (probabilities of outgoing transitions in every state must sum up to one). These probability values are numerical parameters and represent, for example, the fraction of times the system fails when it is in a given configuration.

Figure 5.6 shows a simple DTMC where:

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

$S = \{a, b, c, d\}$  and  $\bar{s} = a$

$AP = \{Init, Connect, Fail, Success\}$

$L(a) = Init, L(b) = Connect, L(c) = Fail, L(d) = Success$

### 5.3.2 Continuous Time Markov Chains

Continuous Time Markov Chains are another extension of DTMC. The model is similar to the DTMC one, but the temporal domain is continuous. This means that the time in which a transition occurs is not fixed, but depends on some parameter of the model. The model is specified as a DTMC by means of state and probabilistic transition, but the value associated with each outgoing transition from a state is intended not as a probability but as a parameter of an exponential probability distribution (transition rate).

Formally a CTMC is a 4-tuple  $\langle S, \bar{s}, R, L \rangle$  where:

- $S$  is the finite set of states.
- $\bar{s}$  is the initial state.
- $R : S \times S \rightarrow \mathbb{R}_{\geq 0}$  is the transition function, where  $R(s, s')$  is the coefficient of the distribution function  $1 - e^{-R(s, s') \cdot t}$  standing for the evolving probability from  $s$  in  $s'$  within  $t$  time instants.
- $L \rightarrow 2^{AP}$  is the labeling function, it assigns to each  $s \in S$  the set  $L(s)$  of atomic propositions  $a \in AP$  holding in  $s$ .

Whenever in a state two or more output transitions are defined, a race-condition occurs. More than one transition is enabled and the output is selected as follows. First of all, the probability that the residence time in state  $s$  is less than  $t$  is computed as  $(1 - e^{-E(s) \cdot t})$ , where  $E(s) = \sum_{s' \in S} R(s, s')$  is the sum of all outgoing coefficients. If the transition is triggered, then the probability of going from  $s$  to  $s'$  is computed as  $\frac{R(s, s')}{E(s)}$ . By replacing in CTMC model  $R(s, s')$  with the normalized value  $P(s, s')$  computed as

$$P(s, s') = \begin{cases} \frac{R(s, s')}{E(s)} & \text{if } E(s) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

we obtain a discrete time model called *Embedded DTMC*, which represents the equivalent system at discrete time. This coincides with value of  $P(s, s', t)$  that is  $\lim_{t \rightarrow \infty} P(s, s', t)$ .

Figure 5.7 shows an example of CTMC representing a generic server and its associated finite queue. The numbers on the arcs represent the job arrival rates (4/3) and the job service rates (2/3), respectively.

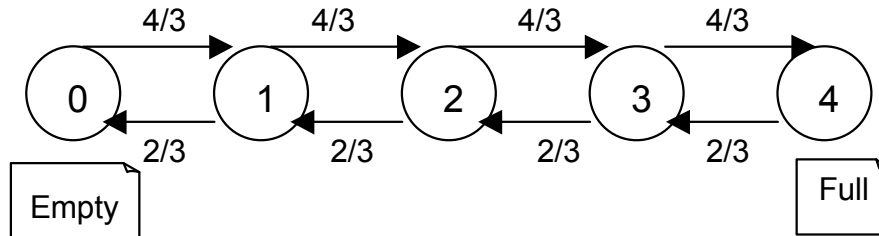


Figure 5.7: An example CTMC.

### 5.3.3 Markov Decision Processes

Markov Decision Processes [58] are an extension of DTMC (or of CTMC) that allow multiple probabilistic behaviours to be specified as output of a state, considering these behaviours non-deterministically. First of all, the particular behaviour to follow is selected in every state; consequently, the next state is selected using the probabilities described in the previously selected behaviour. MDP are characterized by a discrete set of states representing possible configurations of the system being modeled and transitions between states occur in discrete (or continuous, if they extend CTMC) time-steps, but in each state there is also a non-deterministic choice between several discrete probability distributions over successor states.

Formally a MDP is a 4-tuple  $\langle S, \bar{s}, STEPS, L \rangle$  where:

- $S$  is the finite set of states.
- $\bar{s}$  is the initial state.
- $STEPS : S \rightarrow 2^{ACT \times Dist(S)}$  is the transition function, where  $ACT$  represents the set of available non-deterministic choices and  $Dist(S)$  is the set of probability values over the set  $S$
- $L \rightarrow 2^{AP}$  is the labeling function, it assigns to each  $s \in S$  the set  $L(s)$  of atomic propositions  $a \in AP$  holding in  $s$ .
- $\sum_{s' \in S} P_a(s, s') = 1$  for all choices  $a$  associated with a state  $s$

Figure 5.8 shows a simple MDP, with transitions happening at discrete time instants and where:

$S = \{a, b, c, d\}$  and  $\bar{s} = a$

$AP = \{Init, Connect, Fail, Success\}$

$L(a) = \{Init\}$ ,  $L(b) = \{Connect\}$ ,  $L(c) = \{Fail\}$ ,  $L(d) = \{Success\}$

Steps(a) = (0.9 →b, 0.1→c )

Steps(b) = (0.25 →c, 0.75→d ), (1 →b)

Steps(c) = (1 →c )

Steps(d) = (1 →d)

Note that, non-deterministic choices are linked with a dashed line.

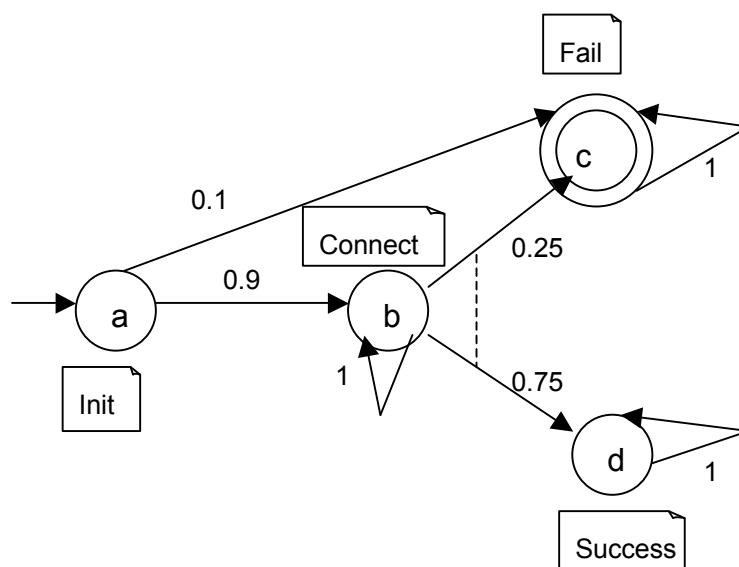
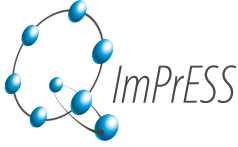


Figure 5.8: An example of MDP.

### 5.3.4 Stochastic Model Checking

Stochastic model checking is an automatic procedure for establishing if a desired property holds in a probabilistic system model. Conventional model checkers start from a description of a model and a specification (using a state-transition system and a formula in some temporal logic, respectively) and return a boolean value, indicating whether or not the model satisfies the specification. In the case of probabilistic model checking, the models are probabilistic (obtained as a variant of Markov chains) and they add a probability to the transitions between states. In this way it is possible to calculate the likelihood of the occurrence of certain events during the execution of the system. This, in turn, allows quantitative analysis of the system, in addition to the qualitative statements made by conventional model checking. Probabilities are modeled via probabilistic operators that extend conventional (timed or untimed) temporal logic.

The key point of probabilistic model checking is the ability to combine probabilistic analysis and conventional model checking in a single tool. The first extension of model checking algorithms

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

to probabilistic systems was proposed in the 1980s. However, work on implementation and tools did not begin until recently, when the field of model checking matured [59, 57]. Probabilistic model checking draws on conventional model checking, since it relies on reachability analysis of the underlying transition system, but must also entail the calculation of the actual likelihoods through appropriate numerical methods, such as those employed in performance analysis tools [59, 57].

Available model checker tools are mainly based on two probabilistic temporal logics, called Probabilistic Computation Tree Logic [60] and Continuous Stochastic Logic [61].

### 5.3.5 A running example

In the following we provide an example of DTMC for the evaluation of the reliability of the reference client-server architecture. In the single node deployment scenario, the system can be modelled as a DTMC which includes three states (see Figure 5.9). The state  $a$  indicates that the node is idle, state  $b$  indicates that the system is executing a request successfully, while state  $c$  represents the failure state which is reached in case of a fault of the hardware or the software components (server or database). The arcs can be labelled with transition probabilities which can be derived from the SAMM quality annotations.

Figure 5.10 shows the second deployment scenario. The state  $a$  models that the whole system is idle. The state  $b$  models that the system is working (at least one server node or the database are serving a request, here we assume that servers at the first layer have the same characteristics and the load is evenly distributed). The states  $c_1$ ,  $c_2$ , and  $c_3$  correspond to a failure of one, two, three server at the application tier. In a similar way,  $cDB$  models a failure at the database tier. Both  $c_3$  and  $cDB$  are final states.

Note that, from the reliability point of view, it is not necessary to differentiate the model to represent the *Whois* and *Whoare* behaviours. Indeed, a failure of a single node has the same impact on the reliability properties independently of the performance class of the users.

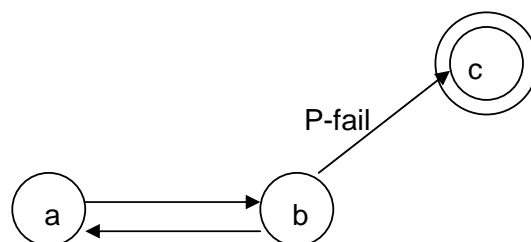


Figure 5.9: DTMC model for the single tier client-server example.

### 5.3.6 Solution Techniques

The solution of Markovian models aims at determining the system behaviour as the time  $t$  approaches to infinite. It consists of the evaluation of the stationary probability  $\pi_s$  of each state  $s$  of

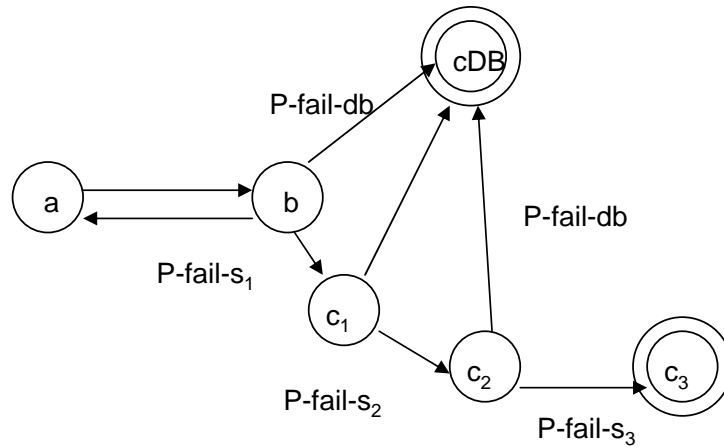


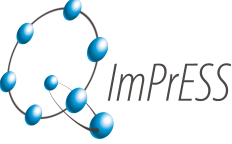
Figure 5.10: DTMC model for the two-tier client-server example.

the model. The solution techniques differ according to the specific model and to the underlying assumptions (e.g., transient or non-transient states, continuous vs. discrete time, etc.). For example, the evaluation of the stationary probability  $\pi_s$  of a DTMC model can be obtained by solving a linear set of equations  $\pi = \pi P$  with the normalizing condition  $\sum_{s \in S} \pi_s = 1$ . In other words,  $\pi$  is the left eigenvector associated with the dominant eigenvalue of  $P$ . In this particular case, the dominant eigenvalue is equal to 1, since the  $P$  matrix is stochastic [39, 37]. Hence, the evaluation of the stationary probabilities requires the solution of a linear system whose size is given by the cardinality of the state space  $S$ . The exact solution of such system can be obtained only if  $S$  is finite or when  $P$  has a specific form. DTMCs including transient and absorbing states necessitate a more complex analysis for the evaluation of the average number of visits and absorbing probabilities. The detailed derivation is discussed in [27].

### 5.3.7 Strength and Weakness

The main problem of Markov models is the explosion of the number of states when they are used to model real systems [27]. On the other hand, Markov models are very general since they can include other modelling approaches as special cases. As an example, under the assumptions of the BCMP theorem, the single service center queue of Figure 5.1, known also as M/M/1 queue, can be modelled by a CTMC with an infinite number of states (each state representing the number of customers in the system), and the closed formulas computed by QN theory are obtained by determining the probability stationary conditions of the underlying Markov chains. Furthermore, CTMC can be used to compute the solution of non-product-form QNs. Other models that can be reduced to Markov models are the families of Stochastic Petri Nets (SPN) and Stochastic Process Algebras (SPAs) [62].

Markov models are very general, they allow estimating both performance and reliability metrics and allow modelling systems without imposing any restriction on the users or resources be-

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

haviour. Hence, more accurate results can be derived with respect to the solution of QN models conforming to the restrictions imposed by the BCMP theorem. This is obtained at the cost of a higher computational effort which is due to the usually large number of states and parameters. Furthermore, with respect to QN models, Markov model generation and interpretation can not be straightforward for representing real complex systems.

Finally, as a general consideration, the accuracy of Markov models depends on the precision of the state transition probability matrix, which may possibly include, a quite large number of parameters.

### 5.3.8 Model Adoption

Similar to QN models, Markovian models can be used at design time to derive performance and/or reliability metrics [63, 64]. Some recent extensions of Markov models address the problem of modelling system transients in order to study burstiness and long range dependency in system workloads. Authors in [65] introduce matrix-analytic analysis to study a Markovian Arrival Process as input and Phase Type distribution as service time in a single serving resource. Other studies (like [66]) focus on the analyses of non-renewal workloads by means on Markov models but, due to the analysis complexity, only small size models based on one or two service centers can be dealt with so far. As discussed above, in real systems Markov models may suffer for high computation overhead.

### 5.3.9 Tools for Model Derivation

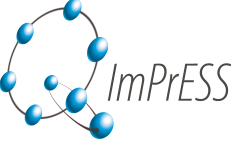
In the literature, there exists several contributions which propose transformation techniques for Markov models derivation. DTMC, for example, can be automatically derived starting from a description of the system behaviour using the methods and algorithms proposed in [63, 64]. Besides, Markov models and Markov decision processes are derived through ad-hoc transformations in [67, 68].

Some indirect transformations have been defined starting from software models and deriving SPN [69, 70, 71], SPA [62], or models for stochastic model checking like PRISM [72, 73]; the resulting models are then analyzed via the numerical solution of the underlying Markov chain.

The aforementioned ad-hoc methods follow transformation patterns similar to the ones used for QN models.

## 5.4 Automata Models

*Timed (finite) automata* (TA) [74] model the behavior of real-time systems over time. The TA framework provides a simple, yet powerful, way to annotate state-transition graphs with timing constraints using finitely many real-valued clocks. A timed automaton accepts timed words — infinite sequences in which a real-valued time of occurrence is associated with each symbol. In the following, we study extensions of timed automata suitable for performance and reliability analysis of service-oriented systems.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

### 5.4.1 Priced Timed Automata

*Priced (or weighted) timed automata* [75, 76] framework is an extension of timed automata [74] with prices/costs on both locations and transitions.

Let  $X$  be a finite set of clocks and  $\mathcal{B}(X)$  the set of formulas obtained as conjunctions of atomic constraints of the form  $x \bowtie n$ , where  $x \in X$ ,  $n \in \mathbb{N}$ , and  $\bowtie \in \{<, \leq, =, \geq, >\}$ . The elements of  $\mathcal{B}(X)$  are called *clock constraints* over  $X$ .

A linearly Priced Timed Automaton (PTA) over clocks  $X$  and actions  $Act$  is a tuple  $(L, l_0, E, I, P)$ , where:

- $L$  is a finite set of locations,
- $l_0$  is the initial location,
- $E \subseteq L \times \mathcal{B}(X) \times Act \times 2^X \times L$  is the set of edges,
- $I : L \rightarrow \mathcal{B}(X)$  assigns invariants to locations, and
- $P : (L \cup E) \rightarrow \mathbb{N}$  assigns prices (or costs) to both locations and edges. In the case of  $(l, g, a, r, l') \in E$ , we write  $l \xrightarrow{g, a, r} l'$ .

The semantics of a PTA is defined in terms of a timed transition system over states of the form  $(l, u)$ , where  $l$  is a location,  $u \in \mathbf{R}^{|X|}$ , and the initial state is  $(l_0, u_0)$ , where  $u_0$  assigned all clocks in  $X$  to 0. There are two kinds of transitions: delay transitions and discrete transitions. In delay transitions,

$$(l, u) \xrightarrow{d, p} (l, u \oplus d)$$

the assignment  $u \oplus d$  is the result obtained by incrementing all clocks of the automata with the delay amount  $d$ , and  $p = P(l) * d$  is the cost of performing the delay. Discrete transitions

$$(l, u) \xrightarrow{a, p} (l', u')$$

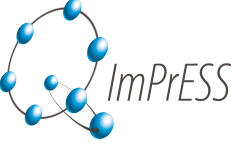
correspond to taking an edge  $l \xrightarrow{g, a, r} l'$  for which the guard  $g$  is satisfied by  $u$ . The clock valuation  $u'$  of the target state is obtained by modifying  $u$  according to updates  $r$ . The cost  $p = P((l, g, a, r, l'))$  is the price associated with the edge.

A timed trace  $\sigma$  of a PTA is a sequence of alternating delays and action transitions

$$\sigma = (l_0, u_0) \xrightarrow{a_1, p_1} (l_1, u_1) \xrightarrow{a_2, p_2} \dots \xrightarrow{a_n, p_n} (l_n, u_n)$$

and the cost of performing  $\sigma$  is  $\sum_{i=1}^n p_i$ . For a given state  $(l, u)$ , the minimum cost of reaching  $(l, u)$  is the infimum of the costs of the finite traces ending in  $(l, u)$ . Dually, the maximum cost of reaching  $(l, u)$  is the supremum of the costs of the finite traces ending in  $(l, u)$ .

A network of PTA  $A_1 || \dots || A_n$  over  $X$  and  $Act$  is defined as the parallel composition of  $n$  PTA over  $X$  and  $Act$ . Semantically, a network again describes a timed transition system obtained from those components, by requiring requiring discrete transitions to synchronize on complementary actions (i.e. the output action  $a?$  is complementary to the input action  $a!$ ).

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

## 5.4.2 Multi Priced Timed Automata

The class of Multi Priced Timed Automata (MPTA) is an extension of PTA in which a timed automaton is augmented with more than one cost variable [77, 78]. In the case of two costs associated with a PTA, the minimal cost reachability problem corresponds to finding a set of minimal cost pairs  $(p_1, p_2)$  (both  $p_1$  and  $p_2$  are minimized) reaching a goal state. Since the costs contributed from the individual costs can be incomparable, when, e.g. for the costs of two traces, say  $(p_1, p_2)$  and  $(p'_1, p'_2)$ ,  $p'_1 < p_1$  and  $p_2 < p'_2$ , the solution is a set of pairs, rather than a single pair. In this setting, the minimal cost reachability problem is to find the set of incomparable pairs with minimum cost, reaching the goal state. Dually, the maximization problem is defined as finding the set of incomparable pairs with maximal cost reaching the target location, or to conclude  $(\infty, \infty)$  if the target location is avoidable in a path that is infinite, deadlocked, or has a location in which it can make an infinite delay. A specific problem is the optimal conditional reachability problem, in which one of the costs should be optimized/maximized, and the other bounded by an upper/lower bound. We refer the reader to [77] for a thorough description of optimization problems in MPTA.

## 5.4.3 Weighted CTL

Let  $AP$  be a set of atomic propositions. In order to specify properties of PTA, Weighted CTL (WCTL) has been introduced [78]. WCTL extends Timed CTL with resets and testing of cost variables. Its syntax is given by the following grammar:

$$WCTL \ni \phi ::= \mathbf{true} \mid a \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{E} \phi \mathbf{U}_{\mathbf{P} \sim c} \phi \mid \mathbf{A} \phi \mathbf{U}_{\mathbf{P} \sim c} \phi$$

where  $a \in AP$ ,  $\mathbf{P}$  is a cost function,  $c$  ranges over  $\mathbb{N}$ , and  $\sim \in \{<, \leq, =, \geq, >\}$ .

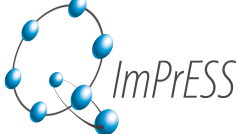
We interpret formulas of WCTL over labeled PTA, that is, PTA having a labeling function  $l$  that associates with every location  $q$  a subset of  $AP$ .

Let  $A$  be a labeled PTA. The satisfaction relation of WCTL is defined over configurations  $(q, v)$  of  $A$ , as follows:

$$\begin{aligned}
(q, v) & \models \mathbf{true} \\
(q, v) \models p & \Leftrightarrow a \in l(q) \\
(q, v) \models \neg\phi & \Leftrightarrow (q, v) \not\models \phi \\
(q, v) \models \phi_1 \vee \phi_2 & \Leftrightarrow (q, v) \models \phi_1 \text{ or } (q, v) \models \phi_2 \\
(q, v) \models \mathbf{E} \phi_1 \mathbf{U}_{\mathbf{P} \sim c} \phi_2 & \Leftrightarrow \text{there is an infinite run } \varrho \in A \text{ from } (q, v) \\
& \text{such that } \varrho \models \phi_1 \mathbf{U}_{\mathbf{P} \sim c} \phi_2 \\
(q, v) \models \mathbf{A} \phi_1 \mathbf{U}_{\mathbf{P} \sim c} \phi_2 & \Leftrightarrow \text{any infinite run } \varrho \in A \text{ from } (q, v) \\
& \text{satisfies } \varrho \models \phi_1 \mathbf{U}_{\mathbf{P} \sim c} \phi_2 \\
\varrho \models \phi_1 \mathbf{U}_{\mathbf{P} \sim c} \phi_2 & \Leftrightarrow \text{there exists } \pi > 0 \text{ position along } \varrho \\
& \text{such that } \varrho[\pi] \models \phi_2, \text{ and for all positions } 0 < \pi' < \pi \\
& \text{on } \varrho, \text{ such that } \varrho[\pi'] \models \phi_1, P(\varrho \leq \pi) \sim c
\end{aligned}$$

If  $A$  is not clear from the context, we may write  $(q, v), A \models \phi$  instead of simply  $(q, v) \models \phi$ .

We will use shortcuts as  $\mathbf{E} \mathbf{F}_{\mathbf{P} \sim c} \phi \equiv \mathbf{E} \mathbf{true} \mathbf{U}_{\mathbf{P} \sim c} \phi$ , or  $\mathbf{A} \mathbf{G}_{\mathbf{P} \sim c} \phi \equiv \neg \mathbf{E} \mathbf{F}_{\mathbf{P} \sim c} \neg\phi$ . Moreover, if the cost function  $P$  is unique or clear from the context, we may write  $\phi \mathbf{U}_{\sim c} \psi$  instead of  $\phi \mathbf{U}_{\mathbf{P} \sim c} \psi$ .

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

#### 5.4.4 A Running Example

In the following, we are focusing on evaluating the performance and reliability of the reference client-server architecture of the Q-ImPrESS Deliverable D2.1, in the PTA framework [75, 76]. We are modeling and analyzing the client-server example using the model-checker UPPAAL CORA, an extension of the UPPAAL model-checker to support cost-based reachability formal analysis of PTA networks. UPPAAL CORA finds optimal/worst-case paths that match given goal conditions expressed as WCTL reachability formulas.

For simplicity, we assume the one-node deployment scenario for the server and database, and are interested in analyzing the energy consumption (during program execution) represented as a cost variable in the given system model. Here, we assume that energy consumption is directly proportional to execution time, which is a simplifying assumption that might give us an underestimated value for energy consumption. As such, estimating worst-case energy consumption could be regarded as a performance estimation for the client-server example. For the calculation of the execution time of a service, we have to take into consideration the type of CPU instruction, that is, to differentiate between *load* or *store* instructions (for now, in this example, we ignore other possible types of instructions correlated to server's administration). The separation into types of instructions is justified by the fact that not all instructions are executed within the same number of CPU cycles: the number of CPU cycles needed for "load instruction" (*NoCycLoad*) is 5, whereas for the "store instruction" (*NoCycStore*) is 4. Besides these, we also assume that multiple instructions are executed per server request; in this particular model, we assume 25 load instructions (*NoIload*), and 20 store instructions (*NoIstore*). Then, the execution time of a request is given, in a simplified form, as:

$$ET = (w * NoCyc) / Frequency,$$

where

- $w$  is the weight that expresses the relative importance of various service execution times;
- $NoCyc$  represents the total number of cycles of all instructions. It is used to describe an aspect of the CPU performance. In our case, since we are dealing with load and store instructions  $NoCyc$  is as follows:  $NoCyc = NoCycLoad * NoIload + NoCycStore * NoIstore$ ;
- Frequency is the server's CPU frequency.

Alternatively, given the relation between frequency and period, ( $Frequency[Hz] = 1/Period[1/s]$ ),  $ET$  may be expressed as follows:

$$ET = w * NoCyc * Period$$

The server's energy consumption per each request is calculated as  $E_i = PW * ET_i, i \in [1..n]$ , where  $PW$  is the server's CPU power consumption. In this example, we assume that  $PW$  is known for the given server processor. The total, accumulated energy consumption can be modeled as a cost variable, as follows:

$$C_{energy} = \sum_i E_i$$

The modeling of the system's behavior is an aggregation of the behavioral models of its components, that is, a network of three non-deterministic PTA, the Client, the Server, and the Database. For this model, we assume that there are three clients trying to get the server's services. In the following, we briefly describe the PTA models of the components.

Figure 5.11 depicts the Client modeled as a PTA. The automaton has two locations: *Start* and *WaitsForData*. The synchronization with the Server PTA is modeled by using three channels: *ReqSyn* (models the synchronization that represents sending a query request), *RespSyn* (models the synchronization that represents the query response from the server), and *FailSyn2* (models the synchronization that shows a failure occurrence in the system). The clock *t1* is bounded from above in location *WaitsForData*, constraint (called location invariant) that ensures that the Client does not wait more that 10 time units for its request. In case a failure occurs because of timing issues, the Server is informed about that via channel *FailSyn2*, and the bounded integer variable *NoReqFailed* is updated accordingly. If a failure occurs because of faulty retrieve of information from the database, the variable *dbFailure* becomes 1, and the same synchronization channel is used to report failure, while variable *NoReqFailed* is updated. After any of such scenarios completes, a similar query is being sent to the Server, again.

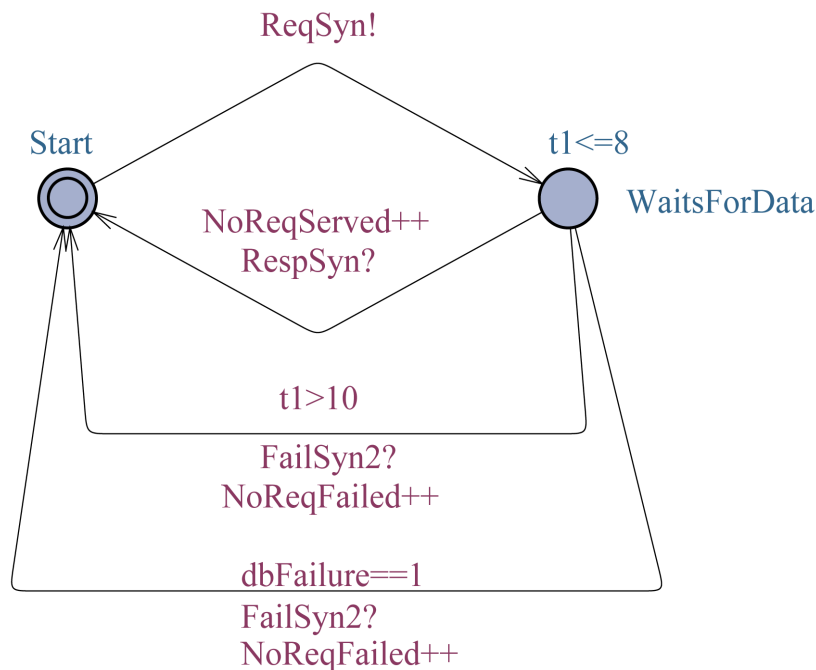


Figure 5.11: The Client Model as a PTA

The PTA model of the Server is given in Figure 5.12. As one can observe, it consists of four locations. Synchronization channels are used for ensuring communication with the Client and the Database. Channels *ReqSyn*, *ResSyn*, and *FailSyn2* model communication with the Client. The synchronization on a query request is modeled via *ReqSyn*, while channel *RespSyn* is used to model query response; *FailSyn2* models failures that are correlated with clock *t2* and integer

variable `dbFailure`. The clock variable `t2` ensures that the server does not wait for data from the database longer than 8 time units. The channels `dbReqSyn` and `FailSyn1` are used to communicate with the database. The first one sends the database response, whereas the second reports an occurred failure.

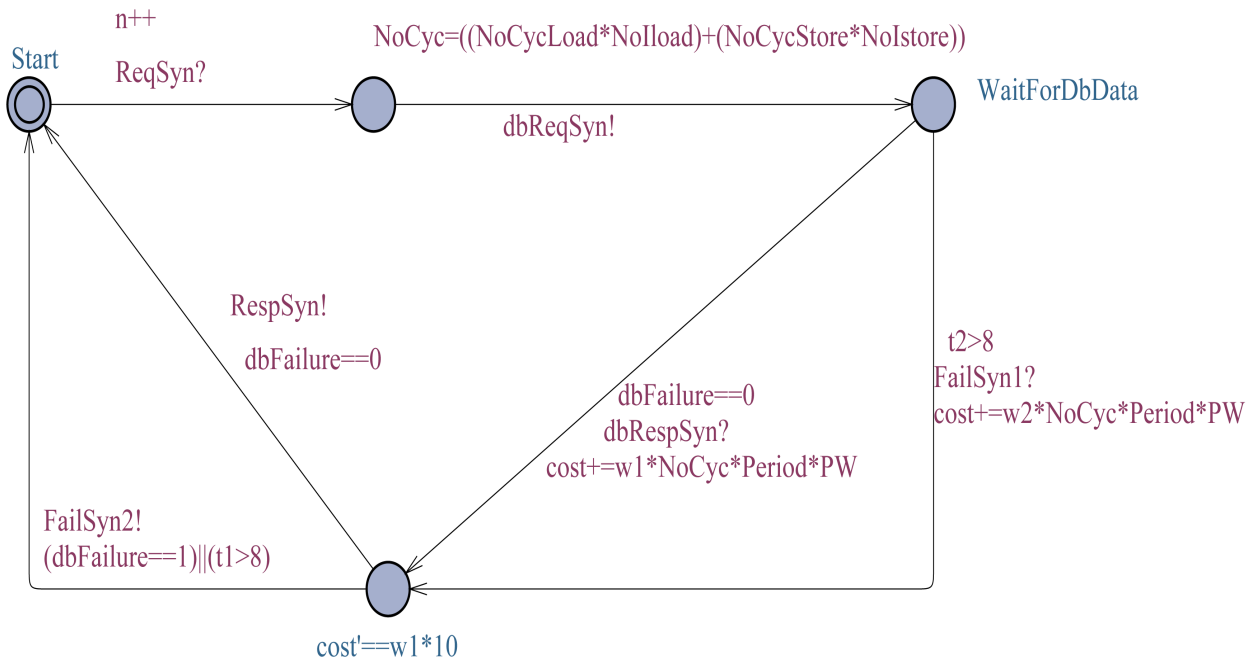
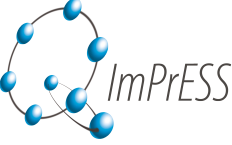


Figure 5.12: The Server Model as a PTA

Figure 5.13 gives an overview of the PTA model of the database. It consists of two locations `Start` and `GatheringData`. As before, the channels `dbReqSyn`, `dbRespSyn`, and `FailSyn1` model the communication with the server. The channel `dbReqSyn` establishes the communication with the server, by accepting the server's query request. If clock variable `t3` becomes greater than 6, that is, if more than 6 time units pass after starting to gather the requested data, a failure occurs. The integer variable `dbFailure` is updated, while channel `FailSyn1` informs the server about the failure. The response to the requested query is modeled through synchronization channel `dbRespSyn`. When the next database request comes, variable `dbFailure` is reset.

The timing constraints for each of the components are chosen arbitrarily.

Performance analysis based on the energy consumption model described above has been carried out on the composition of the above PTA models. Considering a scenario in which ten requests are sent to the server, in basic query mode, and assuming also that the server runs on a platform equipped with a 2GHz processor, without any scheduling policy of serving the clients' requests, model-checking the above composition gives us an estimated execution time of 2,03 ms. This value would be higher if one considered administration-based instructions on the server side (rather than just load and store instructions). This is justified by the fact that additional instructions bring additional clock cycles per instruction. These values can be improved by deploying the system on

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

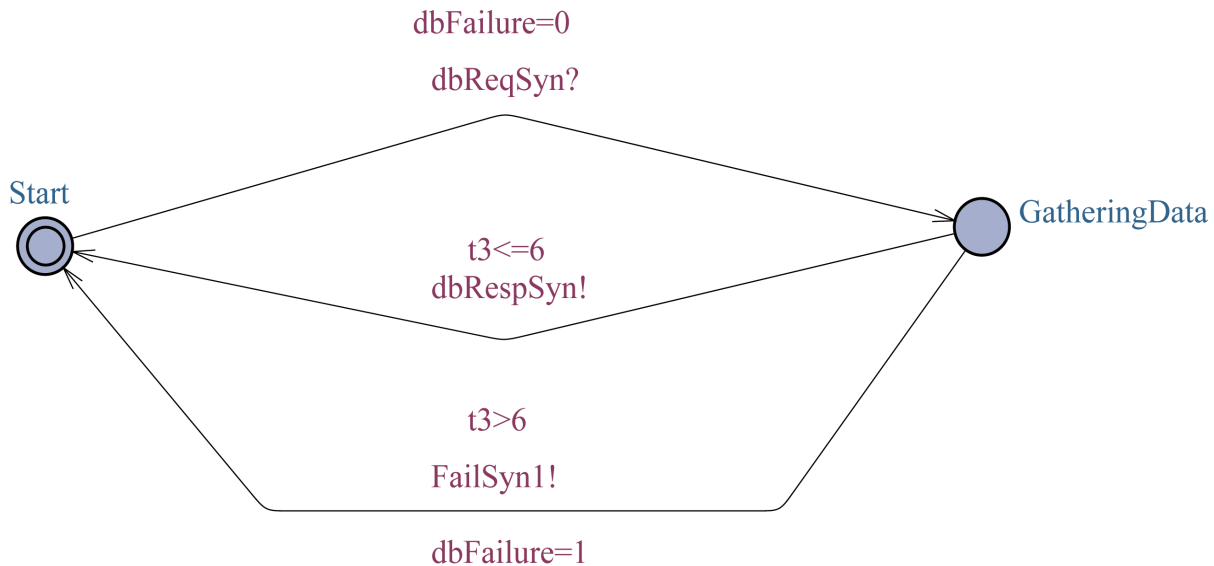


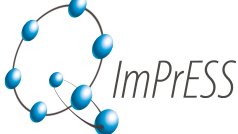
Figure 5.13: The Database Model as a PTA

super-scalar processors of the new generation. Note that we have assumed that each request is a simple query that can be easily served. Modeling large queries would impact on the computed execution time, by increasing it proportionally.

We also consider predicting reliability, measured by the number of system failures. Since we do not support reliability analysis on a probabilistic model, we have simply defined a bounded integer variable that counts the number of failures occurred in those cases when requests were not served within given deadlines. By verifying the modeled system described above with respect to the number of failures that might occur during the client-server communication, one can infer that such a system has high reliability regarding the possible failures. If one considers the mass query mode, the reliability increases slightly, due to big query requests being split in smaller queries that can easily be executed within given time bounds.

### 5.4.5 Solution Techniques

The solution of PTA/MPTA models aims at determining the system's behavior in case continuous consumption of resources (e.g. energy) are modeled and analyzed. It consists of computing the cost of a PTA's execution, as the maximum or least price/price pairs of reaching a state in a set of goal states,  $G$ , along the execution, or as  $\infty$  if the entire execution avoids states in  $G$  [77, 78]. Since clocks are defined over the non-negative reals, the priced transition system generated by a PTA can be uncountably infinite, thus an enumerative approach to the cost-optimal reachability problem is infeasible. In order to effectively analyze priced transition systems, the algorithms for synthesizing the maximal/minimal reachability costs of achieving some goal set are defined over priced symbolic states. Priced symbolic states provide symbolic representations of possibly infinite sets of actual states and their association with costs. The idea is that during exploration, the

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

infimum/maximum cost along a symbolic path (a path of symbolic states) is stored in the symbolic states itself. If the same state is reached with different costs along different paths, the symbolic states can be compared, discarding the more expensive/cheaper state. The priced symbolic states that one encounters for PTA are representable by simple constraint systems over clock differences (often referred to as a clock-zone in the timed automata literature). The cost is given by an affine plane over the clock-zone.

In UPPAAL CORA, cost-optimal reachability analysis is performed using a standard branch and bound algorithm. Branching is based on various search strategies implemented in UPPAAL CORA which, currently, are breadth-first, ordinary, random, or best depth-first with or without random restart, best-first, and user supplied heuristics. The latter enables the user to annotate locations of the model with a special variable called *heur* and the search can be ordered according to either largest or smallest *heur* value.

#### 5.4.6 Strength and Weakness

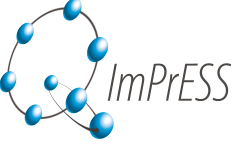
Automata models are very expressive, allowing estimation of both performance (in terms of service response time) and reliability (in terms of global number of failures) of systems with or without nondeterministic behavior. Timed automata-based techniques offer accuracy potentially at the expense of speed. Exact upper bounds on the timeliness properties can be found (with the UPPAAL model checker) for a number of usage scenarios. The results have been compared with three other performance modeling techniques [79]. This comparison shows that, if the state space of the model is tractable, UPPAAL gives the most accurate results out of the considered techniques, yet at a similar cost. Generating TA/PTA models might be intricate sometimes, however the modeling strategy can be automated, which mitigates the difficulty and error-proneness of manually constructing timed automata models.

#### 5.4.7 Model Adoption

Timed Automata models can be used at design time to compute performance and/or reliability metrics [80, 81, 82], but also for run-time monitoring of services [83, 84]. Recently, timed automata have been used to model and analyze timeliness properties of embedded system architectures [79]. Using a case study inspired by industrial practice, a suitable timed automata model is composed and presented in detail.

QoS evaluation and admission control, based on the modeling of both system behavior and QoS requirements, in a TA framework, has been investigated [80]. Two aspects are considered. The first one refers to QoS management, and to a component-based architecture for QoS evaluation. The second one illustrates the approach with the help of a case study based on a Personal Area Network that includes a Wireless router connected to the Internet. The compatibility of the mechanism with architectures that promote QoS management, such as ITSUMO, is also highlighted.

Report on the use of timed automata and the UPPAAL libraries for the verification of web service orchestrations can be found in [81]. Timeliness properties of orchestrations, prior to deployment, are analyzed. Most recent work on correlating resource usage of component-based

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

systems with performance and reliability analysis has been carried out within the framework of PTA/MPTA [82]. As demonstrated in a small accompanying example, the approach allows for rigorous predictions of performance and/or reliability, depending on the prices of using various resources, such as CPU share, memory etc.

A description of how TA can be used as a formalism to support efficient online monitoring of timeliness, reliability, and throughput constraints expressed in web-service SLAs is provided in [83, 84]. In particular, the question of whether an SLA is violated is reduced to the acceptance of a timed word by a timed automaton. It is proved that this is decidable in quadratic time and the approach introduces only a linear run-time overhead for a web service invocation.

### 5.4.8 Tools for Model Derivation

In the literature, there exist several contributions that propose automated techniques for TA/PTA model derivation. TA models, for example, can be automatically derived starting from a description of the system behavior [85]. Transformations of a newly introduced resource model for embedded/service-oriented systems, REMES [86], into PTA, have been recently defined and are in the process of being automated [87].

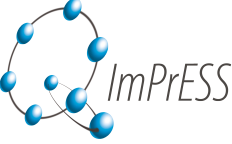
The implementation of the verification of web service orchestrations provides ways of systematically constructing TA models from Orc models [81]. An experimental tool is implemented to automate the approach.

## 5.5 Simulation Models

Simulation is a very general and versatile technique to study the evolution of a software system, which is represented by means of a simulation model. Simulation can be adopted at design time in order to evaluate performance and reliability metrics both in steady state and in transient conditions.

Simulation requires the development of a simulation program that mimics the dynamic behaviour of the system by representing the system components and interactions in terms of functional relations. Non-functional attributes are estimated by evaluating the values of a set of observations gathered in the simulation runs.

Simulation results are obtained by performing statistical analyses of multiple runs. If the goal of the analysis is the steady state of the system, simulation output requires statistical assurance that the steady state has been reached. The main difficulty is to obtain independent simulation runs with exclusion of an initial transient period. The two techniques commonly used for steady state simulation are the “batch means method”, and “independent replication” [88, 37]. None of these two methods is superior to the other in all cases. Their performance depends on the magnitude of the traffic intensity. The other available technique is the “regenerative method”, which is mostly used for its theoretical nice properties; however, it is rarely applied in actual simulation to obtain the steady state output numerical results [88, 37].

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

The steady state values which can be obtained as simulation output are characterized by confidence intervals which give an estimated range of values which is likely to include the average values of performance/reliability metric of interest for the system. The width of the confidence interval expresses uncertainty about the quality metric. A very wide interval, e.g., may indicate that more data should be collected during the simulation because nothing very definite can be said about the analysis. The confidence level is the probability value  $(1 - \alpha)$  associated with a confidence interval. It is often expressed as a percentage. For example, say  $\alpha = 0.05 = 5\%$ , then the confidence level is equal to  $(1 - 0.05) = 0.95$ , i.e. a 95% confidence level. If the interest of the analysis is the estimate of a metric distribution, the concept of confidence interval can not be easily defined [89].

Very often, simulation is used to evaluate performance metrics of non-product-form QNs or if the metric of interest is not the mean response time but other types of quartiles. In this case, the simulation program allows the definition of service centers and network topology and allows analysing in detail the components behaviour which violates BCMP theorem assumptions. For example, simulation allows analysing the impact of blocking conditions, or particular routing algorithms (e.g., the execution at the shortest queue among multiple parallel components). Furthermore, non-Poisson arrival rates for incoming workload, or heavy tail distributions (e.g., Pareto) for the service demands characteristics for Web systems can also be considered. Therefore the class of simulation models is much more general than the class of analytical product-form models. However, the major drawback of simulation with respect to QN models evaluation is their computational cost [37].

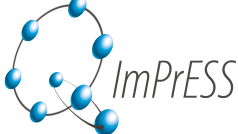
### 5.5.1 Strength and Weakness

The main simulation advantage is its inherent flexibility which allows modelling the system possibly with a very fine grain. Furthermore, very accurate results could be usually achieved. For example, some recent proposals can obtain results at an instruction cycle level precision, even for service center environments [90].

Accuracy and flexibility are obtained at the cost of a higher computational effort with respect to the solution other models (e.g., QN). Furthermore, an estimate of the result precision can be obtained only for some point estimators like means. Vice versa, if the aim of the analysis is obtaining the whole metric distribution, the goodness-of-approximation is mathematical difficult to define.

### 5.5.2 Model Adoption

Simulation models are adopted only at design time since the computational effort is significant both for the model derivation and for the computation time, especially if a simulation stop condition is selected which requires a lot of simulation data to be collected. For example, this applies if for point estimators the confidence intervals are narrow. Simulation models have been adopted at the design time in [91, 92] to evaluate the quality of service of composed BPEL processes, starting from the quality profile of the component Web services. Simulation models are largely adopted

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

also for the validation of the results of new bounding or approximate analytical solution of QN models [34].

### 5.5.3 Tools for Model Derivation

Ad-hoc methods have been proposed for the automatic transformation of software models into simulation models. Examples can be found in [43, 23]. Translations usually start from UML-like specification of the system and are mapped into internal representations, built on ad-hoc or general purpose simulation libraries.

In the Q-ImPrESS project, the SimuCom simulation is available via the Palladio Component Model [14]. SimuCom transforms a PCM instance in an OSGi plugin via a model-2-text transformation. The resulting plugin is then executed to derive the simulation results. SimuCom focuses on deriving the *distribution function* of the PCM instance's response time. SimuCom as a PCM solver supports arbitrary distributions for random variables which makes the resulting queueing network too complex for any known analytical solution technique. The resulting distribution function is suited to analyse service level agreements which can be directly retrieved from the cumulative distribution function (CDF).

## 5.6 Control-Oriented Models

In recent works presented in the literature [25], control oriented techniques have been introduced in order to estimate performance model parameters such as QN demanding times or resource utilizations. Control oriented models will be adopted in the Q-ImPrESS project in order to calibrate design-time models parameters and to validate the design-time analysis predictions. Indeed, this model family allows generalizing the results obtained in a single running experiment in more multiple scenarios, e.g., characterized by different incoming workloads and/or resource demands.

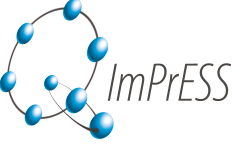
As an example, a discrete time, linear time invariant (LTI) model can be represented by the following set of equations:

$$\begin{aligned}
 x_{k+1} &= \mathbf{A}_k x_k + \mathbf{B}_k u_k + \mathbf{K}_k e_k \\
 y_k &= \mathbf{C}_k x_k + \mathbf{D}_k u_k + e_k,
 \end{aligned}
 \tag{5.1}$$

where  $k$  is the discrete time index,  $x \in \mathbb{R}^n$  is the state vector,  $u \in \mathbb{R}^m$  is the vector of control inputs,  $y \in \mathbb{R}^l$  is the vector of measured outputs, and  $e_k$  is a white process noise. For example, the output could be the response time or utilization of a given component or node, the input could be the incoming workload and the state could represent the current number of requests in the system.

Genuine control-oriented modelling approaches can accurately model software system transients and are very effective over fine grained time scales, e.g., minutes or seconds.

The first application of system-theoretic modelling methods applied to the management of Web services are reported in [93, 94, 95, 96]. Early works focused on system identification techniques with the goal to estimate the constant matrices which model the system behaviour.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

Identification procedures can be either performed off-line or on-line. In the former approach, ad-hoc performance tests injecting varying workload and working conditions on the software system are needed before the production deployment [97]. Vice versa, in the latter case, on-line methods can determine and adjust the values of the model parameters (matrices in equation (5.1)) while the software system is running [98]. In any case, dynamic models can be built and adopted only when the software is running in the target environment.

Linear Parameter Varying (LPV) models [99] have been recently proposed to model linear time-varying systems whose state space matrices  $\{A(\delta_k), B(\delta_k), C(\delta_k), D(\delta_k)\}$  are fixed functions of some vector of varying parameters  $\delta_k$ . LPV models have been adopted by Qin and Wang [100, 101] in order to identify a model of a Web server.

### 5.6.1 Strength and Weakness

The point of strength of control oriented models is the accuracy level which can be achieved. For example, in [97, 98] it is shown that LPV models introduce an average error in the estimation of response time around 20% with a 10 seconds time granularity.

Frequently, the accuracy is traded-off with the time scale; usually the lower the time granularity the greater is the accuracy at a cost of a greater monitoring overhead. The models can effectively describe a software system behaviour under several working conditions, this encourage the adoption of this model family as a validation method of the design-time predictions.

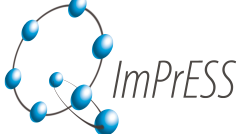
On the other hand, control oriented models require data gathered on an advanced prototype of the system. Furthermore, the time granularity to be adopted in order to achieve a given accuracy level is dependent on the workload intensity and variability. More variable and intense workloads require finer time grain.

### 5.6.2 Model Adoption

Control-oriented models are adopted at run time for the implementation of closed-loop controllers with the aim to adapt the system configuration to environment changing conditions. The use of control-oriented models will be investigated for validation and calibration purposes of design-time models.

## 5.7 Model Comparison and Discussion

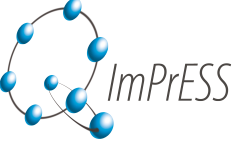
The models presented in the previous sections are analyzed here according to a set of characteristics that are relevant for the Q-ImPrESS project stakeholders in order to drive model selection. To complete the analysis carried out in Sections 5.1-5.6, we provide a qualitative comparison, since the models are heterogeneous (e.g., some are oriented to performance evaluation while others to reliability) and can be used at different abstraction layers and/or at different time granularity. The characteristics are:

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

- *Adaptability.* Prediction techniques should support efficient performance prediction under architecture changes where: (i) components are modified, e.g., by introducing faster components, (ii) homogeneous components (i.e., with the same physical characteristics, e.g., CPU speed, etc.) are added and the load is evenly shared among them, (iii) heterogeneous components are added and the load is not evenly shared (usually faster components receive higher load).
- *Cost effectiveness.* The approach should require less effort than prototyping and subsequent measurements.
- *Composability.* Prediction techniques should be able to make quality predictions based on the quality characteristics of single components, which together build the system. For example SOA systems are intrinsically structured hierarchically, hence quality prediction techniques should be able to exploit this structure in order to determine the QoS metrics of the whole systems by, possibly, exploiting the results obtained by the analyses performed on single components.
- *Scalability.* Software systems are typically built either with a large set of simple components or utilize few large-grain, complex components. To predict performance attributes, analysis techniques need to be scalable to handle both cases.
- *Resource sharing modelling.* An important goal of the Q-ImPrESS project is the modelling of resource sharing for resources not directly represented as entities in the individual performance models. For these resources, external resource models based on resource sharing experiments from Q-ImPrESS Deliverable 3.3 will be employed. Here, we will discuss the ability of the different models to support resource sharing analyses.

The comparison is summarized in Table 5.2. We adopt a qualitative discrete scale for the evaluation of the aforementioned characteristics, i.e., *High*, *Medium* and *Low*; the evaluation is justified by the following discussion.

Considering the adaptability characteristic, we have to analyse possible changes in the systems in different ways, i.e., modifying components, adding homogeneous components, and adding heterogeneous components, since these activities have different implications on the models. Here we consider the adaptability with the aim to determine new results by using always the same class of models and we do not consider the possibility to obtain new results through automatic transformations. In other words, the goal is to revise the model and obtain a new solution, always remaining in the same model family. Usually in the QN model family, bounding techniques and non-product-form models have a high level of adaptability. Vice versa, product-form models are adaptable if faster components or new homogeneous components are introduced into the system, while introducing heterogeneous components usually implies routing mechanisms which violate the assumption of the BCMP theorem. In that case, a system update requires to move from product to non-product-form models or a different class of models (e.g., simulation). Considering the Markovian models, they usually provide a high level of adaptability, since a system update can be modelled simply as a faster service rate or a different probability distribution of the state space.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

Simulation is intrinsically adaptable since a system change can be implemented by modifying the program description of the added or updated components. Control-oriented models, on the other hand, have a lower level of adaptability. In particular off-line black-box models require ad-hoc measurement which cannot be generalized, since the physical meaning of the system parameters is unknown.

Queueing and Markov models have a high level of cost effectiveness since the model solutions have a little effort if compared to the prototyping. Simulation has an intermediate level of cost effectiveness, since it requires a detailed description of system components behaviour. The effort required is anyway lower than the one for the prototype development. On the other hand, control-oriented models have a low level of cost-effectiveness since they are based on the analysis of real data, which have to be experimentally determined on a real implementation environment.

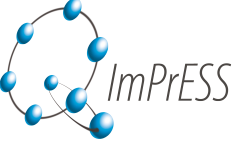
With respect to composability, the models which are structured hierarchically can be composed more easily. On the contrary, flat models or models based on very detailed descriptions (e.g., simulation models, if they are not structured in a modular way) or based on measurements (control-oriented models) have a lower level of composability.

With respect to scalability, bounding techniques can provide results for systems composed by several nodes or classes. Open product-form models are scalable, while closed models have a lower scalability. The models adopted only at design time require considerable computation time to provide a solution and have a low level of scalability. Scalability of simulation models depends on the required level of accuracy. Indeed, the higher is the level of accuracy and narrower are the confidence intervals, the lower is the scalability. For example, some recent approaches can simulate up to 100 physical servers in real service center environments, but in order to provide a solution in a limited time, they need to be supported by 40 nodes [90]. Control-oriented models are also characterized by a high level of scalability. Usually each software component can be described by a local model.

Finally, resource sharing plays a central role in Q-ImPrESS; hence this characteristic is discussed more in depth for each class of model. In Q-ImPrESS, the external resource models will be solved together with the performance models in a bidirectional feedback loop, with the resource models providing information on resource behaviour and the performance models providing information on resource sharing. The modelling potential of the solution naturally depends on the ability of the performance models to provide useful information to the resource models.

**QN:** The basic closed-form QNs are limited by the hypotheses introduced by the BCMP theorem; therefore resource sharing of individual software components can be only roughly estimated. In most cases, this would limit the use of the closed-form models with the external resource models, because the information provided by the performance model is too coarse grained.

Some information that is still of potential value is the average queue lengths of the service centers, which can translate to the average number of concurrent requests handled by a service implementation, and consequently to the average number of service components sharing resources such as memory. Where simplified queue policies are used to approximate more complex process scheduling policies, the external resource models can use additional scheduling information to

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

describe resource sharing behaviour specific to the particular process scheduling policy even when the performance model simplifies the policy. Hence, the ability of QN models to represent resource sharing issues is considered low/medium.

**LQN:** The layered queueing network models present an improvement over the basic closed-form QN models in that they can provide more fine grained information, such as task service times and task completion throughput. In this way it is possible to describe resource sharing at component level in the external resource model.

With more complex performance models, the ability to connect the external resource models to the performance models also depends on the tools used to solve the performance models, as some of the tools can provide limited output data. In general, only the task service times and the task completion throughput can be relied upon, since they are the data of interest in performance modelling. The ability of LQN models to represent resource sharing issues is considered medium.

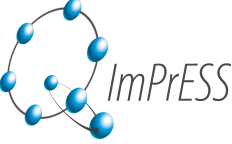
**Markov Models:** The use of the Markov models with the external resource models very much depends on the correspondence between the states of the Markov models and the behaviour of the software services and the service components. Some very specific constructions of the Markov models, where particular states correspond to selected occurrences of resource sharing by the service components, could be highly useful to the external resource models. Such methods of Markov model construction, however, might not necessarily be employed in the Q-ImPRESS project.

As an observation related to the Markov model construction, the external resource models can potentially identify service components whose resource sharing is of particular interest. This information could be used in Markov model construction to focus on appropriate model states only. Therefore the ability of Markov models to represent resource sharing issues is considered medium.

**Automata Models:** Where combination with the external resource models is concerned, the automata models combine some properties of the earlier models. As with the Markov models, the exact use depends on the mapping of the software behaviour to the states of the automata model. As with the other models, the utility also depends on the output provided by the tools that solve the automata model.

Since the behaviour of the software services and the service components is often described by models that are close to automata models, it is highly likely that there will be a good correspondence between the Q-ImPRESS service model elements and the states of the automata models. This should help relate the output of the performance model to the resource sharing at software component level, which is useful for the external resource models. As Markov models, the ability of automata models to represent resource sharing issues is considered medium.

**Simulation Models:** As outlined, the simulation models are the most general and most versatile performance models considered in Q-ImPRESS. Assuming that the output provided by the simulation will be detailed enough, the simulation models can provide the external resource models with the necessary data on sharing of resources limited only by the level of detail of the simulation

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

models. Of particular interest in the Q-ImPRESS project is a tighter version of coupling between the simulation models and the external resource models, where the data is exchanged between the models continuously, rather than once per model solution cycle. Hence, the ability of simulation models to represent resource sharing issues is considered high.

**Control-Oriented Models:** The aim of control model in Q-ImPRESS is validation and parameters tuning. Current literature proposals are limited to black-box models and it is not easy to derive a correspondence between the internal state representation and resource sharing. Hence, the ability of control models to represent resource sharing issues is considered low. If grey/white box control-oriented models will be available, this issue can be revisited.

## 5.8 Model Selection

After the comparison performed in Section 5.7 and following the suggestions of our project reviewers, we present here the selected models for performance and reliability prediction and analysis.

**Performance:** The prediction and analysis of performance attributes in the Q-ImPRESS project will be based on simulation models because they are the most general and versatile performance models. With respect to the characteristics outlined in Section 5.7, simulation models are *adaptable* since a system change can be implemented by modifying the program description of the added or updated components; have an intermediate level of *cost effectiveness*, and their *composability* and their *scalability* depends on the required level of accuracy.

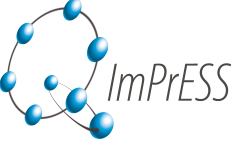
With respect to *resource sharing*, which is a key factor for the Q-ImPRESS project, the simulation models can provide the necessary data on sharing of resources limited only by the level of detail of the simulation models.

**Reliability:** The prediction and analysis of reliability in the Q-ImPRESS project will be based on Markov models because they are general and there exist several analysis tools based on these models. With respect to the characteristics outlined in Section 5.7, Markovian models provide a high level of *adaptability* (since a system update can be modelled simply as a faster service rate or a different probability distribution of the state space), have a high level of *cost effectiveness* while their *composability* and their *scalability* depends on the required level of accuracy

With respect to *resource sharing*, the ability of Markov models to represent resource sharing issues is considered medium since it depends on the correspondence between the states of the Markov models and the behaviour of the software services and the service components.

Model Family	Model	Adaptability	Cost Effectiveness	Composability	Scalability	Resource Sharing
Queueing Models	Bound Product	High	High	High	High	Low
	Non-product LQN	Medium	High	High	Medium/High	Low
		High	High	Medium	Low	Low/Medium
		High	High	High	Medium	Low/Medium
Markov Models	DTMC	High	High	Low	Low	Medium
	CTMC	High	High	Low	Low	Medium
	MDP	High	High	Low	Low	Medium
	SMC	High	High	Low	Low	Medium
Automata Models	TA/PTA/MPTA	High	High	Low	Low	Medium
	Simulation	High	Medium	Medium/High	Low/Medium	High
Simulation Approaches	LTI/LPV	Medium/Low	Low	Low	High	Low

Table 5.2: Quantitative Models Qualitative Comparison.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

## 6 Maintainability Analysis

During their lifetime, software systems need to be frequently adapted to new or changed requirements. The costs of maintenance activities due to requirement changes can be high. The easier the changes can be incorporated, the lower the costs of a system adaptation are.

As already explained in Section 2.3 the *maintainability* of a software system is the *capability of being modified*, hence the capability with which changes can be implemented within a software system. If a specific change can be easily incorporated, the maintainability (with respect to this change) is good. If additional work (like refactoring) is necessary, the maintainability (with respect to this change) is worse. In the worst case, the change can only be incorporated by restructuring and reimplementing large parts of the system, which represents a very bad maintainability with respect to that change. A system can hardly support any kind of change equally well. Hence, it is preferable to focus on changes that are likely to occur and improve the maintainability for these cases. Therefore, we focus on analyzing the maintainability with respect to specific *change requests*. We define a *change request* as a change of requirements, that has to be implemented in a given software system. In Q-ImPrESS we don't focus on corrective maintenance activities, as defined in [102], like bugfixing or exception handling.

The maintainability with respect to a specific change request is measured using the effort necessary to implement the corresponding change. As we specify in Section 2.3, we consider several metric categories which cover maintenance effort and maintainability benefits. The maintenance effort metrics are divided into maintenance workload, maintenance time and maintenance costs.

### 6.1 Running Example

In the next section we introduce the maintainability prediction process. In order to get a better understanding of the approach we illustrate it on a running example. In Q-ImPrESS Deliverable D2.1 in Chapter 5 an example system is already introduced. We use this example and modify it slightly for demonstration purposes.

In Figure 6.1 we show the architecture overview of the system. It is a system with a client-server architecture where a client communicates with a server to retrieve data about users stored in a database. The software system is deployed on several hardware nodes. There are 100 Client components deployed on different machines.

### 6.2 Maintainability Prediction Process

In this section we introduce the maintainability prediction process. The objective of our approach is to enable a software architect to analyse, whether a software architecture incorporates an appropriate maintainability with respect to a *single change request* or a list of *multiple change requests*. More precisely he or she should be enabled to evaluate how a certain architectural alternative

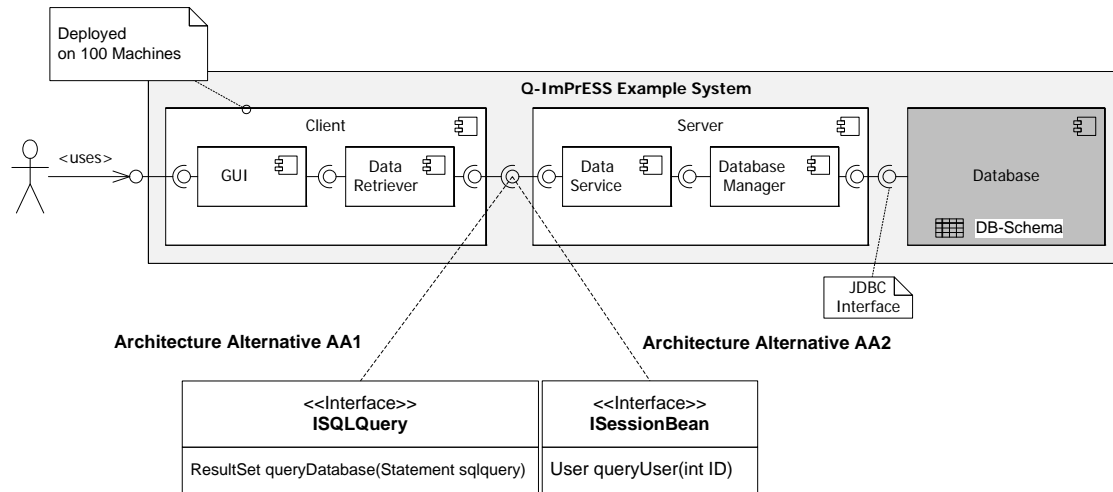


Figure 6.1: Running Example Architecture Overview

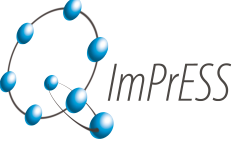
impacts the maintenance effort for implementing the considered change request. By estimating maintenance effort and comparing metric results, the software architect can determine the best architectural alternative with respect to a certain change request.

## 6.2.1 High-level Process Description

For prediction of architecture maintainability we combine a top-down work plan derivation approach with a bottom-up change effort estimation approach. In a traditional process, software architects analyse maintenance efforts manually. In the first phase of the change effort estimation process architects get information about the change request and decide how to react and incorporate the change into the system. More exactly they determine affected parts of the software architecture and split up the work into work packages. Work packages are described in a work plan and consist of work activities. Work activities are assigned to development teams.

In order to estimate the work effort, a software architect inquires feedback from involved development teams about how difficult their share of work will be, how much time it will need, and how much resources will be spent. The idea to ask developers for effort estimates for their work share is in literature known as bottom-up estimation approach ([103]). The advantage of bottom-up estimation is that effort estimates for low-level activities can be made more precisely than for high-level coarse grained change requests. In addition, we assume, that factors influencing personal productivity are implicitly covered by asking developers directly. Thus historical project data for calibration of model parameters is not necessary.

Our change effort estimation approach guides a software architect in breaking down change requests into work activities, creation of a work plan, refinement of work activities into low-level activities and deriving implied work activities (i. e. follow-up activities). As soon as a work plan is derived, the architects have to give bottom-up time estimates for low-level activities. Time

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

estimates finally are aggregated. The process results in an instance of a work plan and a collection of metrics values. In order to get a more precise work plan description and better effort estimates, knowledge of several responsible persons (software architects and software developers) has to be considered. The described process will be integrated into the Q-ImPRESS tool suite.

As can be seen in Figure 6.2 the maintainability prediction process consists of three phases for *Preparation*, *Analysis*, and *Result Interpretation*. We explain these phases and sub-phases below.

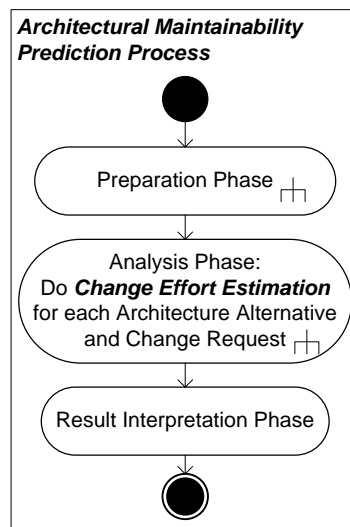


Figure 6.2: Phases of Maintainability Analysis Process

## 6.2.2 Preparation Phase

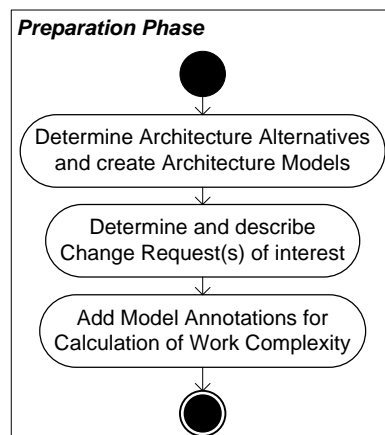
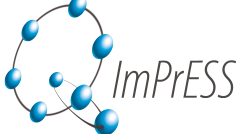


Figure 6.3: Preparation Phase

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

The preparation phase, as shown in Figure 6.3, consists of three phases: 1) Determination and Description of Architecture Alternatives, 2) Determination and Description of Change Requests, and 3) Model Annotations for Calculation of Work Complexity. Each phase is explained in detail below.

**Determine Architecture Alternatives and create Architecture Models** In the first step the software architect determines Architecture Alternatives and sets up a *software architecture description* for each alternative. This means she or he builds a software architecture model (i.e., an instance of the Q-ImPRESS Service Architecture Meta-Model) for each Alternative.

*Example:* According to our running example this means that an instance of the Q-ImPRESS SAMM of the client-server architecture is created. They determine two alternatives. In *Architecture Alternative 1 (AA1)* the clients specify SQL query statements and use Interface ISQLQuery to submit queries to server-side components (Data Service, Database Manager) which delegate queries to the database. In *Architecture Alternative 2 (AA2)* the clients use Interface ISessionBean to query for users with a certain ID. In this alternative components Data Receiver and Data Service deal with business objects, whereas component Database Manager converts queries into SQL statements. The database schema is considered as a Data Type in the architecture model. Since there are two alternatives the architects create two models.

**Determine and Describe Change Request(s)** In a second preparation step the architect describes the considered *change requests*. A description of change request contains 1) *a name*, 2) *an informal description of change cause*, and 3) *a list of already known affected architecture elements*.

*Example:* The architects expect the database schema to be changed during the lifetime of system. They want to analyse the impacts of a change to database schema. Hence the following change request description is given:

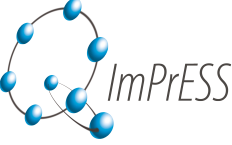
- *Name:* CR-DB-Schema
- *Change Cause:* System needs to be adapted to cope with changed database schema.
- *List of known affected architecture elements:* Data Type "DB-Schema".

**Add Model Annotations for Calculation of Work Complexity** In a third preparation step the architects annotate complexity information to elements in architecture models. Several types of complexity annotations are given in Section 3.3.7. These annotations are used to calculate work complexity (see below).

*Example:* In our running example architects annotate deployment information to client component, i.e. *deployed on 100 machines*.

### 6.2.3 Maintainability Analysis Phase

In the maintainability analysis phase for each architectural alternative and each change request a *Change Effort Estimation* is done. As can be seen in Figure 6.4 the Change Effort Estimation

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

Process consists of four sub-phases: Work plan derivation, Calculation of Work Complexity, Collection of Time Estimates, and Calculation of Cost Estimates.

*Example:* The architects want to analyse which architecture alternative (AA1 or AA2) needs less effort to implement change request *CR-DB-Schema*.

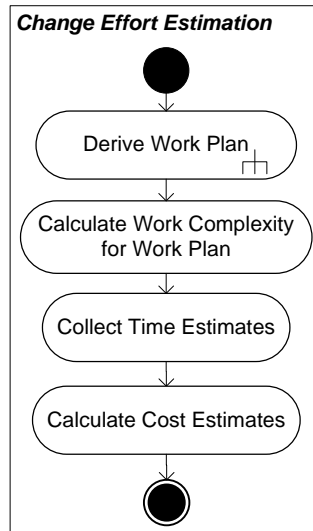


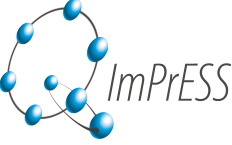
Figure 6.4: Change Effort Estimation Process

**Work Plan Derivation:** In this phase the change request is split up into work activities and a work plan containing these work activities in a structured way is created. For definition and examples of work activity types please see Section 2.3.

In our architecture model there are several types of relations between architectural elements. These relations help to derive a work plan and systematically refine change requests and work plans. In the following, these different types of relations are defined, their role in work plan refinement is discussed and examples are given.

*Include- / contains-relations:* Architectural elements which are contained in each other, are in a contains-relationship. This means, that any change of the inner element implies a change of the outer element. Also, any work activity of the outer element can be refined in a set of work activities of the inner elements. *Examples:* A System consists of Components. A Component has Interface Ports (i.e. Implementation of Interface). An Implementation of Interface contains Implementations of Operations. In our running example a change to the database schema also implies a change to database component. Hence we get another work activity: *Change Component Implementation of "Database" Component*.

*Use- / references-relations:* Architectural elements which use or reference other elements are related by a use-/reference-relation. Such relations are the base for architectural change propagation analyses. Such analyses allow to find potential follow-up changes. This means, that the using entity which references a used entity is potentially affected by a change of the used entity. The

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

direction of the change propagation is reversed to the direction of the use-relation. This implies, that on a model level one needs to navigate the backward direction of use- / reference-relations. *Examples:* An Interface Port references a Definition of Interface. A Definition of Operation uses a Data Type. A Definition of Interface uses transitively a Data Type.

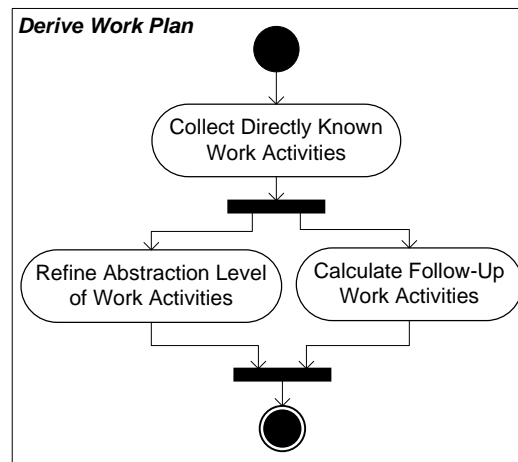
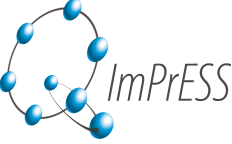


Figure 6.5: Work Plan Derivation

The *work plan derivation phase* itself is a process which consists of several steps. At first our approach helps with mapping the change request to the architecture model and with identifying affected parts in the architecture. A work plan is created for implementing the given change request with respect to the given architectural alternative. A work plan is a list of work activities, where each work activity describes a change in the architecture and can be allocated to a role in the work organization. Therefore model annotations about the work organization (e.g., responsible development teams and developers) is necessary. Each work activity can be clearly mapped to an architectural artefact and a basic activity type (i.e. *Add*, *Change*, *Remove*) can be specified.

In Figure 6.5 we show details of work plan derivation phase. In the following we describe each phase in detail.

- *Collect Directly Known Work Activities:* In this step the architects provide information about architectural work activities which are directly known with respect to the change request description. *Example:* In our running example architects know already that Database Scheme is going to be changed. Hence they specify following directly known work activity: *Change of Data Type "DB-Schema"*.
- *Refine Abstraction Level of Work Activities:* In this step already present work activities in work plan are refined to a lower abstraction level. For example activities on components can be refined to activities on interface ports or operation implementations.
- *Calculate Follow-Up Work Activities:* The work plan should not only cover directly known changes, but also follow-up activities should be detected. In this step we use *use-/reference-relations* in a dependency analysis to calculate potential follow-up activities. *Example:* In

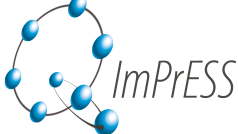
	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

our running example we examine which interfaces use data type "DB-Schema". In alternative AA1 interfaces JDBC and ISQLQuery are affected. Since Client and Server components implement these interfaces they are subject to follow-up activities. In alternative AA2 only component Server is affected.

**Change Effort Estimation** After work plan derivation the change effort estimation process defines the following phases.

- *Calculate Work Complexity for Work Plan:* In this step our approach calculates work complexity information for work activities. For this it uses complexity annotations in the architecture model (see Section 3.3.7). *Example:* Complexity metrics in our example is the number of affected deployments (100 components need to be redeployed). Other examples are the number of involved teams or developers, code/design properties like the number of affected classes or number of affected files.
- *Collect Time Estimates:* In this step the architect provides time estimates for fine-grained activities in the work plan.
- *Calculate Cost Estimates:* In this step cost estimates are calculated by multiplying time estimates with appropriate cost factors.

**Result Interpretation Phase** Finally the process summarises results, compares architecture alternatives with respect to effort estimates from previous phases and presents them to the software architect.

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

## 7 Conclusions

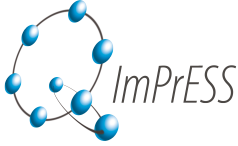
This document introduces the prediction models which support the estimate of the quality attributes considered in Q-ImPRESS and that will be adopted by the project tool suite.

The performance and reliability models have been discussed according to the Model Driven Development framework and their points of strength and weakness, adaptability, cost effectiveness, composability, scalability, and ability to represent resource sharing have been analyzed.

Besides, maintainability quality models and metrics representing the relationship between architectural alternatives and their impact on maintenance efforts have been presented.

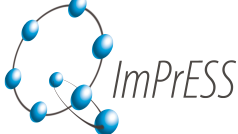
An example of use of the models for the description of the client-server architecture introduced in Q-ImPRESS Deliverable D2.1 has also been discussed. Finally, the interrelationships among the model parameters and the Q-ImPRESS SAMM framework has been also presented.

Future work will focus on the estimate of the QoS metrics of the industrial case studies provided by ABB, Ericsson Nikola Tesla, and Itemis and on the development of the Q-ImPRESS tool chain.

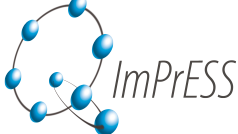
	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

## Bibliography

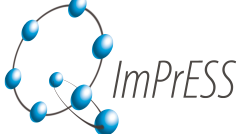
- [1] Atkinson, C., Kuhne, T.: Model-driven development: A metamodeling foundation. *IEEE Software* **20**(5) (2003) 36–41
- [2] Object Management Group: OMG model driven architecture. <http://www.omg.org/mda/> (2006)
- [3] Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.E.: Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.* **1**(1) (2004) 11–33
- [4] Koziolok, H.: Introduction to performance metrics. Springer Verlag (2008)
- [5] Becker, S.: Performance related metrics in iso 9126. Springer Verlag (2008)
- [6] Menasce, D.A., Almeida, V.A.: Capacity Planning for Web Performance: Metrics, Models and Methods. Paperback (2001)
- [7] Metha, V.: A Holistic Solution to the IT Energy Crisis. <http://greenercomputing.com/> (2007)
- [8] Kansal, A., Zhao, F.: Fine-grained energy profiling for power-aware application design. *SIGMETRICS Perform. Eval. Rev.* **36**(2) (2008) 26–31
- [9] Laprie, J.C., ed.: Dependability: Basic concepts and terminology. Springer Verlag (1992)
- [10] Lyu, M., ed.: Handbook of Software Reliability Engineering. IEEE Computer Society Press, New York : McGraw Hill (1996)
- [11] Goševa-Popstojanova, K., Trivedi, K.: Architecture-based approach to reliability assessment of software systems. *Perform. Eval.* **45**(2-3) (2001) 179–204
- [12] ISO/IEC: Software Engineering - Product Quality - Part 1: Quality. ISO/IEC 9126-1:2001(E) (1990)
- [13] Basili, V., Caldeira, G., Rombach, H.D.: Encyclopedia of Software Engineering, chapter The Goal Question Metric - Approach. Wiley (1994)
- [14] Becker, S.: Coupled Model Transformations for QoS Enabled Component-Based Software Design. Volume 1 of The Karlsruhe Series on Software Design and Quality. Universitätsverlag Karlsruhe (2008)
- [15] Tratt, L.: Model transformations and tool integration. *Software and System Modeling* **4**(2) (2005) 112–122

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

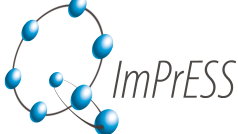
- [16] Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. *IBM Systems Journal* **45**(3) (2006) 621–646
- [17] Mens, T., Gorp, P.V.: A taxonomy of model transformation. *Electr. Notes Theor. Comput. Sci.* **152** (2006) 125–142
- [18] Di Marco, A., Mirandola, R.: Model transformation in software performance engineering. In: *QoSA*. Volume 4214 of *Lecture Notes in Computer Science.*, Springer (2006) 95–110
- [19] Cortellessa, V., Di Marco, A., Inverardi, P.: Integrating performance and reliability analysis in a non-functional mda framework. In: *FASE*. Volume 4422 of *Lecture Notes in Computer Science.*, Springer (2007) 57–71
- [20] OMG, O.M.G.: UML Profile for Schedulability, Performance and Time. <http://www.omg.org/cgi-bin/doc?formal/2005-01-02> (2005)
- [21] OMG, O.M.G.: UML Profile for Modeling and Analysis of Real-Time and Embedded Systems. *ptc/07-08-04* (2007)
- [22] Smith, C.U., Williams, L.G.: *Performance and Scalability of Distributed Software Architectures: an SPE Approach*. Addison Wesley (2002)
- [23] Editors, W.P.: *Wosp : Proceedings of the international workshop on software and performance*. ACM (1998–2007)
- [24] Ardagna, D., Ghezzi, C., Mirandola, R.: Rethinking the use of models in software architecture. In: *QoSA*. *Lecture Notes in Computer Science*, Springer (2008)
- [25] Zheng, T., Woodside, C.M., Litoiu, M.: Performance model estimation and tracking using optimal filters. *IEEE Transactions on Software Engineering* **34**(3) (2008) 391–406
- [26] Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: *Quantitative System Performance: Computer System Analysis Using Queueig Network Models*. Prentice-Hall (1984)
- [27] Bolch, G., Greiner, S., de Meer, H., Trivedi, K.: *Queuing Network and Markov Chains*. John Wiley (1998)
- [28] Baskett, F., Chandy, K.M., Muntz, R.R., Palacios, F.G.: Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM* **22**(2) (1975) 248–260
- [29] Jackson, J.: Jobshop-like queueing systems. *Management Science* **10**(1) (1963) 131–142
- [30] Gordon, W.J., Newell, G.F.: Closed queueing networks with exponential servers. *Operat. Res* **15** (1967) 252–267
- [31] Ardagna, D., Trubian, M., Zhang, L.: SLA based resource allocation policies in autonomic environments. *Journal of Parallel and Distributed Computing* **67**(3) (2007) 259–270

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

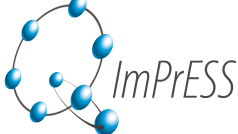
- [32] Urgaonkar, B., Pacifici, G., Shenoy, P.J., Spreitzer, M., Tantawi, A.N.: Analytic modeling of multitier Internet applications. *ACM Transaction on Web* **1**(1) (2007)
- [33] Abrahao, B., Almeida, V., Almeida, J., Zhang, A., Beyer, D., Safai, F.: Self-Adaptive SLA-Driven Capacity Management for Internet Services. In: *Proc. NOMS06*. (2006)
- [34] Casale, G., Muntz, R., Serazzi, G.: Geometric bounds: A noniterative analysis technique for closed queueing networks. *IEEE Trans. Comput.* **57**(6) (2008) 780–794
- [35] Kerola, T.: The composite bound method for computing throughput bounds in multiple class environments. *Performance Evaluation* **6**(1) (1986) 1–9
- [36] Casale, G.: An efficient algorithm for the exact analysis of multiclass queueing networks with large population sizes. In: *SIGMETRICS/Performance*. (2006) 169–180
- [37] Jain, R.: *The Art of Computer Systems Performance Analysis—Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience (1991)
- [38] Papoulis, A., Pillai, S.U.: *Probability, Random Variables, and Stochastic Processes*. 4th ed. edn. McGraw-Hill (2002)
- [39] Kleinrock, L.: *Queueing Systems*. John Wiley and Sons (1975)
- [40] Marzolla, M., Mirandola, R.: Performance prediction of web service workflows. In Overhage, S., Szyperski, C.A., Reussner, R., Stafford, J.A., eds.: *QoSA*. Volume 4880 of *Lecture Notes in Computer Science*., Springer (2007) 127–144
- [41] Cardellini, V., Casalicchio, E., Grassi, V., Mirandola, R.: A framework for optimal service selection in broker-based architectures with multiple QoS classes. In: *Services computing workshops, SCW 2006*, IEEE computer society (2006) 105–112
- [42] Menascé, D.A., Dubey, V.: Utility-based qos brokering in service oriented architectures. In: *ICWS*. (2007) 422–430
- [43] Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey. *IEEE Trans. Software Eng.* **30**(5) (2004) 295–310
- [44] Becker, S., Grunske, L., Mirandola, R., Overhage, S.: Performance prediction of component-based systems - a survey from an engineering perspective. In: *Architecting Systems with Trustworthy Components*. Volume 3938 of *Lecture Notes in Computer Science*., Springer (2006) 169–192
- [45] Di Marco, A., Inverardi, P.: Compositional generation of software architecture performance queueing network models. In: *WICSA*, IEEE Computer Society (2004) 37–46
- [46] Cortellessa, V., Mirandola, R.: PRIMA-UML: a performance validation incremental methodology on early UML diagrams. *Sci. Comput. Program.* **44**(1) (2002) 101–129

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

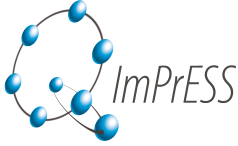
- [47] Balsamo, S., Marzolla, M.: A Simulation-Based Approach to Software Performance Modeling. In: ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering, New York, NY, ACM Press (2003) 363–366
- [48] Gu, G.P., Petriu, D.C.: From uml to lqn by xml algebra-based model transformations. In: WOSP '05: Proceedings of the 5th international workshop on Software and performance, New York, NY, USA, ACM Press (2005) 99–110
- [49] Becker, S., Koziolk, H., Reussner, R.: Model-based performance prediction with the palladio component model. In: WOSP, ACM (2007) 54–65
- [50] Grassi, V., Mirandola, R., Sabetta, A.: Filling the gap between design and performance-reliability models of component-based systems: A model-driven approach. *Journal of Systems and Software* **80**(4) (2007) 528–558
- [51] Woodside, M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Merseguer, J.: Performance by unified model analysis (puma). In: WOSP '05: Proceedings of the 5th international workshop on Software and performance, New York, NY, USA, ACM Press (2005) 1–12
- [52] Rolia, J.A., Sevcik, K.C.: The method of layers. *IEEE Transactions on Software Engineering* **21**(8) (1995) 689–700
- [53] Koziolk, H.: Parameter Dependencies for Reusable Performance Specifications of Software Components. PhD thesis, University of Oldenburg (2008)
- [54] Woodside, M., Petriu, D.C., Siddiqui, K.H.: Performance-related Completions for Software Specifications. In: Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA, ACM (2002) 22–32
- [55] Woodside, C.M., Neilson, J.E., Petriu, D.C., Majumdar, S.: The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions on Computers* **44**(1) (1995) 20–34
- [56] Neilson, J.E.: Parasol: A simulator for distributed and/or parallel systems. Technical Report Technical Report SCS TR-192, School of Computer Science, Carleton University, Ottawa, Ontario, Canada (1991)
- [57] Kwiatkowska, M.: Quantitative verification: Models, techniques and tools. In: Proc. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM Press (2007) 449–458
- [58] Puterman, M.L.: Markov Decision Processes. Wiley (1994)

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

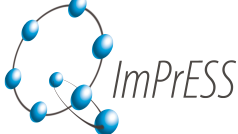
- [59] Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In Bernardo, M., Hillston, J., eds.: *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*. Volume 4486 of LNCS (Tutorial Volume), Springer (2007) 220–270
- [60] Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. *Formal Aspects of Computing* **6**(5) (1994) 512–535
- [61] Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Verifying continuous time markov chains. In: *CAV*. Volume 1102 of *Lecture Notes in Computer Science*, Springer (1996) 269–276
- [62] Clark, A., Gilmore, S., Hillston, J., Tribastone, M.: Stochastic process algebras. In: *SFM*. Volume 4486 of *Lecture Notes in Computer Science*, Springer (2007) 132–179
- [63] Grassi, V.: Architecture-based reliability prediction for service-oriented computing. In: *WADS*. Volume 3549 of *Lecture Notes in Computer Science*, Springer (2004) 279–299
- [64] Sato, N., Trivedi, K.S.: Stochastic modeling of composite web services for closed-form analysis of their performance and reliability bottlenecks. In: *ICSOC*. Volume 4749 of *Lecture Notes in Computer Science*, Springer (2007) 107–118
- [65] Riska, A., Squillante, M., Yu, S.Z., Liu, Z., Zhang, L. In: *Matrix-Analytic Analysis of a MAP/PH/1 Queue Fitted to Web Server Data*. In *Matrix-Analytic Methods: Theory and Applications*, G. Latouche and P. Taylor Ed. World Scientific (2002) 335–356
- [66] Shwartz, A., Weiss, A.: Multiple time scales in markovian ATM models i. formal calculations (1999)
- [67] Grassi, V., Mirandola, R.: Uml modelling and performance analysis of mobile software architectures. In: *UML*. Volume 2185 of *Lecture Notes in Computer Science*, Springer (2001) 209–224
- [68] Grassi, V., Mirandola, R.: Derivation of markov models for effectiveness analysis of adaptable software architectures for mobile computing. *IEEE Trans. Mob. Comput.* **2**(2) (2003) 114–131
- [69] Bernardi, S., Donatelli, S., Merseguer, J.: From uml sequence diagrams and statecharts to analysable petri net models. In: *WOSP '02: Proceedings of the third international workshop on Software and performance*, New York, NY, USA, ACM Press (2002) 35–45
- [70] Bernardi, S., Merseguer, J.: Performance evaluation of uml design with stochastic well-formed nets. *Journal of Systems and Software* **80**(11) (2007) 1843–1865
- [71] Pooley, R.: Software engineering and performance: a road-map. In: *ICSE - Future of SE Track*. (2000) 189–199

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

- [72] Gilmore, S., Kloul, L.: A unified tool for performance modelling and prediction. *Reliability Engineering and System Safety* **89** (2005) 17–32
- [73] Gallotti, S., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Quality prediction of service compositions through probabilistic model checking. In: *QoSA. Lecture Notes in Computer Science*, Springer (2008)
- [74] Alur, R., Dill, D.L.: A theory of timed automata. *Theoretical Computer Science* **126**(2) (1994) 183–235
- [75] Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-Cost Reachability for Priced Timed Automata. In Benedetto, M.D.D., Sangiovanni-Vincentelli, A., eds.: *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control*. Number 2034 in *Lecture Notes in Computer Sciences*, Springer–Verlag (2001) 147–161
- [76] Alur, R.: Optimal paths in weighted timed automata. In: *In HSCC’01: Hybrid Systems: Computation and Control*, Springer (2001) 49–62
- [77] Larsen, K.G., Rasmussen, J.I.: Optimal reachability for multi-priced timed automata. *Theor. Comput. Sci.* **390**(2-3) (2008) 197–213
- [78] Brihaye, T., Bruyere, V., Raskin, J.F.: Model-checking for weighted timed automata. In: *Proc. of FORMATS-FTRTFT’04*. Number 3253 in *Lecture Notes in Computer Science*, Springer–Verlag (2004) 277–292
- [79] Hendriks, M., Verhoef, M.: Timed automata based analysis of embedded system architectures. In: *Proc. of the 20th International Parallel and Distributed Processing Symposium, IPDPS 2006*, IEEE (2006)
- [80] Bordbar, B., Anane, R., Okano, K.: A timed automata approach to qos resolution. *International Journal of Simulation, Systems, Science and Technology* **7**(1) (2006) 46–54
- [81] Dong, J.S., Liu, Y., Sun, J., Zhang, X.: Verification of computation orchestration via timed automata. In: *In Proc. of the 8th Int. Conference on Formal Engineering Methods (ICFEM06)*. Volume 4260 of *LNCS.*, Springer Verlag (2006) 226–245
- [82] Causevic, A., Pettersson, P., Seceleanu, C.: Analyzing resource-usage impact on component-based systems performance and reliability. In: *Proceedings of International Conference on Innovation in Software Engineering - ISE08*, IEEE USA (2008)
- [83] Raimondi, F., Skene, J., Emmerich, W., Wozna, B.: A methodology for online monitoring non-functional specification of web-services. In: *Proc. of the First International Workshop on Property Verification for Software Components and Services (PROVECS’07)*, ETH Technical Report COLOSS Team - University of Nantes (2007) 50–59

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

- [84] Raimondi, F., Skene, J., Emmerich, W.: Efficient online monitoring of web-service slas. In: Proc. of the 16th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2008), ACM (2008)
- [85] Madl, G., Dutt, N., Abdelwadeh, S.: Performance estimation of distributed real-time embedded systems by discrete event simulations. In: Proc. of the 7th ACM & IEEE International Conference On Embedded Software, ACM (2007) 183–192
- [86] Pettersson, P., Seceleanu, C., Vulgarakis, A.: Remes: A resource model for embedded systems. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-232/2008-1-SE, Mälardalen University (2008)
- [87] Sentilles, S., Håkansson, J., Pettersson, P., Crnkovic, I.: Save-ide : An integrated development environment for building predictable component-based embedded systems. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008). (2008)
- [88] Law, A.M., Kelton, W.D.: Simulation Modeling and Analysis. McGrawHill (2000)
- [89] Efron, B., Tibshirani, R., eds.: An Introduction to the Bootstrap. Monographs on Statistics and Applied Probability, Stanford University series (1994)
- [90] Lim, K., Ranganathan, P., Chang, J., Patel, C., Mudge, T., Reinhardt, S.: Understanding and designing new server architectures for emerging warehouse-computing environments. In: International Symposium on Computer Architecture. (2008) 315–326
- [91] Zeng, L., Benatallah, B., Dumas, M., Kalagnamam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. on Software Engineering **30**(5) (2004)
- [92] Ardagna, D., Pernici, B.: Adaptive Service Composition in Flexible Processes. IEEE Transactions on Software Engineering **33**(6) (2007) 369–384
- [93] Abdelzaher, T., Shin, K.G., Bhatti, N.: Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. IEEE Transactions on Parallel and Distributed Systems **15**(2) (2002)
- [94] T. Abdelzaher and, J.S., Lu, C., Zhang, R., Lu, Y.: Feedback Performance Control in Software Services. IEEE Control Systems Magazine **23**(3) (2003) 21–32
- [95] Abdelzaher, T., Lu, Y., Zhang, R., Henriksson, D.: Practical application of control theory to web services. In: Proceedings of the 2004 American Control Conference, Boston, USA. (2004)
- [96] Robertsson, A., Wittenmark, B., Kihl, M., Andersson, M.: Admission control for web server systems - design and experimental evaluation. In: 43rd IEEE Conference on Decision and Control. (2004)

	D3.1: Prediction Models Specification (revised version)	
	Version: 2.0	Last change: September 15, 2009

- [97] Tanelli, M., Ardagna, D., Lovera, M.: LPV model identification for power management of web service systems. In: 2008 IEEE Multi-conference on Systems and Control, San Antonio, USA. To Appear. (2008)
- [98] Tanelli, M., Ardagna, D., Lovera, M.: On- and off-line model identification for power management of Web service systems. In: 47th IEEE Conference on Decision and Control, Mexico. To Appear. (2008)
- [99] Lee, L., Poolla, K.: Identification of linear parameter-varying systems using nonlinear programming. *ASME Journal of Dynamic Systems, Measurement and Control* **121**(1) (1999) 71–78
- [100] Qin, W., Wang, Q.: Modeling and control design for performance management of web servers via an LPV approach. *IEEE Transactions on Control Systems Technology* **15**(2) (2007) 259–275
- [101] Qin, W., Wang, Q.: An LPV approximation for admission control of an internet web server: identification and control. *Control Engineering Practice* **15**(12) (2007) 1457–1467
- [102] ISO/IEC: Software Engineering - Software Life Cycle Processes - Maintenance. ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998 (2006)
- [103] Paulish, D.J., Bass, L.: *Architecture-Centric Software Project Management: A Practical Guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)