Empowering the Service Economy with SLA-aware Infrastructures

Project no. FP7- 216556
Instrument: Integrated Project (IP)
Objective ICT-2007.1.2: Service and Software Architectures, Infrastructures and Engineering

Deliverable D.A6a
Predictable / Manageable Service Engineering Methodology and Prediction Services

Keywords:
QoS Meta-Model, Design-time prediction, Run-time prediction, Manageability Design

Due date of deliverable: 31st May 2009
Actual submission to EC date: 31st May 2009

Start date of project: 1st June 2008
Duration: 36 months

Lead contractor for this deliverable: FZI Forschungszentrum Informatik
Revision: V.0.8 (3rd June 2009)

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

<table>
<thead>
<tr>
<th>Dissemination level</th>
<th>PU</th>
<th>Public</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Document Status

<table>
<thead>
<tr>
<th>Deliverable Status</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliverable Lead</td>
<td>FZI</td>
</tr>
<tr>
<td>Reviewer 1</td>
<td>Alfonso Castro (TID)</td>
</tr>
<tr>
<td>Reviewer 2</td>
<td>Liliana Pasquale (PMI)</td>
</tr>
<tr>
<td>PMT Reviewer</td>
<td>Wolfgang Theilmann (SAP)</td>
</tr>
<tr>
<td>Complete version submitted to reviewers</td>
<td>2009-05-05</td>
</tr>
<tr>
<td>Comments of reviewer 1 received</td>
<td>2009-05-21</td>
</tr>
<tr>
<td>Comments of reviewer 2 received</td>
<td>2009-05-14</td>
</tr>
<tr>
<td>Revised deliverable submitted to PMT</td>
<td>2009-05-25</td>
</tr>
<tr>
<td>PMT Approval</td>
<td>2009-06-02</td>
</tr>
</tbody>
</table>

Contributors

<table>
<thead>
<tr>
<th>Partner</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>FZI</td>
<td>Christof Momm, Jens Happe, Franz Brosch, Samuel Kounev</td>
</tr>
<tr>
<td>CITY</td>
<td>Davide Lorenzoli</td>
</tr>
<tr>
<td>INTEL</td>
<td>Phil Tian, Joe Butler</td>
</tr>
</tbody>
</table>

Notices

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law. Copyright 2009 by the SLA@SOI consortium.

* Other names and brands may be claimed as the property of others.

Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>2009-03-31</td>
<td>Franz Brosch</td>
<td>Outline created</td>
</tr>
<tr>
<td>0.1</td>
<td>2009-04-15</td>
<td>Franz Brosch</td>
<td>Added FZI contributions (run-time prediction, manageability)</td>
</tr>
<tr>
<td>0.2</td>
<td>2009-04-17</td>
<td>Davide Lorenzoli</td>
<td>Added CITY contribution (service level prediction and CITY monitor)</td>
</tr>
<tr>
<td>0.3</td>
<td>2009-04-27</td>
<td>Phil Tian</td>
<td>Added INTEL contributions</td>
</tr>
<tr>
<td>0.4</td>
<td>2009-05-01</td>
<td>Christof Momm, Jens Happe</td>
<td>Completed FZI contributions</td>
</tr>
<tr>
<td>0.5</td>
<td>2009-05-21</td>
<td>Davide Lorenzoli</td>
<td>Addressed review comments for CITY contribution</td>
</tr>
<tr>
<td>0.6</td>
<td>2009-05-22</td>
<td>Christof Momm, Franz Brosch, Jens Happe</td>
<td>Addressed review comments for FZI contribution</td>
</tr>
<tr>
<td>0.7</td>
<td>2009-05-24</td>
<td>Phil Tian</td>
<td>Addressed review comments for INTEL contribution</td>
</tr>
<tr>
<td>0.8</td>
<td>2009-06-03</td>
<td>Jens Happe</td>
<td>Addressed review comments of PMT, finalized deliverable</td>
</tr>
</tbody>
</table>
Executive Summary

This document, SLA@SOI deliverable D.A6a, represents the first of three annual reports documenting the progress of the topics and challenges addressed by Work Package (“WP A6 - Predictable Service Engineering”).

This work package specifically focuses on the engineering process for software systems that considers the additional requirements of an SLA-driven management at run-time. To this end, A6 leverages model-based techniques both from the engineering and the system management field to provide performance prediction for complex SOA stacks at design-time and system management at run-time.

In this regard, WP A6 has divided the overall goal into a number of distinct areas. For each of these areas, this deliverable document provides a description of the activities, and illustrates the progress and results achieved during the course of the first year.

Design-time Prediction

Design-time prediction comprises model-driven techniques, which enable service providers to determine quality of service properties they can guarantee on the provided services before their actual deployment. A design-time QoS metamodel allows capturing a performance (completion time / throughput) abstraction of the (service-oriented) system. Following the model-driven approach, transformations generate simulation models that can be used for performance prediction. The simulation is offered in terms of a prediction service that can be called by the service provider. So far, we support estimates of completion times, throughput, and resource utilisation.

Run-time Prediction

Run-time prediction targets the dynamic re-provisioning and migration of services and computing resources. At infrastructure level, run-time prediction supports the dynamic adjustment of resource allocation. This can save operational and utility cost by suspending unutilized resources. On the application level, run-time prediction can be used to proactively avoid SLA violations. Based on the recent behaviour of a service, we can predict the probability of an SLA violation in the near future and start countermeasures if appropriate.

Regarding infrastructure level prediction, this document presents three improved algorithms based on the existing work, which can realize higher prediction accuracy and be capable of long time step prediction. Experimental results already show the effectiveness of the improvements in this early development stage.

Concerning application-level prediction, the deliverable first introduces the theoretic foundation and application of monitoring policies. Afterwards, it describes implementation and performance evaluation of the corresponding monitoring bus, which represents the basis for the run-time prediction service we target in Y2.
Manageability Design

Manageability Design targets to support additional development steps required for implementing monitoring and control capabilities that are necessary for an SLA-driven management. These additional development steps include (i) the identification of runtime measurements necessary for monitoring the performance as well as control operations used for adjusting the performance at runtime; (ii) the implementation of a corresponding instrumentation of the service components in terms of sensors and effectors; (iii) the configuration of the SLA management framework (at least with SLOs, metrics as well as data computation and monitoring/control rules).

In this document, we introduce an engineering methodology for manageable service components and, as a first step towards complete support for this development process, a manageability configuration metamodel, which supports the configuration of monitoring capabilities for system designs based on the SCA assembly model. This constitutes the starting point for the configuration of the manageability infrastructure with scenario-specific monitoring requirements. So far, we focus on monitoring capabilities, while we plan to provide control capabilities in year 2. Similarly, we also foresee to extend the design and implementation of the corresponding manageability infrastructure with control capabilities.
# Table of Contents

1 Introduction ............................................................................................................. 9

2 Design-time Prediction .............................................................................................. 11
  2.1 Overview ............................................................................................................. 11
  2.2 Preliminaries ...................................................................................................... 12
  2.3 State of the Art .................................................................................................. 14
    2.3.1 Model-based Performance Prediction ......................................................... 14
    2.3.2 Performance Prediction Techniques for Component-based Systems ........ 14
    2.3.3 Measurement-based Approaches to Performance Prediction ................. 16
    2.3.4 Performance Prediction Techniques for Web Services and Service-oriented Architectures ................................................................. 16
  2.4 Predictable Service Engineering Methodology ................................................... 18
    2.4.1 Roles ........................................................................................................... 18
    2.4.2 Prediction Service Workflow ...................................................................... 19
  2.5 QoS Metamodel for Performance Prediction ...................................................... 20
    2.5.1 Components and Interfaces ....................................................................... 21
    2.5.2 Composite Components and Systems ....................................................... 22
    2.5.3 Component Behaviour ............................................................................... 23
    2.5.4 Resource Allocation .................................................................................. 25
    2.5.5 Usage Profiles ........................................................................................... 26
  2.6 Design-time Prediction Service ............................................................................ 27
    2.6.1 Prediction Process ..................................................................................... 28
    2.6.2 Input Data Format ...................................................................................... 29
    2.6.3 Prediction Result Data Format ................................................................... 30
    2.6.4 Implementation ......................................................................................... 34
  2.7 Demonstrator based on ORC .............................................................................. 34
    2.7.1 Service Component Model ........................................................................ 35
    2.7.2 Infrastructure Model .................................................................................. 39
    2.7.3 Allocation Model ....................................................................................... 40
    2.7.4 Usage Model ............................................................................................. 41
  2.8 Summary ............................................................................................................ 43

3 Run-time Prediction ................................................................................................ 44
  3.1 Introduction ....................................................................................................... 44
  3.2 State of the Art .................................................................................................. 45
    3.2.1 Infrastructure level Prediction .................................................................... 45
    3.2.2 Service-Level Prediction .......................................................................... 47
  3.3 Infrastructure level Prediction (INTEL) ............................................................ 50
    3.3.1 Technical evaluation ............................................................................... 50
    3.3.2 Improvement and experimental analysis ............................................... 54
  3.4 Service level Prediction (CITY) .......................................................................... 64
    3.4.1 Introduction ............................................................................................... 64
    3.4.2 Monitoring policies ................................................................................... 65
    3.4.3 Prediction .................................................................................................. 66
    3.4.4 Implementation ........................................................................................ 71
    3.4.5 Evaluation ............................................................................................... 73
  3.5 Challenges/Conclusions .................................................................................... 73

4 Manageability Design .............................................................................................. 75
  4.1 Overview ........................................................................................................... 75
  4.2 State of the Art .................................................................................................. 76
    4.2.1 Model-Driven Monitoring Approaches ..................................................... 76
  4.3 Manageable Service Engineering Methodology .............................................. 77
  4.4 Manageability Configuration Metamodel for SCA .......................................... 79
    4.4.1 Manageability Configuration Metamodel ................................................. 82
4.4.2 Sample Manageability Model .......................................................... 85
4.5 Manageability Design for ORC / Ad Hoc Demonstrator .................. 87
  4.5.1 Scenario and SCA-based ORC design ......................................... 87
  4.5.2 Service Development/Design .................................................. 89
  4.5.3 Service Provisioning ............................................................ 92
  4.5.4 Service Operation ............................................................... 93
5 Conclusions ..................................................................................... 94
6 References ...................................................................................... 96
Glossary ............................................................................................ 105
Appendix A: Abbreviations ................................................................. 107
### Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overview to WP A6 Contributions and their Interrelations</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Abstract workflow of the prediction service</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>Components, Interfaces, and Roles</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>Interface Signatures</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>Composite Components and Systems</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>Service Effect Specifications</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>Resource Demanding and External Actions</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>Overview of Actions</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>Resource Environments</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>System Allocation</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>Scenario Behaviours</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>Usage Scenario Workloads</td>
<td>27</td>
</tr>
<tr>
<td>13</td>
<td>Design-time Prediction Process</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>Prediction Service Input Model</td>
<td>29</td>
</tr>
<tr>
<td>15</td>
<td>Prediction Input Example</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>Deployment Options Model for Prediction</td>
<td>31</td>
</tr>
<tr>
<td>17</td>
<td>Deployment Options Example</td>
<td>31</td>
</tr>
<tr>
<td>18</td>
<td>Prediction Service Result Model</td>
<td>32</td>
</tr>
<tr>
<td>19</td>
<td>Prediction Service Result Example</td>
<td>33</td>
</tr>
<tr>
<td>20</td>
<td>SLA@SOI Framework and Design-time Prediction</td>
<td>34</td>
</tr>
<tr>
<td>21</td>
<td>Open Reference Case Repository</td>
<td>36</td>
</tr>
<tr>
<td>22</td>
<td>Payment Service Component</td>
<td>36</td>
</tr>
<tr>
<td>23</td>
<td>Inventory Service Interface</td>
<td>37</td>
</tr>
<tr>
<td>24</td>
<td>Open Reference Case System</td>
<td>37</td>
</tr>
<tr>
<td>25</td>
<td>Payment Service RDSEFF</td>
<td>38</td>
</tr>
<tr>
<td>26</td>
<td>Book Sale RDSEFF</td>
<td>39</td>
</tr>
<tr>
<td>27</td>
<td>ORC Infrastructure Model</td>
<td>40</td>
</tr>
<tr>
<td>28</td>
<td>Distributed ORC Allocation</td>
<td>41</td>
</tr>
<tr>
<td>29</td>
<td>ORC Usage Model</td>
<td>42</td>
</tr>
<tr>
<td>30</td>
<td>Book Sale Usage Scenario</td>
<td>42</td>
</tr>
<tr>
<td>31</td>
<td>Flow Chart of Improved MPI</td>
<td>55</td>
</tr>
<tr>
<td>32</td>
<td>Experimental Results of MPI on 5S Dataset</td>
<td>56</td>
</tr>
<tr>
<td>33</td>
<td>Experimental Results of Improved MPI on 5S Dataset</td>
<td>56</td>
</tr>
<tr>
<td>34</td>
<td>Experimental Results of MPI on 10S Dataset</td>
<td>56</td>
</tr>
<tr>
<td>35</td>
<td>Experimental Results of Improved MPI on 10S Dataset</td>
<td>57</td>
</tr>
<tr>
<td>36</td>
<td>Experimental Results of Improved MPI for 10 Step Prediction on Mem_Free 240S</td>
<td>57</td>
</tr>
<tr>
<td>37</td>
<td>Experimental Results of Improved MPI for 10 Step Prediction on Pack_In 240S</td>
<td>58</td>
</tr>
<tr>
<td>38</td>
<td>Experimental Results of Improved MPI for 10 Step Prediction on CPU 5S</td>
<td>58</td>
</tr>
<tr>
<td>39</td>
<td>Flow Chart of Improved Fuzzy Logic-Based Approach</td>
<td>59</td>
</tr>
<tr>
<td>40</td>
<td>Experimental Results of Improved PLC on MEM_Free 240S</td>
<td>60</td>
</tr>
<tr>
<td>41</td>
<td>Experimental Results of Improved PLC on Byte_In 240S</td>
<td>60</td>
</tr>
<tr>
<td>42</td>
<td>Experimental Results of Improved PLC on CPU_Idle 30S</td>
<td>60</td>
</tr>
<tr>
<td>43</td>
<td>Experimental Results of Improved PLC for 10 Step Prediction on Mem_Free 240S</td>
<td>61</td>
</tr>
<tr>
<td>44</td>
<td>Experimental Results of Improved PLC for 10 Step Prediction on Byte_In 240S</td>
<td>61</td>
</tr>
<tr>
<td>45</td>
<td>Experimental Results of Improved PLC for 10 Step Prediction on CPU_Idle 30S</td>
<td>61</td>
</tr>
</tbody>
</table>
Figure 47: Experimental Comparison of Improved PP and Original Algorithm on Byte_In ................................................................. 63
Figure 48: Experimental Comparison of Improved PP and Original Algorithm on a Synthesised Dataset ........................................ 63
Figure 49: Experimental Results of Improved PP for 100 Step Prediction on Byte_In .................................................................. 64
Figure 50: Experimental Results of Improved PP for 100 Step Prediction on a Synthesised Dataset .................................................. 64
Figure 51: The formal definition of a formula in EBNF .................................................. 66
Figure 52: Belief graph for Rule-2 .......................................................... 69
Figure 53: Algorithm for computing overall threat belief ........................................ 71
Figure 54: Monitor high level architecture ................................................... 71
Figure 55: CITY monitoring framework APIs .............................................. 72
Figure 56: Monitoring interface .................................................................. 73
Figure 57: Solution building blocks ................................................................ 77
Figure 58: Overview to Engineering Methodology for Managable Services ...... 78
Figure 59: Sample Service Transactions in SCA-based applications.................. 80
Figure 60: Usage of Manageability Configuration Metamodel ........................ 81
Figure 61: Fundamental Elements of the Metamodel ...................................... 81
Figure 62: Manageability Configuration Meta-Model for SCA ......................... 83
Figure 63: Using the Manageability Configuration Metamodel ......................... 85
Figure 64: Abstract example of an SCA-based Application Design ................. 86
Figure 65: Sample Manageability Model ..................................................... 87
Figure 66: SCA-based Design of the ORC Retail Solution .............................. 88
Figure 67: ORC Deployment Options .......................................................... 89
Figure 68: SCA-based Design of the Payment Service .................................... 90
Figure 69: Sample SLAs for Payment Service ............................................. 90
Figure 70: Manageable Service-oriented Retail Solution ................................ 91
Figure 71: Manageability Model for ORC .................................................. 92
Figure 72: Configuration of Static Management Information for ORC .......... 93
1 Introduction

SLA@SOI aims at developing a comprehensive multi-layer SLA management framework for service-oriented software systems. Across all layers of an IT stack and across the various stakeholder perspectives the framework will support the provision of software systems or parts of them as services with a contractually fixed quality based on service-level agreements. Noteworthy, quality attributes span across multiple non-functional domains such as security, performance, availability, and reliability.

The work package A6 ("Predictable Systems Engineering") specifically focuses on the engineering process for software systems that are subject to an SLA-driven management at runtime. In such a scenario, traditional component-based and service-oriented development techniques are not sufficient any more. Depending on which Quality-of-Service (QoS) guarantees a service provider wants to offer, the responsible software provider has to consider additional predictability and manageability requirements while developing the necessary service components. To this end, A6 leverages model-based techniques both from the engineering and the system management to provide performance predictions for complex SOA stacks at design-time and flexible system management at run-time.

Objectives of A6 are classified in three major areas:

- **Design-time prediction**, which covers the challenge of predicting the actual quality of service already at design-time, without having to deploy and operate the service components
- **Run-time prediction**, which addresses the question of how to predict SLA violations and how to optimize infrastructure usage at run-time
- **Manageability design**, which is concerned with supporting management capabilities required for an SLA-driven management

Figure 1 provides an overview to the various WP A6 contributions and illustrates their interplay in terms of a topic map. This includes major interrelations to other work packages. Thereby, the question marks indicate non-trivial interrelations between different areas, which so far have not been fully understood. The clarification of these dependencies and (if applicable) the integration of the different approaches will be addressed during the course of the project.

The engineering methodologies for predictable and manageable service components both leverage model-driven techniques. This in the first place allows for a clear separation of concerns and a reduction of complexity by abstracting from implementation details, like choice of technological platform, or management standards, but also concrete prediction models. Accordingly, in both cases the particular implementation (which might also be the configuration of a framework) is generated from tailored design-models by applying appropriate transformations. This on the one hand comprises a design-time prediction service and on the other hand components of the manageability infrastructure targeted by WP A3. The manageability infrastructure generated in the latter case offers a manageability interface (or service), which is used by the SLA management framework to monitor and control the managed resources. As one major SLA management feature, we target the ability to predict SLA violations at run-time, which for instance is used by SLA adjustment and autonomic service management to initiate counter measures in a timely manner. Within this context, WP A6 accounts for developing cutting-edge run-time prediction models and algorithms. Here, we distinguish between service level and infrastructure
level prediction. Both approaches will mainly operate on basis of run-time measurements. In case of the service level prediction, this information is accessible through the manageability interface and the SLA@SOI monitoring bus, whereas the infrastructure level prediction uses the infrastructure management infrastructure provided by WP A4. Design-time prediction may in addition to this use the manageability infrastructure for calibrating and validating the prediction models. Within scope of this conceptual topic map, major open issues are the interrelation between run-time prediction on service and infrastructure level as well as the general question, to which degree it is possible to integrate design-time and run-time prediction approaches.

Figure 1: Overview to WP A6 Contributions and their Interrelations

Besides these integration issues, WP A6 pursues the following concrete objectives. The area of Design-time prediction targets to
- Create a domain-specific metamodel that allows to model abstractions on the performance of a component- based system. Our metamodel must include all information required for predicting the system’s QoS.
- Provide a design-time prediction service supporting the prediction of a system’s QoS properties without having to fully implement and execute the system.

Concerning Run-time prediction the major objective is to
- Provide prediction models and services for run-time system management. They must support a short-term prediction of QoS properties either for software services or at the infrastructure level, where only partial knowledge of the software stack is available.

Regarding the manageability design, A6 particularly plans to
- Extend existing component models to support the configuration of management capabilities required for run-time system management.

The structure of this deliverable document follows the abovementioned classification. For each of these areas this document provides a description of the activities, progress and achievements during the course of Y1. The overall direction of Y1 activities was to build an “ad-hoc framework” which satisfies the requirements of the “Open Reference Case – ORC” and the “Ad-hoc demonstrator”, as described in deliverable D.B2a. For this purpose, the ORC served as driving scenario during the development of runtime prediction, design-
time prediction and the manageability framework in A6. With its fixed set of deployment options and its specific services, the ORC enabled the development of a prototype that touches all relevant aspects of A6.

This document is structured as follows. Chapter 2 presents achievements regarding design-time prediction, including an overview of the predictable service engineering methodology adopted and a description of its major parts: The QoS metamodel and the prediction service. Chapter 3 shifts focus to run-time prediction, where actual measurements are available, but only a small time frame for prediction exists. For both the software-level and the infrastructure level, this chapter introduces the main achievements in Y1, including first designs and implementations of formal languages, run-time monitors and prediction algorithms. Chapter 4 is concerned with design-time issues related to monitoring and control capabilities (manageability) required for an SLA-driven run-time system management. After giving an overview of the engineering methodology for manageable service components, the deliverable introduces a metamodel for configuring the required monitoring capabilities.

## 2 Design-time Prediction

### 2.1 Overview

In the vision of SLA@SOI, service providers can give some guarantee on specific quality of service properties for the services they offer. Such guarantees concern, for example, the response time and throughput of a service. However, the quality of a service depends on all roles (customers, service providers, software providers, and infrastructure providers) involved in the service life-cycle. Therefore, strict contracts, called Service Level Agreements (SLAs), between the roles are an essential part of SLA@SOI. SLAs determine the quality properties that can be expected from different parties. For example, an SLA captures the response time of a service for a specific customer. SLAs are negotiated before a service is deployed (and may be renegotiated at runtime). For this purpose, service providers need to create general offers for services (i.e., some predefined and unsigned SLA) and react on individual requests of customers. General and individual offers have to be based on sound data to ensure that services can be offered to the conditions that have been negotiated. Service providers have to know in advance which infrastructure resources and external services are required in order to fulfil a customer’s request. This knowledge allows them to estimate costs and to acquire necessary resources. Furthermore, customers need detailed information on the quality of a service. This information helps them, for example, to identify the optimal trade-off between costs and offered quality. Customers may want to choose among multiple offers that provide the same functionality but differ with respect to quality and cost.

The prediction service supports Negotiation (Task A5.3) and Translation (Task A5.4) to determine the quality attributes of services before their deployment. In its current state, it provides realistic estimates of response times, throughput, and resource utilisation. This information can be used to plan infrastructure capacities necessary to fulfil Service-Level Objectives (SLOs, cf. Deliverable D.A1a, Section 3.1). Furthermore, the prediction results support the definition of customised SLAs between service providers and service customers. For this purpose, the prediction service evaluates the performance properties of multiple alternatives that realise a specific service. Each alternative can differ with respect to the allocation of service components to infrastructure, with respect to size and number of virtual machines, or with respect to the respective service components.
used. The prediction service automatically generates different alternatives for a set of virtual machines offered by an infrastructure provider. The prediction results for all alternatives can be used by Negotiation and Translation to determine the optimal offer for a specific customer.

In order to provide such early estimates, the prediction service has to overcome several challenges specific to the service-oriented world. First, it has to retrieve and integrate the data necessary for performance prediction. In a service-oriented environment this information is distributed among the different roles involved in the service life-cycle. Second, the prediction service has to take into account mutual dependencies of services that share physical and/or logical resources. Such dependencies are one of the critical factors determining a service’s quality.

In order to support performance prediction in a service-oriented environment, we grouped needed information into domain-specific modelling languages for service customers, software providers, service providers, and Infrastructure providers. Each model contains the information that can be specified by that particular role. All domain-specific models are integrated. Furthermore, domain-specific models can be extracted automatically from information available in the Software Landscape and the SLA Template Registry. The automatic extraction of information relevant for prediction keeps the modelling effort at a minimum. Based on the information specified in the domain-specific models the prediction service generates different design alternatives for a specific service request. For each alternative, response times, throughput, and resource utilisation are predicted. The results are aggregated and returned to the Negotiation or Translation module which uses them to determine the best solution for a specific request issued by a service customer.

In the context of SLA@SOI, we evaluated the applicability of the prediction service on the Open Reference Case (cf. Deliverable D.B2a, Section 2). We specified the domain-specific models that the prediction service integrates to generate different deployment options. Furthermore, the prediction service evaluates the performance properties for all options, aggregates the results and returns them to the Negotiation and/or Translation module.

In Section 2.2, we describe the preliminaries and the application context of the prediction service. In Section 2.3, we illustrated other works we retain related to and/or relevant for our research. Section 2.4 presents a broad overview of the prediction methodology. In Section 2.5, we introduce the metamodels used in the context of the prediction service. This includes a description of the metamodels for the prediction service. Section 2.6 presents the design of the prediction service’s implementation including interactions with other modules of SLA@SOI. In Section 2.7, we evaluate the prediction service on the Open Reference Case. We describe the performance models for the ORC and give exemplary prediction results. Finally, Section 2.8 concludes this chapter.

### 2.2 Preliminaries

In the following paragraphs, we discuss the application context of the prediction service. We describe the constraints that have to be taken into account for software performance prediction. Furthermore, we explain how the results of the prediction service can be interpreted and used for SLA translation and negotiation.
In order to be meaningful, performance prediction approaches have to consider several constraints that are imposed by the real world. These constraints are essential to retrieve realistic predictions but limit the flexibility with which prediction can be used. For example the performance of a service always depends on its underlying infrastructure, the external services used, the service's usage, and – last but not least – on other services using the same resources. Having this in mind, it is obvious that prediction can only be applied successfully if all this information is available. Even though such a holistic view on the system is not desirable in a service-oriented world, it cannot be omitted for performance prediction. We call pre- and post-conditions the statements determining how predictions can be used and how its results can be interpreted.

Pre- and post-conditions are described as follows:

**Pre-Condition:** For performance (and reliability) prediction, we require a complete view of (a distinct part of) a system. This view has to include information about the usage of the services under study (arrival rate and input parameters), their realisation by service components, the (virtualised) infrastructure, and the allocation of service components to (virtual) machines. Only if all this information is available, performance (and reliability) prediction can be applied.

Services often share hardware and software resources, e.g., they are deployed on the same (virtual) machine. In such settings, the load imposed by one service influences the performance of the other ones. In order to deal with such interdependencies, performance predictions rely on a complete model of (a distinct part of) a system. Performance prediction approaches use information available at lower levels (e.g., infrastructure) to derive performance properties of higher-levels (e.g., service completion times). Therefore, the prediction service can be considered as bottom-up translation [1].

In order to support SLA-translation and negotiation in a reasonable way, the prediction service determines the performance of different predefined contracts the provider can offers to its potential customers. Currently, the variation points for an alternative are the infrastructure size (number and capacity of virtual machines) and the allocation of service components to virtual machines. The prediction service evaluates the performance properties of the services for each alternative. In order to improve the prediction service in the later stages of this project, the search of an optimal solution can be supported by meta-heuristics and search techniques, such as those described in [2].

**Post-Condition:** The prediction service evaluates dependencies between services sharing resources, influences of different infrastructure capacities, and the usage of a specific service on software performance. The result of the prediction service is a discrete set of possible solutions for a request issued by a customer. The Result Model (cf. Section 2.6.3) contains the expected performance properties of all services considered in the prediction for each deployment and infrastructure alternative.

The Result Model also captures dependencies among services. For example, services that are part of the same appliance share hardware and software resources. Therefore, their performance properties can only be considered as a whole. Services deployed in different appliances may be considered independently. It might be possible to change their VM sizes without affecting other services. However, care has to be taken here since other dependencies might exist that are not captured by grouping services into appliances. For example, a shared database can lead to mutual performance influences of services deployed in different appliances.
2.3 State of the Art

2.3.1 Model-based Performance Prediction

Over the last fifteen years many approaches have been proposed for integrating performance evaluation and prediction techniques into the software engineering process. Efforts were initiated with Smith’s seminal work pioneered under the name of Software Performance Engineering (SPE) [6]. Since then a number of metamodels for describing performance-related aspects [7] have been developed by the SPE community, the most prominent being the UML SPT profile [8] and its successor the UML MARTE profile [9], both of which are extensions of UML as the de facto standard modeling language for software architectures. Other proposed metamodels include SPE-MM [10], CSM [11, 12] and KLAPER [13]. The common goal of these efforts is to enable the automated transformation of design-oriented software models into analysis-oriented performance models making it possible to predict the system performance. A recent survey of model-based performance prediction techniques was published in [14]. Furthermore other techniques exploit the combination of different performance including standard queueing networks [15, 16, 17, 18], extended queueing networks [19, 20, 13], layered queueing networks [22, 23], stochastic Petri nets [24, 25], queueing Petri nets [26, 27], stochastic process algebras [28], Markov chains [21], statistical regression models [29, 30] and general simulation models [31]. In recent years, with the increasing success of component-based software engineering (CBSE), the performance evaluation community has focused on adapting and extending conventional SPE techniques to support component-based systems. Since component-oriented technologies are used as foundation for building modern SOA applications, their performance is essential for managing Quality-Of-Service (QoS) in SOA environments. Techniques for component-based performance prediction [32] are surveyed in detail in the next section.

2.3.2 Performance Prediction Techniques for Component-based Systems

A number of performance prediction methodologies and tools for component-based systems have been proposed [32]. One of the first attempts towards compositional performance analysis of component-based systems can be found in [33]. The authors argue that performance-related properties must be integrated with component models in addition to descriptions of their functional behavior. The author sketch an approach mainly based on formal techniques. They admit that an engineering approach to predictability is a necessary ingredient to ensure predictable components. Another approach to integrate component technology with analysis models was presented in [34, 35]. The authors describe a prediction-enabled component technology called PECT which aims to enable the prediction of assembly-level system properties based on certified component descriptions. This is achieved by imposing a set of restrictions on component designs and implementations which permit compositional analysis methods to be applied to component assemblies. Even if it represents a first step towards component-based performance engineering, the proposed component specifications and analysis methods are rather simple and do not cover all the information needed for accurate performance prediction.

In [36, 37], a compositional methodology for automated performance analysis of component-based systems, called CB-SPE (Component-Based SPE) is presented. CB-SPE is a generalization of the conventional SPE method [38, 39, 40] and it
adapts its concepts and methodology to component-based architectures. Annotations based on the UML SPT profile [41] are used to augment component specifications with performance-related properties depending on platform parameters. A disadvantage of this approach is that it does not support nesting component sub-models and each time a component is replaced with another one the modeling steps have to be repeated. In [42, 43], the authors propose a language called Component-Based Modeling Language (CBML) based on XML and UML2, which is designed to describe layered queueing models with embedded components. Component sub-models support parameterization, however, no explicit context model is defined for capturing variations in input parameters, deployment and configuration. A model assembler tool generates a solvable layered queueing model from a system definition with component bindings. Hierarchical component specifications with multiple levels of nesting are supported.

In [44, 45], a compositional method for performance analysis of component-based systems is proposed in which the effect of input parameters on component behavior and resource usage is modeled explicitly. Application developers can explore possible execution scenarios with different parameter initializations and find the worst-case scenarios where the predicted performance does not satisfy the requirements. The proposed approach, however, is still rather limited since it does not consider stochastic parameter specifications and does not provide a comprehensive component context model taking into account system configuration and deployment aspects.

In [46], it is argued that current component models do not sufficiently reflect the influence of the deployment context on the component behavior. For this reason, the authors propose to add an explicit context model as part of the component specification. The context model captures the dependencies of functional and extra-functional properties on the component’s connections to other components, on the execution environment, on the allocated hardware and software resources, and on the usage profile. Context dependencies are specified by means of parametric contracts which can be considered as an adaptation mechanism, modifying a component’s "provides-and-requires-interfaces" depending on its context [47, 48]. A modeling notation based on extensions to the UML SPT profile [41] is proposed in [49] allowing component developers to explicitly specify the influence of parameters on the component resource demands as well as on their usage of external services. A parametric contract in the form of a so-called service effect specification can be defined for those component services describing their behavior and control flow in an abstract and parametric manner. Parameters can be characterized through a probability distribution over their value (for primitive types), sub-types (for Object types), the number of elements (for collection types), the byte-size (for binary data) or the parameter structure (for composite types). Annotated UML models are transformed to stochastic regular expressions that are used for performance analysis. The authors show how their approach can be integrated into the CBSE process model by Cheesman and Daniels [50] to explicitly include early-cycle model-based performance analysis [51].

In [52], the approach described above is extended by introducing constructs for modeling internal parallelism inside a component. Service effect specifications now support forking of threads and can be parameterized in terms of the number of CPUs and CPU cores. Stochastic regular expressions extended with an operator for parallelism are used for performance analysis. The prediction accuracy, however, is still rather limited. A major limitation is that it does not consider resource contention among multiple concurrent requests. The above works were combined in the Palladio Component Model (PCM) [53], a metamodel for
specifying performance-relevant information in component-based architectures. PCM is designed with the explicit capability of dividing the model artifacts among the developer roles involved in the CBSE process.

2.3.3 Measurement-based Approaches to Performance Prediction

In [54, 55], it is argued that while the use of performance models in the early stages of system development could help the identification of bottlenecks in the system design, models often fail to capture important execution aspects that can only be determined at run-time. The authors propose a performance analysis method based on early cycle empirical testing. Application-specific performance tests are derived from architecture designs and executed on the middleware platforms chosen for building the system. The approach, however, provides limited automation and does not consider integrating empirical measurements with performance models. Moreover, since “fake” components are used in place of unavailable system components, the practicality and reliability of the approach is highly questionable.

In [56, 57], the authors use statistical regression techniques to model the relationship between performance-relevant parameters of software components (e.g., use of service calls, input parameters) and their resource demands. Regression models are extracted by running a set of relevant use cases on the components of interest and measuring their performance. The aim is to reuse models, fitted to measurements of existing components, to assess the performance of adapted versions of these components. The proposed method, however, can only be applied if the adapted components are sufficiently “similar” to existing ones.

Another measurement-based approach is proposed in [59, 60]. A simple benchmark is used to extract a performance profile of the underlying component-based middleware (e.g., Java EE or .NET) used to build an application. A generic performance model is then constructed to reflect the interactions between the key performance factors of applications deployed on a selected platform. A significant drawback of this approach is that application-specific behavior is not modeled explicitly and only very rough estimates of the system behavior can be obtained.

In [61, 62], the authors describe a technique to perform automated analysis of system architectures and extraction of performance models based on traces obtained during operation. A limitation is that components having internal parallelism (supporting forking and joining of the execution flow) are not supported. Furthermore, a number of requirements are placed on the tracing tools which make it difficult to apply the technique in large distributed systems spanning multiple administrative domains.

2.3.4 Performance Prediction Techniques for Web Services and Service-oriented Architectures

A number of approaches for introducing QoS support in Web services have been proposed, for example [63, 64, 65, 66, 67]. In [63], an extension of UDDI enabling Web service discovery based on QoS requirements was presented. Similarly, in [64, 65], extensions of WSDL-based Web service descriptions were introduced to support QoS-related information. These studies, however, do not address the issue of how the service provider guarantees its QoS claims. An
approach to dynamically select a service provider that best meets the consumer’s needs is presented in [66]. An agent framework coupled with a QoS ontology is used, however, the framework does not support the ability to reserve the resources required for providing a selected QoS, and therefore again no QoS guarantees are provided.

A number of approaches [63, 64, 65, 66, 67] for introducing QoS support in Web services have been proposed. In [63], an extension of UDDI enabling Web service discovery based on QoS requirements was presented. Similarly, in [64, 65], extensions of WSDL-based Web service descriptions were introduced to support QoS-related information. These studies, however, do not address the issue of how the service provider guarantees its QoS claims. While an approach to dynamically select a service provider that best meets the consumer’s needs is presented in [66], it adopts an agent framework coupled with a QoS ontology. However, since it does not support the ability to reserve necessary resources required for providing a selected QoS, it is not able to give guarantees on provided QoS.

In [65], a lightweight extension to WSDL (Web Service Description Language) introducing QoS characteristics was proposed. It can only be used to model services at a very high-level considering each service as a black box. A much more detailed and fine-grained metamodel is needed in order to cover all relevant aspects necessary for predicting the service performance. In [68, 69, 70], several methods for dynamic selection of services with the goal to optimize the overall QoS of a composition are proposed. In [68] workflow patterns are used, whereas the approaches in [69] and [70] use genetic algorithms and heuristic methods, respectively.

A different approach to QoS brokering and service selection is presented in [71], where analytic queueing models are used to predict the QoS of alternative services that could be selected under varying workload conditions. Service consumers provide to a QoS broker their utility functions and cost constraints on the requested services. The service provider that maximizes the customer’s utility function subject to its cost constraints is selected. In [72, 73], a service discovery system enabling service compositions from semantic descriptions stored in a knowledge base is proposed. A recursive algorithm builds service compositions by adding services in each iteration. The search works backwards, since services are added that produce a certain output regardless of their input parameters. A valid service composition produces a set of queried output parameters and input parameters necessary for the composed services.

An approach to modeling the performance of composite SOA services composed by means of BPEL (Business Process Execution Language) [74] was presented in [75]. Some further approaches based on simulation were proposed in [76, 77, 78]. These approaches, however, only consider static service compositions. Several larger efforts in the Web services arena have focused on describing, advertising and signing up to Web services at defined levels of QoS, for example, HP’s Web Services Management Framework (WSMF), IBM’s Web Service Level Agreement (WSLA) framework, the Web Services Offerings Language (WSOL) and the WS-Policy. These efforts consider QoS in its broader meaning (not limited to performance properties) and specifically target Web service management activities. Performance properties are defined at a very high level and their enforcement at the network and infrastructure layers is not dealt with.
2.4 Predictable Service Engineering Methodology

In this section, we describe the workflow of the prediction service on an abstract level. For performance prediction, it is necessary to collect information from different roles involved in the service lifecycle, namely service providers, software providers, infrastructure providers, and service customers. The prediction service uses this information to evaluate the performance properties of different design alternatives for a specific service. Other modules (such as Negotiation or Translation) can use its results to identify an optimal solution for a specific customer. In the following, we describe the information that prediction service retrieves from the roles involved as well as the workflow of the prediction service.

2.4.1 Roles

Infrastructure providers need to specify the capacity of the available (virtual) machines. This includes the number of processors and/or cores, the clock frequencies, as well as the size of the main memory.

Software Providers specify the architecture of provided services using, for example, the Service Component Architecture (SCA) [136]. Services are realised by service components that can be assembled to composites forming larger systems. For performance prediction, a behavioural specification additionally describes how each service component uses hardware and software resources and calls other services.

Service Customers specify the expected usage of the service they are interested in. The specification includes the expected arrival rates and the parameters passed to the service. For the ORC (Deliverable D.B2a), operation bookSale of the InventoryService might be called 340 times per minute. Furthermore, it may receive 12 items on average that are processed within the request (specified in its parameter salesTO). The number of processed items influences the resource demands needed to process a request and, thus, has to be modelled by the service customer.

Service Providers offer general information on the realisation of services and define or constraint possible design alternatives. For example, they create different appliances for a service and define how dependencies on external services are resolved. In the following section, we describe the workflow of the prediction service in more detail.
2.4.2 Prediction Service Workflow

During the negotiation and translation phase, service providers (implicitly) trigger the prediction service in order to estimate the expected performance properties of a service requested by customers. The prediction service 1) collects the information necessary for performance prediction, 2) generates different deployment alternatives for the service, and 3) predicts the performance properties for each alternative (cf. Figure 2).

Model Collection: The prediction service collects all information necessary to execute performance prediction from all parties involved. We assume infrastructure providers have already stored so-called Infrastructure SLA templates regarding the available VM sizes in a central template repository (cf. Deliverable D.A4a). The prediction service retrieves that information and translates it into infrastructure models that can be processed by the simulation engine of the Palladio Component Model (PCM) [1]. Furthermore, the arrival rates and service parameters specified by service customers are transformed into a usage model of the requested services. We also assume software providers have stored a performance specification of their service components in a central Design-Time Repository. Furthermore, service providers have selected and assembled service components to appliances. The prediction service retrieves information about the service components and their assembly in the Design-time repository. After model collection, all needed information is converted into a format that can be processed by the PCM.

Scenario Generation: In this step, multiple design alternatives for a service are generated. The alternatives vary with respect to the chosen infrastructure and the allocation of service components on (virtual) machines. The performance

![Figure 2: Abstract workflow of the prediction service.](image)
properties of a service change depending on the chosen infrastructure and deployment. The generation of different alternatives allows the service provider to identify the optimal solution for a service customer.

**Performance Prediction:** In this step, the performance properties of a service are predicted for each generated alternative. All results are returned to the caller of the prediction service. They are meant to support Negotiation and/or Translation to identify an optimal setting for the specific customer. So far prediction results include the completion time and throughput of each service operation.

### 2.5 QoS Metamodel for Performance Prediction

In this section, we illustrate the core concepts of the current QoS metamodel. It is worth to note that the current version of the QoS metamodel focuses on performance prediction, and we plan to extend it with reliability features in the next project year.

The QoS metamodel consists of four parts corresponding to different roles involved in the service life-cycle: 1) service component model, 2) infrastructure model, 3) allocation model, and 4) usage model. Every role is responsible for providing its assigned model for QoS analysis. The required models can thereby be specified independently.

**Software providers** create an abstract model of their service components. The model includes a specification of service components, their (service) interfaces, and composition as well as an abstract behavioural description (in terms of so-called *Resource Demanding Service Effect Specification* RDSEFF). The latter describes how a specific service uses the available hardware and software resources, and is thus essential for QoS analysis.

**Infrastructure providers** abstractly define the (virtual) infrastructure that they offer to a specific service provider. The model includes the (virtual) nodes (their processors, hard disk drives, etc.) and their interconnections. Thus, infrastructure providers just specify the execution environment for services and are not concerned with the allocation of service components to (virtual) nodes. The latter is a task of the service provider (see below).

**Service customers** request individual SLOs for specific services. For a priori QoS analysis, they have to specify their expected usage profile. They need to provide information on the expected number of concurrent users, the arrival rate of requests, as well as the size and amount of data that need to be processed by the services.

Finally, **service providers** are responsible for integrating the separate models provided by different parties of the service life cycle. They host the services for customers and also provide prediction and negotiation services. Due to their central role, they have to acquire all necessary models from software providers, infrastructure providers, and customers in order to perform an a priori QoS analysis. They are furthermore responsible to allocate software components to (virtual) nodes offered by infrastructure providers. Together all models allow the service provider to conduct a performance analysis of the system under study.
In the following subsections, we introduce the metamodels for software components and interfaces, composite components and systems, component behaviour, resource allocation and usage profiles.

### 2.5.1 Components and Interfaces

For the static architecture of the QoS metamodel, we follow the concepts of the Service Component Architecture (SCA) (The OASIS Open CSA, 2008), even if our terminology currently deviates from the terms used in the SCA specification. Our notation adopts the terminology of the current QoS metamodel elements and gives the terms used in SCA between brackets.

In the QoS metamodel, service components can either be atomic or composed from other service components. In this subsection, we focus on atomic service components and defer composition to Subsection 2.5.2.

**Figure 3: Components, Interfaces, and Roles**

Figure 3 shows the metamodel excerpt for components, interfaces, and roles. Service components and interfaces are first-class entities, since they can exist independently from other entities. BasicComponents (SCA: Service Components) model atomic components. Roles assign Interfaces to service components. Thereby, ProvidedRoles (SCA: Services) specify the interfaces that are offered by a component, while RequiredRoles (SCA: References) specify those interfaces that allow the component to work properly.
As Figure 4 shows, an Interface contains a list of method Signatures. Each Signature has a list of input Parameters and a return value. Each Parameter, as well as the return value, belongs to a certain DataType, which is either a Primitive-DataType, a CompositeDataType, or a CollectionDataType. Components and interfaces are stored in a design-time Repository.

### 2.5.2 Composite Components and Systems

CompositeComponents (SCA: Lower-level Composites) assemble service components (basic and composite) into higher-level structures. A CompositeComponent does not contain any own business logic or implementation code itself; rather, it is a structuring element to represent different levels of abstraction. It is important to note that composite components can only be deployed as a whole. Thus, their subcomponents cannot be allocated to different nodes. The concealment of internal structures of composite components currently contradicts the concepts of SCA claiming that each subcomponent can be deployed separately.

At the highest level, Systems (SCA: Higher-level Composite) combine components that represent the modelled software system as a whole. Similar to components, Systems provide Interfaces to their users (either human or other systems) and require other software services specified in terms of required
Interfaces (see Figure 5). A System presents a self contained part of the overall QoS model.

### 2.5.3 Component Behaviour

For QoS analysis, behavioural models are essential. They describe how services use available hard- and software resources and thus form the basis for performance and reliability analysis. In the context of the proposed QoS metamodel, *Resource Demanding Service Effect Specifications* (RDSEFFs) model the control and data flow of service components. RDSEFFs specify the behaviour of each method *Signature* provided by a *BasicComponent* (see Figure 6).

#### Figure 6: Service Effect Specifications

Being a *ResourceDemandingBehaviour*, the RDSEFF basically consists of a chain of actions which are either calls to other components (*ExternalCallActions*) or abstract internal computations (*AbstractResourceDemandingActions*). In order to assign an *ExternalCallAction* to a specific method, it references a *Signature* of a required *Interface* or the *BasicComponent* (see Figure 7).

#### Figure 7: Resource Demanding and External Actions

For performance prediction, an *AbstractResourceDemandingAction* needs to specify a resource type and the amount of resource units required. It therefore contains a *ParametricResourceDemand* that stochastically describes the required amount of resource units. For this purpose a *RandomVariable* stochastically specifies possible dependencies to input parameters. In the following, we give a brief introduction to such parametric dependencies. Please refer to (Koziolek,
Happe, & Becker, 2006) or (Becker, Koziolek, & Reussner, 2009) for further
details.

Parameter dependencies allow for representing methods with input and output
parameters, whose values can be characterized with focus on performance-
relevant aspects. Possible characterizations include the VALUE, BYTESIZE,
NUMBER_OF_ELEMENTS, or TYPE of a parameter. The characterisations can be
stochastic, e.g., the byte size of a data container can be specified by a probability
mass function. For example:

data.BYTESIZE = IntPMF[(1000;0.8) (2000;0.2)]

where IntPMF is a probability mass function over the domain of integers. Here
data has a size of 1000 bytes with probability 0.8 and a size of 2000 with
probability 0.2.

Stochastic expressions model data flow based on parameter characterisations.
For example, the stochastic expression:

result.BYTESIZE = data.BYTESIZE * 0.6

specifies that (for instance) a compression algorithm reduces the size of data to
60%. The expression thus yields:

IntPMF[(600;0.8) (1200;0.2)].

Stochastic expressions support arithmetic operations (*,-,+,/,...) as well as
logical operations for Boolean type expressions (==,>,<,AND,OR,...) on random
variables.

Various specialisations of AbstractResourceDemandingActions (see Figure 8)
reflect control flow statements and allow software providers to specify branches,
loops, forks etc. That way, performance-relevant dependencies between the
control flow, external calls, and parameters, can explicitly be expressed in the
model. On the other hand, independent component-internal processing can be
aggregated into InternalActions, keeping the abstraction from the component’s
implementation.

BranchActions represent “exclusive or” splits of the control flow, where only one
of the alternatives can be taken. The choice can either be probabilistic or
determined by a guard. In the first case, each alternative has an associated
probability giving the likelihood of its execution. In the latter case, Boolean

Figure 8: Overview of Actions
expressions on the service's input parameters guard each alternative. With a stochastic specification of the input parameters provided by the caller, the guards are evaluated to probabilities.

**LoopActions** model the repetitive execution of a part of the control flow. A probability mass function specifies the number of loop iterations. For example, a loop might execute 5 times with a probability of 0.7 and 10 times with a probability of 0.3. The number of loop iterations can depend on the service's input parameters. Furthermore, iterations over a collection are also modeled explicitly (**CollectionIteratorAction**) where the number of repetitions depends on the size of a collection.

**ForkActions** split the control flow into multiple concurrently executing threads. The control flow of each thread is modelled by a **ForkedBehaviour**. The main control flow only waits for forked behaviours that are marked as synchronised. Its execution continues as soon as all synchronised **ForkedBehaviours** finished their execution.

**AcquireActions** and **ReleaseActions** model the acquisition and release of limited passive resources, e.g., semaphores or connection pools. Passive resources may have a significant influence on the execution time of a service due to waiting times.

### 2.5.4 Resource Allocation

Infrastructure providers need to model the (virtual) resource environment that shall host a service. In order to separate the specification of service components (which require resources) and the infrastructure (which offers resources) both refer to abstract resource types from a global resource repository. Infrastructure providers instantiate these types to describe their execution environment. The proposed QoS metamodel coarsely distinguishes between processing resource types (e.g., CPU, HD, etc.) and passive resource types (e.g., semaphores etc.).

![Resource Environments](image)

**Figure 9: Resource Environments**

ResourceEnvironments contain a number of resource containers (called nodes in UML) connected by LinkingResources. ResourceContainers bundle a set of resources. They include ProcessingResourceSpecifications (e.g., a CPU with a processing rate of 1000 work units per second) and...
PassiveResourceSpecifications (e.g., a data base connection pool with a capacity of 10). A component that is embedded in a specific software architecture (its so-called AssemblyContext) can be allocated to specific resources. The abstract resources referenced by the RDSEFFs included in the AssemblyContext’s components can then be substituted by the concrete resources from the resource environment to compute actual resource demands.

Within an RDSEFF, ResourceDemandingAction request ProcessingResources (see Figure 7) that always have an associated processing rate. The demand divided by the processing rate yields the processing time of the demand not considering any contention effects.

![Figure 10: System Allocation](attachment:image)

Figure 10 shows the allocation of a System to a ResourceEnvironment. Each component instance included in the System through an AssemblyContext is mapped to a ResourceContainer by an AllocationContext, representing a deployment instance of the component.

### 2.5.5 Usage Profiles

To estimate the expected QoS a priori, Service customers specify a system’s usage in terms of workload (i.e., the number of concurrent users), user behaviour (i.e., the control flow of user system calls), and parameters (i.e., abstract characterisations of the parameter instances users utilise).

![Figure 11: Scenario Behaviours](attachment:image)

Usage models contain multiple UsageScenarios, each of which models a single use case of the system. For each scenario, a workload describes its usage intensity and a behavioural model (ScenarioBehaviour, see Figure 11) its flow of user actions. The scenario behaviour is analogous to RDSEFFs, but does not contain any resource consumptions.
The Workload, as shown by Figure 12, is either an OpenWorkload or a ClosedWorkload (similar to queueing networks). An OpenWorkload specifies an InterArrivalTime, that is, a time interval between the arrivals of two users. Every user who arrives at the system executes the associated ScenarioBehaviour once and exits. By contrast, a ClosedWorkload specifies a fixed number of users who are always present. Every user executes the associated ScenarioBehaviour infinitely, taking a certain ThinkTime between two executions.

### 2.6 **Design-time Prediction Service**

Design-time prediction determines QoS properties of services before they are actually deployed and running. We provide a tool used during the automated SLA negotiation between customers and providers of a service: it allows service providers to calculate feasible QoS parameters they can offer. Given the structure of Service Component Architecture (SCA) [136], the allocation of service components to infrastructure, and the QoS properties of externally required services, prediction can determine if specific QoS requirements issued by a customer under a specific system usage profile are likely to be fulfilled by the system.

Within the SLA@SOI framework, design-time prediction is a service which can be invoked from other components (i.e., negotiation and translation modules). For this purpose, prediction has to be performed *on the fly*, as part of automatic negotiation between service customer and provider. Thus, an interface for invocation has to be offered, and data models have to be defined to capture the input and output data necessary to use the prediction service. Another requirement refers to prediction of *multiple scenarios*. A scenario represents one specific system configuration corresponding to a complete instance of the QoS metamodel (see Section 2.5). Prediction must also be executed for a set of related scenarios, such as those obtained varying a part of the scenario (for example, the allocation of service components in the infrastructure), while keeping the other parts fixed. To some extend, prediction must be able to automatically produce different scenario variants on the basis of parameters given as an input or can be retrieved automatically. The results of all predicted scenario variants are returned to the caller in terms of a result list.
Each entry of the result list contains a reference to the corresponding scenario variant, as well as the predicted results themselves.

A third requirement is the support for distributed prediction, since, in general, prediction is performed as part of a distributed negotiation process across several service providers. The providers are connected to each other through their provided and required services (Services and References in terms of SCA). Since each provider owns part of the architecture, it is in charge to perform predictions for that part. Prediction must translate QoS requirements of provided services into a usage profile, and deduce the QoS parameters values that can be guaranteed from simulation results.

### 2.6.1 Prediction Process

Within the SLA@SOI framework, design-time prediction is implemented as a service for other components of the SLA@SOI framework. The prediction service offers one central routine to predict QoS properties of a scenario or a set of scenarios.

![Figure 13: Design-time Prediction Process](image)

Figure 13 shows the main steps of the prediction process. During service negotiation, prediction is called by the negotiation or translation components on the side of the service provider. The input data include references to the services a customer wants to use, along with a desired throughput (see Section 2.6.2). Prediction consists of three phases, namely scenario data collection, evaluation of scenarios, and aggregation of prediction results. During *scenario data collection*,
prediction service retrieves all necessary information from two repositories available in the SLA@SOI framework: the design-time repository, which keeps the models provided by the software provider that describe the service component architecture (SCA), and the SLA template registry, which keeps information about offerings provided by infrastructure providers, describing virtual machines and their QoS properties.

From the collected data, prediction builds one or multiple scenarios (i.e., QoS metamodel instances). The variability in the construction of scenarios stems from multiple possible infrastructure configurations, as well as a variety of possible allocations of service components to infrastructure.

Once the scenarios have been built, prediction enters the evaluation phase and performs one QoS prediction per scenario. Therefore, a simulation is performed based on the QoS metamodel instance, and results are collected and stored. In year one, simulation results consist of punctual throughput values and response time probability distributions for all services that are involved in the system service execution, including basic services. The last step in the prediction process comprises aggregation of prediction results into statistical values like mean values and percentiles. The overall prediction result consists of a list of predicted scenarios, where each entry in the list describes a scenario and its predicted QoS parameters (see Section 2.5).

2.6.2 Input Data Format

The QoS metamodel consists of four parts, as discussed in Section 2.5. One of these parts is the usage model, which describes the service usage profile of its potential customers.

For design-time prediction, usage models are important since they contain information that can impact prediction results. However, to ease the invocation of prediction service, a general usage model template is prepared in advance, and adjusted to actual needs when prediction is invoked.

Figure 14: Prediction Service Input Model
Prediction service receives a ServiceUsageModel, whose structure is illustrated in Figure 14. It includes a number of ServiceUsages, each describing the usage of one provided service, referenced by its name (attribute serviceName). Each ServiceUsage is composed by a list of OperationUsages, describing the service operations that are invoked. An ArrivalRate (taken from the WP A1 common terms metamodel) determines the frequency of invocations of each operation.

Since input parameters of service operations have impact on quality, as well as arrival rates, they also need to be associated to characterize a OperationUsage element. Such a specification is given by a ParameterCharacterisation, which references a certain input parameter by name, and specifies the type of characterisation by a ParameterCharacterisationType. The differentiation between types of characterisation helps to determine the specific characteristic of a parameter that influences quality. This can be the VALUE of the parameter itself. Other characteristics can be important as well. If the input parameter is a file, the BYTESIZE of the file might influence the duration of processing. For collections, the NUMBER_OF_ELEMENTS might be a critical factor for performance. Currently, parameter characteristics can be specified by constant values. In the long term, specification through a probability distribution might be desirable.

![Figure 15: Prediction Input Example](image)

Figure 15 shows an example of a service input model, we used for the ORC. The model is visualized by an EMF editor. The ServiceUsageModel contains two ServiceUsages for the inventory service and the payment service. Three operations are invoked: getProductDetails() and bookSale() for the inventory service, and handlePayment() for the payment service. Arrival rates are specified for each operation by the number of requests arriving per minute (frequency). Furthermore, the bookSale() operation contains a specification of the number of elements of the salesTO parameter (which represents the list of sold items).

### 2.6.3 Prediction Result Data Format

As explained above, during design-time prediction, multiple prediction scenarios are generated and evaluated (see Section 2.6), each of them producing its own prediction results. Scenarios are generated depending on the available infrastructure SLA templates, and the possible deployment options of service.

In order to limit the range of the possible allocation variants, a set of possible deployment options for the service component architecture is determined in
advance. Prediction takes only these options into account assigning components into the infrastructure.

![Figure 16: Deployment Options Model for Prediction](image)

Figure 16 depicts the model for possible deployment options. The DeploymentRepository as the topmost container class describes the possible DeploymentOptions of a service-oriented architecture. Each DeploymentOption models a specific deployment of services (and their implementing components). Through the explicit specification of DeploymentOptions, all deployment variants that do not comply with any option are implicitly marked as invalid. Each DeploymentOption is meant to comprise the whole system, i.e., all service (and potentially legacy) components that make up the system. To describe a certain option, services are grouped into DeploymentUnits. Services within a unit are referenced by name. Each service should only be included in one DeploymentUnit within each DeploymentOption. Each DeploymentUnit is associated with an Appliance (coming from the WP A1 common terms metamodel). All services belonging to the same unit shall be deployed on its associated Appliance and allocated to the virtual node that hosts the Appliance. An Appliance cannot be associated with two different DeploymentUnits, but is exclusively used for one unit. A DeploymentUnit adds further information to the Appliance. For example, it includes a list of services offered by its referenced Appliance.

![Figure 17: Deployment Options Example](image)

In the first project year, a deployment options model like the one illustrated in Figure 17 are used for the ORC. In this example, two DeploymentOptions are considered: a centralized deployment with all components on one virtual node
(CentralORCDeployment), and a distributed deployment, where the database is separated from the other components (DistributedORCDeployment). Each DeploymentUnit lists the services included by itself (not visible in the figure) and is associated with an Appliance that shall hold the services.

Figure 18: Prediction Service Result Model

After generating and evaluating all prediction scenarios, the results are aggregated into a result model and returned to the caller of prediction service. For each evaluated scenario, the result model contains a scenario description and a collection of individual result values. The scenario description contains information about the allocation of service components to virtual nodes, and a specification of each virtual node’s characteristics.

The root element of the model is ResultSet. It is composed by multiple PredictionResults, describing the infrastructure adopted to perform prediction on SLAs, the chosen deployment configuration of the service component architecture, and the prediction results for each operation of each service.

PredictionResult contain a set of VirtualNodes to express the infrastructure adopted to predict the expected performance for an SLA. A VirtualNode basically encapsulates a VirtualMachine, which is a basic element of SLA@SOI's core model. A VirtualMachine can include a set of CPUs and/or cores, as well as information about the available main memory. The surrounding class VirtualNode contains additional information about a VirtualMachine, such as a name or predicted performance and reliability metrics. For example, VirtualNodes can include the utilisation of the VirtualMachine's CPUs for a given scenario.

The chosen deployment is modelled by the class Assignment referenced by PredictionResult. It is a simple association class that links a DeploymentUnit (and its appliance) to a VirtualNode. DeploymentUnits, VirtualNodes, and
their Assignments provide enough information for the Negotiation Module to construct a request for the infrastructure provider.

The PredictionResult provides various performance metrics depending on the infrastructure templates and specific DeploymentOptions associated to a service. For year one, we focus on CompletionTime (more specifically the percentile 95 of the CompletionTime) and Throughput. Both terms are specified in the term model. In order to associate the prediction results to the correct service operations, we grouped them depending on the services and the operations they refer to. The PredictionResult is also composed of a set of ServiceResults, each of them including a set of OperationResults. The latter represents the actual performance metrics that have been predicted. The class ServiceResult associates a set of OperationResult. Furthermore, the class OperationResult stores the predicted performance metrics (CompletionTime & Throughput) for a specific service operation (e.g., operation bookSale() of the inventory service can have a CompletionTime of 190 ms in 95% of all cases and a mean throughput of 345 requests per minute).

![Figure 19: Prediction Service Result Example](image)

Figure 19 shows an instance of the Result model generated for the ORC. The model contains two PredictionResults associated to two corresponding deployment options of the ORC (assuming that there is only one infrastructure SLA template to apply). Each PredictionResult contains a description of VirtualNodes and ServiceResults representing the concrete prediction result.
values for service operations `getProductDetails()`, `bookSale()` and `handlePayment()`.

### 2.6.4 Implementation

The implementation of the design-time prediction service is open source based on an Eclipse / OSGi platform [143]. We used the Eclipse Modelling Framework (EMF) [144] to generate java code from the QoS metamodel. The construction of QoS metamodel instances is supported by EMF editors that permit to visualize the tree of modelled elements. Furthermore, we provide a set of graphical editors based on the Graphical Modelling Framework (GMF) [145]. The Eclipse platform with its modelling support (EMF and GMF) proofed to be an efficient and elegant way to realise model-driven performance predictions in the context of SLA@SOI.

For the purpose of prediction, the input model is automatically transformed into a java code skeleton. The code generation process is realized based on the OpenArchitectureWare (OAW) framework [146]. The generated code is instrumented to collect performance and utilisation data. Performance evaluation is done through simulation using SSJ simulation engine. During simulation, the generated code is executed, and measurements are collected. After simulation, statistical properties, such as mean values or percentiles, are calculated on the collected data and they are returned to the caller of the prediction service.

![Diagram of SLA@SOI Framework and Design-time Prediction](image)

**Figure 20: SLA@SOI Framework and Design-time Prediction**

Figure 20 shows the connections between the design-time prediction components and the SLA@SOI framework. SLA@SOI framework components are based on Spring [147]. The prediction service can be invoked by the SLA Negotiation / Translation components of SLA@SOI framework. The prediction service is exposed through JAX-WS [148]. A prediction client invokes prediction on the server for each scenario to predict. All model parts belonging to the scenario are given as an input to the server; the results of prediction constitute the return value.

### 2.7 Demonstrator based on ORC

To demonstrate the capabilities of design-time prediction within the context of SLA@SOI, the Open Reference Case (ORC), which is developed in WP B2, has been modelled as a QoS metamodel instance. This instance represents an abstraction of the existing implementation capturing the QoS-relevant characteristics of the service component architecture (SCA). It includes all parts...
that are necessary for a qualified QoS prediction: service component model, infrastructure model, allocation model and usage model (see Section 2.5).

The Open Reference Case is based on a trading system dealing with the various aspects of handling sales at a supermarket. This includes the interaction at the cash desk with the customer, including product scanning and payment, as well as accounting the sale at the inventory. Furthermore, the trading system deals with ordering goods from wholesalers, and generating various kinds of reports.

In the context of the ORC, the trading system is extended into a service oriented retail solution including IT support for retail chains in general, covering enterprise headquarter (central management issues), stores (local management) and cash desks. The system exposes services that are implemented on top of the existing legacy components, and owned by a service provider.

A retail chain takes the role of service customer. Multiple supermarkets within the chain are connected to the service provider. The range of potentially provided services includes inventory management, credit card payments, preferred customer club card, accounting, and others. The service provider is in turn connected to several external providers such as bank, wholesale centre, CRM supplier etc. The ORC services run on top of an IT infrastructure offered by one or multiple infrastructure providers.

In the first project year, the ORC focuses on the sales process at the individual cash desks. Several services are invoked during each sales process. For each item captured by the scanner, product information has to be retrieved from the inventory database operated by the service provider. If the customer at the cash desk decides to pay with credit card, the card has to be validated and debited according to the summarized sales value. This requires an additional bank service provider, which in turn is used by the ORC service provider. Finally, the sales process is accomplished by booking the sale with the inventory database, transferring information about all sold items (ID’s and quantities).

The sales process as described above is supported by the inventory service and the payment service, where the latter is a composition of two basic services: card validation service and payment debit service. The service components for these services are included in the QoS metamodel instance to support QoS prediction for the sales process. Thereby, prediction focuses on the response times and throughput of the involved service operations.

This section is structured following the individual parts of the QoS model – service components (Section 2.7.1), infrastructure (Section 2.7.2), component allocation (Section 2.7.3), and system usage model (Section 2.7.4).

**2.7.1 Service Component Model**

The ORC service component model contains a specification of the service components that are involved in the sales process, as well as the software components of the underlying legacy application (the trading system). The component specifications include the provided and required interfaces, behavioural specifications, and composition of components.
Figure 21: Open Reference Case Repository

Figure 21 shows the component repository, visualized by an EMF editor. The ORC model in its current state comprises 15 components – 8 service components and 7 legacy components. The service components include basic services (such as the inventory service), and composed services (such as the payment service). Access to the inventory database is modelled through a database component. Furthermore, the repository contains all interfaces that are provided and required by the components. Interfaces are first-class entities in the model (see Section 2.5.1). Components and interfaces inherit from the metamodel class Identifier and thus each of them has a unique identifier within the QoS metamodel instance.

Figure 22: Payment Service Component

Within the component repository, each component is specified with its provided and required service interfaces, as well as its RDSEFFs (see Section 2.5.3) for provided service operations. Forexample, Figure 22 shows the payment service component, which provides the payment service interface, and in turn requires the card validation and payment debit service interfaces. In addition, the handle payment RDSEFF specifies the control flow of the handlePayment() service operation (which is also illustrated in Figure 25).
Interfaces are specified with their signatures, containing parameters and return values. Figure 23 shows the inventory service interface specification. Amongst others, the interface contains a signature for the `bookSale()` operation, which takes the list of sold items in input, and has no output parameters.

The model can be simplified compared to the implementation, and it can only include those parameters that have an impact on the performance of the service at runtime. For this reason it is not necessary to model all input parameters and return values of the service operations.

Figure 24 illustrates the set of components belonging to a System element (see Section 2.5.2), which, in this case, represents the service-oriented retail solution as a whole. The system provides a set of interfaces (e.g., the payment interface), and also requires a service interface (the bank interface), which indicates the need of a service provided by an external provider. System element includes connections among components: the service components (left-hand side) on top of the legacy components (middle), and database access at the bottom of the use hierarchy (right-hand side). Figure 24 shows that the payment service is composed of the card validation and payment debit services, which in turn make use of the bank service (offered by an external provider). The inventory service is based on the application (store) legacy component.
As an example for a Resource Demanding Service Effect Specification (RDSEFF, see Section 2.5.3), Figure 25 depicts the control flow for the `handlePayment()` operation of the composed payment service. Two calls to the card validation and payment debit services are surrounded by internal actions representing internal processing of the payment service.

The internal actions are annotated with resource demands specified through probability distributions. The `handlePayment_Preprocessing` action, for example, specifies that up to 10 CPU work units are needed with a probability of 0.15; whereas 10 to 16 work units are needed with probability 0.85. The description of CPU demands through abstract work units enables independent component and infrastructure specifications. Concrete time demands can be deduced from the combination of these specifications, as soon as the allocation of components is defined.

The resource demands specified in RDSEFFs can be based on estimations of the software provider who implements a service component, or by measurements obtained during test bed runs.
A second RDSEFF example is given by Figure 26. It depicts the control flow for the `bookSale()` call within the `Application.Store` component. Any call to the inventory service operation `bookSale()` is forwarded to the `Application.Store` component and processed there.

The control flow includes several calls to the data access layer. In addition to calls concerning transactions and persistence contexts, which are invoked exactly once, there is a call to query the stock amount of an individual sales item. This call is invoked for each item and thus appears within a loop. The loop count equals the size of the items list that has been given as an input parameter to the call. Thus, the loop count is specified using a parametric dependency (see Section 2.5.3).

Furthermore, the pre-processing captured by an internal action right at the start is also parametrically specified, consuming up to 18 CPU work units per item in the given items list.

### 2.7.2 Infrastructure Model

The ORC infrastructure model describes the resource environment to which service and legacy components are allocated to. For QoS prediction in SLA@SOI, the model is not fixed in advance, but used by the design-time prediction service as a template. Concrete values are inserted into the model from the infrastructure SLA templates received through queries to the SLA template registry (see Section 2.6.1).
Figure 27 shows the model contents visualized by an EMF editor. The model is prepared to support both deployment options considered in the first project year: the complete retail solution running on one virtual machine, or service and legacy components separate from the database. Accordingly, three resource containers (see Section 2.5.4) have been specified that can be used for deployment – the CompleteRetailSolution_VM, Database_VM, and ServicesAndLegacyComponents_VM. Each resource container is equipped with a CPU resource. The processing rates of the CPUs are set to the default value 1.0 (which means one CPU work unit per time unit). Design-time prediction substitutes these values with concrete processing rates taken from infrastructure SLA templates.

To support the distributed deployment option, a connection between the Database_VM and ServicesAndLegacyComponents_VM is needed. Therefore, a linking resource is specified to connect these containers. Latency and throughput values of the connection are specified and assumed to be fixed in the first project year.

2.7.3 Allocation Model

The allocation of service and legacy components to resource containers allows calculation of concrete time demands from abstract resource demands specified in component RDSEFFs. It is therefore a necessary part of any prediction scenario (see Section 2.6.1).
Eventually, the allocation model is dynamically constructed by the design-time prediction service from the deployment options model (see Section 2.6.2). However, in the first project year, two allocation models are fixed in advance and directly used for prediction. One allocation puts all service and legacy components to a single resource container; the other allocation – shown by Figure 28 – separates the database from the other components.

The figure shows the allocation of all components that make up the service-oriented retail solution (see Figure 24) to the resource containers specified in the infrastructure model (see Figure 27). The connection between the two resource containers is not part of the allocation model and thus not shown in the figure, but has been specified as part of the infrastructure model and is taken into account by design-time prediction.

### 2.7.4 Usage Model

In analogy to the allocation model described in the preceding section, the usage model is not fixed. It depends on the service customer’s request for usage of system services, and is given to design-time prediction as an input (see Section 2.6.2). In the first project year, a template is constructed in advance and adjusted by prediction service as needed.
Figure 29: ORC Usage Model

Figure 29 shows this template visualized by an EMF editor. System usage in year one is restricted to the `bookSale()` and `getProductDetails()` operations of the basic inventory service, as well as the `handlePayment()` operation of the composed payment service. Both services are provided by the retail solution, as can be seen in Figure 24.

All operations are invoked concurrently, each with an open workload specifying an inter-arrival time, i.e., a time interval between two consecutive invocations (see Section 2.5.5). Design-time prediction takes all interferences between the concurrently executed service operations into account. The predefined inter-arrival time values are placeholders for the actual values filled in during the construction of prediction scenarios.

Figure 30: Book Sale Usage Scenario

Each of the three specified usage scenarios is not only equipped with an open workload, but also with a usage behaviour specifying how the service operation is invoked. As an example, Figure 30 shows the usage scenario for the `bookSale()` operation of the inventory service. The corresponding behaviour only consists of the start and stop actions, and the `EntryLevelSystemCall` (see Section 2.5.5) to the `bookSale()` operation. The call includes a description of the list of sales items given as an input parameter to the operation. It specifies that the list has 1 item with probability 0.3, 5 items with probability 0.4, and 10 items with probability 0.3. The specification of number of items is necessary as it influences the control flow of service execution (see Figure 26). Again, the specification included in the model has to be replaced by actual demands during construction of prediction scenarios.
2.8 Summary

In this chapter, we presented the current state of the design and implementation of the prediction service. We described the general prediction methodology in service-oriented environments, the information necessary from the different roles involved in the service-life, the domain-specific metamodels for each role, the collection and integration of models, the generation of different variations to realise a service, and the aggregation of results. In its current state, the prediction service provides the estimates for different deployment options of the Open Reference Case. These estimates support Negotiation to identify the optimal solution for a specific customer request. Furthermore, Translation can establish an SLA-hierarchy based on the results.

For the future of the project, we plan to improve the prediction quality for service-oriented infrastructure as well as the generation of different alternatives. Furthermore, the performance metrics returned by the service shall be extended so that more information is available for the comparison of different design alternatives.
3 Run-time Prediction

3.1 Introduction

SLA@SOI requires a monitoring framework to collect information across the software, infrastructure and service landscape for detecting SLAs violations. The monitoring framework not only provides reports about a SLA's property violation, it also provides information about a likely property violation. In other words, it predicts a property violation. Monitoring and prediction thereby may either take place on the infrastructure level (infrastructure provider) or on the (software) service level (service provider). In Y1 we investigated two independent prediction methods that focus on one specific level (infrastructure or service) and do not take into account the information available at other level. Starting from Y2, we will investigate beneficial combinations of the approaches across the different levels.

Regarding the service level prediction (CITY, Section 3.4), we focused on designing a monitoring framework for collection information about the functional and non-functional behaviour of the respective business services (i.e. Quality of Service, QoS) and their implementing components. This information is used for predicting the services' non-functional properties in the near future.

In this context, the QoS properties prediction is based on run-time data measurement and behavioural models of the involved services. Behavioural models are used to guide the search for correlations between recorded runtime events that could be subsequently used to establish the probability of occurrences (or absences) of additional events affecting the satisfaction of QoS properties in SLAs.

At infrastructure level (INTEL, Section 3.3), runtime prediction is to forecast the dynamic computational resource demand or usage in the near future. With this prediction, provisioning managers can proactively migrate service, reprovision or release computing resources. It can avoid SLA violation by resource re-provisioning when necessary and save operational and utility cost by temporarily suspending unutilized capacity.

To achieve good predictions at infrastructure level, we use methods and techniques from machine learning. Given the strong capability of machine learning technology in data modelling and prediction, we assessed some machine learning algorithms. We selected and implemented three candidates after technical review of several algorithms. Based on the experimental analysis, we improved the original algorithms. The experimental results obtained on ORC and real data sets show the effectiveness of our improvements.

The remainder of this chapter is structured as follows. Section 3.2 summarises the current state of the art for infrastructure level prediction and service level prediction. In Section 3.3, we present our current infrastructure level prediction approach focussing on the experimental evaluation of different approaches. Service level prediction is presented in Section 3.4. Finally, Section 3.5 concludes this chapter.
3.2 State of the Art

3.2.1 Infrastructure level Prediction

Run-time prediction plays an important role in delivering timely application services and averting SLA violations. But most SLA management systems found in the literatures focused only on some specific actions, such as negotiation and reservation, or reservation and monitoring; few extend support for run-time prediction and adaptation. Some related work on workload analysis, resource demand run-time prediction as well as dynamic resource provisioning is summarized as follows.

Rajarshi Das et al developed an energy-conserving solution under the condition that does not sacrifice performance objectives. A prototype of an integrated data center power management solution was also developed [80]. One problem of the solution is only one application was used for the experiments and the experimental results need to be verified further. Another problem is that the policy in the model derived from historical observations is not flexible enough.

David Vengerov presented a general framework for performing adaptive reconfiguration of a distributed system based on maximizing the long-term business value through reinforcement learning, a machine learning technique [81]. This model is suitable for scenarios where the objective functions are defined as the discounted sum. One problem of the algorithm is that the definitions of fuzzy set and membership function in this paper are quite special that may raise inaccuracy to some extent. Another is that convergence of reinforcement learning was only proved in Markovian domains. Is this solution fit for non-Markovian domains?

Jian Zhang et al described a prototype classifier for application centric Virtual Machines, which classified applications based on extracted features (e.g. CPU, memory, disk and network) and the classifiers would be used to assist multi-dimensional resource scheduling [82]. But the performance metrics are selected manually based on expert knowledge. Besides, this study only focused on sequential and single-stage applications. Base on the previous application classification framework, Jian Zhang et al proposed to automatically perform metric selection from numerous performance metrics collected from some monitoring tool [83]. But the solution also faces the problem that it cannot deal applications with multi-stage behaviours.

Profiling the execution phases of an application can lead to optimizing the utilization of the underlying resources. Based on this idea, Jian Zhang et al presented a novel system-level application resource demand phase analysis and prediction prototype to support on-demand resource provisioning [84]. The solution only focused on long running application. Furthermore, the paper did not describe the details about what clustering method was used, how about the parameters in the cost function, what is the training method etc.

Integration of multiple predictors can be expected to achieve higher prediction accuracy compared with single predictors. A general integration framework was proposed in [85] that used K-Nearest Neighbor (k-NN) algorithm for selecting K best predictors for prediction. Some other predictive models can be easily integrated into the general framework in this paper. We have integrated ARM (for relatively long range prediction) and MMa (for short range prediction) models into this framework.
A two-level resource management system was proposed to dynamically allocate resources to individual virtual containers, through the adoption of fuzzy logic techniques [86]. In this model, the decisions at second level are easily obtained through the prediction at the first level. And also, it can predict resource demand with both workload and historical resource consumption as inputs to pursue more accurate prediction.

Efficient scheduling schemes require modeling and prediction of rendering workload. N.D Doulamis et al addressed the problem combining fuzzy classification and neural network model [87]. Neural networks can approximate non-linear relationship between inputs and outputs with very high accuracy. But the fuzzy set used in the paper may cause inaccuracy to some extent. Besides, more details need to be clarified, such as what feature vector was used, how to determine parameter Q.

Understanding the nature of enterprise workloads is crucial to properly designing and provisioning current and future services. Gmach et al [88] characterized workload demand patterns and predicted its future resource demand according to its periodic behavior. The experiments in the paper demonstrated very high predictive accuracy. But the solution only fits for applications with characteristic of periodic behaviours.

An Agent-based Simulator for Compute (resource) Allocation (ASCA) was described in [89]. ASCA simulated a set of pre-configured business and system policy setting for the purpose of providing a risk-free environment. The approach can discover the relationship between job slot utilization and average job waiting time. The approach also adopted dynamic job slot configuration capability over existing static job slot configuration capability in Netbatch. But the paper did not describe how to determine the parameters in detail.

Tan et al [90] discussed the details of the compute resource management strategy that was central to the success of the DCV program. The program results showed a significant increase in compute utilization and a large amount of dollar saving. The approach realized reservation-based sharing of global resources through a combination of HA and SA. But the approach focused on batch design computing demand, not generalized job requests. And also the experimental results need to be verified and clarified further.

A novel and systematic approach is proposed to profile services for resource optimization and capacity planning in [91]. By service profiling, the weakest points that may deteriorate system performance can be discovered and removed or optimized. The approach can only find invariant linear relationships between measurements and complicated nonlinear or stochastic models need to be built manually to link the other disconnected measurements.

Hui Li addressed the challenges in Grid scheduling and performance evaluation [92]. In his thesis, he described the basic statistical methods, analyzed statistical properties of workload and pseudo-periodic job arrivals, discussed long range dependence (LRD), fractal behavior modelling and correlated workload attributes modeling, carried out simulation studies of grid scheduling strategies and proposed a performance predictions framework based on machine learning technology.

Padgett et al presented an SLA management architecture for negotiation, monitoring and policing of SLAs, which provided SLA manager rule based adaptation to generate run-time prediction and track application service progression [93].
Snowdon et al [94] presented a methodology, based on off-line hardware characterization and runtime workload characterization, for the generation of an execution time model. The model can be used to implement operating-system level dynamic voltage and frequency scaling schemes for embedded systems.

Smith et al [95] addressed the problem of predicting how long applications will wait in a queue until they receive resources and the problem of improving scheduling performance using run-time predictions.

De Jonge et al [96] concentrated on predicting memory consumption in component-based applications. A model was proposed in which individual resource estimations of components can be combined, which were then used in scenarios (which model run-time behavior) to predict memory consumption of applications.

Carrera et al [97] addressed the problem of managing heterogeneous workloads in a virtualized data center. The technique dynamically modified workload placement by leveraging control mechanisms such as suspension and migration, and strived to optimally trade off resource allocation among these workloads.

3.2.2 Service-Level Prediction

Monitoring the preservation QoS properties during the operation of systems at runtime is an important verification measure for checking if the current service usage is compliant with the agreed SLAs. However it does not always provide sufficient scope for taking control actions against violations as it only detects problem after they occur. Prediction techniques aim to detect potential violations of QoS properties before they occur for taking control actions so that to avoid them to occur.

Intrusion Detection Systems (IDS) first attempted to solve the problem of identifying a violation of a system’s rule. An IDS is software and/or hardware designed to detect unwanted attempts at accessing, manipulating, and/or disabling of computer systems, mainly through a network, such as the Internet. These attempts may take the form of attacks, as examples, by crackers, malware and/or disgruntled employees.

Many different IDS techniques and approaches have been studied [104]. Existing approaches to intrusion detection have been distinguished into anomaly-based and misuse-based. Anomaly-based approaches [98, 99, 100] assume that attacks involve some abnormal behavior of the system that is being monitored. Intrusions are, thus, detected as deviations from the expected normal behavior of the system. Misuse-based approaches [101, 102, 103], on the other hand, are based on models of known attacks.

Unfortunately, these approaches are only able to detect an intrusion/violation at the moment it happens not always providing sufficient scope for taking control actions against violations. Therefore, something more it is needed, something that provides the system with enough time to react before a violation happening. Predictive techniques are thus needed.

Both academic and industry have been spending their efforts in studying the challenging field of predictive analytics. Predictive analytics is an area of statistical analysis that deals with extracting information from data and using it to predict future trends and behaviour patterns. The core of predictive analytics
relies on capturing relationships between explanatory variables and the predicted variables from past occurrences, and exploiting it to predict future outcomes.

The approaches and techniques used to conduct predictive analytics can broadly be grouped into regression techniques and machine learning techniques.

Regression models are the mainstay of predictive analytics. The focus lies on establishing a mathematical equation as a model to represent the interactions between the different variables in consideration. Depending on the situation, there is a wide variety of models that can be applied while performing predictive analytics. Some of them are: linear regression model [105], discrete choice models [106, 107], logistic regression [108], probit regression [109], time series models [110], survival analysis [111], and multivariate adaptive regression splines [112].

Machine learning was originally employed to develop techniques to enable computers to learn. Today, since it includes a number of advanced statistical methods for regression and classification, it finds application in a wide variety of fields including medical diagnostics, credit card fraud detection, face and speech recognition and analysis of the stock market. In certain applications it is sufficient to directly predict the dependent variable without focusing on the underlying relationships between variables. In other cases, the underlying relationships can be very complex and the mathematical form of the dependencies unknown. For such cases, machine learning techniques emulate human cognition and learn from training examples to predict future events. Some machine learning techniques are: neural networks [113], radial basis functions [114], support vector machines [115], naïve Bayes [116], and k-nearest neighbours [117].

Even if both regression and machine learning techniques can be used for runtime prediction of future system's behaviours, there are many factors which affect the decision of using one technique instead of another, e.g., its application fields, its preconditions, and its computational time. The latter property is very important in failure/violation prediction. Indeed, the smaller is the computational time required for making a prediction, the bigger is the scope for taking control actions against likely failure/violation.

From a temporal point of view, prediction techniques can be roughly classified in long-term and short-term prediction. Long-term prediction techniques use statistical models to foresee events that might happen after months or years from the time moment the prediction is made. These techniques are mostly used in financial and social sciences to predict economic or social trends, e.g., stock market quotations or populations evolution. On the other hand, short-term prediction techniques use statistical model to foresee events that might happen after days, hours, minutes or even seconds. These techniques are mostly used in fields characterised by a high level of dynamicity, e.g., networking and runtime systems. Both long- and short-term prediction techniques base them predictions on past data.

The most of the approaches discussed above focus on the prediction of single properties, e.g., CPU load, throughput, and disk usage. This kind of properties can be well modelled using statistic models. Moreover, historical data about these properties are easy to collect using many available monitoring techniques (static and dynamic code instrumentation) and tool (Ganglia [118], Nagois [119], MonALISA [120]).

There are many other properties that are useful to be predicted at service level, e.g., correctness of an interaction protocol, and risk degree of some action.
Unfortunately, these properties are often complex and cannot be predicted with the above techniques, if used separately.

Approaches addressing these issues can be found in the autonomic computing field [121] that, according to IBM, can be divided in the following four functional areas:

- **Self-Configuration**: Automatic configuration of components;
- **Self-Healing**: Automatic discovery, and correction of faults;
- **Self-Optimization**: Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements;
- **Self-Protection**: Proactive identification and protection from arbitrary attacks.

The two fields that most are related to predictive issues are self-healing and self-protecting. Moreover, that latter explicitly take into account proactive abilities to identify likely threats that might lead the system to an unstable state.

A self-protective approach for identifying likely dangerous actions is presented in [122]. The approach is named From Failure To Vaccine (FFTV). FFTV observes values at relevant program points. When the technique detects a software failure, it uses the collected information to identify the execution contexts that lead to the failure, and automatically enables mechanisms for preventing future occurrences of failures of the same type. Thus, failures do not occur again after the first detection of a failure of the same type.

FFTV uses different techniques combined together to produce its prediction. It uses interaction and data model. The former is represented using Extended Finite State Machines (EFSM), whilst the latter is represented using invariants over collected data.

A predictive approach using predictive data mining for intrusion detection (ID) and www prediction (WWW) applications is presented in [123]. Data Mining is an analytical process to analyze, explore, and summarize large amounts of data in order to uncover new patterns and/or to discover new relationships between variables. Predictive data mining is the most common type of data mining and it has the most important business applications.

ID uses Support Vector Machines (SVM) for classification. The SVM is one of the most successful classification algorithms in the data mining area, but its long training time limits its use. The authors also present a study for enhancing the training time of SVM, specifically when dealing with large data sets, using hierarchical clustering analysis. They use the Dynamically Growing Self-Organizing Tree (DGSOT) algorithm for clustering. Clustering analysis helps find the boundary points, which are the most qualified data points to train SVM, between two classes. They present a new approach of combining SVM and DGSOT, which starts with an initial training set and expands it gradually using the clustering structure produced by the DGSOT algorithm.

WWW Prediction is the problem of predicting the next page(s) a user might visit after surfing a web site. The improvement of many applications depends on surfing prediction. They propose a hybrid model that combines three classification techniques, namely, Support Vector Machines, Markov model, and Artificial Neural Networks, to resolve prediction using Dempster’s Rule. Such fusion overcomes the inability in predicting the unseen data in the case of Markov model and the complexity of multi-class problem in the case of Artificial Neural Networks and Support Vector Machines, especially when dealing with large number of classes.
A predictive approach for intrusion detection is presented in [124]. The authors present a network intrusion attempts prediction model based on fuzzy neural network which is based on the observation of network packet sequences.

Also in this case the authors do not use only one technique for addressing the consider issues but use a combined approach based on classic neural network enhanced with the fuzzy logic theory.

### 3.3 Infrastructure level Prediction (INTEL)

Intel focuses mainly on infrastructure level prediction and monitors computing resource consumption at infrastructure level with Ganglia system.

#### 3.3.1 Technical evaluation

Based on the review and analysis of about 20 technical papers and a PhD thesis, we made a matrix for algorithm contrast as shown in Table 1.
<table>
<thead>
<tr>
<th>Reference</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Techniques Involved</th>
<th>Parameters to be determined</th>
<th>Accuracy</th>
<th>Efficiency</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Ability to deal with our data</th>
<th>Assumption</th>
<th>Availability of Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>[85]</td>
<td>CPU load, Memory Usage, I/o activity etc</td>
<td>same as inputs</td>
<td>PCA, KNN, AR, MModel-a, ARM, ten-fold validation</td>
<td>slice window m, order p</td>
<td>improvement in almost half cases</td>
<td>no statement, may be reasonable</td>
<td>Higher accuracy, Less prediction time than other integrated model</td>
<td>higher training computational cost</td>
<td>Yes</td>
<td>none</td>
<td>most</td>
</tr>
<tr>
<td>[88]</td>
<td>resource usage demand</td>
<td>resource demand</td>
<td>Autocorrelation function, Fourier transformation, Chi-Square test</td>
<td>none</td>
<td>about 90%</td>
<td>no statement, may be high</td>
<td>high accuracy</td>
<td>only suit for data with periodic property</td>
<td>Partly</td>
<td>Data has periodic property</td>
<td>Most</td>
</tr>
<tr>
<td>[86]</td>
<td>workload, or resource usage</td>
<td>resource demand</td>
<td>Fuzzy set, Center clustering method</td>
<td>none</td>
<td>about 95%</td>
<td>no statement, may be very high</td>
<td>Very simple idea</td>
<td></td>
<td>Yes</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>[87]</td>
<td>rendering (general, object) descriptor</td>
<td>workload</td>
<td>fuzzy set, Neural network Q: Number of fuzzy sets of each dimension.</td>
<td>about 90%</td>
<td>no statement, should be very high</td>
<td>Neural network can approximate any non-linear function with high degree of accuracy.</td>
<td>The manual assignment of Q may degrade the accuracy</td>
<td>Yes</td>
<td>none</td>
<td>most</td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Inputs</td>
<td>Outputs</td>
<td>Techniques Involved</td>
<td>Parameters to be determined</td>
<td>Accuracy</td>
<td>Efficiency</td>
<td>Advantages</td>
<td>Disadvantages</td>
<td>Ability to deal with our data</td>
<td>Assumption</td>
<td>Availability of Implementations</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>---------</td>
<td>---------------------</td>
<td>-----------------------------</td>
<td>----------</td>
<td>------------</td>
<td>------------</td>
<td>--------------</td>
<td>-----------------------------</td>
<td>------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>[92] Ch. 8</td>
<td>Job attributes, resource state attributes, policy attributes</td>
<td>Running time, waiting time</td>
<td>Fuzzy set, clustering, genetic search, M-tree</td>
<td>Pivot attribute</td>
<td>30%</td>
<td>Run time: 20ms; Wait time: 300ms</td>
<td>Adapted tuning can guarantee good accuracy. Pivot attribute can be easily identified</td>
<td>Adapted tuning may cause inefficiency</td>
<td>partly</td>
<td>None</td>
<td>Most</td>
</tr>
<tr>
<td>[81]</td>
<td>CPU usage, memory usage</td>
<td>resource provisioning policy</td>
<td>Fuzzy set, Temporal difference, reinforcement learning</td>
<td>Transfer cost, backlog of partition, etc.</td>
<td>N/A</td>
<td>no statement, should be high</td>
<td>resource reallocation without resource demand prediction.</td>
<td>Backlog of partition may cause inaccuracy.</td>
<td>Partly</td>
<td>Decomposable Objective function, Input Markovian propert</td>
<td>no</td>
</tr>
<tr>
<td>[82]</td>
<td>application performance data</td>
<td>Application class, cost</td>
<td>PCA, k-NN</td>
<td>simple idea</td>
<td>about 95%</td>
<td>15 ms/sample</td>
<td>resource management base on several classes may be not accurate</td>
<td>not sure</td>
<td>none</td>
<td>most</td>
<td></td>
</tr>
<tr>
<td>[83]</td>
<td>application performance data</td>
<td>Application class, cost</td>
<td>Bayesian Network, k-NN</td>
<td>automatical data selection</td>
<td>about 95%</td>
<td>&lt; 20 ms/sample</td>
<td>same as above</td>
<td>not sure</td>
<td>none</td>
<td>most</td>
<td></td>
</tr>
<tr>
<td>[84]</td>
<td>system level performance data</td>
<td>next phase clustering algorithms, phase classifier training method</td>
<td>detailed definition of the total cost function</td>
<td>no</td>
<td>no statement, should be high</td>
<td>Maybe only suitable for long running application</td>
<td>not sure</td>
<td>none</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Inputs</td>
<td>Outputs</td>
<td>Techniques Involved</td>
<td>Parameters to be determined</td>
<td>Accuracy</td>
<td>Efficiency</td>
<td>Advantages</td>
<td>Disadvantages</td>
<td>Ability to deal with our data</td>
<td>Assumption</td>
<td>Availability of Implementations</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>---------</td>
<td>---------------------</td>
<td>----------------------------</td>
<td>----------</td>
<td>------------</td>
<td>------------</td>
<td>--------------</td>
<td>-----------------------------</td>
<td>------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>[80]</td>
<td>Performance attributes, power attributes</td>
<td>management policy</td>
<td>optimization</td>
<td>none</td>
<td>N/A</td>
<td>no statement, should be very high</td>
<td>very simple idea</td>
<td>the utility function may not fit the real data</td>
<td>not sure</td>
<td>none</td>
<td>easy to implement</td>
</tr>
</tbody>
</table>

Table 1: A matrix for algorithm contrast
3.3.2 Improvement and experimental analysis

By analysis of and comparison among the algorithms in the matrix and also according to our requirement, we proposed three improved algorithms each is suitable for a specific usage scenario. Our improvements targeted two drawbacks of the existing algorithms, low prediction accuracy and incapability of long time step prediction. We evaluated effectiveness of the three algorithms by experimental analysis on datasets from ORC.

An improved framework of multiple predictor integration (Improved MPI)

We presented an improved algorithm Based on the work of Jian Zhang et al [85], which integrated more predictive models and can make long time step prediction. The improvements to the original algorithm are as follows. For more choice of predictor, we added a ARM model and a Multi-Resource Model into the framework, which can support multi-dimensional inputs.

1. As for the particularity of similarity calculation of time series, we adopted an improved similarity measurement.

2. To enable the algorithm long time step prediction, we reconstructed the learning process of the algorithm.

The flow chart of the improved algorithm is shown in Figure 31.
Experimental Analysis

We implemented the algorithm in Java. The experiments were carried out on a laptop with a 1.86GHz CPU and 1G RAM, and running Windows XP Professional. We used in our experiments the datasets collected from Ganglia system that monitored Open Reference Case.

The first group of experiments is to evaluate improved MPI compared with the MPI in terms of one step predication, where the predicted metric is CPU_Idle. The average absolute errors (average(abs(x'-x))) was used to evaluate the accuracy of the two algorithms. Two data sets were obtained by generating one sample per
5 seconds and 10 seconds respectively. Each data set includes about 5000 samples, in which 4000 samples for training and 1000 samples for testing. So the time ranges of the two samples are about 7 and 14 hours long respectively.

The experimental results on 5S datasets are shown in Figure 32 and Figure 33. The average absolute errors of MPI and improved MPI on 5S datasets are 11.196 and 7.972 respectively. The experimental results on 10S datasets are shown in Figure 34 and Figure 35. The average absolute errors of the two algorithms on 10S datasets are 15.622 and 10.546 respectively. By contrast of the experimental results shown in Figure 32 to Figure 35, we can see the effectiveness of our improvements to MPI. By analyzing the experimental results and the models selected for prediction in different time slice, we can find there isn’t one predictor that performs best for all data samples.

![Figure 32: Experimental Results of MPI on 5S Dataset](image)

![Figure 33: Experimental Results of Improved MPI on 5S Dataset](image)

![Figure 34: Experimental Results of MPI on 10S Dataset](image)
To evaluate the improved MPI in terms of long time step prediction, we performed prediction for 10 steps with respect to three metrics, including Mem_Free, Pack_In and CPU_Idle. Three datasets, Mem_Free 240S, Pack_In 240S, and CPU 5S were used in the experiments. The reason we chose datasets of Mem_Free and Pack_In with interval of 240 seconds is because the metrics in the two datasets changed little with small intervals.

The experimental results on the three datasets are shown in Figure 36 to Figure 38. The average relative errors of improved MPI for 10 step prediction on Mem_Free 240S, Pack_In 240S, and CPU 5S are 23.57%, 16.61% and 45.04% respectively. Overall, the accuracy of 10 step prediction of improved MPI is reasonable, but the algorithm showed poor performance when resource consumption changed very sharply, such as that in Figure 36 and Figure 38.
An improved algorithm based on fuzzy logic and clustering (improved FLC)

Based on the work of Jing Xu et al [86], we presented an improved algorithm, which made the following improvements to the original algorithm.

1. In the original algorithm, the sum of the membership of an object isn’t equal to 1, which is unreasonable and also raised dramatic huge errors in our experiments. We redefined the membership function to fix this problem.

2. For better clustering results, we improved the method for tuning clustering parameters.

3. For dealing with little change of the input attributes, we integrated the Last Model into the original algorithm. When the input is similar with the last one, then directly makes prediction with the last output without clustering.

The flow chart of the improved algorithm is as follows.
Experimental analysis

We implemented the algorithm in Java and carried out the experiments in the same hardware and software environment. We used in our experiments the datasets from Open Reference Case. The average relative error was used to evaluate the accuracy of the two algorithms.

We evaluated the improved algorithm compared with the original algorithm in terms of one or two step prediction in the first group of experiments. Three metrics, including Mem_Free, Byte_In and CPU_Idle were predicted with three datasets, Mem_Free 240S, Byte_In 240S, and CPU_Idle 30S from ORC. The reason we chose datasets of Mem_Free and Byte_In with interval of 240 seconds is also because the metrics in the two datasets changed little with small intervals.

The experimental results on the ORC datasets are shown from Figure 39 to Figure 42. The average relative errors of the improved algorithm on Mem_Free 240S, Byte_In 240S and CPU_Idle 30S are 10.40%, 28.51% and 23.99% respectively. But the original algorithm got very huge prediction error on the three datasets, the relative errors were even beyond 100%. The reason of the low performance is mainly because of the unreasonable definition of membership function.

Figure 39: Flow Chart of Improved Fuzzy Logic-Based Approach
To evaluate the improved PLC in terms of long time step prediction, we also performed prediction for 10 steps with respect to Mem_Free, Byte_In and CPU_Idle. The experimental results are shown in Figure 43 to Figure 45. The average relative errors of improved PLC for 10 step prediction on Mem_Free 240S, Byte_In 240S, and CPU_Idle 30S are 22.50%, 22.75% and 25.01% respectively. The accuracy of 10 step prediction of improved PLC is also reasonable on the whole, but the algorithm also showed poor performance when resource consumption changed very sharply, such as those in Figure 43 and Figure 45.
Besides, comparing Figure 43 to Figure 45 with Figure 36 to Figure 38, we can see the prediction accuracies of improve MPI and improved PLC are comparable.

**Figure 43: Experimental Results of Improved PLC for 10 Step Prediction on Mem_Free 240S**

**Figure 44: Experimental Results of Improved PLC for 10 Step Prediction on Byte_In 240S**

**Figure 45: Experimental Results of Improved PLC for 10 Step Prediction on CPU_Idle 30S**
An algorithm for prediction of resource usage with periodicity (improved PP).

We presented an improved algorithm for resource usage prediction with periodicity, namely improved PP, based on the algorithm by Daniel Gmach et al [88]. We modified the calculation method of period candidates for getting the real period. Concretely, we expanded the period candidate set by adding left and right point of the existing candidates for capturing rough periodic behaviours. We deleted the ACF values that are less than 0 and the candidates with less than 3 ACF values from the calculation of average period energy. We improved the weighting formula in calculation of periodic pattern. We adopted a multinomial replacing the linear function to fit more complicated trends of periodic patterns. The flow chart of the improved algorithm is shown in Figure 46.

![Figure 46: The Flow Chart of Improved PP](image-url)
Experimental analysis

We implemented the algorithm in Java and carried out experiments in the same hardware and software environment. In our experiments, we used only one Byte_In dataset from Open Reference Case because other datasets didn’t show the periodic property. We also generated a synthesised dataset for experiments. The average relative error was used to evaluate the accuracy of the algorithms.

We evaluated the improved algorithm compared with the original algorithm on Byte_In dataset and a manually synthesised dataset. The experimental results on the two datasets are shown in Figure 47 and Figure 48. In Figure 47, we can see significant improvement of the prediction accuracy caused by improved PP compared with the original algorithm. The average relative errors of improved PP and the original algorithm are 16.91% and 21.30% respectively. While in Figure 48, the average relative errors of improved PP is 15.27% compared with 16.74%, that of the unimproved counterpart. We can also see somewhat improvement to the original algorithm in accuracy.

![Figure 47: Experimental Comparison of Improved PP and Original Algorithm on Byte_In](image)

![Figure 48: Experimental Comparison of Improved PP and Original Algorithm on a Synthesised Dataset](image)

We also evaluated the improved PP in terms of long time step prediction. We performed prediction for 100 steps on the same datasets. The experimental results are shown in Figure 49 and Figure 50. The average relative errors of improved PP for 100 step prediction on Byte_in dataset and the synthesised dataset are 20.20% and 18.94% respectively. We can see that improved PP
achieved good accuracy even for 100 step prediction, which are comparable with those by improve MPI and improved PLC for 10 step prediction. That means improved PP can look much further ahead than improved MPI and improved PLC when the resource usages have periodic property.

Figure 49: Experimental Results of Improved PP for 100 Step Prediction on Byte_In

Figure 50: Experimental Results of Improved PP for 100 Step Prediction on a Synthesised Dataset

3.4 Service level Prediction (CITY)

3.4.1 Introduction

Our requirements monitoring framework has been designed with the objective to support two different monitoring scenarios for service-based systems (SBS) using a non intrusive approach. The two key features of this approach are that monitoring is performed in parallel with the operation of an SBS without affecting its performance and does not require the instrumentation of the composition process of an SBS system or the individual services deployed by it.

In the first of the assumed monitoring scenarios (Scenario 1), a human user (typically the provider of an SBS) can request the framework to monitor whether the runtime operation of the system satisfies certain requirements and view any deviations from these requirements as soon as they are detected.
In the second scenario (Scenario 2), the monitoring can be requested by the environment that executes the process of an SBS. In this scenario any deviations of the requirements which are being monitored are reported back to the system which requested the execution of the monitoring activity.

In both these scenarios, the input to the monitoring framework is a *monitoring policy* that contains the formulas to be monitored and other monitoring parameters.

### 3.4.2 Monitoring policies

The behavioral properties and assumptions (monitoring policies or rules) of SBSs that need to be monitored are expressed in event calculus (EC).

EC is a logic language based on first-order predicate calculus that can be used to represent and reason about the behavior of dynamic systems. In EC, system behaviour is specified in terms of *events* and *fluents*. An event is something that occurs at a specific instance of time and may change the state of a system (e.g., invocation of an operation, assignment of a value to a variable) while a fluent signifies a system state. In an SBS, such states are expressed as conditions over the values of variables of the system composition process.

Events and fluents, and the effects that the former have to the latter, are specified using the following EC predicates:

- *Happens*(\(e, t, \mathcal{R}(t_1, t_2)\)) - This predicate signifies the occurrence of an event \(e\) at some time \(t\) that is within the time range \(\mathcal{R}(t_1, t_2)\).

- *InitiallyP*(\(f\)) - This predicate signifies that a fluent \(f\) holds at time 0.

- *Initializes*(\(e, f, t\)) - This predicate signifies that a fluent \(f\) starts to hold after the event \(e\) at time \(t\).

- *Terminates*(\(e, f, t\)) - This predicate signifies that a fluent \(f\) ceases to hold after the occurrence of event \(e\) at time \(t\).

- *Holds*(\(f, t\)) - This predicate signifies that the fluent \(f\) holds at time \(t\).

Fluents are specified using the following terms:

- *equalTo*(\(x, y\)) - This term signifies that the value of the fluent variable \(x\) is equal to \(y\).

- *greaterThan*(\(x, y\)) - This term signifies that the value of the fluent variable \(x\) is greater than \(y\).

- *lessThan*(\(x, y\)) - This term signifies that the value of the fluent variable \(x\) is less than \(y\).

Events and fluents can be composed together in formulas. A formula represent a monitoring policy and its logical syntax in Extended Backus Naur Form (EBNF) [125, 126] is presented in Figure 51.

---

\(^1\) *Happens*(\(e, t, \mathcal{R}(t_1, t_2)\)) in our framework is equivalent to the formula:

\[\text{Happens}'(e, t) \land (t_1 \leq t) \land (t \leq t_2)\]

where *Happens*(\(e, t\)) is the predicate that signifies an event occurrence in standard event calculus.
The rules to be monitored at runtime are specified in terms of the above predicates and have the general form \( \text{body} \Rightarrow \text{head} \). The meaning of a rule is that if its \text{body} evaluates to True, its \text{head} must also evaluate to True. The \text{Happens} predicates in a rule with no constraints for their lower and upper time boundaries are what we call “unconstrained” predicates. During the monitoring process, rules are activated by events that can be unified with the unconstrained \text{Happens} predicates in them. When this unification is possible, the monitor generates a rule instance to represent the partially unified rule and keeps this instance active until all the other predicates in it have been successfully unified with events and fluents of appropriate types or it is deduced that no further unifications are possible. In the latter case, the rule instance is deleted. When a rule instance is fully unified, the monitor checks if the particular instantiation that it expresses is satisfied.

For instance, let's considering the scenario in which a master controller wants to check the availability of a specific resource. To perform this check the master control sends a signal to the desired resource and then it waits for 10 time units for a replay. If within 10 time units the master controller does not receive any response, then the rule is violated. This would be a bounded availability check which can be expressed in \text{EC-Assertion} by the following monitoring rule:

**Rule-1:**

\[
\text{Happens}(e(_e, _\text{masterController}, _\text{resource}, \text{REQ}, \text{ping}), t_1, \mathcal{R}(t_1, t_1)) \\
\Rightarrow (\exists t_2: \text{Time}, e_2: \text{String}) \\
\text{Happens}(e(_e_2, _\text{resource}, _\text{masterController}, \text{RES}, \text{pong}), t_2, \mathcal{R}(t_2+1, t_2+10))
\]

In the Rule-1 an event \( e \) is defined by the tuple \{eventId, sender, receiver, eventType, operation\}.

### Prediction

In some cases, the detection of violations of SLA monitoring rules after they occur might not be sufficient to guarantee a quality service. This is because the required action for doing so can be expensive to take or because no such action may be possible, since the violation has already occurred. Thus, in addition to the detection of occurred violations, it is also important to be able to predict whether violations of SLA monitoring rules are likely to occur in some future state during the operation of a system. CITY monitoring framework provides support for predicting potential violations of monitoring rules, referred to as threats.
A threat is defined as a potential violation of an SLA monitoring rule and is associated with a belief measure indicating how likely the violation is, given the current state of the system that is being monitored.

**Belief probabilities**

A key objective in threat detection is to estimate the threat likelihood for a particular rule before a violation occurs. To measure this likelihood we use belief measures founded in reasoning framework of the Dempster Shafer theory of evidence [128]. This is due to the need to cope with uncertainty regarding the events that have been seen at the different monitoring states of the system, which makes the use of classic probabilistic reasoning inappropriate for our needs.

The calculation of threat likelihood requires the measurement and combination of beliefs of three different types:

1. basic probabilities in the genuineness of events that have been recorded in the log of the monitor,
2. basic probabilities in the occurrence of an event of a specific type within a time range that is determined by another event,
3. basic probabilities in the validity of the derivation of the negation of an event when another event’s occurrence indicates that the time range within which the former event should have occurred has elapsed.

The calculation of basic probabilities of the first of the above types (i.e., the basic probabilities of the genuineness of events) is based on a basic probability assignment function that has been defined in [129]. The second type of basic probabilities used in threat detection provides measures of the likelihood of the occurrence or not of an event when another event E_j that E_i is temporally constrained by has occurred. The basic probability function that is used to provide these measures is defined as follows:

**Definition 1:**
The conditional basic probability in the occurrence of an event E_i within the time range determined by another valid event E_j that has occurred, m_{ij}, is defined as:

\[
\begin{align*}
m_{ij}(X) &= \begin{cases} 
      k_{ij} = \frac{\sum_{e \in \text{Log}(E_j)} m(e) \sum_{e \in \text{Log}(E_i \setminus \text{Log}(E_j))} m(e) (-1)^{|\text{Log}(E_i \setminus \text{Log}(E_j))|} \prod_{e \in \text{Log}(E_i \setminus \text{Log}(E_j))} m(e)}{\sum_{e \in \text{Log}(E_i)} m(e)} & \text{if } X = e_i \\
      k_{i}' = \frac{\sum_{e \in \text{Log}(E_i)} m(e) \sum_{e \in \text{Log}(E_i \setminus \text{Log}(E_j))} m(-e_i) (-1)^{|\text{Log}(E_i \setminus \text{Log}(E_j))|} \prod_{e \in \text{Log}(E_i \setminus \text{Log}(E_j))} m(e)}{\sum_{e \in \text{Log}(E_i)} m(e)} & \text{if } X = -e_i \\
      1 - k_{ij} - k_{i}' & \text{if } X = e_i \lor -e_i 
   \end{cases}
\end{align*}
\]

In the above formula:
- \( \text{Log}(E_j) \) is the set of events of type \( E_j \) in the event log up to the time when \( m_{ij} \) is calculated. The size of \( \text{Log}(E_j) \) is a parameter of the algorithm. Furthermore, the events that constitute this set are randomly selected.
- \( \text{Log}(E_i | e) \) is the set of the events of type \( E_i \) in the event log that have been occurred within the time period determined by \( e \) and up to the time point when \( m_{ij} \) is calculated. The size of \( \text{Log}(E_i | e) \) is a parameter of the algorithm.
- \( I \in \wp(\text{Log}(E_i | e)) \) denotes any set that is an element of the powerset of \( \text{Log}(E_i | e) \).
• \(m(e)\) is the basic probability assignment \(m_j(e)\) in the case of non negative events \(E_j\) or the basic probability assignment \(m^{NAF}_{ji}\) in the case of negative events \(-E_j\).

Function \(m_{ji}(X)\) in the above definition measures the basic probability of the occurrence of a genuine event of type \(E_i\) within the time range determined by events of type \(E_j\). This is measured as the average belief of seeing a genuine event of type \(E_i\) within the time range determined by a genuine event of type \(E_j\). To measure this belief, for each occurrence of an \(E_j\) event, \(m_{ji}(X)\) calculates the basic probability of seeing at least one genuine event of type \(E_i\) within the period determined by \(E_j\). Assuming that the set of such \(E_i\) events is \(Log(E_i|e)\), this basic probability is calculated by the sub-formula:

\[
\sum_{I \in \phi(\text{Log}(E_{i} | \text{e}))) \land I \neq \phi} (-1)^{|I|+1} \prod_{e_i \in I} m(e_i) .
\]

This sub-formula measures the basic probability of at least one of the events in \(Log(E_i|e)\) being genuine, i.e., an event that has at least one explanation confirmed by other events in the log of the system, and uses the basic probabilities of individual events \(m(E_i)\) for positive events, or the basic probability \(m^{NAF}_{ji}\) that is defined in Definition 2 below for negative events \(-E_j\). Thus, \(m_{ji}(X)\) discounts occurrences of events of type \(E_i\) which are not considered to be genuine. Also, by virtue of its functional form, the higher the number of genuine events of type \(E_i\) within the period determined by \(E_j\), the larger the basic probability of the occurrence of at least one genuine event of type \(E_i\) that \(m_{ji}(X)\) generates. \(m_{ji}(X)\) takes also into account the basic probability of the genuineness of each occurrence of an event of type \(E_j\) within the relevant period (this is measured by \(m_j(e)\)) and uses it to discount the evidence arising from \(E_j\) events which are not assessed to be genuine.

The basic probability functions that we have introduced above do not cover cases where the absence of an event is deduced by the negation as failure (NAF) principle. CITY monitoring framework uses this principle to deduce the absence of an event \(E\) (i.e. \(-E\)) that is expected to occur within a specific time range \([t_U, t_L]\) when it receives another event \(E'\) from the same event captor that should sent \(E\) with a timestamp \(t'\) that is greater than \(t_U\) \((t' > t_U)\) and has not received \(E\) up to that point. Considering, however, that the event \(E'\) that would trigger the application of the NAF principle in such cases might not be a genuine event itself, it is necessary to estimate the basic probability of \(-E\). The function that measures this basic probability is defined as follows:

**Definition 2:**

The basic probability in the absence of an event \(E_i\) or, equivalently, \(-E_i\) due to the application of the NAF principle when another event \(E_j\) occurs is defined as:

\[
m^{NAF}_{ji}(X) = \begin{cases} 
  m(e_j) & \text{if } X = e_j \\
  1 - m(e_j) & \text{if } X = e_j \lor e_j \\
  0 & \text{Otherwise}
\end{cases}
\]

Where \(m_j(E_j)\) is the basic probability in the genuineness of the event \(E_j\), see [129].
Threat Likelihood

To illustrate the mechanisms for threat detection we will use the following example of a monitoring rule Rule-2 that is specified to monitor the logging activity of the users of a system that is accessible from different distributed client devices.

Rule-2:
\[ \forall \_U: \text{User}; \_C1, \_C2: \text{Client}; \_C3: \text{Server}; t1, t2: \text{Time} \]
\[ \text{Happens}(e(\_e1, \_C1, \_C3, \text{REQ, login}(\_U, \_C1), \_C1), t1, R(t1,t1)) \land \]
\[ \text{Happens}(e(\_e2, \_C2, \_C3, \text{REQ, login}(\_U, \_C2), \_C2), t2, R(t1,t2)) \land \_C1 \neq \_C2 \]
\[ \Rightarrow \exists t3: \text{Time} \]
\[ \text{Happens}(e(\_e3, \_C1, \_C3, \text{REQ, logout}(\_U, \_C1), \_C1), t3, R(t1+1,t2+1)) \]

The rule states that if a user \_U logs on the system from some client device \_C1 (i.e., when the event \( e(\_e1, \_C1, \_C3, \text{REQ, login}(\_U, \_C1), \_C1) \) happens) and later he/she logs on from another device \_C2 (i.e., when the event \( e(\_e2, \_C2, \_C3, \text{REQ, login}(\_U, \_C2), \_C2) \) happens), by the time of the second login (t2), he/she must have logged out from the first device (i.e., an event \( e(\_e3, \_C1, \_C3, \text{REQ, logout}(\_U, \_C1), \_C1) \) must have occurred).

Monitoring Rule-2 can be used to prevent users from logging on from different devices simultaneously and, therefore, reduces the scope for masquerading attacks. Simultaneous logging provides scope for such attacks since a user who is logged on from different devices simultaneously may leave one of them unattended. When this happens, however, some other user may start using the unattended device with \_U’s credentials. Blocking logging attempts that violate Rule-2 would prevent such cases. Furthermore, monitoring Rule-2 could detect cases where some user gets hold of the credentials of another user \_U and tries to use them to log on with the identity of \_U at the same time when \_U is logged from a different device.

To represent the different ways of combining basic belief functions at run time in order to calculate the overall belief in a monitoring rule threat, CITY monitoring framework constructs a belief graph for each rule. The vertices of this graph represent the different event occurrence predicates in violation signature of the rule (i.e., the \text{Happens} predicates) and the directed labelled edges between the vertices indicate dependencies between the time variables of these events. The edges of the graph are derived from the boundaries of the time variables of the rule predicate which constrain these variables and, hence, the occurrence of events in the rule. The graph edges indicate how evidence can be propagated at runtime by combining the different basic belief functions that are associated with the observed events.

![Figure 52 Belief graph for Rule-2](image-url)
An example of a DS belief graph is shown in Figure 52. The graph represents the different paths of combining the basic belief functions for the events of Rule-2 and reflects the time dependencies between the different events. The occurrence of $E_2$ in the rule, for instance, depends on the occurrence of $E_1$ since the range of the time variable of $E_2$ (i.e., $\mathcal{R}(t_1,t_2)$) refers to the time variable of $E_1$ but not vice versa (the range $\mathcal{R}(t_1,t_1)$ of $t_1$ indicates that $E_1$ is an event with a not constrained time variable). Thus, an edge from $E_1$ to $E_2$ labelled by $m_{2|1}$ has been inserted in the graph as well as another edge from $E_2$ to $E_1$ labelled by $m_{1|2}$. Similarly, as the time range of the event $\neg E_3$ (i.e., $\mathcal{R}(t_1+1,t_2-1)$) refers to the time variables $t_1$ and $t_2$ of the events $E_1$ and $E_2$, the graph contains edges from $E_1$ to $\neg E_3$ and $E_2$ to $\neg E_3$. Note, however, that the graph does not contain an edge from $\neg E_3$ to $E_2$ or from $\neg E_3$ to $E_1$ as the former event cannot be derived by NAF unless $E_1$ and $E_2$ are received first. Finally, the graph includes edges from the starting node to $E_1$ and $E_2$. These edges are labelled by $m_1$ and $m_2$ representing the basic belief functions that are to be used when the occurrence or absence of the events $E_1$ or $E_2$ is established from the starting node.

At runtime, belief graphs are used to record the events matched with a given rule and violation signature and determine the combination(s) of basic belief functions that will be needed to compute the overall threat belief for the rule. In general, given a set of received and a set of unknown events, the overall belief for a rule is evaluated by combining the basic beliefs of the received events that match with the rule’s violation signature and the conditional beliefs for the unknown events. It should be noted that in such cases, there may be more than one known events in the graph which are linked directly with an unknown one. In such cases, the conditional belief in the unknown event $m_{ij}$ is computed by considering all paths which start from some known event $e_i$ and end in the unknown event $e_j$, without passing through any other known events (this ensures that known events will not be considered as supporting evidence for unknown ones multiple times). The algorithm for evaluating the overall belief in a rule threat given a belief graph is shown in Figure 53.

```
Compute_Threat_Belief(E_0, DSG_R, R)
1. find the sets of the known events KE and the set of unknown events UE in DSG_R
2. m = basic_belief (<start, E_0>)
3. CombinedBPA = {}

/* combine the basic beliefs of events in KE */
4. for each E_i in KE do
5.   m = m ⊕ basic_belief (<start, E_i>)
6. CombinedBPA = CombinedBPA ⊔ basic_belief (<start, E_i>)
7. end for

/* combine the BPAs of paths to unknown events */
8. for each e_i ∈ UE do
9.   insert all the paths from e_i to e_j which do not include any event in KE, into P_i
10. for each p ∈ P_i do
11.   for each edge L in p do
12.     if basic_belief (L) ∉ CombinedBPA then
13.       m = m ⊕ basic_belief (L)
14.     CombinedBPA = CombinedBPA ⊔ basic_belief (L)
15.     end if
16.   end for
17. end for
18. mark E_i as a known event in DSG_R
19. return (m(events(¬R), m(events(R))))
20. end Compute_Threat_Belief
```
3.4.4 Implementation

The monitor has been developed using the Java™ Standard Edition 6 platform (developer version number 1.6.0). Other third party libraries have also been used in order to speed up the development process, e.g., libraries for parsing an XML document and for connecting to databases.

Figure 54 shows the monitor high level architecture: the monitor main components and tool, and its databases. The monitor has two main components: Event receiver and Analyser, and three tools: Detection Of Violation Tool (DOVT), Threat Detection Tool (TDT), Event Genuineness Belief Tool (EGBT).

- Event Receiver: it is the monitor's component in charge for communicating with the outer world. It receives event coming from the monitored services and store them into the Event DB. Once a received event has been stored, it notifies the Analyser.
- Analyser: it is the monitor’s component in charge for reasoning over the monitoring rules. It knows how to handle event calculus formulas and uses DOVT for detecting a rule violation.
- DOVT: it is in charge for detecting whether or not a rule is violated

The EGBT and TDT are used for determining the genuineness of an event and detecting the threat likelihood associated to a particular instance of a monitoring rule, respectively.

![Figure 54: Monitor high level architecture](image)

Figure 55 shows the monitor APIs. The full external library list is shown in the three boxes on the right hand side of the figure. These third part libraries are provided by well known and reliable projects or companies, e.g., Apache project, MySQL, and DB4O. In Figure 55 we have omitted the JUnit libraries that were used for testing purposes, only.
On top of the monitor core APIs, that provide access to the monitors’ functionalities, we have implemented the EGBT, TDT, belief graph tool (BGT) and a set of interfaces to access the deployed data bases (DB).

The libraries MySQL and DB4O have been represented explicitly in the APIs schema to point out that the monitor has been re-designed to operate transparently with different data storage solutions. To achieve this flexibility the monitor provides its own interfaces for managing data storage and these APIs are provided by the interfaces in the package database. The actual implementation supports the relational database MySQL, the object database DB4O, and RAM primary memory used as storage.

Each of these storage solutions are mutual exclusive. For instance, if MySQL database is selected, then all data collected and computed by the monitor, e.g., event genuineness and threat probability, and its tools, e.g., TDT and EGBT, are stored into the database MySQL. Other solutions can be easily added just providing the specific implementations of the database interfaces.

The monitor exposes an interface trough witch it is possible to interact with it. It also provides a GUI interface for management purposes. Figure 56 shows the monitor infrastructure, inside the dotted box, and how the monitor is notified with events and how it communicates monitoring results.
3.4.5 Evaluation

The monitor has been evaluated in a series of experiments that have focused on the performance of the core monitoring process that is realised by the framework and the effect that it has on the performance of the systems that it monitors. A detailed account of this evaluation may be found in [127]. In the following, however, we summarise the main findings of the evaluation experiments of the framework to enable a better understanding of its capabilities and limitations.

More specifically, the evaluation has demonstrated that in the general case the time required to detect violations of monitoring rules after all the events that would enable this become available, increases exponentially with the number of the events that are sent to the monitor.

The violation detection time depends on the number of instantiated partially unified rules when the last event that enables making a decision about the violation or not of a rule becomes available. The latter number depends on the exact form of the rules that are being monitored and, thus, it may be reduced substantially for specific types of rules.

Finally, previously conducted experiments have indicated that the performance of the monitor is not significantly affected by the size of the domains of the variables used in monitoring rules. Also the evaluation in [127] has indicated that the overhead of event capturing on the performance of the system that is being monitored depends on the type of the deployed capturer. This overhead ranges from an 18% - 20% drop in performance, when events are captured from the execution platform of the application, to 800%, in cases where event capturers are implemented as wrappers of components of the system that is being monitored [127].

3.5 Challenges/Conclusions

In this chapter, we presented the current status of our runtime prediction approaches at infrastructure level and service level. At infrastructure level (INTEL, Section 3.3), run-time prediction plays an important role in dynamically re-
provisioning and migrating compute resources. Dynamic adjustment of resource allocation can be realized with assistance of run-time prediction. This can save operational and utility costs by suspending unutilized compute resource. At service level (CITY, Section 3.4), runtime prediction can be adopted for SLA violation avoidance. This reduces the risks for service providers and provides more stable QoS to service customers.

There are still some challenges we are facing. One is how to improve the prediction accuracy further. Currently, we only use historical resource usage and workload to train predictive models, how to take into account more useful metrics, such as some job attributes policies, and design time info to learn more accurate models is a research issue. Besides, the predictive algorithms should also be adjusted and adapted according to the particular requirements and data characteristics.

Another challenge is how to look far ahead with certain accuracy and confidence interval, such as from seconds to minutes. Currently, we can predict 10 time steps later in most cases, 100 time steps later in some cases with relatively high accuracy, but in most cases, after 10 time steps, the prediction accuracy will decline significantly. This is another research issue we should work on next.

The last challenge is how to integrate run time prediction with other prediction services, saying design-time prediction. Moreover, we want to take advantage of information coming from different levels, i.e., software, service and infrastructure. This information can be used to discover interplays among SLA violations and, for instance, data exchanged of operation performed so that to enhance our predictive models.

Currently, the relationship and interface between runtime prediction and other components is not clear. This is also an important research problem that should be paid efforts to.
4 Manageability Design

4.1 Overview

In SLA@SOI, we consider a scenario where an application is provided in terms of software services by a dedicated service provider. We assume that the application design is based on the principles of service-oriented architectures using web services technologies. Basic functionality is offered through atomic web services, whereas customer-specific processes are implemented by means of web service compositions using the BPEL standard. For designing and modelling such applications from a functional point of view, we decided to use the Service Component Architecture (SCA) Metamodel as a core metamodel as basis. Thus, all available services (atomic or composite) are modelled as service components offering their functionality through a provided service interface. The required Interfaces represent the dependencies between different service components. In case of a composite service, the component’s implementation is set to BPEL, which defines the component’s internal behaviour. This BPEL specification particularly specifies the order in which the required service operations are executed in a process-oriented manner.

SLA@SOI targets the development of a framework that allows for an SLA-driven management of such a service-oriented application. A customer may choose from a set of services and negotiate non-functional properties in terms of service level parameters and objectives (SLOs). This negotiation eventually results in a service level agreement (SLA) between the customer and the service provider. The provider is responsible to instantiate the service components required for fulfilling the services he has agreed on. In this context, he collaborates with an infrastructure provider who offers virtualized resources used for executing the services. Once they completed the provisioning of the service providing resources, the customer may use the service. During this service operation phase the service provider has to ensure that the provided services are compliant to the SLOs defined in the SLA. As a first step, we limit these SLOs to performance-related properties, like response time and throughput.

To determine the service levels of the involved service components or to predict likely SLA violation (see chapter 3.4) it has to be possible to monitor and control the performance-related properties of the components. The implementation of these management capabilities requires additional development steps and activities. This includes (i) the identification of WSC runtime measurements necessary for monitoring the performance as well as control operations used for monitoring, adjusting or predicting the performance at runtime; (ii) the implementation of a corresponding instrumentation of the service components in terms of sensors and effectors; (iii) the configuration of the SLA management framework (at least with SLOs, metrics as well as calculation and controlling rules).

Here, we face several challenges:

- The monitoring requirements are specific to the regarded service-oriented application and the scope of SLA management (in particular the scope of considered service level parameters). Therefore, it is not possible to provide a standard set of management capabilities. They rather have to be tailored to the particular needs.
- There are different service-oriented applications (one for each industrial use case) that are supposed to be managed by the SLA management


framework. As there is no common standard for instrumenting components and it cannot be foreseen, which management capabilities are already available, we have to find a way to deal with this heterogeneity.

- There are intrinsic interdependencies between the functional and the monitoring implementation. If for instance a service composition is changed, the corresponding monitoring and instrumentation in most cases has to be changed as well. Thus, there has to be support for maintaining coherency between the functional design/implementation and the manageability design/implementation.

Hence, we argue that a systematic development approach is necessary to take into account these manageability requirements from the very first.

The following section discusses relevant related work. After that, we introduce the general idea of an engineering methodology for developing manageable service components and describe its main parts, available so far.

In year 1 of the SLA@SOI project, we focused on the design of the manageability configuration metamodel, which serves as a basis for configuring the corresponding manageability infrastructure with scenario-specific monitoring requirements. The manageability configuration metamodel is limited to monitoring capabilities, whereas its extension by control capabilities will be addressed in year 2. The same holds for the design and implementation of the corresponding manageability infrastructure, which is presented in the A3 deliverable (Section on manageability design).

4.2 State of the Art

4.2.1 Model-Driven Monitoring Approaches

In [128] and [129] approaches are presented that promote an integration of Quality of Service (QoS) concerns into a model-driven development process for component-based applications. This particularly includes the annotation of component specification by QoS monitoring requirements and the automated generation of a corresponding monitoring infrastructure. [128] is aligned with the Common Information Model (CIM) and supports the automated generation of a WBEM-based QoS monitoring infrastructure including component instrumentation, whereas [129] still lacks a concrete implementation. Both approaches are promising but have to be adapted to SCA-based applications, particularly regarding the monitoring model and the instrumentation. Monitoring of complex transactions through compositions of components as well as the monitoring of the components’ internal behaviour/implementation are not possible at all. Also, control capabilities are not taken into account. [128] is even limited to a limited set of predefined QoS parameters and may not be extended by custom SLA parameters.

[130] focuses on the model-driven specification of SLAs as an activity that is independent from the functional design. This approach includes the definition of SLA parameters along with the required management metrics/indicators and the rules for calculating them. It is assumed that there already is a management infrastructure delivering the required (elementary) metrics. The instrumentation issue as well as a template based indicator specification is not addressed. Thus, our approach should be considered as supplementary to this.
In [131] a very similar approach is presented for implementing business performance monitoring requirements. Although the monitoring of Key Performance Indicators is not subject to SLA@SOI, it is still related to the problem we face. The authors introduce metamodels for specifying observation models for monitoring business performance and transformations to executable models. But again, the instrumentation of the managed resources and a templates-based specification are not included.

### 4.3 Manageable Service Engineering Methodology

This section introduces the methodology to provide a solution for the requirements outlined above. For the development of the solution, we propose a model-based approach according to the basic principles of Model Driven Software Development [132][133].

The solution consists of two parts (see Figure 57):
- The development of a Manageability Framework, and
- the development of models and guidelines for the actual Manageability Design of a certain service-oriented application.

The artefacts of the Manageability Framework are independent of a certain service-oriented application and are developed once (in the scope of this project). In contrary to this, the artefacts of the Manageability Design depend on a certain service-oriented application and have to be created again for each service-oriented application, i.e. for each use case.

![Figure 57: Solution building blocks](image)

The Manageability Framework comprises the Manageability Configuration Meta-Model and the Manageability Infrastructure (including the Manageability Data Model and the Instrumentation Guidelines). The development of the Manageability Infrastructure is subject to Task 3.4 in Workpackage A3 and its documentation can be found in the corresponding deliverable.

The Manageability Design has to be created by a Manageability Designer and an Instrumentation Developer according to a specific Functional Design. The Manageability Design includes the Manageability Models and the Instrumented Functional Design. To support this activity, guidelines and examples for the development of the Manageability Design are given.

Figure 58 shows a detailed, graphical illustration of our approach to the development of manageable service-oriented applications.
The use of SCA is a common means to hide the technological heterogeneity during the design phase of a service-oriented application [134]. Hence, SCA is used to build the Functional Design of the applications and the SCA Meta-Model influences the design of the Manageability Configuration Meta-Model.

The Manageability Configuration Meta-Model allows to define the objects relevant for the management of an SCA-based application system as well as the relations between these objects (Requirement 1). Thereby, the model has to cover all aspects of an SCA-based application design without limiting the flexibility of an SOA. To this end, the structure of the management information has to be independent of the actual service component structure (Requirement 3). The fact that one service composition or one service component can have multiple instances must also be considered by the Manageability Configuration Meta-Model. With this model at hand, the Manageability Designer is able to exactly specify the parts of the Functional Design relevant for the monitoring of SLAs. This management configuration process results in a concrete Manageability Model for the particular Functional Design of a service-oriented application. Thus, the Manageability Model is an instance of the Manageability Configuration Meta-Model.

The Manageability Model is, on the one hand, used by the Manageability Infrastructure Administrator to supply the Running Manageability Infrastructure with the static meta management information about the managed objects. Hereby, the role of the Manageability Infrastructure Administrator could also be replaced by an automated model-transformation. A transformation could, for example, create a script that automatically writes the data into the data repository of the responsible Management Agent within the Manageability Infrastructure. On the other hand, the Manageability Model provides information to the Instrumentation Developer about how and where he has to extend the Functional Design, i.e. the Manageability Model provides guidance on developing the Instrumented Functional Design (Requirement 2). Furthermore, additional Instrumentation Guidelines concerning the communication between the

---

**Figure 58: Overview to Engineering Methodology for Managable Services**

The use of SCA is a common means to hide the technological heterogeneity during the design phase of a service-oriented application [134]. Hence, SCA is used to build the Functional Design of the applications and the SCA Meta-Model influences the design of the Manageability Configuration Meta-Model.

The Manageability Configuration Meta-Model allows to define the objects relevant for the management of an SCA-based application system as well as the relations between these objects (Requirement 1). Thereby, the model has to cover all aspects of an SCA-based application design without limiting the flexibility of an SOA. To this end, the structure of the management information has to be independent of the actual service component structure (Requirement 3). The fact that one service composition or one service component can have multiple instances must also be considered by the Manageability Configuration Meta-Model. With this model at hand, the Manageability Designer is able to exactly specify the parts of the Functional Design relevant for the monitoring of SLAs. This management configuration process results in a concrete Manageability Model for the particular Functional Design of a service-oriented application. Thus, the Manageability Model is an instance of the Manageability Configuration Meta-Model.

The Manageability Model is, on the one hand, used by the Manageability Infrastructure Administrator to supply the Running Manageability Infrastructure with the static meta management information about the managed objects. Hereby, the role of the Manageability Infrastructure Administrator could also be replaced by an automated model-transformation. A transformation could, for example, create a script that automatically writes the data into the data repository of the responsible Management Agent within the Manageability Infrastructure. On the other hand, the Manageability Model provides information to the Instrumentation Developer about how and where he has to extend the Functional Design, i.e. the Manageability Model provides guidance on developing the Instrumented Functional Design (Requirement 2). Furthermore, additional Instrumentation Guidelines concerning the communication between the
Instrumentation and Management Agents are given. This allows capturing the specified management information during service execution by the Running Instrumented Service Components.

The Manageability Data Model describes an executable database schema that allows storing the information specified in the Manageability Model. It is derived from both the Manageability Configuration Meta-Model and the Common Information Model (CIM) standard (see Deliverable for WP A3, Task 3.4). The Manageability Configuration Meta-Model provides information about which kinds of managed elements have to be gathered and persisted. The CIM standard is widely-used and provides a common representation of a management information model. In order to reuse the relevant concepts of the standard and support a simple mapping to the standard, CIM forms the basis for the Manageability Data Model.

The Manageability Infrastructure exposes the gathered management data stored in the Manageability Data Model through a Manageability Interface. The interface provides methods to create, read, update and delete managed element instances of the Manageability Data Model. Thereby, it follows the structure of the CIM provider interfaces, but also provides additional operations not covered by CIM. Nevertheless, the proposed solution provides both a general and standard-based Manageability Interface (Requirement 6). According to the model-based procedure, the Manageability Data Model is at first designed standard independently and later mapped to the CIM standard.

During runtime, the Instrumentation of the functional service components captures the information about the running service instances, which is required for the SLA management. This information contains runtime information about each single instance e.g. start/end time as well as the information about the relation between the instances e.g. service component instance X operation 1 calls service component instance Y operation 2. This information is necessary to find, e.g. in case of an SLA violation, the error-prone component or the bottleneck within the service stack (Requirement 5).

The Running Manageability Infrastructure gathers the information provided by the Manageability Infrastructure Administrator and the Running Instrumented Service Components. This information is processed according to the specifications in the Manageability Model and is then persisted in a runtime instance of the Manageability Data Model (Requirement 4).

### 4.4 Manageability Configuration Metamodel for SCA

This chapter introduces the Manageability Configuration Meta-Model used to define management capabilities, which a service component should offer at runtime. The approach so far is limited to the specification of monitoring capabilities required for monitoring performance-related parameters. Control functions and the monitoring of availability or reliability are not considered yet.

Furthermore, we assume the application that is subject to the monitoring to be modelled by means of SCA. Thus, instances of the manageability configuration metamodel form a complementary view to an SCA-based application design. SCA is used as foundation for the manageability configuration since the specification is tailored to the concepts of SOA. It gives developers a single programming model for using services for all aspects of the service lifecycle, as well as a single a single programming model for using data sources. Furthermore, SCA is not tied to
a specific programming language, protocol, technology, or runtime environment. Nevertheless, it could also be modelled by any other modelling language, such as UML.

The goal of the Manageability Configuration Meta-Model is to enable the Manageability Designer to specify the elements within an SCA-based service-oriented application relevant to SLA Management, which for instance comprises monitoring adherence of an SLA or analysing SLA violations. As already mentioned, the focus is thereby on the monitoring of performance-related properties, more precisely response time and throughput. Within an SLA various aggregated metrics and statistical values are used to describe the performance-related requirements of a service. In order to calculate these values, raw measurement data is needed. This data describes an instance of an “action” or a “Unit of Work” (as known from the CIM Metrics Model [135][136]) executed by the application system. Each action thereby belongs to a certain managed element (e.g. a service component in SCA). As a managed element mostly includes multiple actions and often depends on other managed elements, these actions form complex transactions that span over multiple managed elements. Since we assume that the application system is modelled by means of SCA, the Manageability Configuration Meta-Model has to provide a management view on such actions and transactions within SCA models. Figure 59 shows a sample SCA model and points out a transaction that could be monitored.

![Figure 59: Sample Service Transactions in SCA-based applications](image)

The example clarifies that a transaction can include various components and implementations that are possibly deployed on multiple systems. The Manageability Configuration Meta-Model provides a unified management view on such a heterogeneous scenario by building management abstractions of the actions that can be part of a transaction. As shown in Figure 59 this can be an invoke activity of a BPEL process, a reference operation call of an external service, a database transaction, etc.

The model is used in two ways, as illustrated in Figure 60.
Manageability is meta-model of Manageable Data Model forms basis for Manageable Model

The Manageability Designer uses the Manageability Configuration Meta-Model to specify the elements of a certain Functional Design that need to be monitored. This specification results in the Manageability Model required by the Instrumentation Developer. Additionally, the Manageability Configuration Meta-Model forms the basis for the Manageability Data Model used within the Manageability Infrastructure architecture described in the WP 3 deliverable.

The Manageability Configuration Meta-Model has to define a unified management view on the application system. Within the IT-Management the management abstractions of a resource are called managed elements (or managed objects). A managed element is an instance of the Adapter pattern \[137\] for an individual resource, that translates its particular interface into one shared by all resource instances \[138\]. Thus, it enables management operations to be performed using a single interface. This well-known concept is used to define the basic structure of the Manageability Configuration Meta-Model. Figure 61 points out the fundamental elements of the metamodel.

The central object of the model is the ManagedElement described above. A ManagedElement represents a management abstraction of a physical or logical "Real-world" object and thus, it is the basis for all concrete classes in the configuration metamodel. As for the considered scenario, several managed elements may exist for each service component - one for each deployed and running instance.

In the following the meaning of the other objects is explained.
- **Property**: In order to allow monitoring by a management application, the service provider has to make information available regarding the ManagedElements. This management relevant information is specified by means of Properties describing a ManagedElement. These Properties can either have static content, meaning the value is valid for all instances of a
**ManagedElement**, or dynamic content. The dynamic content describes the current or the historical state of a managed element respectively.

- **Operation**: A ManagedElement can provide Operations in order to supply the management application with control functions. However, in the first project year we focus on the monitoring aspect only. Nevertheless, the model should be extendable so that control operations can be included.
- **Association**: Associations allow the allocation of complex coherencies to multiple ManagedElements. Furthermore, they are a means to integrate the relationships of the actual system into the configuration metamodel.

At design-time, such a basic approach can be used to specify the management capabilities of arbitrary resources including service components. In this case, for each service component the corresponding managed element has to be modeled. However, when using this kind of general purpose metamodel, the developer gets no information about the actual available properties, operations and dependencies in case of service components. Thus, the service component Manageability Configuration Meta-Model should introduce particular types of managed elements, which are tailored to SCAs service components. This eases, on one hand, the creation of valid Manageability Models for service components. On the other hand, it represents a crucial precondition for an automated generation of the corresponding Instrumentation. A generic metamodel is too imprecise and therefore does not allow the definition of meaningful transformations.

### 4.4.1 Manageability Configuration Metamodel

Figure 62 shows the Manageability Configuration Meta-Model for SCA service components. Since the metamodel design is based on the previously introduced concept, the monitoring view on service components is modeled through corresponding managed elements.
The model defines which runtime properties exist and thus can be monitored. When creating an instance of the model, the actually needed properties are specified. As the model focuses on performance-related aspects of service components, and the measurements are always included within service transactions, all the monitored elements are considered as units of work according to the unit of work concept as introduced in the CIM Metrics Model [136]. Within the configuration model a UnitOfWork is defined as one managed unit of a business transaction. In case of a service component the class that represents the fundamental unit of work is the ServiceComponentAction. The class ServiceComponent is the actual managed object and holds references to all of its service component actions, i.e. units of work. Additionally, the Manageability Designer is enabled to specify the type of implementation (e.g. Java, BPEL, etc.) used by the service component. A service component action may include any functional element of the SCA model. For each action that should be monitored, two kinds of information can be specified. The ManagementMetaInfo class holds information related to the general definition of the service component action, such as configuration settings or static management information (e.g. the ID of the corresponding SLA). The classes derived from the InstanceProperty class reflect information about each executed service component action instance, i.e. the runtime monitoring information.

The ServiceComponentAction has two subclasses InternalAction and OperationCall. An internal action is the management abstraction of an element of a service component which is used during the internal processing of the component’s application logic. Examples of internal actions provided by the configuration model are conditional flows (ConditionalFlow), loops (Loop), and variables (Variable). The class OperationCall allows the specification of management capabilities concerning the externally observable behavior of service components. Since it should be possible to monitor both provided and required actions, the OperationCall class provides a mechanism to define the type of operation that a service component is capable of performing. This includes the ability to specify the kind of monitoring data that should be collected for each operation, such as the number of requests, response times, or error rates.

Figure 62: Manageability Configuration Meta-Model for SCA
operation calls (or service and reference operation calls according to SCA naming), the two subclasses ServiceOperationCall and ReferenceOperationCall are available.

The scope of monitoring can be defined by adding the available property types (e.g. StartTime, EndTime, IncomingMessage, etc.) to the service component action elements. For all execution instances of type ServiceComponentAction (and thus also for all its subclasses), it is possible to monitor the time-related properties StartTime, EndTime and ElapsedTime. Moreover, it can be specified that a mutual context ID (MutualContextId) is made available for the management at runtime. According to [139] the mutual context ID can be used to facilitate the following two situations: (1) Displaying all ServiceComponentAction instances that participate in a business transaction as a whole; (2) Direct retrieval of ServiceComponentAction instances that participate in the same context by using one query expression.

For OperationCall subclasses there is additionally the ability that the actual binding as well as the content of the operation calls is made available for the management at runtime by adding the properties Binding, IncomingMessage or OutgoingMessage.

The InternalAction subclasses also have additional properties that can be made available for the management at runtime. Loop classes provide the ability to gather information about the state of the loop condition (CondStatus), the current number of loop iterations (LoopCount), and the elapsed time of the last loop iteration (LastLoopElapsedTime). For ConditionalFlow classes the property CondStatus, which contains the result of the condition, can be supplied for management. And finally, the content of a variable can be made available for management by specifying the property VariableContent in the Variable class.

By adding the required Properties to the corresponding ServiceComponentActions, a developer may configure the level of monitoring for each service component. And, because of the ability to define associations, he is also able to define the static structure and hierarchies of the managed elements which belong to the management meta-information. In addition to this, the Manageability Configuration Meta-Model supports capturing dependencies between different running service component instances. These runtime dependencies can be derived from the functional design. If a reference interface that refers to a service interface of another component is specified in the design model, the corresponding managed elements have to be associated as well. Furthermore, the dependencies between a service interface and the internal actions included in the application logic of the implementation of the provided service interface can be captured. This leads to managed element associations as depicted in the figure below.
This clarifies that the Instrumentation has to take into account that it has to supply the next managed child action instance with the ID of the last managed parent operation call instance. The Instrumentation gets this information from the Manageability Model which is an instance of the Manageability Configuration Meta-Model. The following section gives an example of such a Manageability Model.

### 4.4.2 Sample Manageability Model

As aforementioned, the Manageability Designer uses the Manageability Configuration Meta-Model to create a Manageability Model. This Manageability Model is then used by the Instrumentation Developer to appropriately create the Instrumented Functional Design. Furthermore, the Manageability Model provides the static management meta-information which has to be delivered to the Running Manageability Infrastructure before the first runtime instance properties are captured by the Running Instrumented Service Component. Figure 63 depicts this scenario.

![Figure 63: Using the Manageability Configuration Metamodel](image-url)
The role of the Manageability Infrastructure Administrator can also be taken over by a model-to-text transformations. A transformation could, for example, create a script that automatically writes the data into the database.

In the following an abstract example of the created artefacts shown in the figure above is given. The basic artefact of the scenario is the Manageability Model created by the Manageability Designer using the manageability configuration model described above and a certain functional model. The functional model of the simple, abstract example process is depicted in Figure 64.

Figure 64: Abstract example of an SCA-based Application Design

The example depicted above consists of two service components. SomeServiceComponent is a service composition which describes a process flow implemented in BPEL. Depending on a certain condition, either, a variable is assigned with a specific value and the process ends, or an operation of the service component AnotherServiceComponent is called and the process ends. The service component AnotherServiceComponent is implemented in Java and contains, among other elements, a loop which does something useful.

The Manageability Model for this abstract example, created by a Manageability Designer, might look as follows.
With this model at hand, the Instrumentation Developer knows which information to capture. For example, the start time and the end time of each operation call (service and reference) should be captured. Another example is that the incoming and outgoing messages for operation \textit{op-A} should be monitored, while this information has not to be captured for the other operation calls. Furthermore, the Instrumentation Developer knows, on the basis of the given dependencies, the parent-child relationships important for the management.

How this information is captured, and how it is supplied to the Manageability Data Model is described in the A3.4 Deliverable.

\subsection*{4.5 Manageability Design for ORC / Ad Hoc Demonstrator}

This chapter demonstrates a manageability design for the SLA@SOI Open Reference Case (ORC) as part of the year 1 adhoc demonstrator. Thereby, the benefit of the solutions is outlined within the phases of the SLA@SOI service lifecycle.

The following section introduces the general scenario along with an SCA-based design of the ORC. Afterwards, we present a corresponding instance of the manageability configuration metamodel.

\subsubsection*{4.5.1 Scenario and SCA-based ORC design}

Within the project an Open Reference Case (ORC) is used as a publicly available reference case that demonstrates the capabilities of the SLA framework. The
following scenario forms the foundation of the ORC: A service provider offers services with differentiated, dependable and adjustable SLAs and can negotiate concrete SLAs with individual or groups of customers in an automated fashion. This business goal imposes requirements on software providers, infrastructure providers and the service provider. The software providers have to provide components with predictable non-functional behaviour. The infrastructure providers need to support an SLA aware management of resources and the service provider has to translate and manage SLAs from business level along the IT stack down to the infrastructure.

Within the ORC the service provider offers a service-oriented retail solution. This retail solution is a modification of the Common Component Modelling Example (CoCoME) [140]. It represents a trading system dealing with the various aspects of handling sales at a supermarket. This includes the interaction at the cash desk with the customer, including product scanning and payment, as well as accounting the sale at the inventory. Furthermore, the trading system deals with ordering goods from wholesalers, and generating various kinds of reports.

The offered retail solution consists of five atomic Web services and one additional composite service, which is realized as a BPEL process on top of the atomic services (Figure 66).

Within the Adhoc Demonstrator the following assumptions are made:

- A customer always needs the complete ORC solution.
- The invocation of atomic and composed Web services by the customer always follows the same sequence, i.e. the same sequence is used in the workload generator and the prediction components.
- Customer invocations include both, the composed service and some atomic services.
- Customers have the following non-functional requirements:
  - Completion time for each individual service call (at the access point of the service not including public network), e.g. completion time should be below 2 seconds.
  - Throughput requirement for each service, e.g. 2 services have up to 10 requests per minute; 1 service has 100 requests per minute.
Security requirements: Here the customer has the option that the whole solution must be physically separated from other customers/competitors
- Requirements are specified in an inelastic manner, i.e. with fixed parameters; no burst capacities are considered
- Service provider also owns infrastructure
- All atomic services share one database.
- Users are employees of the customer

These assumptions imply the following deployment options:

![Diagram of deployment options]

**Figure 67: ORC Deployment Options**

Either, the whole Service-oriented Retail Solution and the Database run on the same virtual machine (Deployment Option 1), or both run on their own virtual machine (Deployment Option 2), or the Composite Services and the Atomic Services of the Service-oriented Retail Solution run on different machines, whereby the Atomic Services and the Database could run on the same system (Deployment Option 3) or on different systems (Deployment Option 4).

In the following the use of the introduced contributions within the scenario is described in terms of a detailed example.

### 4.5.2 Service Development/Design

The SCA model in Figure 68 shows the functional design of the payment service.
The *Payment Service* is implemented in BPEL by the service provider. Thereby, the composition depends on the atomic services *Card Validation Service* and *Payment Debit Service* provided by the software provider. The SLAs between the service provider and the software provider have to be derived from the SLA the service provider negotiated with the customer. Further SLAs have to be negotiated between the service provider and the infrastructure provider since all service run on top of the infrastructure offered by the infrastructure provider. However, since this thesis focuses on the management of service compositions and atomic services, the infrastructure is not considered at this point.

Figure 69 shows sample SLAs as they could be negotiated in the scenario explained above.

<table>
<thead>
<tr>
<th>SLA</th>
<th>Method Name</th>
<th>Usage Profile [requests/second]</th>
<th>Percentage (Completion Time)</th>
<th>Completion Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Payment Service</em></td>
<td>handle</td>
<td>0.69</td>
<td>90</td>
<td>200</td>
</tr>
<tr>
<td><em>Card Validation Service</em></td>
<td>validate</td>
<td>0.69</td>
<td>90</td>
<td>70</td>
</tr>
<tr>
<td><em>Payment Debit Service</em></td>
<td>debit</td>
<td>0.65</td>
<td>90</td>
<td>90</td>
</tr>
</tbody>
</table>

The customer declares that it requires the *Payment Service* in a maximum frequency of 0.69 requests per second. The service provider guarantees a completion time of less than 200ms in 90 percent of the service calls if the customer adheres to the obligated usage profile. In order to offer this service level objective (SLO), the service provider translates the objective into sub-objectives for the atomic services the *Payment Service* depends on. Thus, it negotiates appropriate SLAs with the software provider (and the infrastructure provider).

Once the SLAs are negotiated, and the services are running and in use, the service provider has to take care that it achieves the SLO offered to the customer. Therefore, it requires a comprehensive SLA management that monitors...
the metrics defined in the SLA. In order to calculate these aggregated metrics of an SLO the SLA management needs structured basic management information. This information is configured with the manageability configuration metamodel and effectively provided by the manageability infrastructure and through the manageability interface we presented in the A3.4 deliverable. Figure 70 shows an overview to the manageable service-oriented retail solution we implemented for this scenario. Details can be found in the A3.4 deliverable.

Accordingly, the Manageability Agent gathers the structured basic management information provided by the Instrumentation (I) of the service components and exposes this information at the unified Manageability Interface (M).

In order to create the instrumentation, the service provider needs a means to specify the management information he requires for the monitoring of the SLAs and the analyzing of an SLA violation. The implementation of this instrumentation follows the Manageability Configuration Meta-Model presented in chapter 4.4, which defines the information that should be captured by the instrumentation and thus processed and stored in the data repository of the Management Agent. This step is executed during the service design phase. Figure 71 shows a sample instance of the Manageability Configuration Meta-Model for the monitoring and analysis of the Payment Service and the corresponding sub-services.
Figure 71: Manageability Model for ORC

With this model at hand, the Instrumentation Developer knows where to extend the functional design. For example, the start time and the end time of each operation call (service and reference) should be captured. Another example is that the incoming and outgoing messages of service operation call SOC_ValidateCard should be monitored, while this information has not to be captured for the service operation call SOC_DebitCard. Furthermore, the Instrumentation Developer knows, on the basis of the given dependencies, the parent-child relationships that are important for the analysis of SLA violations.

4.5.3 Service Provisioning

During the service provisioning phase the instrumented functional design as well as the management agents are deployed on the corresponding systems. Moreover, the static management meta-information is extracted from the Manageability Model described above. This information has to be loaded to the responsible Management Agents before the first instance of the instrumented functional components is started. Details on the implementation of this component can be found in the A3.4 deliverable. Figure 72 outlines this information for the sample model shown above.
Figure 72: Configuration of Static Management Information for ORC

The static meta-information comprises the meta-attributes, such as the description of the HandlePayment service operation call, as well as the parent/child relationships between the definitional elements.

4.5.4 Service Operation

Within the service operation phase, the Manageability Infrastructure monitors the service-oriented application. Using the definitional meta-information (Figure 72) and the dynamic management information regarding the executed instances of these definitional elements, the Management Agent offers a unified management view on the heterogeneous service-oriented retail solution. Thus, the SLA management application is capable of calculating the aggregated metrics and statistical data specified in the SLOs. However, detecting an SLA violation is only one issue the SLA management has to deal with. In fact, the service provider must also be able to identify the cause of an SLA violation in order to start corresponding countermeasures. Since the solution we developed also allows specifying structural dependency information, and information concerning the status of critical internal actions of an operation (e.g. the state of a condition or the number of loop iterations), the Manageability Interface eases the analysis of those SLA violations. In the A3.4 deliverable we provide examples for possible SLA violation causes and explain on how to detect them using the manageability interface. This information in the first place is used by SLA adjustment, which provides means for recovering from SLA violations during service operation in an automated way. As the collected information can be persisted by the manageability infrastructure, the solution may also be used beyond the service operation phase for a (continuous) service improvement (according to the IT Infrastructure Library).
5 Conclusions

In this document, we presented the current status of our work targeting design-time prediction and runtime prediction for service-oriented architectures as well as the design and development of management capabilities required for an SLA-driven management.

Design-time Prediction

Design-time prediction enables service providers to determine quality attributes of services before their actual deployment. In its current state, design-time prediction provides realistic estimates of response times, throughput, and resource utilisation. In order to support SLA-translation and negotiation, design-time prediction evaluates the performance properties of multiple infrastructure sizing alternatives that can be used to realise a service.

The information which is necessary to predict the performance of a service is distributed among different roles involved in the service life-cycle. Therefore, we defined domain-specific modelling languages for service customers, software providers, service providers, and infrastructure providers. Design-time prediction can collect and integrate the information stored in these models to derive realistic performance estimates.

In the context of SLA@SOI, we evaluated the applicability of the prediction service on the basis of the Open Reference Case (ORC). For this purpose, we created the models of the ORC necessary for performance prediction. Design-time prediction integrates the models and generates different alternatives of its architecture. For all alternatives, it predicts the performance properties and aggregates the results.

Currently, design-time prediction is working for the ORC with the desired accuracy. However, many of the concepts developed so far need to be generalised so that design-time prediction can deal with arbitrary services. Also the interaction with other modules of SLA@SOI, such as the Service Landscape or the Template Registry, needs to be improved in order to make design-time prediction applicable in practice.

Beyond these practical steps, we plan to optimise the exploration of different deployment options. This includes the automatic generation of different deployment options using model-driven techniques. More specifically, we plan to use model transformation languages (such as Query View Transformation, QVT [149]) to implement the generation of different service variants. Furthermore, the search for an optimal solution can be supported by meta-heuristics such as proposed in [5]. Furthermore, the integration of model-driven and measurement-based performance prediction techniques can help us to improve the accuracy of design-time prediction. Service demands need to be calibrated based on measurements. Using measurements in combination with predictions will help us to improve the overall prediction accuracy.

Run-time Prediction

Run-time prediction targets the dynamic re-provisioning and migration of services and compute resources. At infrastructure level, run-time prediction supports the dynamic adjustment of resource allocation. This can save operational and utility
cost by suspending unutilized compute resource. On the application-level, run-time prediction can be used to proactively avoid SLA violations. Based on the recent behaviour of a service, we can predict the probability of an SLA violation in the near future and start countermeasures if appropriate.

In the context of SLA@SOI, several challenges with respect to run-time prediction remain for the time being. In the current state, our approaches for infrastructure-level prediction and service-level prediction are clearly separated. However, first experiments demonstrated that infrastructure level prediction can profit from monitoring data at the service level and vice versa. Therefore, we need to address the question of how a possible integration of both approaches can be achieved. Since both approaches follow different philosophies, a full integration is most likely impossible. However, both approaches may profit from the concepts and information available at the other level.

For infrastructure-level prediction, machine learning and advanced filtering techniques, such as Kalman or LPV filters, have been widely used in. Based on the experimental analysis, we already improved the original algorithms. However, we already plan to further improve existing techniques and integrate them into our SLA-aware infrastructure management (Deliverable D.A4a).

For service-level predictions, we need to improve the prediction accuracy for longer periods during run-time, in order to allow more flexible actions to avoid SLA violations. This may require taking into account other useful metrics, such as job attributes policies, and design-time information. Furthermore, the predictive algorithms should be adjusted and adapted according to the particular requirements and data characteristics.

Manageability Design

The manageability framework for service-oriented solutions allows service providers and service customers to monitor the compliance of the system to its SLOs with respect to performance-related properties, like response time and throughput. They can monitor and control the performance-related properties of service components. In its current state, our management framework supports the identification of WSC runtime measurements necessary for monitoring performance, control operations to adjust the performance at run-time, and the instrumentation of the service components in terms of sensors and effectors.

In this document, we have presented the design of the manageability configuration metamodel. It serves as a basis for the configuration of the manageability infrastructure with scenario-specific monitoring requirements. We focused on monitoring capabilities, whereas its extension by control capabilities will be addressed in year 2. Similarly, the design and implementation of the corresponding manageability infrastructure will be extended towards control capabilities.
6 References


[87] Doulamis, N.D., Doulamis, A.D., Non-linear 3D rendering workload prediction based on a combined fuzzy-neural network architecture for grid computing applications, ICIP03(III: 1069-1072).


[120] MonALISA home page: [http://monalisa.cacr.caltech.edu/monalisa.htm](http://monalisa.cacr.caltech.edu/monalisa.htm) (visited April 2009)
[136] Open SOA, "Service Component Architecture Specifications ".


[146] Open Architecture Ware (OAW). http://www.openarchitectureware.org/


## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Service</td>
<td>Any service not yet instantiated (service instance) is called an abstract service.</td>
</tr>
<tr>
<td>Agreement Initiator</td>
<td>An agreement initiator is a party to a service level agreement. The initiator creates and manages an agreement on the availability of a service on behalf of either the service customer or service provider, depending on the domain-specific signalling requirements.</td>
</tr>
<tr>
<td>Agreement Offer</td>
<td>An offer is the description of the agreement relationship that is sent from agreement initiator to agreement responder during agreement creation, indicating the relationship which the initiator would like to form.</td>
</tr>
<tr>
<td>Agreement Responder</td>
<td>The agreement responder is a party to a service level agreement. The responder implements and exposes an agreement on behalf of either the service provider or service customer, depending on the domain-specific signalling requirements.</td>
</tr>
<tr>
<td>Agreement Template</td>
<td>An agreement template is an XML document used by the agreement responder to advertise the types of offers it is willing to accept.</td>
</tr>
<tr>
<td>Agreement Term</td>
<td>Agreement terms define the content of a service level agreement.</td>
</tr>
<tr>
<td>Business Service</td>
<td>A business service is exposed/invokable via at least some non IT elements.</td>
</tr>
<tr>
<td>External Service</td>
<td>External services are exposed across the boundaries of an organization, i.e. across at least two administrative domains.</td>
</tr>
<tr>
<td>Guarantee Term</td>
<td>Guarantee terms define the assurance on service quality associated with the service described by the service definition terms. They refer to the service description that is the subject of the agreement and define service level objectives, qualifying conditions and business value expressing the importance of the service level objectives.</td>
</tr>
<tr>
<td>Infrastructure Provider</td>
<td>A specific kind of service provider who focuses on the provisioning of infrastructure services.</td>
</tr>
<tr>
<td>Infrastructure Service</td>
<td>An infrastructure service is a specific IT service which exposes resource/hardware-centric capabilities.</td>
</tr>
<tr>
<td>Internal Service</td>
<td>Internal services are exposed within the boundaries of an organization, i.e. within one administrative domain.</td>
</tr>
<tr>
<td>IT Service</td>
<td>An IT service is exposed/invokable by means of information technology. Specific classes of IT services may be software services, infrastructure services or media services.</td>
</tr>
<tr>
<td>Offered Service</td>
<td>An abstract service which is offered by a specific Service Provider to its Service Customers.</td>
</tr>
<tr>
<td>Service</td>
<td>A means of delivering value to Customers by facilitating Outcomes Customers want to achieve without the ownership of specific Costs and Risks. See also service interface type, service stage, service exposure</td>
</tr>
</tbody>
</table>
Service Stage  
The stage a service reaches over time from fully abstract to actually instantiated.  
See also → abstract service, → offered service, → service instance

Service Consumer  
Person(s) who actually consume/use the provided services. Typically they belong to the → service customer.

Service Customer  
Someone (person or group) who orders/buys services and defines and agrees the service level targets.

Service Description Term  
Service Description Terms describe the functionality that will be delivered under the → service level agreement. The agreement description may include also other non-functional items referring to the service description terms.

Service Exposure  
Services can be exposed either internally (within the same administrative domain) or externally.  
See also → internal service, → external service

Service Instance  
A concrete implementation of an → offered service which is ready for consumption by service users.

Service Interface Type  
Describes the nature of an actually exposed service, i.e. about the nature of his invocation interface.  
See also → business service, → IT service, → composed service

Service Level Consequence  
An action that takes place in the event that a → service level objective is not met.

Service Level Agreement  
An agreement defines a dynamically-established and dynamically managed relationship between parties.  
The object of this relationship is the delivery of a service by one of the parties within the context of the agreement. The management of this delivery is achieved by agreeing on the respective roles, rights and obligations of the parties. The agreement may specify not only functional properties for identification or creation of the service, but also non-functional properties of the service such as performance or availability. Entities can dynamically establish and manage agreements via Web service interfaces.

Service Level Objective  
Service Level Objective represents the quality of service aspect of the → agreement. Syntactically, it is an assertion over the → agreement terms of the agreement as well as such qualities as date and time.

Service Provider  
An organization supplying services to one or more internal customers or external customers.

Software Provider  
An organization producing → software components which might be used by a → service provider to assemble actual → services.

Software Service  
A software service is a specific → IT service which is exposed/invokable by means of software entities such as Web services, user interfaces, or software-based business processes.

Software Component  
Software components are the entities produced at design-time by a → software provider.
## Appendix A: Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACF</td>
<td>Auto-Correlation Function</td>
</tr>
<tr>
<td>ASCA</td>
<td>Agent-Based Simulator for Computing (resource) Allocation</td>
</tr>
<tr>
<td>BGT</td>
<td>Belief Graph Tool</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>CBML</td>
<td>Component-Based Modelling Language</td>
</tr>
<tr>
<td>CBSE</td>
<td>Component-Based Software Engineering</td>
</tr>
<tr>
<td>CB-SPE</td>
<td>Component-Based Software Performance Engineering</td>
</tr>
<tr>
<td>CIM</td>
<td>Common Information Model</td>
</tr>
<tr>
<td>CoCoME</td>
<td>Common Component Modelling Example</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DGSOT</td>
<td>Dynamically Growing Self-Organizing Tree</td>
</tr>
<tr>
<td>DOVT</td>
<td>Detection of Validation Tool</td>
</tr>
<tr>
<td>EBNF</td>
<td>Extended Backus-Naur Form</td>
</tr>
<tr>
<td>EGBT</td>
<td>Event Genuineness Belief Tool</td>
</tr>
<tr>
<td>EMF</td>
<td>Eclipse Modelling Framework</td>
</tr>
<tr>
<td>FFTV</td>
<td>From Failure To Vaccine</td>
</tr>
<tr>
<td>GMF</td>
<td>Graphical Modelling Framework</td>
</tr>
<tr>
<td>HD</td>
<td>Hard Disk</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection Systems</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>Java EE</td>
<td>Java-Platform, Enterprise Edition</td>
</tr>
<tr>
<td>JAX-WS</td>
<td>Java API for XML – Web Services</td>
</tr>
<tr>
<td>LRD</td>
<td>Long Range Dependence</td>
</tr>
<tr>
<td>MBC</td>
<td>Model-Based Clustering</td>
</tr>
<tr>
<td>MMPP</td>
<td>Markov-Modulated Poisson Process</td>
</tr>
<tr>
<td>MWN</td>
<td>Multi-fractal Wavelet Model</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>OAW</td>
<td>Open Architecture Ware</td>
</tr>
<tr>
<td>ORC</td>
<td>Open Reference Case</td>
</tr>
<tr>
<td>OSGi</td>
<td>Open Services Gateway initiative</td>
</tr>
<tr>
<td>PCM</td>
<td>Palladio Component Model</td>
</tr>
<tr>
<td>PECT</td>
<td>Prediction-Enabled Component Technology</td>
</tr>
<tr>
<td>PP</td>
<td>Prediction of Periodicity</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RDSEFF</td>
<td>Resource-Demanding Service Effect Specification</td>
</tr>
<tr>
<td>SBS</td>
<td>Service-Based System</td>
</tr>
<tr>
<td>SCA</td>
<td>Service Component Architecture</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SLO</td>
<td>Service Level Objective</td>
</tr>
<tr>
<td>SPE</td>
<td>Software Performance Engineering</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOI</td>
<td>Service-Oriented Infrastructure</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSJ</td>
<td>Stochastic Simulation in Java</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>TDT</td>
<td>Threat Detection Tool</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>UML MARTE</td>
<td>Modelling and Analysis of Real-time and Embedded Systems</td>
</tr>
<tr>
<td>UML SPT</td>
<td>Schedulability, Performance and Time Specification</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>WSLA</td>
<td>Web Service Level Agreement</td>
</tr>
<tr>
<td>WSMF</td>
<td>Web Services Management Framework</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WSOL</td>
<td>Web Services Offerings Language</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>