Faculty of Computer Science, Electronics and Telecommunications

PROACTIVE ADAPTATION OF MULTI-LAYERED SERVICE ORIENTED SYSTEMS

Ph.D. Dissertation
Daniel Žmuda

Supervisor:
Prof. dr hab. inż. Krzysztof Zieliński

Kraków, 2015
Wydział Informatyki, Elektroniki i Telekomunikacji

PROAKTYWNA ADAPTACJA WIELOWARSTWOWYCH SYSTEMÓW ZORIENTOWANYCH NA USŁUGI

Rozprawa doktorska
Daniel Żmuda

Promotor:
Prof. dr hab. inż. Krzysztof Zieliński

Kraków, 2015
I would like to dedicate this dissertation to my family, Anna, Aleksander and Aneta. This thesis would not have happened without their support and patience.
Acknowledgments

I would like to express my gratitude to my supervisor, Professor Krzysztof Zieliński, for his extremely helpful support, guidance and motivation while carrying out the research described in this dissertation.

Special thanks to the members of the Distributed Systems Research Group at the Computer Science Department of AGH University of Science and Technology, especially to Marek Psiuk for suggestions and fruitful discussions during writing this dissertation.

The research was partially carried out within the European Union within the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

I acknowledge that during the work on my thesis I have been a scholarship fellow of the "Doctus - Małopolski fundusz stypendialny dla doktorantów" project cofunded by EU funds within European Social Fund.
# Table of Contents

Acknowledgments .................................................. i

List of Tables ...................................................... v

List of Figures ...................................................... vi

Listings ............................................................. ix

1 Introduction ......................................................... 1
   1.1 Motivation and Thesis Statement .......................... 2
   1.2 Scope of Research ........................................... 4
   1.3 Thesis Contribution .......................................... 4
   1.4 Dissertation Organization .................................. 5

2 Technological Background and Related Work ..................... 7
   2.1 Service Oriented Architecture Paradigm ..................... 8
      2.1.1 SOA-related models ..................................... 8
      2.1.2 S3 Model ................................................. 10
      2.1.3 SOA Environments ....................................... 12
   2.2 Adaptability Concepts ........................................ 14
      2.2.1 Control Loop Building Blocks .......................... 16
      2.2.2 Taxonomy of Self-Adaptation ............................ 18
   2.3 Research Related to Proactive Adaptation of Multi-Layered SOA Systems ........................................... 22
      2.3.1 Online Prediction and Analysis .......................... 23
      2.3.2 Overview of Existing Solutions ......................... 28
   2.4 Summary ...................................................... 31

3 Concept of Proactive Adaptation in Multi-Layered, Distributed SOA Environments ........................................... 32
   3.1 System Requirements .......................................... 33
# Table of Contents

3.2 Abstraction of Proactive Adaptation in Multi-Layered Environments ... 34  
3.3 Realization of Proactive Adaptation Process ........................... 37  
   3.3.1 Property Relationship Model Evaluation Details ................. 39  
   3.3.2 Online Prediction of SLA Violations Process Details .......... 44  
3.4 Adaptive Controller Concept and Design ............................... 47  
3.5 Summary ........................................................................... 52  

4 Design of Proactive Adaptation Framework for Service Oriented Sys-  
   tems .................................................................................... 53  
   4.1 PRIDE Realization of Adaptation Process ............................ 54  
   4.2 PRIDE System Architecture ............................................. 56  
      4.2.1 Instrumentation Layer .............................................. 60  
      4.2.2 Monitoring Layer .................................................. 62  
      4.2.3 Management Layer ............................................... 64  
      4.2.4 Exposition Layer .................................................. 66  
      4.2.5 Policies Layer ...................................................... 67  
   4.3 PRIDE System Functional Description ................................. 69  
      4.3.1 Execution Environment Part .................................... 69  
      4.3.2 SLA Online Prediction and Adaptation Part ................. 73  
   4.4 Summary ........................................................................ 78  

5 PRIDE Implementation ............................................................. 79  
   5.1 Platform Specific Model Selected Technologies ..................... 80  
      5.1.1 PRIDE Integration Process ...................................... 83  
   5.2 Implementations Details ..................................................... 84  
      5.2.1 Execution Environment Part .................................... 87  
      5.2.2 Online Prediction and Adaptation Part ....................... 90  
   5.3 Summary ........................................................................ 93  

6 System Evaluation and Experimental Results ............................. 95  
   6.1 Evaluation Methodology ..................................................... 96  
   6.2 Functional Evaluation ...................................................... 100  
      6.2.1 Experiment 1 - Utilization of Relationship Model in Adaptation Process ........................................... 105  
      6.2.2 Experiment 2 - Influence of Adaptive Controllers Communication Scheme on Adaptation Enforcement Results .............................................. 108  
      6.2.3 Experiment 3 - Online Prediction Aspects in Adaptation Enforcement ....................................................... 113  
      6.2.4 Experiment 4 - Behaviour of the system under varying SLA ......................................................... 122  
   6.3 System Overhead and Scalability Aspects .............................. 127  
   6.4 Summary ....................................................................... 129  

7 Conclusions ........................................................................... 131
LIST OF TABLES

2.1 Comparison of selected related frameworks ............... 30
5.1 Fulfilment of service oriented systems properties by selected technology stacks ........................................ 81
6.1 Application scenario services description .................. 98
6.2 Experiment 4 - scenario 1 - SLA definition changes ........ 123
6.3 Experiment 4 - scenario 2 - SLA definition changes ........ 123
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Positioning of SOA-related standards</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>SOA Solution Stack</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>High-level representation of adaptation loop</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>Hierarchical taxonomy of self-adaptation</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>The closed control loop vs. the adaptive control loop</td>
<td>23</td>
</tr>
<tr>
<td>2.6</td>
<td>The predictive control loop compliant with Model Proactive Control method</td>
<td>24</td>
</tr>
<tr>
<td>2.7</td>
<td>Exemplary SLA metric values in different adaptation enforcement types</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Concept map presenting the simplified version of SOA execution environ-</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>ment model adopted from the S3</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>High-level concept map of adaptation process in service oriented environ-</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>ments</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Detailed concept map of adaptation process realization</td>
<td>36</td>
</tr>
<tr>
<td>3.4</td>
<td>Realization of proactive adaptation process</td>
<td>38</td>
</tr>
<tr>
<td>3.5</td>
<td>Foundations of Property Relationship Model</td>
<td>39</td>
</tr>
<tr>
<td>3.6</td>
<td>Use case of simple service oriented system</td>
<td>41</td>
</tr>
<tr>
<td>3.7</td>
<td>Representation of relationships between properties in a sample service</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>oriented system</td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Symptom matchmaking realization phases</td>
<td>45</td>
</tr>
<tr>
<td>3.9</td>
<td>The block diagram of the Adaptive Controller</td>
<td>48</td>
</tr>
<tr>
<td>3.10</td>
<td>Possible cooperation models of Adaptive Controllers</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>PRIDE High-level design of proactive adaptation process realization</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>Layered architecture of PRIDE framework</td>
<td>58</td>
</tr>
<tr>
<td>4.3</td>
<td>The structure of PRIDE Instrumentation Layer</td>
<td>60</td>
</tr>
<tr>
<td>4.4</td>
<td>The Interceptor Socket instrumentation mechanism</td>
<td>61</td>
</tr>
<tr>
<td>4.5</td>
<td>The structure of PRIDE Monitoring Layer</td>
<td>63</td>
</tr>
<tr>
<td>4.6</td>
<td>The structure of PRIDE Management Layer</td>
<td>65</td>
</tr>
<tr>
<td>4.7</td>
<td>The structure of PRIDE Exposition Layer</td>
<td>66</td>
</tr>
</tbody>
</table>
List of Figures

4.8 The structure of PRIDE Policies Layer .................................................. 68
5.1 PRIDE components functional categorization ........................................ 85
5.2 Separation of communication channels between PRIDE and dependant SOA system ................................................................. 86
6.1 The high-level view of the Plan and Book Trip (PBT) application flow .... 97
6.2 The testbed configuration used during experiments evaluation .......... 100
6.3 Load generated while conducting experiments in the functional evaluation functional evaluation experiment ........................................ 101
6.4 Observed services average invocation count (moving average - 2s) during functional evaluation experiment ........................................ 102
6.5 Initial values of response time metric values without enforcement of adaptation ................................................................. 103
6.6 Initial values of CPU consumption metric values without enforcement of adaptation ................................................................. 104
6.7 Experiment 1 - Correlation matrix visualisation computed during nominal system run before occurrence of SLA violation .......................... 106
6.8 Experiment 1 - Correlation matrix visualisation computed during nominal system run after disappearance of SLA violation ................. 107
6.9 Experiment 2 - SLA related CPU consumption metric values utilizing reactive ACs in layers independent approach ................................ 110
6.10 Experiment 2 - SLA related CPU consumption metric values utilizing reactive ACs in global approach ........................................ 111
6.11 Experiment 2 - SLA related response time metric values utilizing reactive ACs in layers independent approach ................................ 112
6.12 Experiment 2 - SLA related response time metric values utilizing reactive ACs in global approach ........................................ 112
6.13 Experiment 3 - SLA related CPU consumption metric values utilizing proactive ACs in layered approach .................................. 115
6.14 Experiment 3 - SLA related CPU consumption metric values utilizing proactive ACs in global approach .................................. 116
6.15 Experiment 3 - SLA related response time metric values utilizing proactive ACs in layered approach .................................. 117
6.16 Experiment 3 - SLA related response time metric values utilizing proactive ACs in global approach .................................. 117
6.17 Experiment 3 - Proactive adaptation behaviour with different prediction window lengths - CPU consumption metric - logarithmic scale .... 119
6.18 Experiment 3 - Proactive adaptation behaviour with different prediction window lengths - response time metric ....................... 120
6.19 Experiment 3 - Comparison of adaptation decisions enforcement frequency with different lengths of prediction window ............... 121
6.20 Experiment 3 - SLA cost comparison .................................................. 121
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.21</td>
<td>Experiment 4a - SLA related CPU utilization metric values</td>
<td>125</td>
</tr>
<tr>
<td>6.22</td>
<td>Experiment 4b - SLA related CPU utilization metric values</td>
<td>126</td>
</tr>
<tr>
<td>6.23</td>
<td>Experiment 5 - CPU overhead caused by deployment of instrumentation</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>mechanisms and activation of adaptation process</td>
<td></td>
</tr>
<tr>
<td>6.24</td>
<td>Experiment 5 - Network bandwidth overhead caused by deployment of</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>instrumentation mechanisms and activation of adaptation process</td>
<td></td>
</tr>
<tr>
<td>6.25</td>
<td>Experiment 5 - CPU utilization of single Adaptive Controller managing</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>different number of containers with increasing number of services</td>
<td></td>
</tr>
</tbody>
</table>
A.1 Aspect Weaving OSGi BundleContext .......................... 135
A.2 OSGi Interceptor Socket ........................................ 136
A.3 Implementation of GenericMeasurement class ............... 136
A.4 Implementation of VMManager class ......................... 137
A.5 Implementation of ManagementAgent class .................. 138
A.6 Implementation of CoherenceCacheConnector class ....... 139
A.7 CEP statement used for linear regression computation .... 140
A.8 CEP statement used for average metric value computation 140
A.9 CEP statements used for trends estimation ................. 140
A.10 Implementation of CorrelationManager class ............. 141
A.11 CEP statements used for measuring SLA violations in a reactive manner 142
A.12 Implementation of SLAReactiveListener class ........... 142
A.13 Implementation of SLAProactiveListener class .......... 143
A.14 Timed eviction of predicted SLA violations ............... 144
A.15 Symptom matchmaking patterns processing ............... 144
A.16 SLA violations handling by the AdaptationManager .... 144
# List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Discovery process realized by Adaptive Controller</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>Instrumentation of layer specific resources</td>
<td>72</td>
</tr>
<tr>
<td>3</td>
<td>Property Relationship Model Computation</td>
<td>74</td>
</tr>
<tr>
<td>4</td>
<td>Trends estimation process</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>Symptom matchmaking process realized by Adaptive Controller</td>
<td>77</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

The Service-Oriented Architecture (SOA) has become the mainstream of building enterprise applications in the recent years. It has been promoted as a cost-effective and efficient solution capable of addressing the organizations business needs which enterprise environments are often characterized by high heterogeneity, ineffective hardware utilization and duplication of software functionality. Thanks to the improvement of business processes and IT infrastructure alignment, the SOA solved the problem of reducing the total costs of maintenance and ownership of companies environments.

The existing SOA solutions are concentrated on providing tools and environments to ease the way of creating distributed systems and applications deployed at a big scale. The increased complexity and scale of those systems as well as their layered nature entail the aspects of efficient infrastructure management (including both software and hardware) to be perceived as one of the most important challenges. Moreover, current needs of the end users show that they crave for high quality of offered services. It results in the very restrictive contracts signed between the service providers and their consumers which are often characterized by a high cost of not maintaining the Service Level Agreement (SLA). The manual governance of the whole SOA environment in order to meet the desired SLA requires a huge amount of work. Furthermore, it may result in many errors or bad decisions, especially when we are dealing with highly dynamic environments. In such cases, to fulfil the specified contract there may be a necessity to switch different instances of services or modify the hardware resources at runtime.

Because of the aforementioned, there is still a need for the solutions which enable service oriented systems with capabilities of automatic management decisions enforcement while maintaining a given service contract. Such solutions are often called the adaptive systems and mostly focus on the development of systems that react to changes in their execution environment. It is worth to emphasize that such approach is far more advanced than ordinary system reconfiguration as the adaptive system (despite of performing standard
management actions) modifies the internal system structure or behaviour to achieve a given goal. More advanced solutions gather the knowledge during system operation and utilize it while enforcing the adaptation to improve the decision making process. The most sophisticated solutions allow to anticipate the future system behaviour and avoid possible violations of agreed service contract as well as dynamically fit to application needs and adapt their execution environments accordingly. The latter is the newest trend in the industry and is called the Application Centric Infrastructure (ACI).

This dissertation addresses the identified needs by proposing the concept of the proactive adaptation of service-oriented systems with special emphasis on layered nature of such environments.

1.1 Motivation and Thesis Statement

Latest trends in software engineering insist on creating the IT systems in accordance with the Software as a Service (SaaS) concept (which in turn is built on top of the Platform as a Service (PaaS) and Infrastructure as a Service (IaaS)).[1] It entails the situation where newly created IT solutions are highly distributed and more complex than before. Such situation demands the usage of more sophisticated tools which enable a given system with adaptive capabilities. Moreover, the multi-layered nature of such systems is also directly reflected in the agreed contract between the service providers and theirs consumers which causes that the actual SLA definition also refers to multiple layers of the given system instead of a single one. Unfortunately, most of the existing SOA adaptation solutions are only capable of addressing issues related only to a single technology or system layer. Such approach, if used in isolation, is not capable of dealing with the real-world deployments, where changes occurring in one layer do often affect other layers. Thus, the information from multiple layers is essential to truly understand the problems of a given domain and develop a comprehensive solution. Such an approach seems to be correct as many of the proposed SOA models emphasize their layered architecture and direct relationship between some of the layers. One of such models is the SOA Solution Stack (S3) model proposed by IBM, which specifies that there exist specific layers that enable the multi-layered execution environment with capabilities of integration, Quality of Service (QoS) or governance. Yet, the presented concepts are very abstract and do not specify in depth the aspects of adaptability in such systems.

Another important aspect is related to growing demands of the end users, dynamic changes of their requirements as well as the dynamism of the SOA environments itself. It creates the situation in which the service providers have to adapt to ongoing changes much more quicker. Moreover, the agreements between the service providers and service consumers are becoming more and more restrictive which potentially implies the high cost of the agreed SLA violation. This has led to the situation where a reactive approach for maintaining the agreed contract ceased to be profitable and in some cases may
generate more losses than gains, especially in complex multi-layered environments. In
order to partially solve these issues, there were research efforts related to the area of
the **SLA** contract negotiation [2]. Some of them also covered the aspects of adaptive
renegotiation of a given agreement [3] at runtime. However, this still left the aspects
of responding to imminent problems and issues related to **SLA** violation not solved.
Therefore, the assessing and predicting of service realization quality at runtime is still
a top research challenge.

The multi-layered nature of **SOA** environments entails the application of adaptation
enforcement techniques which are aware of cross-layer dependencies taking place in such
systems. Moreover, to further improve the service realization quality in such systems,
the realization of adaptation process in many cases should expose the capabilities of
predictive behaviour. This dissertation focuses on providing a generic model which could
be applied in multi-layered service oriented systems in order to enforce the adaptation
process when the agreed service contract is not met or is anticipated to be violated. Such
approach comes with several assumptions for the dependant execution environments as
it is mainly done utilizing the closed control-loop approach. The work presented in this
dissertation is formulated on three different levels of abstraction, which are compliant
with the Model-driven Architecture (**MDA**) principles: conceptual, platform independent
and platform specific. Each of them concretizes the previous one, which makes the
presented work easy to adopt in case of very abstract as well as implementation specific
scenarios. The thesis statement is as follows:

*Multi-layered service oriented systems can be enriched at run-time with uniform
configurable mechanisms enabling realization of a proactive adaptation process
which allows goal-driven improvement of service realization quality.*

The declared capabilities of the proposed concept, i.e. proactivity and multi-layered
nature of the dependent systems, are defined in the following way:

**proactive adaptation** - enforcing the adaptation process utilizing the online pre-
diction techniques, which are essential to proactively avoid foreseen **SLA** violations.
In order to perform it, there is the need for continuous estimation of **QoS** metrics
related with defined **SLA** entries. If violation is anticipated then the enforcement
of adaptation actions should occur.

**multi-layer dependencies** - the changes occurring in one layer may often affect
other layers. Therefore the identification of casual relationship model between
system elements located in different layers is very important. The acquisition
of such data allows for more accurate and efficient realization of the adaptation
process.

In order to address the challenges introduced in the thesis statement this dissertation
proposed to utilize the concept of predictive reasoning which is driven by means of
the declarative adaptation policy. The declarative specification of high-level adaptation goal allows to mitigate the risks associated with explicit referencing of services (or other system elements) currently operating in the runtime environment by referring to the system execution model constructed and updated at runtime. Moreover, in case of more complex deployments such policy can be automatically transformed into multiple layerspecific policies enforcing realization of adaptation process in given sections of the whole system.

1.2 Scope of Research

This dissertation focuses on the realization of proactive adaptation process in service oriented systems, which are typical examples of the multi-layered systems. The realization of management and adaptation in each layer of such systems separately is in the most cases inefficient and may lead, in the worst case, to the adverse results. Therefore, this dissertation tackles the aspects of adaptation enforcement from a multi-layered perspective, thus overseeing the system from a global view. In order to minimize the occurrences of the SLA violations the adaptation process itself is steered in a proactive manner, which entails foreseeing of a given QoS related metrics.

It is assumed, that the dependant systems are instrumented which enables the transparent, runtime configuration of adaptation mechanisms in accordance with the declarative, policy-driven specification of high-level adaptation goal. The policy is provided by the system administrator and refers to the defined SLA which describes the dependant system non-functional capabilities.

The work presented in this dissertation proposes the utilization of graph-based model (Property Relationship Model (PRM)) for describing the casual relationships between different system elements. The proposed model is used for automatic identification of dependencies between elements exposed in different layers of the system. The PRM is further evaluated by the decision engine used for Adaptive Controller (AC) adaptation process implementation. The dissertation also investigates the application of a hierarchical structure of AC during the realization of aforementioned process which is steered by a declarative adaptation policies.

Finally, the author analyses several technological stacks, standards and frameworks as the execution environment of the proposed solution. The work describes the enhancement process of a dedicated SOA environment with an extension that allows for realization of the proactive adaptation process.

1.3 Thesis Contribution

The main contributions of this dissertation are as follows:
The critical analysis of the current state of the art in the area of the multi-layered environments proactive adaptation. The analysis identifies the capabilities and limitations of existing solutions in the scope which is relevant from the perspective of this dissertation.

Specification of the methodology for realization of proactive adaptation process in multi-layered, distributed environments. The methodology assumes utilization of [PRM] model, which expresses dependencies between different system elements of a dependant system and can support decision making while performing online adaptation enforcement process. The methodology includes the declarative specification of the adaptation policy which is automatically transformed into the low-level representation, furthermore dynamically configured in form of execution mechanisms deployed in supervised environments.

Proposal of goal-driven adaptation process model which can be adopted in multi-layered service-oriented environments. It is supported by the Adaptive Controllers abstract design and deployment scheme (driven by the structure of the [PRM]), which enables to model the decision making flow while enforcing the adaptation in multi-layered, distributed [SOA]-compliant environment.

Proposal of a Platform Independent Model (PIM) layered architecture for solution supporting proactive adaptation in service-oriented environments, which exposes capabilities of predictive reasoning through event patterns matchmaking.

The prototype implementation of the Proactive Adaptation Framework (PRIDE), which is further utilized to evaluate the requirements stated in the dissertation. The prototype is realized with the use of current state of the art technologies and its utilization during the experimental evaluation proves the pointed thesis statement.

Practical evaluation of proposed concepts in the dedicated [SOA] environment which is used for the purpose of possible profits and losses comparison in the context of proactive adaptation enforcement.

Design of adaptation mechanisms and architectural patterns that are compliant with the Application Centric Infrastructure concept.

1.4 Dissertation Organization

The dissertation organization is outlined in what follows. Chapter 2 presents the technological background aspects which are related to the dissertation scope. It covers the aspects of the [SOA]-related standards (including the more detailed description of the most mature one - IBM’s [S3] model) and solutions used to
create and maintain SOA environments. It also includes the analysis of the state of the art research related to control loop and proactive adaptation realization techniques.

The content of Chapter 3 presents the concept of adaptation enforcement in service oriented environments with special emphasis on proactivity aspects of such process. The introduced idea assumes the utilization of an Adaptive Controllers hierarchy which are supported by Property Relationship Model constructed on the basis of a dependant system. The controllers utilize the predictive measurements to maintain the specified adaptation goal which typically refers to the agreed service contract in form of the SLA.

Chapter 4 contains the design and architecture of the created Proactive Adaptation Framework. It results from the current technological background as well as requirements defined for the solution enabling proactive adaptation of multi-layered service oriented systems. The architecture is divided into layers (in accordance to the Adaptive SOA Solution Stack (AS3) pattern), which are further categorized into Execution Environment Part and Online Prediction and Adaptation Enforcement Part. The chapter describes not only the main system components but also the behaviour expressed in a set of algorithms that should be implemented by concrete realization of this architecture.

The prototype implementation of the framework supporting proactive adaptation in multi-layered environments is described in depth in Chapter 5. It includes the justification of the concrete technologies used for the prototype which is mainly based on the analysis of technical background, stated requirements and architectural decisions presented in Chapter 4.

Chapter 6 presents the evaluation of the introduced concepts. It also verifies the correctness of the stated thesis in a set of experimental scenarios, in which the PRIDE prototype is used to enrich the dependant system with the capabilities of proactive adaptation.

Finally, the dissertation is concluded in Chapter 7. It includes the discussion of the achieved results and points to the possible directions of the future research.
Chapter 2

TECHNOLOGICAL BACKGROUND AND RELATED WORK

The main scope of this dissertation is the proactive adaptation of layered software environments compliant with SOA principles. There are several aspects that have to be tackled in the context of technological background and related work. First of all, the analysis of the SOA models is performed. Since the scope of this dissertation includes not only abstract architectural design but also prototype implementation, it is also crucial to analyse industry related SOA standards and environments. Another important aspect is associated with the research related to the adaptive and autonomous systems. It has to be explored in order to identify the current challenges relevant to the scope of this dissertation. Finally, the related works in the area of service oriented systems proactive adaptation have to be analysed, to prove the novelty of the proposed concepts.

In this chapter the discussion of all aforementioned aspects is done, which provides strong foundations for the dissertation content included in the following chapters.

The structure of this chapter is as follows. The first section discusses a number of SOA related standards and presents in details the IBM’s S3 model, which is the most mature and production ready among all of them. It also includes the analysis of various SOA environments that are crucial in the context of the prototype implementation presented in Chapter 5. The second section presents the concept and methods in the context of adaptation realization state of the art. The discussion includes the elaboration on the hierarchical taxonomy of adaptation and points out the elements which are the most important in the scope of this dissertation. The third section discusses the research related to the concept of proactive adaptation in a multi-layered environments. It is also devoted to the analysis of the most mature solutions, including their capabilities and limitations. Finally, the chapter is summarized.
2.1 Service Oriented Architecture Paradigm

The SOA paradigm was first introduced in [5] by Thomas Erl and is build upon eight service orientation principles which enable to characterize the service-oriented system in a technology agnostic manner. These principles are as follows: Standardized Service Contract, Service Loose Coupling, Service Abstraction, Service Reusability, Service Autonomy, Service Statelessness, Service Discoverability and Service Composability. All these principles taken together allow for easier adaptation of SOA compliant applications by system integrators. Moreover, these principles are further supported by a well-known patterns [16] and concepts [17] which ease the (re-)development process and runtime modification changes.

2.1.1 SOA-related models

A. Fattah [6] provides the classification scheme used to position different Reference Architectures (RAs) related to SOA paradigm. The positioning process is realized in accordance with two aspects: the level of abstraction of a given RA and its applicability in given areas (such as presentation, integration or security). Although, part of the presented solutions is proprietary (mostly IBM-centric view) the proposed methodology was adopted in the white paper presenting a survey of SOA open standards and the specifications [7]. The paper contains the analysis of the specifications proposed by such organizations as OASIS, Object Management Group (OMG) or The Open Group (OpenGroup) which includes among others, the following:

- Reference Model for SOA [8] - OASIS
- SOA Governance Framework [10] - OpenGroup

The survey also included such positions as SOA Modeling Language (OMG), SOA Ontology or Service Integration Maturity Model (OpenGroup). However, because of the fact that these aspects are not relevant to the scope of this dissertation they are omitted in further discussions.
Fig. 2.1: Positioning of SOA-related standards

Figure 2.1 presents the positioning of the SOA-related standards and their categorization in the context of abstraction and coverage, as presented in the white paper [7]. The partial category includes the RAs which cover specific subsystems of the whole environment like integration or security. The end-to-end category includes the RAs which offer the comprehensive solution in the all layers of the SOA systems. As shown, the OASIS Reference Model of SOA and Reference Architecture Foundation for Service Oriented Architecture are the most abstract ones. They offer only a partial level of coverage, yet still very close to the end-to-end solutions. The more concrete specification is the OpenGroup Governance Framework. However, it only refers to the governance aspects of SOA solutions. The most comprehensive specification presented in the paper is the SOA Reference Architecture from the OpenGroup. Figure 2.1 depicts also the industry related standards presented in both [6] and [7]. It includes ones proposed by the Association for Retail Technology Standards, Hotel Technology The Next Generation Association or the IBM company.

This dissertation aims to propose a generic solution for proactive adaptation, yet with a working implementation prototype which meets the enterprise-class systems quality. Moreover, the concepts proposed in this thesis expose vast capabilities of distributed, hierarchical deployment which entails that the selected standard should comply with them as much as possible. Taking into account that the IBM’s S3 model is the most comprehensive one from all of the introduced models, has hierarchical design and is defined on a proper level of abstraction it was decided that the work presented in this dissertation will be based on top of it. Selected S3 model is described in more details in the following section.
2.1.2 S3 Model

The S3 model proposed by the IBM is one of the main models of SOA application development and deployment. It provides an architectural definition of the SOA which is built upon nine layers. Their structure and relationships are depicted in Figure 2.2. Each of the described layers introduces its own logical (building blocks, design decisions etc.) and physical (applicability of the logical elements in reference to concrete technologies and products) aspects. The S3 model is based on assumption that a given SOA objective is described by a set of functional and non-functional service requirements. Each of such specific service requirements can be fulfilled by a combination of nine proposed layers through layer-specific mechanisms.

![SOA Solution Stack](image)

**Fig. 2.2: SOA Solution Stack**

The layers of the S3 stack are as follows (in a bottom-up order): Operational Systems, Service Components, Services, Business Process, Consumer, Integration, QoS, Information Architecture, and Governance and Policy. A brief description of each S3 layer is following:

*Operational Systems* - includes all legacy applications and hardware assets used in an IT operating environment. Typically, it includes a virtualized IT infrastructure which results in improved resource utilization and manageability. Such property could be effectively used in the development of an adaptive infrastructure, which guarantees the specified level of computational resources accessibility.

*Service Components* - contains the software components, which represent the operations of services. Service components reflect the functionality and QoS of the service they represent. Each component flexibly supports the composition and layering of atomic services. They also help to conceal the low-level implementation details from higher layers elements.
Chapter 2. Technological Background and Related Work

**Services** - consists of all the services defined within SOA environment. In the S3 model, a service is defined as an abstract specification of one or more business-aligned IT functions. It provides the consumers with sufficient information to invoke the endpoints exposed by service providers. The services are either implemented by assembling components (atomic) or other services (composite). Because the process of assembly can be realized dynamically it is possible to support it by the adaptability mechanisms.

**Business Process** - provides the mechanisms for the service assembly process. The assemblage of the services may be realized either through a choreography or orchestration [13]. As a result, a business process is defined which can be exposed to the consumers. The invocation flow in the process can also be constructed dynamically in accordance with a specific adaptation policy.

**Consumer** - provides mechanisms that handle interaction with the customer including the communication protocols to access specific services or business processes.

**Integration** - integrates horizontal layers through built-in capabilities (supported by an Enterprise Service Bus (ESB) [14]) which enable mediation, transporting and routing service requests from different endpoints defined in the ESB.

**Quality of Service** - specifies the needs for the QoS governance mechanisms which are driven by the characteristics of SOA environments such as: heterogeneous computational infrastructures, loose coupling of services or decentralized SLAs.

**Information Architecture** - covers the key aspects related to information-issues involved in developing business intelligence with the use of data warehouses. It includes stored metadata, which is needed to correctly interpret business-valued information.

**Governance and Policy** - defines all aspects of business operations lifecycle management. It includes different types of policies, from manual governance to the enforcement of autonomous policies. The governance framework specifies that the layer-specific policies include service-level agreements based on QoS and key process indicators of a given layer. However, the Governance and Policy layer can be superimposed onto all other layers of the S3 model.

The layered architecture of the S3 model makes it possible to develop all the layers separately and exposes which of them influences each other. For example, the quality of observed services is not only the result of the components activity but also depends on computational resources used during their execution. It illustrates the role of the Operational Systems layer on the higher ones and leads to the facilitation of the multilayer adaptability.
2.1.3 SOA Environments

This section describes the environments, technology stacks as well as abstract patterns which are used to implement service-oriented systems. Their analysis will be used to verify whether some of them could be useful while implementing the S3 model extensions in the area of SOA-compliant systems proactive adaptation.

One of the most commonly used patterns during integration of SOA environments is the ESB\[18,19\]. The ESB is an architectural pattern which was first defined by Roy Schulte in [20] as ”... a new architecture that exploits Web services, messaging middleware, intelligent routing, and transformation ...”. It allows for effective integration of legacy systems and applications and their transformation into a SOA compliant ones. The pattern is technology agnostic, however there exists a variety of vendors which offer software exposing the ESB capabilities. The most recognized ones are MuleESB [21], Apache ServiceMix [22], or Microsoft BizTalk [23]. The scale of the ESB implementation efforts by the industry companies can be seen in a sequence of Forrester Research reports [24–26]. As can be seen, the leading companies are the top ones from the IT industry, including Software AG, Oracle, IBM and Progress Software.

Similarly to the ESB pattern there was also an effort to propose the Java related standard for implementing the SOA architecture. It resulted in two Java Specification Requests (JSRs) committed to the Java Community Process (JCP): Java Business Integration (JBI) 1.0 (JSR 208) [27] and JBI 2.0 (JSR 312) [28]. The idea was to propose a pluggable architecture for the containers that host components used in the integration process. The external services could connect to the container using the Binding Components (BCs) and put a request to a given Service Engine (SE) for an execution. The normalized messages traversed through the Normalized Message Router (NMR) in accordance with a specified Message Exchange Pattern (MEP) (which is taken from the WSDL 2.0 specification [29]). The JBI defines standardized packaging for BC and SE, which allows its components to be interchangeable between different vendors of JBI-compliant containers. Although, the first version of specification was successfully finished and implemented in such products as GlassFish (OpenESB [30]) or Apache ServiceMix, it did not gain a wide publicity. It resulted in the withdrawal of the second version of JBI in 2010.

Another approach to implementing the applications that follow the SOA principles is the Service Component Architecture (SCA) [31]. The specification defines a model for composing applications and systems through assembly process of low level system services and components. Those applications can contain not only new services but can also encompass business functions from existing solutions, which are reused as a part of composition. As the SCA aims to maintain programming language and application environment neutrality, it allows for accessing the services and components through common standards such as Web Services [32], messaging systems [33] or Remote Procedure Call (RPC) [34]. The separate technical committees that oversee the work on the SCA specification proposed a set of extensions for such languages as C++, Java, PHP or
BPEL. The SCA is widely adopted in the industry which is supported by the fact that multiple open source projects implement this specification. It includes such projects as: Apache Tuscany [36], FraSCAti [35] or Fabric3 [37].

The OSGi technology [38] is a component-oriented framework which is commonly called the "SOA-inside the JVM". Thanks to such features like modularity, life-cycle management and dynamism of created applications it is becoming more and more popular. Its architecture is designed in accordance with service-orientation principles while maintaining a rich set of enterprise-class capabilities (specified by such extensions as OSGi Service Platform [39] and OSGi Enterprise [40]). There is a lot of research work realized in the context of communication mechanisms between the OSGi container like [41] or [42]. The former proposes its own lightweight binary protocol, the latter utilizes the Message Oriented Middleware (MOM) approach. The OSGi is widely accepted and utilized in different aspects of the IT industry i.e. in the internals of application servers like JBoss AS, GlassFish and many more.

All of the aforementioned technologies and patterns represent the means for implementing the SOA compliant environment. However, there are also several more comprehensive solutions (which in some cases rely on the technologies already introduced) including the open sourced as well as proprietary ones.

JBoss Enterprise SOA Platform [43] is a platform (certified by Red Hat) for creating an Enterprise Application Integration (EAI) and SOA solutions. It is a standard-based solution build on top of JBoss Enterprise Application Platform (JBoss Developer Studio, JBoss Web Framework Kit, JBoss AS - OSGi-based), JBoss Fuse ESB [44], Business Process Management engine (jBPM) [45] and a Rules engine (JBoss Drools) [46]. Altogether, the platform provides the core infrastructure for developing and executing SOA applications in form of messaging, clustering, remoting and management services. Thanks to the provided Integrated Development Environment (IDE) integrated with all of the provided solutions it simplifies the process of developing, testing, deploying and maintaining applications in the runtime environment.

The SOA Platform Project [47], maintained by Eclipse, is another open sourced platform that supports creation of the SOA applications. It aims to provide the solution compliant with such standards as JBI, BPEL, WSDL, BPMN, WS-*, OSGi and SCA. However, there is a very poor activity in the development of the platform and some of its sub-projects were archived in January 2014. It makes an impression that the whole project will be withdrawn soon.

The IBM SOA Foundation [48] is a platform that supports the realization of the S3 model. It encompasses the tools, programming model and techniques for implementing the applications and middleware infrastructure hosting them. It includes the solutions from such software families like: WebSphere, Rational, DataPower, DB2 and Tivoli. Altogether, the platform offers a huge functionality which overcomes all of their open sourced equivalents. Yet, the solution is proprietary and its price is also tremendous.
The Oracle SOA Suite \cite{49} is the comprehensive and highly integrated solution for building, maintaining and monitoring distributed SOA implementations. It enables a single, unified IDE (JDeveloper) which offers a very rich set of tools supporting the design and analysis of the developed applications. The platform enables to define the policies related to end-to-end security and unified metadata management. Moreover, it is compliant with multiple standards including the SCA and WS-*. The most important software packages that build this platform are as follows: JDeveloper, Oracle Service Bus, Oracle Policy Manager, BPEL Process Manager, Oracle Business Process Manager and Oracle Business Rules. When compared to other SOA solutions it scores the top places.

The reviewed technologies and comprehensive solutions provide extensive capabilities for SOA application development and management. However, none of them supports the capabilities of adaptation enforcement, even on the conceptual level. Yet, those examples act as good reference points for discussion when selecting a proper technological stack for the prototype implementation presented in this dissertation’s Chapter \cite{5}.

Before evaluating the research related to the concept of proactive adaptation of multi-layered systems (c.f. Section \ref{sec:adaptability_overview}) it is crucial to define the scope of the research conducted in this dissertation in the area of adaptability. In order to do so, the evaluation of adaptability concepts and methods is necessary. The description of this research efforts, which include the identification of the most closely related areas from the adaptability taxonomy, is realized in the next section.

\section{2.2 Adaptability Concepts}

This section describes the concepts related to the adaptability of the software and the methods of its realization. The description is made with particular emphasis on two aspects of adaptation control loop: detecting when adaptation has to occur and deciding what actions should be enforced.

The aspects of adaptability has been widely studied in the research work in the last years. First, it is crucial to answer the question why the adaptability is needed and then evaluate how it can be achieved. The primary reason for the need for the adaptive capabilities of the software is the increasing cost of maintaining specified goals in the context of complex software systems \cite{50}. Moreover, in recent years, there has been an increasing demand to deal with the given goal maintenance at runtime. The increased level of heterogeneity (of both software and hardware components) and more frequent changes of goals and requirements cause that the adaptation mechanisms are needed. However, such mechanisms are expected to take actions at a minimal cost and in a timely manner. In order to do so, it is crucial to monitor the system itself as well as its execution environment (context) to detect the changes and act accordingly.
Current status-quo of the literature studies shows that there exist a few concepts related to the software adaptability. The most mature one is the autonomic computing\[51, 52\], next there is the adaptive software\[55, 56\] and the self-adaptive software\[58, 59\] which in fact depict the same concept but are named differently. All of these concepts are described in detail in the following paragraphs\[1\].

The autonomic computing concept was first announced by the IBM in 2001 in\[51\] and further described in a more generic way by Kephart and Chess in\[52\]. In general, the autonomic computing systems expose the properties of self-management. Such a system continually monitors its own execution and, if necessary, reconfigures itself. When it detects errors it will revert to the last working version and try to isolate the source of error through automatic problem-determination algorithms. The self-management capabilities of autonomic computing system refer to the following aspects:

- **Self-configuration** - instead of manual installation, reconfiguration and integrations the system follows the high-level policies. Such policies enforce the automated configuration of components and allow the system wide adjustments in an automatic and seamless manner;

- **Self-optimization** - system components and software elements continually seek for the improvements of their performance, efficiency and cost of maintenance;

- **Self-healing** - typically the software systems only inform about errors or other problems. In the autonomic computing, system not only automatically detects but also diagnoses and repairs the localized software and hardware problems;

- **Self-protection** - despite the existence of firewalls and other intrusion detection software the system automatically defends against the attacks or cascading failures (in places where self-healing was not able to help). Moreover, the system anticipates the problems and takes steps to avoid them.

The concept of autonomic computing was further widely elaborated in the literature. The Huebscher and McCann\[54\] provided a broad evaluation of applicability and evolution of this concept in the different research areas. In addition, Kephart\[53\] shows that there are still many research challenges related to this concept. It includes forecasting logic of autonomic elements and interactions among multiple autonomic elements to achieve system-level goals which are the most interesting from the perspective of this dissertation.

In\[56\] Chen et al. defined the adaptive software as a software that can react to changes in the execution environment or changes of user requirements by switching algorithms at runtime. To create such software it is crucial to address the following issues:

\[1\]The details about the origin of adaptive control are provided in the further sections of this chapter.
Chapter 2. Technological Background and Related Work

- Change detection - ability to detect changes and evaluate if they are significant enough to trigger adaptation. In multi-layer distributed systems it requires the global agreement to determine whether an adaptation should be made;

- Adaptation policy - ability to decide which components within a layer should make an adaptation when the change is required. Typically there can be multiple options which offer different trade-off;

- Coordination - ability to enforce an adaptation in a coordinated manner so that the system functionality is preserved while the adaptation is in progress.

Aforementioned issues, are realized by the combination of adaptive components which are coordinated by the inter-layer adaptation controller. A more generic view of the adaptive composition of software is presented in [55] by McKinley et al. The authors propose very similar definition of adaptive software and evaluate the possibilities of its realization. The authors describe that the adaptation may refer to the modification of program variables (parameter adaptation) or to the structure or behaviour of the system components (compositional adaptation).

The concept of the self-adaptive software is very similar to the adaptive one. In fact the Oreizy et al. [58] defined the self-adaptive software as a software which modifies its own behaviour in response to changes in its operating environment (anything that is observable by the software system). This is almost the same definition as already mentioned one by Chen et al. [56]. In [60] Salehie and Tahvildari point out slight differences between all three concepts (autonomic computing, adaptive software and self-adaptive software) which results in a conclusion that the autonomic computing is the broadest concept as it takes into account all the layers of the systems while the self-adaptive software is mainly focused on the topmost layers (applications and middleware). They also state that the self-adaptive software is a closed-loop system with a feedback loop aiming to adjust itself to changes during its operation. It means that every system compliant with autonomic computing properties is a self-adaptive one but not the contrary. In both cases it is necessary for the system to monitor its execution context, detect changes, decide and react in accordance with specified policies.

2.2.1 Control Loop Building Blocks

In [60] authors present a high-level representation of adaptation loop which constitutes activities realized during the adaptation, as presented in Figure 2.3.
As can be seen, the whole process is built on top of the sensors and effectors. Sensors monitor software in order to generate a collection of data reflecting the state of the system, while effectors rely on the system internal mechanisms to apply changes to it. The four activities can be summarized as follows:

- Monitoring - process responsible for collecting data from sensors. In this process data is converted into behavioural patterns and symptoms. It can be achieved by the event correlation, threshold checking, etc.;

- Detecting - process responsible for analysing the symptoms created by the monitoring. It is done in order to detect when a change in the system is necessary and to publish a proper request for transition. It helps to identify the root source of a transition to a new system state which is not a deviation from the given goals;

- Deciding - process responsible for selecting actions that have to be taken in order to achieve the best outcome of transiting system into the different state. The same outcome can be realized by performing a different set of actions;

- Acting - process responsible for applying the actions selected by the deciding process. It includes the mapping of actions to what is provided by the effectors and their underlying adaptation techniques.

Presented high-level representation of adaptation loop was realized in different manners in the literature. However, in all of them the meaning is derived from the one presented in Figure 2.3. In the context of autonomic computing the adaptation loop is realized by the autonomic element. Such an element consists of the autonomic manager which supervises the control loop (the MAPE-K loop) in order to adapt the behaviour of one or more managed resources [52]. The MAPE-K loop constitutes of four phases: Monitoring, Analysing, Planning and Executing and is extended by the shared utilization of Knowledge. On the other hand, Dobson et al. [61] define a slightly different realization of control loop. They define the term of autonomic control loop in the context of autonomic communication which constitutes of Collect, Analyse, Decide and Act phases.
Particular phases of proposed approach are also extended in the area of user interaction and reporting. Oreizy et al. [58] refer to the adaptation loop as adaptation management. This concept is composed of several steps that support planning/enacting changes, collecting/evaluating observations and deploying change descriptions. The whole process is realized in parallel to constant changes of managed system which allows for dynamic evolution of given adaptation strategies.

It is assumed that the concepts presented in this dissertation will comply with the MAPE-K pattern. It is supported by the fact that the selected [S3] model naturally fits into that pattern, which was a basis for its construction. The managed resources of the MAPE-K pattern represent the entities proposed by the [S3] model (services, service components, etc.) and enforcement of the control loop enables them to be transformed into their adaptive equivalents. The Knowledge part of this pattern plays extremely important role while performing decision making in the adaptation process. In the context of proactive systems the proper acquisition and representation of the knowledge are the most crucial steps as they stand as a core element of the whole adaptation process.

### 2.2.2 Taxonomy of Self-Adaptation

There is a large amount of papers that evaluate and specify the taxonomy related to the self-adaptation concepts. In the [63], Bradbury et al. describe a variety of approaches for specification of dynamic software architectures, specifically self-managed architectures. Although, the paper does not propose the exact taxonomy definition it offers a well structured formal categorization from structural and behavioural adaptation perspectives. On the other hand, McKinley et al. [62] propose a taxonomy of compositional adaptation. The paper evaluates a set of representative projects and describes a taxonomy that distinguishes approaches for software composition from following perspectives: i) how - what specific mechanisms are used to enable compositional adaptation; ii) when - at what time the adaptive behavior is composed with the business logic; iii) where - in which place the adaptation logic is inserted (middleware or application). The taxonomy proposed in the paper was further evaluated from the service oriented systems perspective by the author of this dissertation in the [64]. The most comprehensive proposal of self-adaptation taxonomy is proposed by Salahie and Tahvildari in [60]. In this survey paper authors propose the hierarchical taxonomy that unifies the current state of the art classification in the area of adaptation. Figure 2.4 illustrates the structure of proposed taxonomy.
Fig. 2.4: Hierarchical taxonomy of self-adaptation

The division of the introduced taxonomy is realized on two levels. The first level includes such elements as: Object to adapt, Realization issues, Temporal characteristics and Interaction concerns. Each of introduced elements contains the second, and in some cases third level of division (which is omitted in this description).

The Object to adapt facet deals with the following issues:

- what attributes or properties of the system can be changed through enforcement of adaptation actions;
- which layer and in which extent need to be changed during the adaptation.

These aspects are described in details in three subcategories: Layer, Artifact & Granularity and Impact & Cost. The first subcategory refers to the issues of deciding which layers of a given system should be taken into account during enforcement of the adaptation actions. It may include only a high-level division of layers, such as: middleware and application, or more decomposed one where the system execution environment is divided into sub-layers (as in the aforementioned S3 model). The second subcategory refers to the aspects of adaptation granularity. The adaptation process may change the system resources behaviour or their architecture. Moreover, the adapted system may be decomposed into services, components, methods or some other subsystems. Therefore it is crucial to specify at which level of granularity, fine or course, the adaptation will be enforced. The third subcategory refers to the actual impact of the adaptation and cost of its enforcement. Typically the adaptation actions can be categorized in two classes: weak or strong. The former adaptation includes modifying system parameters, which is typically a low-cost and limited-impact solution. The latter includes the runtime recon-
configuration of system elements (components, services, etc.) to improve the overall system quality, which is a high-cost and broad-impact solution.

The *Realization issues* facet deals mainly with the aspects related to determining how the adapted resources can be changed and which adaptation actions are appropriate to be applied in a given system condition. It is further divided into two subcategories: *Location* and *Decision Process*. The former one refers to the location of incorporating adaptation mechanisms into the system. It is further divided into following sub-facets:

- **Static/Dynamic Decision Making** - deals with the aspects of how the deciding process can be constructed and modified. In the static approach the process is hard-coded, while in the dynamic one it is externally defined in a form of rules or policies.

- **External/Internal Adaptation** - deals with the aspects of adaptation logic location. In the former there is an adaptation engine which steers the adaptable software, while in the latter software is self-adaptive (embeds the adaptation logic).

- **Making/Achieving Adaptation** - deals with the aspects of obtaining knowledge in the adaptation process. In the former there exists the adaptation strategy which is defined during the development phase. In the latter one there exist the knowledge learning techniques.

The second subcategory, *Decision Process*, refers to the characteristics of the decision making process during adaptation enforcement. It is further divided into following sub-facets:

- **Close/Open Adaptation** - deals with the aspects of introducing new adaptation actions into the system. In the former there exists only a fixed set of adaptive actions while in the latter there is a possibility for extension of the adaptation logic as well as dynamic inclusion of new system entities by the adaptation mechanisms.

- **Model-Based/Free Adaptation** - deals with the aspects of adaptation process model. In the former adaptation mechanisms utilize a model of the system and its context, while in the latter these mechanisms do not depend on a predefined model. It is dynamically learned on the basis of system requirements, goals and possible alternatives.

- **Specific/Generic Adaptation** - deals with the aspects of the generality of the adaptation process. In some cases the process may be technology or application specific. On the other hand, it may also be more generic and matched for a given domain through the configuration and policy management.

The *Temporal characteristics* facet deals with the following issues:
• when the an adaptation actions do need to be applied and how often they have to be enforced: continuously or in a on-demand manner;

• is it enough to enforce the adaptation actions reactively or the online prediction capabilities are needed.

These aspects are described in details in two subcategories: Reactive/Proactive and Continuous/Adaptive Monitoring. The first subcategory refers to the issues of adaptation enforcement timing. In the former, the adaptation actions are produced when a given condition already occurred (i.e. SLA entry was violated, request processing is already throttled etc.). In the latter, the system characterises with anticipatory or predictive properties which allows to act proactively and avoid some of negative situations. Thus, it affects the cost of the system upkeep and maintenance. The second subcategory refers to the aspect of monitoring process realization. When the monitoring is continually collecting the data it causes a higher resources usage (CPU, disk storage or even network bandwidth). In case when it is being adaptive the total overhead can be significantly lower. Thus, not only lowering the cost of monitoring but also affecting the overall fault detection time.

The last facet, Interaction Concerns, deals with the aspects of interacting with operators or other subsystems during the adaptation process. It is divided into three subcategories: Human Involvement, Trust and Interoperability Support. The first subcategory elaborates on the maturity model of the adaptive system which can be one of following: basic, managed, predictive, adaptive and automatic. It includes the interaction of the human from two separate perspectives: i) the involvement of human operator in the adaptation process - should be minimized or none; ii) the ability to fit the adaptation process by the operator through definition of policies or goals - should be as broad as possible. The second category deals with the aspects of the security and trust of the adaptation mechanisms itself. The former is rather self-explanatory, the latter specifies how much human or other systems can rely on the system and how the system status and the visibility of adaptation processes are exposed. The third category deals with the aspects of the system interoperability. It includes not only the interoperability of adaptation mechanisms located in different layers of the system but also the maintenance of data and behaviour integrity across all elements of the distributed system.

As shown in Figure 2.4 there are two elements highlighted in the proposed taxonomy: Layer and Reactive/Proactive. These elements are the most closely related to the thesis and scope of this dissertation. In accordance with the aspects of adaptation in a multi-layered environments are ones of the top research challenges in the last years. The authors state that: "The application of the centralized control loop pattern to a large-scale soft- ware system may suffer from scalability problems. There is a pressing need for decentralized, but still manageable, efficient, and predictable techniques for constructing self-adaptive software systems. A major challenge is to accommodate a systematic engineering approach that integrates both control loop approaches with decentralized
agent inspired approaches”. Moreover, the additional research in this area \cite{66,68} proves that understanding of the control loops adaptation mechanisms interaction in case of distributed complex systems is also a major challenge. Moreover, most of the concrete research projects and frameworks implemented for this purpose are still tackling only a single layer (mostly the application one), omitting the management aspects of layered and hierarchical systems. The situation is similar in the context of temporal characteristics of adaptation process. In most cases the adaptation enforcement is realized in a reactive manner which results in losing the opportunities being the outcome of online prediction mechanisms. The anticipation capabilities of software is one of the major research challenges as presented in \cite{60}, \cite{65} or \cite{67}.

The next section elaborates on the research activities in the aforementioned areas, especially the ones realized in the area of the proactive adaptation of multi-layered service oriented systems.

### 2.3 Research Related to Proactive Adaptation of Multi-Layered SOA Systems

This section elaborates on the research conducted in the area of online prediction and analysis of multi-layered systems. It also focuses on the comparison of comprehensive solutions which is used as a basis for specification of the requirements for the solution supporting proactive adaptation of service oriented systems.

In order to define the concepts of online prediction or cross-layer management it is necessary to introduce the term of the adaptive control which was already heavily utilized in the scope of the control systems. Yet in the late 60’s there was a lot of research in this area such as \cite{121}, \cite{122} or \cite{123}. It can be summarized that the concept of adaptive control relies on the extension of the well-known pattern of closed control loop $^2$. The block diagrams representing the both concepts are depicted in Fig. 2.5. Both of the presented control loops refer to a particular System block as a single monolith solution. In case of service-based systems this concept has naturally evolved into the form in which such a control loop is applied to each layer of the system separately (or to selected subset of them) and the controllers communicate with each other to supervise the dependant system altogether.

Fig. 2.5a illustrates the concept of closed control loop. This approach assumes that the actual input of the System is regulated by Controller on the basis of the output measured by a given set of Sensors. Thanks to that it is possible to control system behaviour depending on the actual output and predefined logic of controller. Fig. 2.5b depicts the concept of adaptive control loop. This approach allows for more sophisticated methods of system control. The Adjustment Mechanisms can change the behaviour or

$^2$Introduced in the Control Theory
Chapter 2. Technological Background and Related Work

Fig. 2.5: The closed control loop vs. the adaptive control loop

The comparison of those two loops makes it possible to describe the relationship between the two types of actions which can occur in the system: control actions and adaptation actions. The control actions enable the possibility of influencing the system state or behaviour in the predefined areas. It involves the modification of the system configuration or parameters in accordance with a specified logic of the controller. The adaptation actions extend the capabilities of control actions in two areas. First, they allow for modification of the controller behaviour itself. It is done by changing either the controller parameters or functionality (used algorithms, internal structure, etc.). Second, they not only interact with the system input to achieve specified goal but also with its system structure. It means that the structure or behaviour of the system itself (not only the controller) is also a target of adaptation. The data related to both of these aspects is gathered from a different set of sensors, which are able to sense the system internal state.

2.3.1 Online Prediction and Analysis

The concept of proactive control in industry processes has been widely elaborated on in the literature. The most standardized approaches are based on the method of Model Proactive Control [120]. It is based on the iterative optimization of given variables in the system in accordance with the cost-minimizing control strategy. The optimization is driven by the internal model of analysed system and is realized in predefined time periods. The shift of time slots in the following iterations of optimization makes this type of system optimization called receding horizon control. The predictive control loop representing the Model Proactive Control method is depicted in Fig. 2.6. Similar to the previous case the System block should be perceived as a single layer of the service-based environment.
Fig. 2.6: The predictive control loop compliant with Model Proactive Control method

The output measured by Sensors as well as the input created by the Controller are used by the Predictor in order to produce the predicted output of the system. Such prediction may influence the system input provided by the Controller.

The presented concept of the proactive control can easily be mapped in the SOA compliant systems. Yet, the service oriented environments are commonly characterized by the fact that there exists a service contract (defined in form of a given SLA which specifies whether observed system behaviour (often represented as a set of metrics) is satisfactory or not [69, 70]. In many cases, aforementioned concepts of the control loops are incorporated in the adaptation process of service oriented system to make sure that the specified SLA is not (or will not be) violated. The introduced models of control loop realization influence the type of the adaptation process which may be realized in two different approaches: reactive and proactive. In both approaches the analysed system is monitored at runtime in order to gather the data important for the process realization.

In the reactive approach the data is used to find the SLA violations that happened. If such violations are found it means that the system is in a state that is not desirable and some adaptation actions should be executed to move the system into the state that does not violate SLA. In the proactive approach the data is analysed in order to anticipate possible SLA violations. If such violations are predicted then it is possible to enforce adaptation actions to prevent them.

Fig. 2.7 depicts the exemplary process of adaptation to meet the specified SLA constraints. The plot presents the value of single metric (service response time) over the time. There are three different states of the system presented:

- $S_{v1}$ – red dashed area - consumer perspective SLA violated state;
- $S_{v2}$ – yellow dashed area - provider perspective SLA violated state;
- $S_a$ – green pointed area - SLA acceptable state.

The desirable one is the state $S_a$, while the states $S_{v1}$ and $S_{v2}$ should be avoided. The values $Y_{max}$ and $Y_{min}$ represent the metric constraints. The former illustrates
the expectations from the consumer perspective and values higher than it means that the system is in state $S_{v1}$. The latter illustrates the expectations from the provider perspective and values lower than it means that the system is in state $S_{v2}$.

Fig. 2.7a presents the values of the exemplary SLA metric (service response time in seconds) while reactive adaptation enforcement takes place. There are two periods in time when the SLA contract is violated, respectively $T_I$ and $T_{II}$. In the period $T_I$ there is a occurrence of SLA upper bound violation ($Y_{\text{max}}$) while in the $T_{II}$ there is the occurrence of SLA lower bound violation. When the SLA violation occurred the reactive approach enforced some adaptation actions (respectively $AA_I$ and $AA_{II}$) which successfully traversed the system into the state $S_a$. The main drawback of the reactive adaptation is that it allows for situation where SLA is violated and only then the adaptation process is triggered.

Fig. 2.7b presents the values of the exemplary SLA metric (service response time in seconds) while proactive adaptation enforcement takes place. Online analysis of monitoring data showed that it is necessary to perform adaptation actions (respectively $A_I$ and $A_{II}$) in order to avoid the SLA violation. The $T_I$ and $T_{II}$ represent periods of time after enforcement of adaptation process. Introduced example has shown that the proactive adaptation successfully predicted the SLA violation and preserved the $S_a$ state.

The following can be stated while summarizing the aforementioned exemplification of different adaptation types. The proactive approach to adaptation, if successfully enforced, leads to more efficient and accurate decision making process, thus minimizing the occurrences of possible SLA violations. Yet, incorrect enforcement of proactive behaviour typically causes the dependant system to fluctuate unexpectedly, thus generating more costs than savings. Therefore, the proactive capabilities should be employed only if the possible gains (i.e. lowering costs related to violating a given SLA) will be bigger than costs (i.e. higher resource consumptions, advanced techniques of knowledge acquisition and processing).

The actual realization of proactive adaptation enforcement always relies on mechanisms...
of online prediction which has been widely elaborated on in the literature. Salfner et al. \[71\] propose a categorization of the online prediction and analysis aspects. Taking into account the scope of this dissertation they can be grouped into two categories:

- **failure-based long- and short-term prediction**;
- **prediction of resource-scarcity and service level agreement fulfilment**.

The **failure-based long- and short-term prediction** category represents areas where prediction mechanisms are used to anticipate an occurrence of the next failure in the system and propose some alternatives to avoid it. Such area is evaluated by Csenki \[72\] where he proposes to utilize the concept of a Bayesian predictors. The predictor is used to estimate the probability of distribution of the next failure through knowledge acquisition realized during previous system failures. Proposed idea relies on Jelinski-Moranda’s software reliability model \[73\] improvement for the better failure estimation. A slightly different approach is proposed by the Pfefferman and Cernuschi-Frias \[74\]. The authors state that the actual failures in observed system may vary depending on resource utilization, changes in configuration etc. Therefore, a stationary estimation (with a given time window) which base on histograms gives a poor results. It is assumed that the failure process is a Bernoulli experiment where the occurrence of a system failure at some time depends on the failure type. Thus, they propose a method to estimate probability of its occurrence using an autoregressive averaging filter (modifying analysed data windows size) which gives a better probability distribution of Time Between Failure (TBF) metric.

There is also a number of techniques related to short-term online (failure) prediction. Chen et al. \[75\] propose to track the requests made to some services (exposed in J2EE application server) and internal components which take part in such invocations. On this basis it is possible to correlate the component invocations with a given failure which occurred in the system and then, using the Jaccard similarity coefficient to estimate the similarity of particular component sets. Finally, in case of a failure it is possible to propose a set of components which can be used alternatively. If none of such sets exists then the failure is imminent. This approach is commonly named path-based prediction and was further evaluated by Chen et al in \[76\] and Kiciman et al. in \[77\]. Another approach is proposed by the Elbaum et al. \[78\] where authors propose to predict on the basis of system model during normal (failure free) behaviour. In this approach, the current measured system behaviour (or state) is compared to the expected one and if there is a significant deviation the failure is predicted. The authors evaluated this concept in a scenario of low-level function calls of a given software (in their case a simple command line mail client). Finally, there is also a research which supports decision making in short-term prediction. In \[79\] Daidone et al. propose a generic solution for supporting the transparent diagnosis process of software components suffering whether permanent, intermittent or transient faults. The approach is based on the utilization of hidden Markov models (to evaluate the state of a given component and its eventual deviation from a nominal behaviour) through continuZous probing of component outputs.
The solution is compared with the existing optimal solutions relying on the Bayesian inference theory and proved to have the higher accuracy.

The second group of aforementioned categorization is the prediction of resource-scarcity and service level agreement fulfilment. In this category the prediction is generally realized through time series analysis and extrapolation of given data. It is realized through the employment of the statistical analysis techniques for trend detection. In [80] Gerg et al. propose a methodology for the estimation of a resource exhaustion time. The solution gathers the system metrics from networked UNIX stations and processes them using the statistical trend detection methods (robust locally weighted regression proposed by the Cleveland et al. [81]). When a given trend is confirmed its slope is estimated using the linear regression method - the least square estimation. Having the slope, the authors estimate when the resources will be exhausted. Similar approach is proposed by the Castelli et al. in [82]. The prediction of resource exhaustion is realized by the curve-fitting analysis and projection algorithm. It operates on a sliding window of the observed data points. The sampling mechanisms are reconfigured accordingly to the length of the window. The implementation was tested and included as a part of IBM Director package named xSeries Software Rejuvenation Agent.

In the context of the service oriented environments the prediction is often related to a given SLA which typically refers to the evaluation of domain specific QoS metrics. Furthermore, the complexity of the SOA compliant environments entails more compound architectures of prediction frameworks. In [83] Ivanovic et al. propose an architecture for prediction of the SLA violations in environments supporting service orchestrations. Proposed solution relies on a constraint-based QoS predictor that is created with the use of Constraint Logic Programming subsystem [84]. Although, the details of high-level architecture are provided there is not enough information about the exact technique used to predict the values of a given SLA. In [85] Leitner et al. propose a PREvent framework that integrates event-based monitoring and prediction of SLA violations using the machine learning techniques. The framework is applied to a sample application constructed with the Web Services. The events gathered during service invocations are published to a metric processor. It is assumed that there exist the checkpoints in which the prediction of a given set of metrics is made. The estimations are computed with the use of the regression classifier and further compared with specified thresholds. To make the prediction more accurate the framework evaluates two metrics (mean prediction error and prediction error standard deviation) which are taken into account during making a decision if a predicted violation is accurate or not. In [86] Aschoff and Zisman propose a ProAdapt framework which can be used for proactive adaptation of service composition. The prediction mechanisms used for evaluation of a given SLA are based on function approximation and failure spatial correlation. Estimations of a given QoS metrics are created with the use of exponentially weighted moving average [87]. The framework was successfully evaluated in the three case studies that were implemented in Apache Axis2 and BPEL4WS technologies.
All of the described papers prove that there is already a tremendous amount of research
done in the area of online prediction and analysis. Yet, in most cases the solutions
are proposed for a very narrow domain and in a limited context. Therefore, in order to
provide a better perspective on the current status quo in the area of proactive adaptation
of multi-layered systems a comparison of more comprehensive solutions is done in the
next section.

2.3.2 Overview of Existing Solutions

The analysis of the research related to the proactive adaptation of software systems
shows that there already exist a few solutions that offer comprehensive functional capa-
bilities. Moreover, current trends in the area of software development process show that
the complexity of newly developed technological stacks directly influences the design of
systems created with their use. It is demonstrated in the form of multiple abstraction
layers which are needed in the case of simple CRUD-like systems. In the context of the
service-oriented systems the software architectures are even more complicated. Such
situation drives the need for solutions that will be capable of enabling the adaptation
capabilities for such highly-layered environments.

Table 2.1 presents the comparison of selected frameworks that are related to the concept
of proactive adaptation of multi-layered systems. The presented features were selected
on the basis of research challenges related to the scope of this dissertation as well as
industry principles for high-quality and scalable software. The comparison was created
on the basis of the following high-level characteristics:

- **Proactive behaviour** - verifies the ability of a given solution to predict the future
  system states, resource consumption or other QoS-related metrics.

- **Standard/Pattern compliance** - evaluates whether the proposed approach is com-
  pliant with some standard or pattern related to adaptation realization.

- **Multi-layered adaptation** - ensures that the evaluated solution architecture is de-
  signed to perform the adaptation in a layered execution environment.

- **Decentralized architecture** - verifies whether the architecture of the proposed so-
  lution is prepared for a distributed deployment and can successfully work in de-
  centralized application scenarios.

- **Runtime reconfiguration** - evaluates whether it is possible to (re)configure the
  proposed solution at runtime or the configuration has to be provided at start-up
  and cannot be changed later.

- **Technology agnostic** - ensures that the proposed approach is technology agnostic
  thus making it possible to deploy it in a variety of concrete scenarios and industry
  domains.
In [88] Heilscher et al. present the PROSA framework which is the result of their work on the S-CUBE project [90]. The novel concept proposed in the framework is to utilize the online testing solutions to proactively enforce adaptation. It is assumed that the individual services or service compositions of a given Service-based Application (SBA) can be externally tested. The test scenario contains a set of test cases that invokes them in parallel and independent of operating applications. The final activity of adaptation triggering depends on the results of those test cases which include deviations from expected functionality or QoS. The definition of the SLA may be provided either externally or built-in a given SBA in the form of meta-data like annotations. The presented concepts are evaluated in multiple scenarios and seem to be a novel approach for proactive adaptations. The authors propose an extended and more generic version of their work in [89] where monitoring and testing areas of the framework are better integrated.

In [91] Cardellini et al. present the MOdel-based SElf-adaptation of SOA systems (MOSES) framework which is aimed for QoS-driven adaptation of service oriented systems. The approach proposed in the paper is compliant with the MAPE pattern with the following phases: M - Runtime monitoring, A - Modelling, P - Adaptation mechanisms, E - Implementation. It is assumed that the analysed workflow of a given SOA-compliant system is modelled in accordance to the MOSES grammar. On top of the modelled flow the QoS metrics and adaptation policies are specified. The MOSES dynamically switches the concrete service invocations (in accordance with specified adaptation policy) in order to maintain the specifics of the QoS for abstract/composite services. In [92] the authors present the extended and more mature version of the framework, especially in the context of adaptation policy and QoS model.

Gjorden et al. [93] propose a solution for middleware managed adaptation. In this approach the services are enriched with a description that specifies their behaviour which is further utilized by the middleware to satisfy the behavioural requirements. The proposed framework, QuA Middleware, adopts the principles of mirror-based service reflection and exposes their meta-data such as implemented interfaces, concrete implementations of the service and desired QoS. The mirror also includes so called QoS predictors which encode the developers knowledge about the QoS delivered by the service implementation under different contextual conditions. Thanks to that, it is possible to anticipate how the services will behave in the future and modify their instances at runtime. In [94] the authors extend the proposed middleware with capabilities of cross-layer adaptation. It is done through the introduction of multiple service platform implementations that provide the meta-data description of adaptation targets. The actual solving of cross-layer actions is realized by a single instance of service planner.

Guinea et al. [95] propose a framework that allows for the installation of multi-layered control loops in the service-oriented systems. The solution is compliant with MAPE pattern and proposes the following phases of its realization: M - Monitoring & Correlation; A - Analysis of Adaptation Needs; P - Identification of Multi-layer Adaptation Strate-
gies; E - Adaptation Enactment. The framework enables the adaptation enforcement in both infrastructure and software layers. The solving of adaptation decision is done by the manager named CLAM (Cross-layer Adaptation Manager) which is capable of determining which adaptation strategy should be applied in a given context. The manager is ranking the available strategies using the decision trees techniques. In \[96\] the authors further evaluate the aspects of adaptation strategies ranked in CLAM, especially in the area of fuzzy logic.

In \[97\] Schmieders et al. propose an approach for cross-layer adaptation in order to prevent SLA violations. The framework architecture is comprised of two subsystems: SALMon (monitoring and evaluating of the service QoS metrics) and SPADE (evaluates the impact of SLA violation on specified application requirements). The solution can be deployed in the distributed environment as it is using multi-agent platform for managing a given service composition instance. The adaptation can be enforced on both service and infrastructure level. The authors of the framework do not provide any specific details of the SLA prediction mechanisms, although they can be found in \[98\].

<table>
<thead>
<tr>
<th></th>
<th>PROSA</th>
<th>MOSES</th>
<th>QUA</th>
<th>CLAM</th>
<th>SPADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proactive behaviour</td>
<td>+</td>
<td>-</td>
<td>+/−</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Standard/Pattern</td>
<td>−</td>
<td>+</td>
<td>+/−</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>compliance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-layered</td>
<td>+/−</td>
<td>+/−</td>
<td>+</td>
<td>+</td>
<td>+/−</td>
</tr>
<tr>
<td>adaptation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decentralized</td>
<td>−</td>
<td>+</td>
<td>?</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>architecture</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime reconfiguration</td>
<td>+/−</td>
<td>−</td>
<td>+/−</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Technology agnostic</td>
<td>−</td>
<td>+/−</td>
<td>+</td>
<td>+/−</td>
<td>?</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of selected related frameworks

+ – solution supports requirement;
− – solution does not support requirement;
+/− – partial support for a given requirement;
? – no information provided.

A brief comparison of the selected frameworks presented in Table 2.1 shows that there are some solutions that provide a subset of specified requirements. Yet, there is no system with decentralized architecture that can be directly used in the area of the proactive adaptation of the multi-layered service-oriented systems which provides the capabilities of runtime reconfiguration.
2.4 Summary

As described in this chapter, the SOA principles have been widely adopted by the industry in the recent years. Although, there is also a lot of research areas which need a thorough attention, especially in the domain of proactive adaptation of multi-layered systems. As defined in the S3 model there is an exact place for such an area, Governance and Policy Layer, which enables to superimpose the desired adaptation mechanism onto other layers of supervised systems. The complementary solution must expose the following features:

- the identification of multi-layered dependencies and its further utilization in the adaptation process;
- the support for predictive reasoning during adaptation enforcement;
- capabilities of adaptation process realization from a global perspective, with special emphasis on inter-layer relationship.

Although, there exists a variety of works on proactive adaptation or cross-layer management, there is no single solution that could be used in the given area as a whole. Following chapters describe a comprehensive solution that is able to solve the stated problem of proactive adaptation of multi-layered service oriented systems. Introduced concepts are designed in accordance with the MDA principles. The work presented in this thesis includes the description of defined requirements, technology agnostic proposal of PIM model and technology specific Platform Specific Model (PSM) model which are built on top of the aforementioned requirements. Designed architectural patterns provide a strong foundation for the solution compliant with the ACI paradigm.

It is assumed that some of the high-level system characteristics (such as decentralized architecture, runtime reconfiguration, etc.) will be constructed on the basis of the concepts and technologies presented in Section 2.1.3 and 2.3.2. Among others, the short- and long-term predictions will be based on well-known trend estimation techniques, the matchmaking capabilities will use the event patterns matching concepts, while the reasoning capabilities will utilize the rule-based approach. Yet, the application of those concepts and techniques will be done in a novel approach which enables the realization of proactive adaptation of service-oriented systems.
Chapter 3

Concept of Proactive Adaptation in Multi-Layered, Distributed SOA Environments

The main goal of this chapter is twofold. First, it tries to define a conceptual reference model which establishes some high-level boundaries related to the aspects of proactive adaptation in service oriented systems. Such boundaries are important as they show in which way the adaptation process itself should be perceived in the context of such complex and dynamic environments. Second, it proposes the high-level abstract models which could be used in the process of adaptation enforcement. These models do not provide a concrete realization techniques but rather point out the most crucial aspects which should be considered during designing and implementation of such systems. The proposed concepts enable the fulfilment of the dissertation goals in the subsequent chapters, which present the more concrete realizations of the proactive adaptation loop of SOA compliant environments.

The structure of this chapter is as follows. The first section introduces the requirements of a system supporting realization of proactive adaptation in multi-layered environments, which result from the analysis of technological background and related work done in the Chapter 2. The abstraction of the adaptation process realized by such system is presented in the second section. It also includes the proposal of Property Relationship Model which stands as a basis while enforcing the adaptation. The third section presents in detail the proactive adaptation process which includes further utilization of the Property Relationship Model (PRM) during online prediction. Section four introduces the concept of Adaptive Controller and its possible structuring schemes. Most of the aforementioned concepts are presented with the use of simple cases, which is valuable from
the perspective of Chapter\textsuperscript{4} in which the platform independent architectural design is proposed.

The core concept used during the adaptation enforcement (regardless of the type of adaptation) is the \textit{PRM} model. It describes the dependencies between the system elements located in different layers of the system. The model is constructed dynamically through correlation of \textit{QoS} metrics specified for a given set of system elements (such as services, components, virtual resources etc.). In the context of multi-layered environments, the lower layer often acts as an execution environment for the higher one, thus it is assumed that dependencies between their elements should be observable.

The \textit{PRM} itself does not offer any capabilities of proactive behaviour. Yet, on its basis the adaptation actions can be taken because it points classes of possible system states which can be achieved through modification of various system element properties. The overall process relies on the weight of the edges between the \textit{PRM} vertices. The higher the weight of the edge, the bigger dependency between the two system elements can be observed. When the \textit{SLA} violation is foreseen (thanks to the utilization of long-/short-term online prediction and events matchmaking) the \textit{PRM} is evaluated to select the adaptation actions set that will traverse the system between those two states. Such a set of actions is then applied by a given hierarchy of \textit{AC}.

3.1 System Requirements

The requirements for the system that allows adaptation enforcement in service oriented environment result from investigation of the existing technologies and current state of the art solutions presented in Chapter 2. The goal of the system is to provide the solution which exposes high-level features specified in Section 2.4. In order to do so, a set of the system requirements has been stated, which are follows:

\begin{itemize}
  \item the solution should propose a generic methodology which can be used to transform the \textit{SOA}-compliant system into the adaptive equivalent. It should expose not only the capabilities of reactive adaptation but also the proactive one;
  \item the designed architecture of the proposed solution should take into account multi-layered, distributed nature of service-oriented systems. Thus, being able to the enforce adaptation process on multiple layers of the system accordingly to the specified adaptation policy;
  \item the adaptation process should be realized in an automatic manner in accordance to the specified high-level declarative adaptation policy which contains rule related to a given \textit{SLA}. The adaptation enforcement itself should take into account the dynamic dependencies between the elements located in different layers of the supervised system;
\end{itemize}
• the system must be able to change its current SLA and adaptation policy at runtime. Thus, it will result in the reconfiguration of adaptation mechanisms and QoS metrics in all the supervised layers accordingly;

• the system should expose capabilities of distributed deployment and communication in a scalable and efficient manner.

### 3.2 Abstraction of Proactive Adaptation in Multi-Layered Environments

This section presents the abstract view of the proactive adaptation concept in the domain of multi-layered service oriented environment. The analysis of the S3 model (described in Section 2.1.2) enabled the possibility to propose a concept map which can be used to describe the execution environment of the multi-layered service oriented systems. The proposed concept map is presented in Figure 3.1.

![Concept map presenting the simplified version of SOA execution environment model adopted from the S3](image)

The presented meta-model assumes that each service oriented system consists of a given number of software layers which are supported by the dedicated infrastructure layer. The infrastructure layer contains a set of infrastructure resources. In all cases, these resources are hosting the software artefacts deployed in other layers of SOA compliant system. The resources present in the infrastructure layer may be either physical or virtualized ones. The fact is that in the recent years more and more platforms or solutions use the latter one. The main reason is that the benefits of hardware virtualization include improved service-level agreements, availability and cost-efficiency. Each of the
virtualized (or physical ) resources has a specific set of properties which may refer to such aspects as: networking, storage, memory and processing capabilities.

The software layers are located on top of the infrastructure. They include a set of Software Artefacts which are used in a modelled SOA compliant system. It refers not only to components, services or business processes, but also to their software execution environments which is used to manage the lifecycle of given artefacts, such as: business processes engine (like BPEL) or services container (like Apache ServiceMix). However, to maintain the clarity of the presented meta-model only the Abstract Service and Abstract Component elements are presented in the Figure. Each of the Abstract Services may use a given set of Abstract Components as well as include the invocation of other Abstract Services. Both of them represent the abstraction of a given functional entity which may be realized by different Concrete Instances. The Concrete Instance of an abstract entity, similarly to the infrastructure resources, exposes a set of properties. These properties reflect the characteristics of a given instance and may refer to such aspects as: memory/CPU utilization, instance availability and response time.

Analysis of a multi-layered service oriented system and specification of its particular components in accordance with presented SOA Execution Environment Meta-Model enable to create a Concrete Execution Model. As shown in Figure 3.1 the Concrete Execution Model refers to the following elements of the presented meta-model: virtualized and physical resources, abstract services and components definitions, concrete instances specification and properties exposed by all of the aforementioned elements. It is important to point out that there may exist some relationships between the properties of specified system elements. Those dependencies, in addition to the concrete model, can be further utilized during the realization of the adaptation process, especially in the proactive manner.

Fig. 3.2: High-level concept map of adaptation process in service oriented environments

Figure 3.2 presents the high-level concept map of adaptation process proposed by this dissertation. The whole concept is based on the foundation that a given multi-layered service oriented system is modelled in accordance with the aforementioned meta-model,
thus it exposes its Concrete Execution Model. Furthermore, the supervised system itself has defined a set of QoS Metrics which are used to specify the agreement in the form of SLA. The adaptation process is driven by the Declarative Adaptation Policy. Typically it refers to the specified SLA and defines how to maintain its fulfilment in the analysed system. The Adaptation Policy is also specifying the Adaptation Action Types which are used when instantiating the concrete Adaptation Actions. These types specify the operations which can be enforced in the supervised system, for example: the addition or removal of service instances, the addition of resources in the infrastructure layer etc. The actual realization of adaptation process is realized by the federation of the Adaptive Controllers. The controllers are driven by the Adaptation Policy and realize the specified adaptation strategies in accordance with the current system state represented by the Concrete Execution Model and QoS Metrics. The actual deployment and communication scheme of the controllers is domain specific and relies on the structure of the supervised system itself. The details of possible configuration will be presented further in this section.

The presented approach is agnostic from the perspective of the adaptation type. It means that it can be used to enforce the adaptation actions in the reactive manner as well as in the proactive one. Figure 3.3 presents the detailed concept map of the adaptation process realization in the proposed approach.

The main assumption is that the information about supervised system can be divided into two categories. One is representing the system state and the other the aspects of system quality measurement. The former exposes the properties specified in the system as well as its Concrete Execution Model (abstract and concrete instances of
system elements). The latter shows aspects related to quality metrics measured in the system including their compliance with agreed service contract, which is mapped onto the compliance with the lower or upper bound constraints defined in the SLA.

As shown in Figure 3.3, the Declarative Adaptation Policy contains references to the supervised system service contract entries. On its basis the Adaptive Controllers enforce the adaptation process. Despite of the aspects related to the system state and quality, which support the decision making of the controllers, there are also two additional elements which are utilized by them: the relationship knowledge about the system elements properties and the predictive measurement of QoS metrics. The relationship knowledge block represents the abstraction of the Property Relationship Model (PRM) which is utilized during adaptation process. As shown it refers directly to two different categories of elements: i) the properties of services, components or resources (expressed in a form of Concrete Execution Model), ii) QoS metrics related to those properties.

The utilization of proper mechanisms of online prediction during the realization of adaptation process enables the Adaptive Controllers to anticipate upcoming violations of the agreed contract and enrich the supervised system with proactive adaptation capabilities. Depending on the context of predicted violations it is either possible to prevent their occurrence or minimize the cost of SLA exceeding. It is a well-known fact that unnecessary adaptation actions can cause a variety of negative effects in the system. In case of adaptation supported by the predictive mechanisms such effects are compounded. Among others, the lack of accurate decisions in the adaptation process may lead to:

- system state fluctuation - adaptation mechanisms are producing bad decisions which entails traversing between different system states;
- adaptation delays - some adaptation actions which are executed unnecessarily entail delays in execution of another ones;
- excessive resource consumption - enforcement of adaptation actions is by default resource consuming therefore should not be done when not needed;
- unpredicted failures - the execution of adaptation actions leads to SLA violations which would not happen otherwise.

### 3.3 Realization of Proactive Adaptation Process

This section presents detailed description of proactive adaptation process steps proposed in this dissertation. Despite of the fact, that it describes the realization of adaptation process in a proactive manner, all of the presented concepts can be easily applied in its reactive counterpart. In order to utilize them during such process a given subset of components responsible for online prediction should be disabled, thus preventing the
system from being able to predict the SLA violations and making it aware only of the observed ones.

Figure 3.4 presents the abstraction of the proactive process. As shown it is comprised of three main phases (Property Relationship Model Evaluation, SLA Violation Online Prediction, State Transition Enforcement) which execution in a continuous manner allows for proactive adaptation of multi-layered service oriented systems.

The Property Relationship Model Evaluation phase is the foundation of the whole process. The phase encompasses two building blocks: Static PRM Definitions and Dynamic PRM Evaluation. The proposed approach assumes that the information about the relationship between different elements of the system may be provided in both static and dynamic way. Static PRM definitions are provided by the system administrator which has the appropriate expertise and is able to specify how specific (particular) system elements metrics are related to each other. The dynamic evaluation of PRM model should correlate the values of a given system element metrics and compute their dependencies in a normalized form. It can be either achieved through the utilization of correlation coefficient or distance metrics. The most recent values stored in PRM model are used in two following phases.

The SLA Violations Online Predictions is the second phase which occurs during realization of proactive adaptation process. On the basis of the multi-layered definition of the SLA, the initial values of the QoS metrics, trends estimation queries and matchmaking
patterns are constructed. Trends are used to foresee whether a current values of the QoS metrics will violate the agreed SLA. The presented model of the prediction process does not determine the realization method of the trends estimation mechanisms as it is the matter of the specific implementation details. In case when QoS metrics or trends show that the SLA is violated (or the violation is foreseen) a proper notification is published. The situation is similar when matchmaking algorithm finds a set of events that preceded a SLA violation.

The occurrence of the SLA violation triggers the third phase of proactive adaptation process which is named *State Transition Enforcement*. In this phase a few steps are executed in order to select a set of adaptation actions which can be applied to remove or avoid given SLA violation. In these steps, the possible adaptation actions are evaluated in order to find the ones that will be acceptable, in term of given SLA violation. The selected adaptation actions are executed in the supervised environment which results in the invocation of effectors deployed in the system during the initial instrumentation. The details about particular steps undertaken in this phase are provided further in this chapter.

### 3.3.1 Property Relationship Model Evaluation Details

The core concept exploited by the adaptation process presented in this dissertation is the PRM model which represents the relationship knowledge block shown in Figure 3.3. The model represents the dependencies between the properties and/or metrics related to different elements of supervised system and is agnostic from the perspective of the adaptation process type (can be used either in reactive or proactive solutions).

![Diagram of Property Relationship Model](image)

**Fig. 3.5: Foundations of Property Relationship Model**

Figure 3.5 tackles the foundations that were assumed while defining the PRM model.
The basis assumption of the PRM model is that there exists the relationship between any two system elements present in the service oriented environment (namely $SE_n$ and $SE_m$) which can be expressed as dependencies between properties exposed by given elements $(P^n_1, P^n_2, P^n_3, P^m_1, P^m_2, P^m_3)$ having a specific value expressed as an edge weight ($w^1$, $w^2$, $w^3$, $w^4$). A single system element represents either a concrete instance of abstract service/component or infrastructure resource which were aforementioned earlier in the Figure 3.1 presenting SOA execution meta-model.

The structure of the PRM can be perceived as a directed weighted graph between different properties. The definition of vertices and edges is as follows:

- $v^k_n$ – a vertex of the graph representing a single system element property tuple. A tuple contains a reference to specific system element $se_n$ and one of its properties $p^k_n$. In this notation, $n$ denotes a unique identifier of the system element (i.e. the number of a graph node), $k$ stands for the number of the $k_{th}$ property exposed by such element;

- $e^nkml$ – an edge of the graph representing a relationship between a property $p^k_n$ and the property $p^l_m$ of the system elements $se_n$ and $se_m$. All edges in the graph are directed (from one property to the other) and describe which property is influencing the other and to what extend in accordance to specified weight $w$.

- $f(v^k_n, v^l_m)$ – a function assigning weight to the edge between two vertices. It results in the correlation coefficient and represents the dependencies between two vertexes: $v^k_n$ and $v^l_m$. Computed value $w$ should be from $< -1, 1 >$ where: $-1$ means that vertexes are perfectly negatively correlated while $1$ perfectly positively correlated. The values closer to zero, the lower the correlation and thus less impact between the given vertexes. The actual weights of the edges are evaluated at runtime and may change during the system uptime thus the structure of the PRM model may change while the execution environment of the supervised system is changing.

In addition, each of the system elements can be labelled with a category tag. The categories act as a logical overlay on all of the system elements allowing to group them. This division is further used while specifying the communication schemes of the Adaptive Controllers (c.f. Section 3.4). The simplest categorisation of the system elements may be expressed in accordance with the layered structure of the supervised system such as: Operational System Layer, Components Layer, Services layers etc.

Such definition of the PRM model expresses the fact that the change of a single property value can affect the values of the others (in accordance with their correlation coefficient). Taking into account that properties are exposed by elements from different categories, which typically will be mapped onto Layers of supervised system, it can be stated that

---

To maintain the clarity of the PRM description further usage of the 'property' phrase represents either property or metric of such property.
changes occurring in one layer (e.g. Operational Systems Layer) can lead to changes in another one (e.g. Services Layer).

The generic form of the proposed model allows for its appliance even when there are complex properties, constructed on top of the other ones. Thanks to that, the end user can specify his high-level expectations in more complex properties, which will trigger the mapping process to their factors in lower levels. It is especially important during defining the agreed service contract. In this case, the top level properties will be directly addressed in a given SLA. In turn, this will trigger the metric value evaluation in accordance with property definition which will map to measurement of basic properties. The work presented in this dissertation introduces the approach which automatically discovers the dependencies between system properties and utilizes this knowledge during SLA prediction and adaptation actions enforcement (c.f. Section 4.3).

The PRM model itself can be constructed in two different ways: static definitions or dynamic identification. The former assumes that the model is provided by the domain expert which models the actual relationships between system elements. The latter, more advanced approach, assumes that the model is dynamically constructed on the basis of the system operation. The dynamic evaluation of PRM values can be typically done through the computation of correlation coefficient between a given QoS-related metrics.

In order to simplify the adoption of the PRM model a clear but still comprehensive example is necessary. Let us assume that we want to construct manually the PRM model for a simple service oriented system presented in Fig 3.6.

We assume that the system elements categories are defined in an exact way as their membership to the given system layers (Operational Services, Components or Services). There are two service types - S1, S2 - present in the Services Layer. The former is bound to two service components types: SC1, SC2, the latter only to component with of SC3. All of the components - SC1, SC2, SC3 - are deployed in resource with type VM1. The
number of instances of particular system elements (services, components or resources) is depicted at the bottom of the graphical representation of each element.

The properties in a given system are defined in a following way:

- **Services Layer:**
  - $P_{S1}$ - cost [eurocents] – a fee that is expected to be paid for a given service invocation;
  - $P_{S2}$ - response time [milliseconds] – a duration of the execution of a given service;
  - $P_{S3}$ - availability [percentage] – a probability that a given service is accessible when required for use at a given moment.

- **Components Layer:**
  - $P_{SC1}$ - cost [eurocents] – a fee that is expected to be paid for usage of a given component;
  - $P_{SC2}$ - CPU consumption [MHz] – a value of CPU resources which is needed for a given component;
  - $P_{SC3}$ - memory consumption [MB] – a value of memory resources which is needed for a given component.

- **Operational Systems Layer:**
  - $P_{VM1}$ - cost [eurocents] – a fee that is expected to be paid for a given resource;
  - $P_{VM2}$ - CPU availability [MHz] – an available CPU that a given resource is exposing;
  - $P_{VM3}$ - memory availability [MB] – an available memory that a given resource is exposing.

The $\text{PRM}$ constructed for a presented use case is as follows:

$$
S = \{ S_1, S_2, SC_1, SC_2, SC_3, VM_1 \}
$$

$$
V = \{
  v^{1}_{s1} = (S_1, P_{S1}), v^{2}_{s1} = (S_1, P_{S2}), v^{3}_{s1} = (S_1, P_{S3}),
  v^{1}_{s2} = (S_2, P_{S1}), v^{2}_{s2} = (S_2, P_{S2}), v^{3}_{s2} = (S_2, P_{S3}),
  v^{1}_{sc1} = (SC_1, P_{SC1}), v^{2}_{sc1} = (SC_1, P_{SC2}), v^{3}_{sc1} = (SC_1, P_{SC3}),
  v^{1}_{sc2} = (SC_2, P_{SC1}), v^{2}_{sc2} = (SC_2, P_{SC2}), v^{3}_{sc2} = (SC_2, P_{SC3}),
  v^{1}_{sc3} = (SC_3, P_{SC1}), v^{2}_{sc3} = (SC_3, P_{SC2}), v^{3}_{sc3} = (SC_3, P_{SC3}),
  v^{1}_{vm1} = (VM_1, P_{VM1}), v^{2}_{vm1} = (VM_1, P_{VM2}), v^{3}_{vm1} = (VM_1, P_{VM3})
\}
$$

where:

$S$ – a set containing all system elements;
Chapter 3. Concept of Proactive Adaptation in Multi-Layered, Distributed SOA Environments

\[ V - \text{a set containing all vertices (nodes) of the graph;} \]

\[ v^y_x - \text{naming convention used to represent vertex of } x^{th} \text{ system element and its } y^{th} \text{ property.} \]

Typically the edges between the vertices will be created dynamically through online metrics correlation of a given system elements properties. Although, in most cases, the typical relationship between CPU related metrics of dependant layers system element could be observed (similar approach can be assumed for the cost-related properties), which results in the creation of the following edges:

\[ E = \{ \]
\[ /* \text{virtualized resources influence on hosted service components} */ \]
\[ (v^1_{vm1}, v^1_{sc1}, w^1), (v^1_{vm1}, v^1_{sc2}, w^2), (v^2_{vm1}, v^2_{sc1}, w^3), (v^2_{vm1}, v^2_{sc2}, w^4) \]
\[ /* \text{virtualized resources influence on hosted services} */ \]
\[ (v^1_{vm1}, v^1_{s1}, w^5), (v^1_{vm1}, v^3_{sc1}, w^6), \]
\[ /* \text{components influence on services that utilize them} */ \]
\[ (v^1_{sc1}, v^1_{s1}, w^7), (v^1_{sc2}, v^1_{s1}, w^8), \]
\[ (v^2_{sc1}, v^2_{s1}, w^9), (v^2_{sc2}, v^2_{s1}, w^{10}) \} \]

where:

\[ E - \text{a set containing the edges of the graph.} \]

The relationships between selected subset of introduced properties are depicted in Fig. 3.7. To maintain the clarity of the figure only the edges which influence properties of service S1 are presented.

Fig. 3.7: Representation of relationships between properties in a sample service oriented system
Chapter 3. Concept of Proactive Adaptation in Multi-Layered, Distributed SOA Environments

As can be seen, the properties of $S_1$ are directly influenced by the properties values of $SC_1$ and $SC_2$. In addition, there is a transitive dependency of $VM_1$ properties. For example, if there is a need to lower the overall cost of the service $S_1$ it can be done by lowering the cost of $SC_1$ (i.e. choosing a cheaper instance) or lowering the cost of $VM_1$ which implies the reduction of $SC_1$ costs. The actual weights of the edges should be computed on the basis of a given correlation coefficient adopted in the analysed scenarios such as Paersons correlation method (when $QoS$ metrics change slowly in accordance with the linear characteristic), Spearmans rank correlation (when $QoS$ metrics values match the monotonic functions) etc.

3.3.2 Online Prediction of SLA Violations Process Details

This section describes the aspects related to the realization of the proactive adaptation. The process related to online prediction of $SLA$ violations is driven by the adaptation policy which directly refers to the specific $SLA$ constraints (as shown in Figure 3.3). Such high-level goal is transformed by the Adaptive Controllers into a multi-layer representation of agreement which is realized either by each controller separately or by a global one. The enforcement of this agreement on the lower levels ensures the fulfilment of the high-level one. A layer specific $SLA$ definition triggers the execution of three processes related to the proactive adaptation, which are executed in parallel: $SLA$ metrics evaluation, trends estimations and symptom matchmaking.

The $QoS$ metrics evaluation mechanisms are activated in accordance with defined goals. Depending on the $PRM$ different metrics may be activated, even those which are not explicitly defined in the $SLA$. It is possible because the $PRM$ model expresses relationships between different system elements through the correlation of $SLA$ related metrics. Thanks to that the system is able to predict how the eventual violations of the $SLA$ propagate in the system. In general the $QoS$ metrics should be activated for all the properties which are dependent on the ones specified in the $SLA$. The goal of $SLA$ metrics evaluation is to produce events in case when there is a violation of the agreements (in a reactive manner). Events representing a violation of the low-level $SLA$ are then processed in order to verify the impact on the $SLA$ of the higher level.

The trends estimation mechanisms that allow for prediction of $SLA$ violations that may occur if the system continues its current execution (in the context of both long- and short-term). The estimation of possible $SLA$ violations should utilize the information gathered during the system execution such as verification whether the generated predictions were correct or not. This ensures that eventual false positives of $SLA$ violation prediction will be eliminated in the future. Similarly to the first process, the utilization of the $PRM$ model allows for estimations of trends which are not directly specified in a given $SLA$ but may affect its fulfilment in the future.

The symptom matchmaking is an additional way of the possible system behaviour pre-
Chapter 3. Concept of Proactive Adaptation in Multi-Layered, Distributed SOA Environments

(a) Symptom-based knowledge acquisition

(b) SLA prevention through predictive match-making

Fig. 3.8: Symptom matchmaking realization phases

diction. The proposed approach assumes that each Adaptive Controller (in addition to previously introduced trends estimation of QoS related metrics) is constantly monitoring the events that occur in the supervised system (as presented in Fig. 3.8a). When a SLA violation occurs the Adaptive Controller correlates the events that preceded it for the purpose of further utilization. Next, when new events are generated in the supervised system, the Adaptive Controller is performing the predictive match-making (c.f. Fig. 3.8b). During that process it evaluates whether the actual sequence of events does match any of previously correlated patterns. If there exists the pattern that has been matched the Adaptive Controller triggers a notification that a SLA violation preventive action is needed.

There are few steps which are realized in order to obtain and analyse data in the three aforementioned sub-processes. According to the SLA defined for a given layer the specific monitoring mechanisms are activated. Each activity that occurs in the system and refers to the system element present in the Concrete Execution Model triggers the generation of an event. This event holds the information about the property (or metric) and contains both old and new value of such property. Next, all the events are formed into the streams. Each stream is related to a single property or metric which allows for correlation of multiple streams in given time period. The events that match SLA related metrics (including indirect ones expressed by the PRM model) are used to construct high-level ones which represent the actual or predicted violation of SLA for a given layer.

This process can be applied for identification of both types of SLA violations (observed and predicted). The observed SLA violation meaning is straightforward and states that the system is in not acceptable state and should be moved into another one. The trends estimation process used for online prediction can be realized with a variety of the distances thus providing information about short- or long-term violations. On the one hand, the greater window size allows for earlier anticipation of possible SLA violations,
however the quality of such prediction is reduced. On the other hand, the narrower window size allows for better prediction quality but leaves a much shorter time period for adaptation process to occur. In fact, the size of the prediction window is application specific and should be specified in accordance with the dynamisms characteristics of the system. The most comprehensive solution should allow for the adaptive adjustment of prediction window length. In case when the prediction window length is zero then the system acts as a reactive one.

In both cases the SLA violations are exposed in a form of complex events to other Adaptive Controllers in accordance with their actual communication model. It allows them to be aware of the actual situation in different layers looks like as well as orchestrate the overall adaptation process from a higher supervising entity. Either way, the creation of SLA violation event triggers the second stage of proactive adaptation named State Transition Enforcement.

The process of the system adaptation is triggered when one or more SLA violations occurred or were predicted. However, this process is not trivial and requires extensive knowledge about the possible actions which can be realized to improve some of the metrics or properties of the system. In order to avoid the SLA violations it is necessary to select proper actions (in a limited time window) to transit current system state into the one that will not violate it. Those actions may be related to two different aspects of influencing the system state. One type of action refers to the control of system resources such as increasing the number of available memory and computational resources. However, in some cases these actions are not sufficient (we are already using the whole of the available resource or the structure of analysed system is limiting our capabilities). Similarly, the more advanced mechanisms have to be applied, the more the structure of the analysed system will be altered e.g. by adding new instances of services or modifying the internal structure of components.

The process of adaptation actions enforcement is steered by a given adaptation policy. Such policy specifies a set of actions which can be executed in a dependant system in the context of different system elements. The exemplary adaptation action types may be as follows: addition of new service instance, modification of virtualized resources etc. Such actions contain information how their execution influences metrics specified in the SLA agreement. For example, the addition of new service instance influences positively response time metric of a given service type. In order to decide which actions should be enforced to prevent the SLA violation, this dissertation proposes the utilization of PRM model from which the information about properties dependencies can be extracted. The overall goal is to select these actions that have (directly or indirectly) positive impact on a given SLA entry and does not influence negatively other entries defined in the service contract.

To achieve such results the process iteratively analyses the PRM model, selects adaptation actions and evaluates them in terms of their correctness and quality. As mentioned
earlier, given action is checked whether its enforcement will not trigger violation of other constraint through online prediction of SLA related metrics. If a satisfactory set of actions is found then they are passed to execution. This behaviour is realized by a dedicated decision engine deployed in an Adaptive Controller which supervises the dependant system. In case when the SLA violation occurs or is predicted the decision engine evaluates whether there exists the adaptation action (specified in the declarative adaptation policy) that can be directly applied to improve the quality of violated metric. If so, it verifies whether it is possible to do so, thanks to the verification of current system monitoring data, available resources etc. If it is possible to improve the quality in such way the decision engine marks such actions as possible for execution. If there is no such possibility then the PRM model is analysed to fetch metrics that are correlated with the one that will be violated and the process is executed once more. Finally, all of the actions marked as possible for execution are validated in the context of whether their enforcement will not violate any other SLA entries specified in the adaptation policy. On the basis of the correlation coefficient the decision engine ranks the selected actions and outputs its best match (the action set that gives the best improvement of the violated SLA metric and the lowest overhead in the context of other SLA entries).

The step of Transition Enforcement is layer and technology dependant therefore it will not be tackled in this chapter. However, it assumes that the selected adaptation actions are realized with the use of Effectors deployed during the realization of System Instrumentation.

The next section introduces the details of architectural pattern that enables the realization of proactive adaptation in the service oriented environments. The pattern is called the Adaptive Controller and its capabilities of hierarchical deployment and communication enables the realization of adaptation process in a flexible and scalable manner. The description includes not only the different configurations of Adaptive Controllers hierarchical structure, but also details of the internal sub-processes enabling the enforcement of given adaptation strategies.

### 3.4 Adaptive Controller Concept and Design

This section describes the structure of the Adaptive Controllers concept, their hierarchical deployment and communication scheme as well as mutual cooperation in order to enforce a given adaptation goal. The process presented in Section 3.3 is realized by a given configuration of Adaptive Controllers which are steered by the Declarative Adaptation Policy, as presented in Figure 3.9. The actual evaluation of the PRM model may be either done by each controller separately or only by a selected subset of supervisory ones.
As shown in Figure 3.4, the Declarative Adaptation Policy specifies the initial configuration of Adaptive Controllers. Such configuration defines the actual communication scheme of Adaptive Controllers (c.f. Fig. 3.10). This configuration should be driven by the structure of the PRM model which expresses the inter-layer dependencies or in case of systems which will express some complex relationship between particular system elements. The inaccurate hierarchy of the Adaptive Controllers may lead to the wrong adaptation decisions which in turn may generate more costs than benefits.

Each of the Adaptive Controllers involved in the adaptation process also performs the instrumentation of a system layer for which it is responsible. Such instrumentation enforces the initial installation of adaptation mechanisms which allow for the dynamic management of sensors and effectors.

The structure of such controller is compliant with the Autonomous Manager concept, which is widely used as an approach for the creation of autonomous systems. Presented design is a generalization of adaptive control loop presented in Fig. 2.5 with extension of

---

4When talking about the layers of dependant system one should be borne in mind that this is just a simplification of a previously introduced categorization of system components. It may happen that multiple Adaptive Controllers will be managing the same layer of the supervised system, yet a different set of components. Such situation may take place when we want to divide the supervisory roles on a per-hardware resources basis.
adaptation strategy which can be applied in more sophisticated software architectures compliant with SOA principles.

The Adaptive Controller constitutes of three functional blocks which altogether enable the realization of proactive adaptation process. These block are named as follows: Goal Coordination, Goal Assessment and Goal Execution.

The Adaptive Controller takes the Adaptation Goal as an input and evaluates it in the Goal Coordination functional block. It encompasses two main building blocks: Policy Evaluation and Strategy Generation. The Policy Evaluation is used to steer the process of Goal Assessment and Goal Execution. The former is realized by providing a given set of Monitoring and Diagnosis Policies and the latter by enforcing a collection of Strategy Selection and Control Policies. The Strategy Generation is used to provide a collection of possible Adaptation Strategies which can be applied in order to fulfil the specified Adaptation Goal.

The Goal Execution functional block is responsible for the enforcement of adaptation and control actions in a resource being the target of adaptation. It consists of two building blocks: Strategy Selection and Strategy Control. The former realizes the process of solving which Adaptation Strategy should be used in the currently observed context of the analysed system. The latter enforces the selected Adaptation Strategy and publishes the control actions to Effectors located in Controlled Layer.

The Goal Assessment functional block realizes the tasks related to the monitoring and analysis of given system layer. It is composed of two blocks: Monitoring and Diagnosis. The Monitoring is receiving the layer specific data measured by the Sensors. If no disturbances of selected strategy are discovered then the Strategy Control continues with currently selected strategy. Otherwise, the violations of low level monitoring goals and policies are published to the Diagnosis block. The Diagnosis analyses the outcomes of the Monitoring and publishes the diagnosed problems and opportunities related to given Adaptation Goal. Such a situation triggers the re-evaluation and selection of the selected Adaptation Strategies. This process is also triggered when the Adaptation Goal changes.

The design of an Adaptive Controller also includes the knowledge utilization during the process of adaptation enforcement. The Knowledge functional block interacts with all the other blocks of the controller. It is composed of two blocks: Prediction Mechanisms and Relationship Model. Both of them are directly related to the concepts presented in Figure 3.3. The former, represents the part of the knowledge which is accommodated by the utilization of online analysis and estimation techniques. Although, it may also be perceived as the knowledge resulting from the enforcement of particular adaptation decisions or provided in a form of historical data about system behaviour. In both cases, the utilization of such knowledge in the process of reasoning allows to predict further system states or SLA violations. The latter, is the representation of the PRM model introduced in the previous section.
The multi-layered nature of service oriented environments implies the layered deployment of Adaptive Controller. Moreover, the structure of a single Adaptive Controller assumes that it can receive the supervisory decisions and other essential information from other controllers. It seems natural to propose a communication model of the Adaptive Controllers as a hierarchical structure in which some of them realize the enforcement of adaptation control loop for specific layers of service oriented systems, while the others acts as supervisory nodes. This assumption is all the more true when we consider the actual dependencies of system element properties reflected in the PRM model. The Adaptive Controllers managing different system elements should communicate with each other when there is a dependency between them. Otherwise, the overall effects of the adaptation enforcement may be negative. It may happen that one of the controllers will execute actions to improve the quality of a given property but will spoil the other one, while another controller will do exactly the opposite. Therefore, it would be desirable for the actual communication scheme of the Adaptive Controllers to be steered by the PRM model structure. Depending on the configuration the Adaptive Controllers may operate on a single/synchronized instance of the PRM model (context sharing or supervisory cooperation model) or may be constructed completely on their own basis (layers independent cooperation model).

The exemplification of possible cooperation models is depicted in Figure 3.10. There are a few different communication schemes between Adaptive Controllers which exercise control over multi-layered service oriented system.

The first approach (c.f. Fig. 3.10a) assumes that each controller is acting independently as an autonomous controller of given layer. The Adaptation Goal which is steering the adaptation control loop is defined for each of such controllers. The main disadvantage of such an approach is that the adaptation process in one layer is realized without knowledge about the others. It leads to the situation when the realization of adaptation goals defined for specific layers may disturb each other, thus leading to bad adaptation actions or event preventing the achievement of given goals.

Fig. 3.10b represents another approach where Adaptive Controllers are communicating with each other. Similarly to the previous scenario each of the controllers is responsible for an adaptation enforcement and is steered by its own adaptation goal. The overall adaptation process realized in the system is still not supervised and it is realized in layer independent manner. However, the Adaptive Controllers exchange the contextual information (their current state, adaptation strategy realization status, etc.) in order to benefit the realization of Adaptation Goal of a given layer.

The most advanced communication model of Adaptive Controllers is depicted in Fig. 3.10c. This approach assumes that the adaptation process in each layer is enforced by separate controller. In addition, there is a supervisory Adaptive Controller which coordinates the realization of this process by supervising the adaptation realized in lower layers. It is done by evaluating the high-level Adaptation Goal and steering the Adaptive
Chapter 3. Concept of Proactive Adaptation in Multi-Layered, Distributed SOA Environments

(a) Layers independent cooperation model
(b) Context sharing cooperation model
(c) Supervisored cooperation model
(d) Hybrid cooperation model

Fig. 3.10: Possible cooperation models of Adaptive Controllers
Controllers accordingly. It is the most advanced approach because the supervisor has the access to all the contextual information of other controllers and can evaluate in which way and in which layers specific adaptation actions should be enforced. While this approach is offering the greatest potential for realization of adaptation in SOA it makes it the most complex and difficult to implement.

The last possibility (c.f. Fig. 3.10d) represents the hybrid approach. In such a situation, the overall model of Adaptive Controllers relationships in the system is a combination of the ones previously introduced.

3.5 Summary

This chapter introduced the high-level concepts and methodology of proactive adaptation process realization in multi-layered service oriented systems. The approach is based on two main concepts. First, the utilization of Property Relationship Model to express the dependencies between the QoS metrics and/or system properties. Second, the deployment of Adaptive Controllers hierarchy in accordance with specified declarative adaptation policy which should reflect the dependencies expressed by the PRM.

The main idea of the PRM construction is to utilize the correlation techniques in order to express the dependencies between different system elements properties. It acts as a basis for the adaptation process realization but can also be further evaluated to support the online prediction mechanisms. Moreover, the evaluation of the PRM makes it possible to select proper adaptation actions in order to prevent a violation of a given SLA agreement. In addition, the PRM model can be analysed to discover the proper structure of the Adaptive Controllers in the context of dependant system. This enables their proper deployment which will reflect the characteristic of the underlying system.

The next chapter introduces the architecture of the proactive adaptation framework which is based on introduced concepts and architectural assumptions. The described solution will be presented from the technology agnostic perspective, which ensures its compliance with the Platform Independent Model principles.
Chapter 4

Design of Proactive Adaptation Framework for Service Oriented Systems

The enforcement of proactive adaptation in service oriented systems is a demanding and complex task. In order to tackle this issue this chapter describes the design decisions of Proactive Adaptation Framework which are based on the concepts of proactive adaptation in SOA systems (Chapter 3). There are three main aspects described in this Chapter. The first describes the high-level design of adaptation process realized by the aforementioned framework including the decomposition into several phases realized by a given hierarchy of Adaptive Controllers. The second is related to the decomposition of system architecture in accordance with the AS3 Pattern thus realizing the adaptation loop compliant with MAPE-K. The third tackles the system behaviour during the realization of adaptation process which includes trends estimation and online predictions in particular.

This chapter describes the design of Proactive Adaptation Framework for Service Oriented Systems named PRIDE. At the beginning, the chapter introduces the PRIDE high-level design of adaptation process realization. It explains how the specific steps of the process described in Fig 3.4 are mapped to the architectural building blocks of the framework. Next, a detailed description of system layers is made. The architecture is compliant with the AS3 Pattern, thus the description is divided in accordance with the following layers: instrumentation, monitoring, management, exposition and policies. Next, the chapter focuses on the system behaviour which enables the realization of adaptation loop. The description includes the introduction of low level mechanisms related to dynamic instrumentation as well as discovery processes, both realized at runtime. Furthermore, the chapter introduces algorithms which are used during the process of
trends estimation which rely on the concepts of events stream processing.

4.1 PRIDE Realization of Adaptation Process

Figure 4.1 presents the high-level design of adaptation process realized by the PRIDE framework. As shown, the overall process is driven by the *Declarative Adaptation Policy*. It is assumed that such a policy is provided by the system administrator and consists of the following elements:

- **SLA Definition** - specifies the SLA defined for the *Supervised System*. The definition includes what the lower- and upper-bounds of the agreed contracts are as well as how particular SLA-related metrics should be computed when there will be multiple instances of given system elements;

- **Adaptation Actions Definition** - specifies the possible actions that can be undertaken by the Adaptive Controllers as well as the possible strategies of their enforcement. Such actions specify which of the SLA-related metrics are affected by their execution. Thanks to that it is possible to select a valid set of actions when the SLA violation is anticipated.

- **Adaptive Controllers Configuration** - specifies the communication and structuring scheme of the Adaptive Controllers.

The deployment of the Declarative Adaptation Policy in the PRIDE framework triggers the set up of the Adaptive Controllers structure. After the initialization of the Adaptive Controller Hierarchy many phases occur simultaneously, yet to maintain the clarity of the description they will be described in an ordered way in accordance with the bottom-up approach.

The most bottom-down phases of the adaptation process realized by PRIDE could be classified as the ones related to the aspects of generic monitoring, which are: *System Instrumentation* and *Observed System State Actuation*. The first one (*System Instrumentation*) realizes the activities related to the enrichment of the supervised system with the software artefacts that allow for its state modification and data extraction in a transparent and non-intrusive way. The second phase (*Observed System State Actuation*) is responsible for acquisition of monitoring data and normalization which influence the execution of other system phases (especially the PRM model actuation and events processing).
Chapter 4. Design of Proactive Adaptation Framework for Service Oriented Systems

specifies
configures
Supervised System
Adaptive Controllers Hierarchy
performs
System
Instrumentation
Observed System State
Actuation
enriches
publishes
data
influences behaviour of
SLA Statements
Initialization
performs
SLA Metrics
Regression
based Trends
Prediction
triggers
SLA Violations
Notifications
Event based Symptom Matchmaking performs SLA Violation Symptom Pattern Management triggers SLA Violations Notifications

supports
Declarative Adaptation Policy
Adaptive Controllers Configuration
Adaptation Actions Definition
SLA Definition
Adaptation Enforcement
Verification and Ranking
performs System State Transition
affects

Fig. 4.1: PRIDE High-level design of proactive adaptation process realization
Chapter 4. Design of Proactive Adaptation Framework for Service Oriented Systems

The PRIDE framework defines a set of phases which main responsibility is SLA verification. It includes the following phases: SLA Statements Initialization, SLA Metrics Regression-based Trends Prediction, Event-based Symptom Matchmaking and SLA Violation Symptom Pattern Management. The former one is responsible for the initialization of events streams and the appropriate statements that will process the gathered events in accordance with a given SLA definition. The initialization of SLA statements triggers the set up of prediction mechanisms which predict the future values of the SLA-related metrics on the basis of regression based trends. If any of the initialized mechanisms match (or predict) a SLA violation, then a proper notification is published.

Next, there are the phases which are directly related to the aspects of adaptation enforcement. They include the following: Adaptation Enforcement Verification and Ranking, Adaptation Actions Evaluation and System State Transition Enforcement. The first of the phrases mentioned above is responsible for constant scoring and the verification of adaptation actions that were previously enforced. It allows the system to gather the knowledge about the correctness of its decision making mechanisms and if needed to alter them automatically. The Adaptation Actions Evaluation is triggered by a notification of SLA violation occurrence. This phase evaluates the possible actions that can be undertaken (in accordance with specified Declarative Adaptation Policy) and heavily utilizes the PRM model (introduced in the previous section) while deciding which of the actions should be applied in the system. The last phase (System State Transition Enforcement) simply enforces the selected actions in the supervised system.

The described phases illustrate the high-level realization of proactive adaptation process in the PRIDE framework. Because all of those concepts should be evaluated in the context of distributed hierarchical system a more detailed explanation is needed. For this purpose, the next sections introduce the details of the PRIDE system internals. The PRIDE System Architecture contains the architectural design of the framework, while the PRIDE System Behaviour section describes the most important algorithms realized in each of the system layers.

4.2 PRIDE System Architecture

The proposed system architecture results from the concepts presented in the previous Chapter as well as literature studies introduced in Chapter. The notation used to describe all architectural figures in this chapter is as follows:

- Resources which are the subject of adaptation are presented as brown rectangles.
  The name is bold and centred on the bottom of the shape:
The layers of the AS3 Pattern are presented as beige rectangles. The name is bold and centred on the bottom of shape:

- Monitoring

Components of the pride system are presented as light beige rounded rectangles. The name is in italics and centred in the middle of the shape:

- Component

The components which used internal repositories are marked with additional black rectangle in the lower right corner.

- Component with repository

The adaptation loop control flow is represented as thick blue arrow.

- The interactions between different components of the system are represented as thin dashed arrows. The colors denote different context of the interactions. The blue one is related to monitoring of external entities, the red one is management of them, while the black one is internal communication between layer components.

- - - - - - -
- - - - - - -
- - - - - - -

It is assumed that the return messages of interactions between components are not presented to maintain the clarity of the figures.

- The data passed between different components of the system are denoted as standardized document shape.

- Data

The architectural design of PRIDE assumes the application of the Adaptive Controllers concept in the domain of service oriented systems. Figure 4.2 depicts the design of a single Adaptive Controller. Furthermore, it can be structured in accordance with the concepts presented in Figure 3.10. It is assumed that the cooperation model of Adaptive Controllers influences the data flow in the PRIDE framework. The management responsibilities lie on the supervisory Adaptive Controller nodes while the data acquisition and interaction with dependant system responsibilities lie on the servant Adaptive.
Controllers. For example, the created models will be actuated always by the topmost controller in a given branch of the Adaptive Controller hierarchy, while the enforcement of adaptation actions will be always realized by the bottommost ones. Presented architecture is compliant with the structure of the AS3 Element \[126\], which in turn is the realization of the MAPE-K closed feedback loop.

Fig. 4.2: Layered architecture of PRIDE framework

The PRIDE architecture is composed of two main parts: \textit{SLA Online Prediction and Adaptation Part} and \textit{Execution Environment Part}. The former is the core of the PRIDE system where analysis and planning processes are realized. The latter encompasses the monitoring and management mechanisms deployed in the analysed execution environment.

The architecture defines a set of system components which are categorized in five groups on the basis of their functional capabilities. Four of the categories are located in \textit{SLA Online Prediction and Adaptation Part} and one is located in \textit{Execution Environment Part}. The specified categories of the components are hereinafter referred to as layers of the PRIDE system and should not be mistaken with the layers of the dependant system. The functional capabilities of each proposed layers are as follows:

- **Instrumentation Layer.** It is the only PRIDE layer which elements are located inside of the analysed system execution environment. The architecture assumes that each layer of the system, which is the subject of adaptation, is enriched with several components of PRIDE Instrumentation Layer that enable further runtime reconfiguration of supervised system. The components of this layer provide not only the crucial information about the analysed system state but also allow for the enforcement of adaptation actions in order to modify its properties, behaviour or state. In each layer of the analysed system there is one \textit{Monitoring and Management Agent} and a given set of \textit{Sensors} (which retrieve data about specific resources.
located in the analysed layer) and Effectors (which manage resources of analysed layer).

- **Monitoring Layer.** In the context of MAPE-K closed loop pattern, the Monitoring Layer is representing its monitoring part. There are three main components located in this layer: Monitoring Manager, Complex Event Processor Engine and Models Builder. The first one is performing a role of the coordinator in the process of data acquisition from Monitoring Agents located in the execution environment of monitored layer. The second one is an event processing engine (including the event processing statements repository) which correlates events from different data streams in accordance with specified statements in a declarative way and can be changed at runtime. Complex events created on the basis of raw monitoring data represent high-level information that is more useful in the prediction process. The last one is responsible for the creation and actuation of PRM model introduced in Chapter 3.

- **Management Layer.** It is representing the execution part of the MAPE-K closed loop pattern. There are two main components located in this layer: Management Manager and State Transition Enforcer. The former is coordinating the process of adaptation actions execution in instrumented layers of the analysed system. It realizes it by using the appropriate Management Agents and Effectors bound to them in each layer. The latter is evaluating what adaptation actions have to be executed in order to transit current system state into a different one, which usually will be triggered by predicted violation of SLA.

- **Exposition Layer.** This layer is performing the activities related to the analysis part of the MAPE-K pattern. There are three main components located which are used in this process: Trends Estimator, Symptom Matchmaker and Predicted State Solver. The first one is realizing the trends estimation on the basis of data obtained from Monitoring Layer as well as historical (or statistical) information about the analysed system behaviour. By processing and correlating the received data it is able to estimate if current metrics or the system state is changing in accordance with some trend (observed previously during the system runtime). The second one is responsible for the matchmaking of symptoms that occur before the violation of a given SLA and it evaluates whether further system states are going to match some of the previously recognized patterns. The third one is a solver used to realize the scoring process of possible system states in order to avoid the violation of SLA (either predicted or actual one).

- **Policy Engine Layer.** This layer is representing both planning and knowledge part of the MAPE-K loop pattern. There are three main components located in this layer: SLA Realization Solver and Rule Engine and PRM. The solver component is used to apply and transform given SLA policies into the layer specific monitoring and management actions. It realizes it by utilizing the capabilities
of Decision Engine. In case when specific Adaptive Controller is acting as a supervisor then this layer is responsible for the mapping process of high-level SLA definition into layer specific one. The PRM represents the knowledge obtained during the system uptime about relationships between its different elements.

Further sections of this chapter describe in detail the internal design of each presented layer as well as collaboration between the components located in them. Each of the presented layer is shown from the perspective of the single Adaptive Controller. In case when some of the functionality needs to be described in the context of the overall controllers structure, a special remark is noted.

4.2.1 Instrumentation Layer

The bottom-most layer of the PRIDE system architecture is the Instrumentation Layer. It provides the functional means of interacting with resources which are the target of instrumentation. Such capabilities are enabled thanks to extensible architecture of the Instrumentation Layer which is depicted in Figure 4.3 and offers runtime reconfiguration of dependant system.

![Fig. 4.3: The structure of PRIDE Instrumentation Layer](image)

There are two different aspects related to the resource interaction: sensing and effecting. Therefore, it is assumed that the two agent components are present in the architecture of the Instrumentation Layer: monitoring agent and management agent. The former is responsible for handling the mechanism related to the monitoring of given layer resources. In particular it instantiates and (re)configures at runtime the Sensors specific
for a given layer. The latter handles activities which affect the state or configuration of the managed resources. Among others, it enables the execution of managing actions by means of *Effectors* invocations. Both of the agents are located in *Execution Environment Part* of the analysed system which means that the communication between the agents and the components of higher layers of the system is carried out remotely. Furthermore, they act as Façades in order to hide the complexity of the instrumentation layer for the components of higher layers.

In order to cope with the dynamic instrumentation and (re)configuration of sensors and effectors the architecture introduces the mechanism called *Interceptor Socket* which is presented in Figure 4.4.

![Fig. 4.4: The Interceptor Socket instrumentation mechanism](image)

As presented in the figure the instrumented resource is tied to the Interceptor Socket. Such socket is a place-holder where interceptors can be plugged. The socket has two chains where interceptors can be placed. The monitoring chain contains the interceptors providing sensing capabilities. The management chain contains the interceptors enabling affecting capabilities on the resources. The assignment process of interceptors is relying on the *TYPE* property of given interceptors. In addition, each interceptor is exposing its *PRIORITY* property. On the basis of this property the actual place in a given chain is evaluated.

The evaluation of a given chain may be triggered in two ways. First, in case when a given resource is an executable artefact (such as service or component) then the invocation of such resource is a trigger. In case of monitoring chain this will be the most common cause of evaluation. Another method of interceptor socket chain evaluation is manual evaluation. In this case there is a request demanding the evaluation of interceptor chain. In order to provide the flexible mechanism for the invocation of selected subset of interceptor such request may contain a filter that will be processed by the interceptor socket. This type of triggering will be mostly used for evaluation of management chain to invoke a selected collection of effectors.
Chapter 4. Design of Proactive Adaptation Framework for Service Oriented Systems

The actual realization of the Interceptor Socket mechanism will strongly depend on the implementation technologies of given execution environment in which Instrumentation Layer is located. If the environment itself does not provide the necessary means for its realization then the techniques of aspects weaving should be used.

As mentioned earlier the interceptor can be instantiated either as a sensor of effector. The sensors are used for collecting the information that is necessary for the adaptation process realization. The sensors may be used in two different scenarios. In the first scenario the sensor is bounded to a resource in a passive way which means it activates only when a specific situation occurs. It is intended to be used during such activities like service or components invocations. In the second one, it is related to the concept of constant monitoring of a given resource where there is a need for constant updates, such as consumed CPU or available memory. In such an approach the sensor is periodically executing the measurement and publishing the gathered data. In both cases it is crucial to ensure that the actual overhead generated by the monitoring process realized by the sensors is minimized. The effectors are used for managing the resource that is instrumented. They can be also used in two different scenarios. The first scenario is similar the one connected with the sensor. The effector is bounded to a resource and enables management operations of its state or its reconfiguration. With the use of such effectors the dedicated agent can manage the resources present in a given layer. However, in some cases it may be necessary to expose more advanced functionalities such as adding a new instances of service or modifying the composition of components. In this scenario the effector is bounded to the execution environment of a given resource thus exposing the aforementioned capabilities. In general there will be a single effector for a given execution environment which will be activated by default during the initial instrumentation process.

4.2.2 Monitoring Layer

The Monitoring Layer of PRIDE architecture is responsible for handling all the activities related to the acquisition and correlation of data gathered from sensors. The actual configuration of the monitoring mechanisms is done in a declarative way and exposes capabilities of runtime reconfiguration. The overview of internal structure of this layer is depicted in Figure 4.5.

The main purpose of this layers is threefold. First of all, it is responsible for providing the information necessary to actuate the PRM model. Secondly, it provides the mechanisms for data processing and aggregation which are relevant from the perspective of a given SLA. Finally, it is used to trigger notifications further used in the symptom matchmaking process. Thanks to the utilization of generic Complex Event Processing (CEP) approach the process of data acquisition and processing can be abstracted and unified for all of the aforementioned aspects.
Chapter 4. Design of Proactive Adaptation Framework for Service Oriented Systems

Fig. 4.5: The structure of PRIDE Monitoring Layer

The data received from the Instrumentation Layer may vary in the context of structure and contained information. Therefore, the Data Receiver acts as an extensible and configurable communication data sink which can be adapted for a variety of communication protocols. Thanks to that, the data sources from different places may be easily integrated in the system. In addition, as soon as the data is received it is the subject of the normalization process. During this process the Data Normalizer transforms the structure of the received data in accordance with specified transformation definitions. It is crucial to provide such definitions, because the Monitoring Agent located in Instrumentation Layer passes the data obtained from sensors in a raw format. The normalized data are then converted into events and published to CEP Engine related to a given layer for which Adaptive Controller is applied.

The actual configuration of CEP Engine is steered by Monitoring Manager. The manager receives the Layer Specific SLA which is then translated into configuration of monitoring mechanisms. Such configuration includes the actual CEP Statements that have to be registered. These statements are the main elements that measure selected metric in the analysed system as well as produce high-level events for the purpose of trends estimation and symptom matchmaking in Exposition Layer. The overall configuration of the Monitoring Layer could be made in declarative way. Thanks to that, when the new system elements are instantiated in the system they will be automatically encompassed with the monitoring process.

The Monitoring Manager is registered as a listener in a CEP Engine, thus it receives the
notifications generated by previously created statements. In the situation when some of the metrics related to Layer Specific SLA show the evidence that the threshold values are or will be exceeded, the Monitoring Manager publishes the appropriate information to higher layers of the PRIDE system. The structure of CEP Statements represents the hierarchical nature of a given SLA. Such an approach allows for seamless mapping of given SLA into monitoring statements that measure the values contained in the SLA related model.

In case when the Adaptive Controller acts as a supervisory node the purpose of the Monitoring Layer does not change. However, in this situation the SLA received by the Monitoring Manager is related not to the layer specific aspects but rather to high-level SLA. Similarly, the data received by the Data Normalizer component is produced by other Adaptive Components instead of the Monitoring Agents localized in the Instrumentation Layer. In addition, if a given Adaptive Controller is the supervisor (or is deployed in a standalone configuration) then the Model Builder component is triggered which in turn publishes the data about SLA-related metrics relationship for the purpose of PRM model actuation.

4.2.3 Management Layer

Another layer distinguished by the PRIDE system architecture is the Management Layer. Its main task is to manage the execution environments components (especially sensors and effectors) in order to fulfill the specified SLA. The functionality of the layer components depends on the role of the Adaptive Controller and may differ if it acts as a servant or supervisor. If a given Adaptive Controller is acting as a supervisor the Management Layer is enforcing the realization of decisions in the servant controllers. Otherwise, the components enforce the decisions imposed by the supervisory controller. The internal structure of this layer is depicted in Figure 4.6.
The main component located in this layer is the Management Manager. First of all, the manager receives the information obtained during the realization of the monitoring process. Such information may include the data about available sensors and effectors but also about other Adaptive Controllers that are currently deployed in the analysed system. This data is crucial for the SLA enforcement process either if a given controller is a supervisor or not. In the first case, the supervisor is able to properly distribute and manage the layer specific SLA. In the second one, it enables the possibility of triggering the selected set of effectors located in managed resources.

The Management Manager is steered by Adaptation Policy (a subset of Declarative Adaptation Policy which is valid for a given Adaptive Controller) which is created and published by higher layers such as Exposition or Policy Layer. The policy may contain predefined configuration that should be applied by Instrumentation Layer. Such Layer Specific Configuration is processed by the Management Agent and results in the initialization of sensors and effectors necessary for the realization of a given SLA. The results of all the actions taken by the Management Manager are obtained by the Monitoring Layer components and forwarded to the manager for the verification or exception handling process.

Another important aspect that is realized by the Management Layer is the enforcement of the actions that realize some Adaptation Strategy. This process is started if the manager receives the Adaptation Decisions developed by the higher layers. The State Transition Evaluator evaluates the Adaptation Decisions and translates them into the set of actions that has to be enforced in order to modify the analysed layer state. Such
a set of actions is then published as a collection of effectors invocations.

All communication that occurs on the behalf of the Management Layer is realized with the use of Action Publisher component. Its main purpose is to provide an extensible and configurable setup of communication channels. It also realizes the process of format translation to the one that is supported by the receiver. Thanks to that the management business logic is decoupled from the communication mechanisms and thus makes it more applicable for service oriented systems which are heterogeneous and distributed environments.

### 4.2.4 Exposition Layer

The Exposition Layer is the place where the evaluation system state and possible SLA violations are taking place. Components located in this layer are gathering the data obtained by the Monitoring Layer, analysing it and producing the actions necessary to be undertaken by the Management Layer. The process of steering is driven by the highest layer of PRIDE architecture named Policies Layer which supervises the realization of high-level service realization agreement. The structure of the Exposition Layer is presented in Figure 4.7.

![Fig. 4.7: The structure of PRIDE Exposition Layer](image)

The Monitoring Layer publishes the data related to a given SLA to higher layers of the PRIDE system. For the purpose of trends estimation the data can be categorized into two groups: online monitoring data and archival (or historical) monitoring data. The former represents the data acquired by monitoring of the analysed system in accordance with specified SLA. The latter contains information gathered in a specific time period of system uptime. Both of such data sources act as an input to the Trends Estimator component. The estimator is evaluating obtained data in order to identify, estimate and
track the trends which occur in the system. The actual method of trends estimation is implementation specific and will be described in further section of this dissertation. All results produced by Trends Estimator are published to the SLA Evaluator.

In addition to the identified trends, the SLA Evaluator also receives the Observed SLA Violations produced by the Monitoring Layer. Moreover, it also receives the notifications about possible SLA violations that were matched by the Symptom Matchmaker. On the basis of the obtained data the evaluator performs the analysis to check whether the current information shows that the given SLA entry will be violated or not. If so, the evaluator publishes the actions which have to be undertaken in order to avoid such violations. The process of deciding about necessary actions is supported by the transition graph. These decisions are further analysed by the Policies Layer in order to check their compliance with other Adaptive Controllers deployed in the system. The SLA Evaluator also notifies the Predicted State Solver about the current trends and predicted violations.

The Predicted State Solver is the main component responsible for the transition graph creation and maintenance process. In the case when the Adaptive Controller is acting as a supervisor, the created graph is related to all layers which are supervised. In other scenarios the graph is much simpler and contains only states which can be achieved by the modification of resources which are managed by a given Adaptive Controller. When SLA Evaluator identifies the SLA violation it notifies the Predicted State Solver to update the transition graph in accordance with the newest state of the system. The graph is supporting the process of evaluation which decisions have to be taken in order to avoid predicted or compensate observed SLA violation.

Another component located in the Exposition Layer is named SLA Enforcer. It enforces the realization of SLA in a specific layer of analysed system (standalone Adaptive Controller) or in a supervised Adaptive Controllers. The process of enforcement is started when the SLA Enforcer receives the SLA Realization Policy. Such policy is mapped into Layer Specific SLA configurations and Adaptation Actions which both are passed to lower layers. The SLA Enforcer also provides the Management Layer with Management Policies which include system wide configuration of sensors and effectors.

4.2.5 Policies Layer

The Policies Layer is the topmost layer of the PRIDE system architecture. Its main responsibility is to coordinate the realization of high-level SLA in a multi-layered service oriented system including avoidance and resolving of possible conflicts during the realization of many Layer Specific SLAs. The structure of this layer is presented in Figure 4.8.
The data produced by the SLA Evaluator (in form of Proposed SLA Decisions) is passed to the Cross-Layer Policy Checker to verify if it is compliant with the system wide realization of the global SLA. The policy checkers of Adaptive Controllers exchange the information about the adaptation process realized in the specific layers of service oriented system. If the controller is acting as a supervisor, it analyses the related metrics computed by the other Adaptive Controllers and analyses them from a multi-layered perspective. In other scenarios it is just pushing the data about the realization status of layer specific adaptation process to other controllers. All decisions or data worked out by the Cross-Layer Policy Checker are published to SLA Realization Solver for the further analysis. The problem of high-level adaptation policy decomposition is a complex and demanding task. Therefore, a few assumptions have been made in this dissertation. First of all, the decomposition of adaptation policy is done in accordance with the communication scheme of Adaptive Controllers. Secondly, the configuration which supports the automatic decomposition of the high-level policy will be provided by the domain expert in a form of business rules. The PRM model provides the solver with the information about the current influences between different system elements. On its basis it is possible to decide which actions should be undertaken. Typically, the ones which improve the violated SLA metric but does not imply negative effects on other metrics.

The SLA Realization Solver is a component which is mapping the High-level Adaptation Goal into the layer specific configurations of system components and adaptation policies. The SLA Realization Policies consist of metric definitions, event processing queries, sensors and effectors configurations as well as layer specific SLA entries. On the topmost level the solver utilizes the Decision Engine which acts as an expert system for a given adaptation scenario and outputs its best matches of adaptation actions in case of SLA.
violation.

4.3 PRIDE System Functional Description

The previous section described the overall architecture of the PRIDE system as well as the internal structure of all of its layers. This section concentrates on the description of the most important algorithms which are realized by the components located in each of the two aforementioned parts: Execution Environment and SLA Online Prediction and Adaptation. It also describes the architectural decisions that enforce the compliance with the system requirements stated in Section 3.1.

4.3.1 Execution Environment Part

One of the most important aspects of the PRIDE system is the proper discovery of all the Adaptive Controllers deployed in the system as well as all the resources which are bounded with them.

The discovery process is realized in two different stages of the system life-cycle. The first stage is the discovery of the Adaptive Controllers deployed in the analysed system for the purpose of data exchange and supervising activities. In particular, it takes into account the discovery of Execution Environment Part components which are deployed directly in some runtime environment. The second stage is related to the discovery of layer specific resources in managed layers and their enrichment for the purpose of sensors and effectors dynamic instantiation.

The first stage of the discovery process is occurring after the deployment of Adaptive Controllers in the runtime environment of the analysed system. The behaviour of the PRIDE system during that process, described from the perspective of a single Adaptive Controller, is presented in Algorithm 1.

The presented algorithm is realized by each of the Adaptive Controllers separately. It assumes that every controller contains initial configuration which in particular defines its role and the communication protocols that will be used. In addition, each of the controllers stores the references of the other ones categorized in accordance with their roles. The realization of algorithm ensures that the process of discovery will take place and all the Adaptive Controllers will be able to communicate with each other for the purpose of maintaining the coordination activities and exchanging the context information.

The discovery process started by each of the Adaptive Controller is represented by the InitializationOfDiscoveryMechanisms function (introduced in line 1). During its execution it is assumed that each of the Adaptive Controllers is spawning two categories threads (Heartbaet Sender and Heartbeat Listener) responsible for sending and listening
Chapter 4. Design of Proactive Adaptation Framework for Service Oriented Systems

Algorithm 1 Discovery process realized by Adaptive Controller

Assume: AC is one of Adaptive Controllers deployed in the system;
- The controllers are pre-configured to act in a role of SUPERVISOR, SERVANT or STANDALONE node;
- DiscoveryProperties – a configuration holding the properties of communication protocols used between controllers;

Ensure: All Adaptive Controllers deployed in the system are dynamically discovered and capable for communicating with each other.

1: function InitializationOfDiscoveryMechanisms(AC)
2:   Ensure that threads responsible for publishing and obtaining of heartbeat messages are initialized.
3:   for each protocol in DiscoveryProperties do
4:     Spawn thread Heartbeat Sender (protocol)
5:     Spawn thread Heartbeat Listener (protocol)
6:   end for
7: end function
8: thread Heartbeat Sender using protocol
9:   while running do
10:     Send heartbeat message using the selected protocol
11:     Sleep for interval
12:   end while
13: end thread
14: thread Heartbeat Listener using protocol
15:   while running do
16:     hb ← Wait for a heartbeat message using protocol
17:     ProcessHeartbeatMessage(hb)
18:   end while
19: end thread
20: function ProcessHeartbeatMessage(hb)
21:   Ensure that the heartbeat message is properly analysed and its producer stored in AdaptiveControllersMap
22:   metadata ← Extract metadata from hb
23:   role ← metadata[ROLE]
24:   AdaptiveControllersMap[role].add(metadata)
25: end function
for heartbeat messages. As some of the protocols may be synchronous it is assumed that such threads are spawned for each supported protocol separately.

The *Heartbeat Sender* thread is presented in line 8. Each of these threads is bounded to a given communication protocol which is used to publish heartbeat message. Such a message is sent to an address specific for particular protocol and contains meta information about the controller such as direct communication address, managed resources, evaluated metrics etc. After publishing of the heartbeat message the thread is sleeping for a given time period.

The *HeartbeatListener* thread is presented in line 14. It uses protocol specific mechanisms to listen for a heartbeat messages from the other *Adaptive Controllers*. When the message (*hb*) is received, the thread is executing the *ProcessHeartbeatMessage(hb)* function introduced in line 20. The function is accepting the heartbeat message as an argument and its main responsibility is to ensure that the information about the producer of such message (other *Adaptive Controller*) is properly extracted and stored in *AdaptiveControllersMap* structure.

Constant execution of described threads ensures that the discovery process of *Adaptive Controllers* will be realized properly and each of the controllers will have the most up to date information about the available supervised system elements and other *Adaptive Controllers*.

Another important aspect related to the discovery mechanisms of *Adaptive Controllers* is gathering information about available layer specific resources which are managed by the controller. This process is presented in Algorithm 2. It is assumed that in accordance with the layer specific *SLA* the controller is instrumenting the resources with the use of Interceptor Socket pattern introduced in the previous section. The application of that pattern ensures that all the resources specified in a given *SLA* are manageable and it allows for the dynamic plugging of sensors and effectors in form of interceptors.

The initial instrumentation enforced by the *Adaptive Controller* is realized by the execution of *InitializationOfInstrumentationMechanisms* function (presented in line 1). For each layer specific resource that is defined in *InstrumentationConfiguration* (which is created on the basis of given *SLA*) the *Adaptive Controller* is deploying the Interceptor Socket. The deployment of such component is depicted in line 10. It can be either realized with the use of instrumentation API (if supported by analysed layer execution environment) or with the use of aspects weaving. In both cases the resource is enriched with the Interceptor Socket which allows for its dynamic instrumentation. Next, the algorithm assumes the activation of sensors and effectors in accordance with configuration stored in *InstrumentationConfiguration*. The execution of that algorithm ensures that the resources available in a given layer of service oriented system are manageable and can be used in the process of adaptation by components of higher layers.
Algorithm 2 Instrumentation of layer specific resources

**Assume:** Given service oriented system layer contains resources that can be managed in the adaptation process;

*InstrumentationConfiguration* – a configuration specifying which resources are going to be instrumented. It may be created on the basis of layer specific SLA;

*InterceptorSocketsMap* – a map containing all Interceptor Sockets deployed in the system.

**Ensure:** All necessary resources of layer managed by Adaptive Controller are manageable and allow for the dynamic activation of sensors and effectors.

1: function **InitializationOfInstrumentationMechanisms**
2:   for each resource in *InstrumentationConfiguration* do
3:     Execute **DeploymentOfInterceptorSocket**(resource)
4:   end for
5:   for each socket in *InterceptorSocketsMap* do
6:     Activate sensors related to socket managed resource
7:     Activate effectors related to socket managed resource
8:   end for
9: end function

10: function **DeploymentOfInterceptorSocket**(resource)
11:   if *LayerExecutionEnvironment* exposes *InstrumentationAPI* then
12:      socket ← Create the Interceptor Socket with the *InstrumentationAPI*
13:   else
14:      socket ← Create the Interceptor Socket with the aspects weaving
15:   end if
16:      *InterceptorSocketsMap*[socket] ← resource
17: end function
4.3.2 SLA Online Prediction and Adaptation Part

The whole functional behaviour of the AC lies in the *SLA Online Prediction and Adaptation Part*. The core system activities are as follows:

- PRM model actuation;
- SLA metrics trends estimation;
- symptom matchmaking.

All of the aforementioned aspects are described in the following paragraphs of this section.

**PRM Model Actuation**

The PRM model construction is the first of the pointed activities. The process of model actuation is based on the evaluation of low level system metrics which refers to the properties of particular system elements. The process of the PRM actuation may utilize different techniques of computing correlation coefficients (such as product-moment or rank based) which may give completely different results during the prediction process.

Algorithm 3 presents the process of PRM model computation. The activation of PRM model creation is preceded by the specification of PRMConfiguration which contains the configuration of correlation intervals as well as the declaration of techniques that should be used for this purpose. The correlation intervals parameter specifies at which frequency the PRM model values should be actuated. Each of the metric (specified for a given set of system elements) acts as an input for DeploymentOfMetricCorrelator along with the desired intervals and technique used for correlating the data.

The DeploymentOfMetricCorrelator spawns a separate listener which instantiates proper CEP statement. The query retrieves all events related to a given metric and correlates its values with other metrics in accordance with selected technique (which may be either some distance metric or correlation coefficient). The resulting vector of correlation values is stored in the PRM model.

**SLA Metrics Trends Estimation**

The architecture of PRIDE system assumes the usage of trends estimation mechanisms for the purpose of the online prediction. The data produced in that process is further utilized during the proactive adaptation of the analysed service oriented system. Algorithm 4 presents the trends estimation process realized by the Adaptive Controller. It is assumed that the layer specific SLA was received by given controller and resulted in the activation of low-level monitoring mechanisms in the form of sensors. In addition, the metrics
Algorithm 3 Property Relationship Model Computation

**Assume:** Proper monitoring metrics are instantiated and used to process the data received from sensors in a form of streams;

- \textit{PRMConfiguration} – the configuration specifying the actual method of PRM correlation coefficient computation and their parameters.

**Ensure:** The Property Relationship Model is created and expresses the information about relationship between different system elements.

1: \textbf{function} \textsc{InitializePRMEvaluation} \\
2: \hspace{1em} \textbf{for each} metric of \textit{DependantSystemProperty} \textbf{do} \\
3: \hspace{2em} \textit{technique} $\leftarrow$ Obtain correlation technique from \textit{PRMConfiguration} \\
4: \hspace{2em} \textit{intervals} $\leftarrow$ Obtain correlation intervals from \textit{PRMConfiguration} \\
5: \hspace{2em} \textbf{Execute} DeploymentOfMetricCorrelator\textit{(metric,technique intervals)} \\
6: \hspace{1em} \textbf{end for} \\
7: \textbf{end function} \\
8: \textbf{function} \textsc{DeploymentOfMetricCorrelator}(\textit{metric, technique, intervals}) \\
9: \hspace{1em} \textbf{Spawn} Metric Correlation listener \textit{(metric, technique, intervals)} \\
10: \textbf{end function} \\
11: \textbf{thread} Metric Correlator using \textit{metric, technique} and \textit{intervals} \\
12: \hspace{1em} \textbf{Ensures} that the given \textit{metric} is correlated in accordance with the specified \textit{technique} and \textit{intervals} with other metrics in the system. \\
13: \hspace{1em} \textbf{Register} CEP statement in accordance with the specified \textit{technique} \\
14: \hspace{1em} \textbf{while running} \textbf{do} \\
15: \hspace{2em} \textit{correlationVector} $\leftarrow$ \textit{technique.correlate(metric,intervals)} \\
16: \hspace{2em} \textbf{Actuate} the \textit{correlationVector} of a given \textit{metric} in the PRM model \\
17: \hspace{1em} \textbf{end while} \\
18: \textbf{end thread}
Chapter 4. Design of Proactive Adaptation Framework for Service Oriented Systems

related to this SLA are activated and they evaluate the data generated by sensors. The realization of the presented algorithm ensures that the actual trends of the monitoring metrics are produced and available for further utilization by the prediction mechanisms.

Algorithm 4 Trends estimation process

Assume: The layer specific SLA is applied in the analysed system which results in the activation of a given collection of sensors;
Proper monitoring metrics are instantiated and used to process the data received from sensors in a form of streams;
TrendsConfiguration – a configuration specifying the actual techniques of trends estimations and their parameters. It may be created on the basis of layer specific SLA.
Ensure: The information about current trends of monitoring metrics is available for the purpose of online prediction;

1: function \textsc{InitializeTrendsEstimationMechanisms}
2: for each \texttt{metric} in \texttt{SLARelatedMetrics} do
3: \hspace{1em} \texttt{technique} ← Obtain trend estimation technique from \texttt{TrendsConfiguration}
4: \hspace{1em} \texttt{intervals} ← Obtain prediction intervals from \texttt{TrendsConfiguration}
5: \hspace{1em} Execute \texttt{DeploymentOfTrendEvaluator}(\texttt{metric}, \texttt{technique}, \texttt{intervals})
6: end for
7: end function
8: function \texttt{DeploymentOfTrendEvaluator}(\texttt{metric}, \texttt{technique}, \texttt{intervals})
9: Spawn Trend Estimator thread \( \texttt{metric}, \texttt{technique}, \texttt{intervals} \)
10: Spawn SLA Evaluator thread \( \texttt{metric} \)
11: end function
12: thread Trend Estimator using \texttt{metric}, \texttt{technique} and \texttt{intervals}
13: \hspace{1em} Ensures that the given \texttt{metric} trend is evaluated in accordance with specified \texttt{technique} and \texttt{intervals}.
14: \hspace{1em} while running do
15: \hspace{2em} \texttt{estimations} ← \texttt{technique.estimate}(\texttt{metric}, \texttt{intervals})
16: \hspace{2em} Publish new \texttt{estimations}
17: \hspace{1em} end while
18: end thread
19: thread SLA Evaluator using \texttt{metric}
20: \hspace{1em} Evaluates whether the SLA for the \texttt{metric} will be violated or not taking into account its actual trend.
21: \hspace{1em} while running do
22: \hspace{2em} \texttt{estimations} ← Wait for new \texttt{metric} trend notifications
23: \hspace{2em} \texttt{violations} ← \texttt{metric.verify}(\texttt{estimations})
24: \hspace{2em} if \texttt{violations} is not \( \emptyset \) then
25: \hspace{3em} Publish information about SLA violations
26: \hspace{2em} end if
27: \hspace{1em} end while
28: end thread

The activation of the trend estimation mechanisms is done by the execution of \texttt{InitializeTrendsEstimationMechanisms} function (presented in line 1). For each metric specified in SLA the actual technique and intervals of trends estimation are obtained from the
configuration. These parameters are passed to \textit{DeploymentOfTrendEvaluator} function which ensures that the proper mechanisms of trends evaluation are instantiated. Such mechanisms concern the spawning of \textit{Trend Estimator} and \textit{SLA Evaluator} threads (depicted in line 12 and 19).

The \textit{Trend Estimator} thread is parametrized by given metric, trend estimation technique and configuration (which should contain the information about the actual trend estimation intervals). This thread estimates given metric trends in accordance with specified intervals. As a result it receives the estimations of given metric trend. These results are then published in order to be processed by the trend evaluation mechanisms.

The \textit{SLA Evaluator} thread is parametrized by a metric bounded to some \texttt{SLA} specific entries. It evaluates whether the values produced by given metric will be violating given \texttt{SLA} or not. This process takes into account the estimations produced by \textit{Trend Estimator}. When some new estimations of metrics are received it verifies if there are some predicted violations. If such violations exist, the evaluator publishes such information for the purpose of further analysis and adaptation enforcement by the component of higher layers.

The important aspect of \texttt{SLA} evaluation is the verification if given metric will produce violations or not (taking into account current estimations). The discussed algorithm evaluates it in the line 23 by the invocation of \texttt{metric.verify(estimations)} method. The actual realization of this method should involve two different processes of predicting violations. First, it should evaluate the direct violations of given metric without the utilization of context information. In such case, this method will provide a collection of \texttt{SLA} violations that may occur if a given trend remains maintained. However, the proposed solution concerns the multi-layered environments and particular metrics may have relationships with each other. This forces the algorithm also to analyse the context information about the estimations of other metrics which are activated in the system. Therefore, producing more \texttt{SLA} violations that are possible to occur taking into account the actual context.

\textbf{Symptom Matchmaking}

The symptom matchmaking feature of proposed solution is used to predict upcoming failures on the basis of previously observed failures. Algorithm 5 presents the realization of such a process by the Adaptive Controller.

The process of symptom matchmaking assumes the constant evaluation of monitoring data in the terms of matching to the previously identified patterns that preceded a given \texttt{SLA} violation. It is assumed that the two separate threads areSpawned (as presented in line 1) in \texttt{InitializationOfSymptomMatchmakingMechanisms} function: \texttt{SymptomMatchmakerKnowledgeAcquisitor} and \texttt{SymptomMatchmakerEvaluator}.

The former (line 6) is responsible for correlating the monitoring data (in terms of pat-
Algorithm 5 Symptom matchmaking process realized by Adaptive Controller

Assume: PreviousSLAViolations – a collection holding the previously occurred SLA violations.

Ensure: Proper notification is generated if the system state matches the one which was observed in conjunction with the previous SLA violations.

1: function InitializationOfSymptomMatchmakingMechanisms(AC)
2:   Ensure that the threads responsible for knowledge acquisition and matchmaking are initialized.
3:   Spawn thread Symptom Matchmaker Knowledge Acquisitor
4:   Spawn thread Symptom Matchmaker Evaluator
5: end function

6: thread Symptom Matchmaker Knowledge Acquisitor
7:   while running do
8:     violation ← Wait for notification about SLA violation
9:     Store violation in PreviousSLAViolations
10:    Correlate preceding monitoring data with violation
11:   end while
12: end thread

13: thread Symptom Matchmaker Evaluator
14:   while running do
15:     data ← Fetch latest monitoring data
16:     matched ← Match data with PreviousSLAViolations correlated data
17:     if matched then
18:       Publish notification about possible violation
19:     end if
20:   end while
21: end thread
terns) with the SLA violations that can be observed. It is assumed that on this basis a set of event patterns can be automatically created which will detect further upcoming of a SLA violation of a similar type. The latter (line 13) is responsible for matching to current monitoring data with previously created patterns. In case when such a pattern is matched, a proper notification about upcoming SLA violation is triggered.

Such approach enables the Adaptive Controller with predictive capabilities to learn about the system behaviour and adapt accordingly. Generated information may also be further used as an advisory information while performing other online prediction activities.

4.4 Summary

In this chapter the design of Proactive Adaptation Framework was presented. The system architecture is built in accordance with the Adaptive SOA Solution Stack model and is divided into five independent layers. Each of the individual layers plays an important role during the adaptation process and is directly related to a particular element of the MAPE-K loop pattern. The system functional behaviour was presented in the context of dynamic discovery, trends estimation and online prediction of SLA violation. The integration process of presented PRIDE framework for a given application scenario will be shown in the further chapters of this dissertation.

The overall concept is based on the Property Relationship Model utilization in the decision process to improve the quality of the performed adaptation actions. The dynamic evaluation of PRM model is possible through two architectural aspects of proposed architecture. First of all, the runtime instrumentation of the dependant system with low-level elements such as sensors and effectors related to a specified SLA entries metrics. Second, the declarative specification of monitoring queries which enable their dynamic application in case when the execution environment of dependant system changes. Moreover, the knowledge represented through the PRM provides valuable data in the context of Adaptive Controller communication scheme. The next chapter presents the prototype implementation of PRIDE system compliant with the architectural design and algorithms related to the proactive adaptation of multi-layered service oriented systems.
The previous chapter introduced a high-level Platform Independent Model architecture of the Proactive Adaptation Framework that supports the concepts presented in Chapter 3. To prove that the presented concepts are valid an experimental evaluation is necessary. For this purpose a deployable and executable implementation compliant with the introduced architectural decisions is needed. The purpose of this chapter is twofold. First, it presents the transformation process from PIM into PSM model. The process itself is driven by the characteristics of the service oriented systems. Second, the chapter presents the implementation details of the PRIDE framework which are based on top of the selected technological stacks that meet the functional capabilities defined in Section 2.4 as well as system requirements specified in Section 3.1.

The structure of this chapter is as follows. The first section presents the transformation process from PIM to PSM. Each of the selected technologies is analysed in the context of service oriented systems characteristics. Next, a broad description of implementation details is made. The section starts with the presentation of concrete realization of high-level architecture introduced in the previous chapter. Then, more detailed work is presented from the perspective of two parts of the framework: Execution Environment Part and Online Prediction and Adaptation Part. The former includes the following: instrumentation, monitoring, management and communication aspects. The latter concentrates on the elements related to data processing, data correlation and online prediction. Finally, the chapter is summarized.
5.1 Platform Specific Model Selected Technologies

The chapter presented the Platform Independent Model as a solution supporting the proactive adaptation in service oriented environments. In order to propose the concrete technologies which could be used as a basis foundation for the prototype implementation, it is necessary to analyse the characteristics of the service oriented systems in the context of non-functional requirements. The main properties of such systems are the following:

- **Resources heterogeneity** - one of the service orientation principles is the service abstraction. It forces not only the functional abstraction (realized with the use of interfaces or contracts) but also the technology information abstraction, which in real-life scenarios triggers the deployment of such solutions on very heterogeneous physical or virtualized resources.

- **Distributed deployment** - the deployment scheme of service oriented environments production versions typically include multiple nodes on which services are hosted. It is driven by either good practices related to the separation of concern or increasing the overall scalability of the system (e.g services load-balancing, high-computation task scheduling etc.)

- **High dynamism** - in every service oriented environment there is a service registry where the details about currently available services are stored. This allows for the dynamic management of the service life-cycle and triggers the redeployment of different parts of the systems at runtime. The execution environment of such a system often changes therefore the proposed solution should be capable of runtime reconfiguration and dynamic instrumentation.

- **Discoverability** - taking into account the existence of service registry it is crucial for such environments to provide the mechanisms for services (re)discovery process in case the underlying execution environment changes. It should also include the runtime reconfiguration of monitoring mechanisms in case the new resources are discovered in the execution environment so that it matches the previously specified monitoring queries.

- **Large scale** - the real-life deployment scenarios may differ in the context of deployment scale. However, in most cases the technologies used for the implementation of such environments allow for easy reconfiguration and adopting to a newly specified scale of the system.

Although, the aforementioned properties do refer to the service oriented system, the proposed solution for enabling proactive adaptation in such systems should also comply with them. Therefore, during the selection of technologies for the framework implementation (which represents the Platform Specific Model of presented concepts)
the characteristic of the service oriented environments properties was the main driving force. It was also supported by the analysis of currently available framework for the SOA environments creation presented in Section 2.1.3.

<table>
<thead>
<tr>
<th>Resources heterogeneity</th>
<th>Java</th>
<th>OSGi</th>
<th>Spring</th>
<th>Esper</th>
<th>Coherence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed deployment</td>
<td></td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>High dynamism</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discoverability</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Large scale</td>
<td>+</td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Fulfilment of service oriented systems properties by selected technology stacks

Table 5.1 present the fulfilment of aforementioned service oriented systems properties by the selected technology stacks. As can be seen, the Java technology addresses the Resources heterogeneity as it could be executed almost in any operating system currently available on the market. Moreover, its mature architecture ensures that solutions created on top of would scale very well. The OSGi technology ensures the fulfilment of the High dynamism and Discoverability properties, thanks to its internal architecture, often called ”SOA inside JVM”, which enables the dynamic management of services life-cycle as well as the capabilities of service tracking. The Spring technology enables the runtime reconfiguration of the implemented solutions as well as enables their distributed deployment, if supported by a proper communication middleware. The Esper technology ensures the compliance with the High dynamism and Large scale properties. The former is realized with the use of generic, declarative monitoring and correlation queries. The latter is possible through the in-memory stream processing, which is efficient and does not produce any high overhead. Finally, the Coherence fulfils all the aspects related to high-scale distributed communication and deployment. In pair with the OSGi technology it may be used to implement the mechanisms compliant with the aspects of distributed OSGi.

The short description of each of the selected technology stacks is as follows:

- Java technology \[117\] is the most commonly used technology in the IT companies especially in the area of enterprise-class systems. It is utilized to create the products supporting multiple areas of the IT domain. It includes such solutions as: applications servers (GlassFish, JBoss) or servlet containers (Tomcat, TomEE); messaging and distributed communication (ActiveMQ, HornetQ) and many others libraries (mostly open source) which affect the development of new IT technologies and challenges. Moreover, it offers the hardware abstraction layer,
Chapter 5. PRIDE Implementation

named as the Virtual Machine, that executes Java bytecode. Because of that, the Java applications can be run in different operating systems without any changes (in most cases).

- **OSGi** is the Java-centric component framework which could be leveraged to build the advanced service oriented systems that need features like the dynamic deployment, registration and discovery of services. Thanks to that capabilities it is also often adopted as the container which hosts low level system services i.e. in GlassFish or JBoss application servers. Moreover, it is the only solution (apart from those analysed in Section 2) that is standardised in form of a specification. There are a few implementations of the OSGi specification but the most mature ones are the Apache Felix 114 and the Eclipse Equinox 116. While both of them provide similar functionalities, the Apache Felix is more widely used which is why it was selected. Additionally, the OSGi-related stack was extended by selecting the Apache Karaf 115 which offers more advanced features for the managed OSGi container (including provisioning and hot deployment).

- Spring Framework 113 was selected as a core technology for the developing management part of the PRIDE framework. Thanks to its Inversion of Control (IoC) feature realized by the means of the Dependency Injection (DI) it makes the process of the software development simpler and more effective. In addition, the possibility to use the declarative configuration (instead of the programmatic approach) allows for much more efficient testing process during the implementation of new parts of the code. It also offers the modules that simplify the process of integration testing which is very desirable during the development of distributed frameworks.

- Esper technology 112 is one of the most mature and highly scalable implementations of the CEP engines. It is an open source library and it has very good documentation which includes a lot of working examples. In addition, it can easily be wrapped as an OSGi bundle (in contrary to other solutions such as Drools Fusion 111). Its advanced features allow for the statistical analysis and correlation of events as well as the dynamic (un)registrations of new event processing queries and listeners.

- Oracle Coherence 110 is the enterprise-level implementation of distributed cache. It is successfully deployed in hundreds of production environments as well as used as the provider of application servers clustering mechanisms etc. Its extensive API interfaces and management capabilities enable the effective implementation of distributed middleware which can be scaled vertically over hundreds of nodes. Moreover, its binary representation in form of a single jar file allows for an easy integration with the OSGi technology.
During further analysis of the technological requirement, capabilities and features it was decided that the following technology stack will be used in the implementation of the PRIDE framework: Java programming language, Apache Felix, Apache Karaf, Spring Framework, EsperTech Esper and Oracle Coherence. Selected technologies represent the top ones from currently used in the IT industries. Such a combination ensures that the implementation of the PRIDE framework itself will be carried out at the very high technological level. It will allow the PRIDE framework to be successfully evaluated and deployed in the environment of the enterprise class.

5.1.1 PRIDE Integration Process

This section presents the steps which have to be executed in order to integrate the PRIDE prototype framework with a given service oriented system. The presented description acts as a concretization of the generic proactive adaptation process presented in Figure 3.4.

In order to employ the features offered by the PRIDE framework an existing service-oriented application has to be used. Moreover, a Service Level Agreement should be defined for such an application, preferably in the form of QoS metric constraints which refer to the specific components, services or other software resources. If these predicates are defined it is possible to perform the integration process which will enrich a given application with the proactive adaptation capabilities exposed by the framework. The process is expressed in a few steps which are as follows:

1. PRM Identification and Adaptive Controllers Setup - in this step the identification of the PRM model is done, which provides useful information related to the communication scheme of Adaptive Controllers. This process is supported by the actual definition of a given system SLA. The PRIDE framework assumes the concrete structure of the SLA in which the QoS metric constraints are defined, which is compliant with high-level concept presented in Figure 3.3.

2. Adaptation Goal Planning - the purpose of this step is to define of a high-level adaptation goal determining which elements of the dependant service oriented system should be the target of the adaptation actions and in what way they can be adapted. The goal is then mapped into the configuration of the Adaptive Controllers, the definition of the layer specific SLA and data processing queries.

3. Sensors and Effectors Implementation - in this step the concrete implementation of sensors and effectors has to be provided. Both, sensors and effectors, refer to the software elements of the system which is the target of adaptation. The former are

---

5 The scheme of the SLA used in the framework can be found in the Appendix A.
6 The structure of the adaptation goal used in the framework is built on top of rule-based approach and can be found in the Appendix A.
used to obtain the data which is beneficial to the QoS metrics measurement. The latter is necessary to provide mechanisms which allow for actuating the dependant system, therefore the implementation of these elements has to be compliant with the technologies used in the dependant system. Typically they will be started in the same execution environment as the software components of the supervised systems.

4. Initial Provisioning - the purpose of this step is to install the agnostic monitoring and management mechanisms which cut across all of the layers of the dependant system. These mechanisms enable the discovery of resources belonging to the elements of the specific layers and the monitoring of the selected QoS parameters. This step initializes the mechanisms for which instantiates and/or deploys the previously implemented sensors and effectors.

When all of the aforementioned steps are executed then the PRIDE framework is fully operational and can be used to enrich the supervised system with the proactive adaptation capabilities. The adaptation will be made in accordance with the specified high-level adaptation goal and defined SLA.

The next section describes the implementation aspects related to the development of the Platform Specific Models on the basis of the Platform Independent Model presented in Section 4. The particular subsections describe in technical details how a given framework capability or feature is realized.

### 5.2 Implementations Details

This section describes the PRIDE framework implementation details from the perspective of the two parts of the system: Execution Environment Part and SLA Online Prediction and Adaptation Part. Unlike the content of Section 4 (which described the high-level architectural technology agnostic concepts), the description in this section contains concrete code listings realized with the use of the aforementioned technology stack.

Figure 5.1 represents the concrete components categorization of the PRIDE framework architecture presented in Figure 4.2. As can be seen, some elements of the system are deployed in the same execution environment as the instances of the services and components of the dependant system.
Typically the dependant system will be deployed in the distributed scheme. As the PRIDE framework should work seamlessly in such networked environment it is crucial to provide the means for efficient and scalable communication. To enable such capabilities of the framework, the implementation imposes the utilization of communication middleware which is based on the distributed cache technologies. For this reason, on the depicted Figure, the two types of components (agnostic from the perspective of functional categorization) are presented:

- **Cache Server** - 3rd party component providing the capabilities of distributed cache. The current implementation of PRIDE framework uses the Oracle implementation of cache named Coherence.

- **Cache Connector** - the component that hides the implementation details of the selected cache realization and provides the domain oriented API.

The first group (located in the Execution Environment Part) represents the components executed in the supervised system execution environment. Among others, it includes the following:

- **Management API** - services used to communicate with VMware hypervisor;

- **Monitoring Scripts** - scripts used to efficient measurement of low-level system metrics related to the CPU and memory utilization;

- **Monitoring and Management Agents** - components used to monitor and manage the application level resources.

The second group of components (located in the SLA Online Prediction and Adaptation Components) represents the components which form the PRIDE ACs. Among others, it includes the following:
Chapter 5. PRIDE Implementation

- **PCC Relationship Matrix** - the representation of the PRM model computed in accordance with Pearson’s correlation coefficient algorithm;

- **Esper Engine** - 3rd party CEP compliant engine;

- **Linear Regression Trends** - the components used to perform the estimations of the QoS metrics and SLA violations in accordance with the linear regression approach;

- **Symptom-based Online Prediction** - the components used to predict the future state of the supervised system utilizing the symptom-matchmaking approach;

- **Adaptation Enforcement** - the components enabling the enforcement of adaptation actions in the supervised system.

Fig. 5.2: Separation of communication channels between PRIDE and dependant SOA system

Figure 5.2 depicts in what way the separation of communication between the framework and dependant service-oriented system is done. As can be seen, there are two distinct communication channels groups: application specific and PRIDE specific. The former ones are the legacy communication channels used in the dependant system to communicate between the application services deployed in the distributed locations. The architecture as well as the implementation of the PRIDE framework do not enforce any modification of the previously used protocols etc. and they are completely transparent for the dependant system. However, the prototype implementation of the PRIDE framework does specify two different categories of communication channels realized on top of the selected technology stack: instrumentation-based management and monitoring.
channels and Adaptive Controllers communication channels. Both of them are realized on top of the selected technology stack which is the Oracle Coherence distributed cache. The former category is used to fetch the monitoring data from the given resources (managed virtualized machines, application containers or software components) and enforce the adaptation actions on them. The latter category is used to exchange the information between a given hierarchy of Adaptive Controllers while enforcing the Declarative Adaptation Strategy in the supervised system.

Such an approach allows for the isolation of Adaptive Controllers in accordance with the specified communication scheme as well as limits the management boundaries of a given Adaptive Controller only to the nodes which should be supervised by it. Moreover, the overall logic and communication between the controllers is realized in parallel to the application level actions. The adopted approach, allows the PRIDE framework to be easily accommodated in a variety of environments only by implementing a specific subset of its components (the ones located in the execution environment of a given service oriented application). The following sections introduce the implementation details of the most important parts of the PRIDE framework.

### 5.2.1 Execution Environment Part

As described in the Section 4, the Execution Environment Part is located in the runtime environment of supervised system. The description in this section is divided into three categories: instrumentation, monitoring and management and communication. Because of the fact that the OSGi was selected as a base execution container for the PRIDE prototype implementation most of the following subsections present the OSGi specific realization of the Execution Environment Part components. Moreover, the prototype implementation focuses only on a few layers of the S3 stack, namely: operational systems, components and services. However, the implementation is realized in a very generic way, which enables flexible extension if needed.

#### Instrumentation

In order to dynamically activate the monitoring strategies or management actions an Aspect Oriented Programming (AOP) instrumentation is needed for a selected OSGi container. It is implemented with the use of the AspectJ library and the code of aspect that is woven into the Fuse OSGi container which is presented in Listing A.1.

The aspect pointcuts the invocation of the BundleContext.getService method. In the OSGi this method is invoked by the clients who want to use the service registered in the registry. In order to allow the calling bundles to load the code of interceptor socket during invocation interception it is necessary to provide a separate class loader - MonitoringProxyClassLoader (lines 11-14). The socket is created in accordance with the
well-known proxy pattern with the use of Java Proxy API [107]. The implementation of the OSGi Interceptor Socket can be found in Listing A.2.

In such a configuration each service in the supervised environment has at least one interceptor socket created during its first invocation. The interceptor socket evaluates the registered interceptors in three steps: i) before service invocation - `invokeStarted` method; ii) after service invocation - `invokeFinished` method; iii) in case of processing error - `invokeError` method.

**Monitoring and Management**

The mechanisms of interceptor socket (as well as interceptors themselves) can be successfully used to monitor metrics related to the software components deployed in a given container such as *invocation count* or *response time*. However, in order to measure the metrics related to the hardware or operating system aspects it is crucial to use the mechanisms that do not generate too much overhead. It was decided that the *sysstat* package will be used at the operating system level to realize those tasks. The *sysstat* package is a set of system performance tools for Linux which contains, among others, the *sar* tool. The *sar* is a system monitor tool that reports various system loads, including CPU activity, memory, device load, network etc. To integrate the results generated by the *sar* a dedicated class responsible for initiating and destroying the measurement process is provided.

Listing A.3 depicts the internals of the *GenericMeasurement* class. In order to make sure that the interaction with the operating system shell processes is realized properly the implementation utilized two Apache Commons libraries: IO [106] and Exec [105]. The former is responsible for handling the input and output streams that read/write from the file system. The latter is responsible for spawning the OS-level processes and their proper termination.

There are two other elements that are deployed in the execution environment of the supervised system: the management of the virtualized resources and the management of the software artifacts. Having the access to the infrastructure managed by the VMware software it was possible to implement the Virtual Machine (VM) management mechanisms. The interaction with the vSphere API is realized with the use of the VI Java [104] open source library. Listing A.5 presents the implementation of the *VMManager* class which performs the communication with the vSphere management interfaces. With the use of the *VMManager* it is possible to obtain a reference to a given virtual machine and dynamically reconfigure its parameters such as available CPUs or RAM amount.

The management of the software artefacts is realized utilizing pure OSGi capabilities. The functionality is implemented in the *ManagementAgent* class. The *ManagementAgent* receives the configuration of the new services and recursively iterates through it looking for service definitions. For each definition that is found, the manager registers...
Chapter 5. PRIDE Implementation

a new OSGi service.

All system elements described in this section are packaged in a single OSGi bundle \( (osgi-monitoring-management-agents) \) with the use of the \( maven-bundle-plugin \) provided by Apache Felix. Their distribution to the supervised containers is realized with the use of Apache Karaf provisioning capabilities. Thanks to that, all the bundles defined in the \( features \) XML file are automatically downloaded from the maven repository and activated when the container is starting.

Communication

The communication between the Monitoring Agent, Management Agent and their coordinators deployed in the Online Prediction and and Adaptation Part of the PRIDE framework (Monitoring Manager and Management Manager) is realized with the use of the Oracle Coherence distributed cache. The choice of the distributed cache as a communication middleware basis was dictated by a wide range of functional capabilities offered by the specific implementation. It enables the seamless setup of the logical communication channels over the physical networks which is highly-scalable and provides built-in discovery mechanisms. Moreover, new nodes of cache servers can be dynamically plugged into the distributed cache at runtime and will be automatically synced with the state of the whole federation.

In order to provide the connectivity to the distributed cache a connector implementation was needed. The connector class used by both parts of the PRIDE system is also packaged as the OSGi bundle and deployed into the execution environment. Listing A.6 presents the core methods of the \( CoherenceCacheConnector \) class. During its startup the connector is connecting to the configured Coherence Servers cluster. The deployment assumes that the cluster servers are started in some external localization and the connector itself is configured not to store the data locally. When the connector joins the cluster it initializes its local \( dispatcherMapListener \). It is done in order to minimize the computational overhead when transferring the data through the cache. Such an approach ensures that there is a single listener per the OSGi container which allows to register the additional second level ActionListeners. Those listeners are notified when changes occur in a given cache. The clients of the connector can by default use two predefined caches: ManagementCache and MonitoringCache. They can also use the connector to put the value into a given cache (as shown in line 37) and it will propagate given value to other members of the cluster.

To minimize the amount of the data which traverses through the cluster a dedicated addressing protocol is implemented. All objects stored in the caches have the same base abstract class - \( BaseAction \). This class contains a property \( addressedNodes \) which represents a set of nodes that are capable of reading the data. During the registration of the aforementioned \( dispatcherMapListener \) a filter containing the name of the current
container is passed. It is filtering the data on the Cache Servers level which results in pushing the data from the cache to a given \textit{dispatcherMapListener} only if its container name matches one of those specified in the \textit{BaseAction addressedNodes} property.

### 5.2.2 Online Prediction and Adaptation Part

As described in Section 4, the \textit{SLA Online Prediction and Adaptation Part} can be located outside the supervised system runtime environment. The content of this section is divided into two categories: \textit{data processing} and \textit{correlation and prediction}.

The communication mechanisms used to put and receive the data from the distributed caches layer are the same as in the \textit{Execution Environment Part}. There is a single difference in form of the connector exposition in the internals of the system. It is not registered as the OSGi service, but wrapped in a Spring component and exposed in accordance with the Business Delegate pattern \cite{103}. In fact, all the system components that are deployed in the analytical part of the PRIDE framework are deployed into the Spring IoC container. The PRIDE implementation heavily utilizes the annotation-based approach to declare which classes will be exposed as Spring components.

#### Data processing

The data processing of monitoring events is realized in accordance with the \textit{CEP} approach which is supported by the Esper \textit{CEP} engine. The PRIDE framework allows for easy and quick reconfiguration of its data processing mechanisms at runtime. It is possible because the Esper engine is instantiated by the \textit{MonitoringManager} and dynamically supplied with the statements that are matched against the published events. The system is implemented in such a way that each of the statements has an associated domain object that extends the abstract \textit{CEPStatement} class. The \textit{CEPStatement} adds the associated statement to the Esper engine and registers itself as an \textit{UpdateListener} for further processing of its results. The PRIDE framework by default specifies three categories of generic events processing statements which are as follows:

- average metric value measurement;
- linear regression computation;
- trends estimation.

Listing A.8 presents the CEP statement used to compute the average metrics value in accordance with the specified parameters. The statement selects four values: average value of a given metric, metric name, timestamp of the last processed event and the period for which the average value is computed.
Chapter 5. PRIDE Implementation

The *GenericMonitoringEvent* is a generic event that contains two metric related fields: name and value. The values reported by the statement are obtained using the built-in view of the Esper engine *stat:uni* (univariate statistics) on the stream that contains this event. The view is processed in a sliding time window with the width equal to `:period` parameter (in seconds). The last part of the statement represents the output rules of data computed by it. It says that the *UpdateListeners* associated with this statement should be notified with the most up to date information every `:output` seconds. By default the discussed statement processes all metrics that are published into the Esper engine in the form of *GenericMonitoringEvents*.

Listing A.7 presents the CEP statement used to compute the metrics linear regression values. The statement among others computes the steepness of a given metric as well as its y-intercept place. Both of them are computed on the basis of the average values stored in the *AverageMetricStream* which are produced by the statement presented in Listing A.8. The computation is using the built-in view of Esper engine *stat:linest*. Similarly to the previous statement, all *UpdateListeners* are notified every `:output` seconds. The results of the linear regression are placed in a new event stream called *RegressionValuesStream*. A single event in this stream is related to the single regression values for a given `:period`. By default the framework computes the regression for three values of this parameter: 30, 60 and 120 seconds. By these means it is possible to evaluate three different trend categories for a given metric which are further utilized in the prediction process.

Listing A.9 presents the CEP statements used to evaluate whether a trend can be observed in the system for a given metric. The system is configured to inform about the ongoing trend for a given metric (either an uptrend or a downtrend) when three events with the same steepness appear in the *RegressionValuesStream* one after another. In case when such a sequence is disturbed by an event with different slope value the pattern is broken and the matching process is starting once again.

The introduced statements allow for efficient processing of the data which is gathered by the *Monitoring Agents* deployed in the runtime environment of the supervised system. In addition, the utilized technologies ensure the high scalability as well as the flexible reconfiguration of the implemented mechanisms even at runtime. Moreover, the generic implementation enables for the automatic creation of low level metrics when new system elements are added to the system.

**Correlation and prediction**

The data processed by the Esper CEP engine is analysed further by the additional components of the PRIDE framework. One of the concepts described in Section 3 is the **PRM** model which describes the relationships between different system elements, usually deployed and executed in different layers of the runtime environment. This
model is based on the correlation coefficient values computed for metrics measured in the system. The implementation of the coefficient computation is realized using the Apache Commons Math library. Listing A.10 presents the core method used during the recomputation of the correlation between metrics.

As shown, the correlation is computed for the last $numberOfEntries$ of measured metric values. The results of the correlation are stored in a hashmap ($correlationValues$) with a key representing the two metrics that are compared. The $recomputeCorrelation$ function is periodically invoked by the $AdaptationManager$ to obtain the most up to date results.

The detection process of SLA violations is realized from two perspectives. The first one is related to the reactive approach and detects the violations that already occurred. Its implementation also utilizes the CEP Esper engine. In accordance with specified SLA definition the system dynamically registers the statements presented in Listing A.11 previously replacing the $:value$ and $:metric$ parameters.

The statements process the AverageMetricStream and trigger the UpdateListeners when the average values are bigger (upper bound of the SLA) or lower (lower bound of the SLA) than the values defined in the SLA. Listing A.12 depicts the part of the code which processes the notifications sent by the Esper engine. The SLAReactiveListener extracts the detailed values from the received event and publishes the notification about the occurrence of the SLAViolation. Because the SLA is expressed in a declarative way, it should automatically be matched again by any new system element that is added to the environment.

The second way of detecting the SLA violations is realized from a proactive perspective. This approach is based on the results produced by the statements presented in Listing A.9. Listing A.13 presents the part of SLAProactiveListener which is handling the registration of PRIDEInternalListener in the aforementioned statements. The listener is created for each SLAEntry and the prediction window length is configured in high-level AdaptationStrategy. As a result, each PRIDEInternalListener is processing the values of a single SLA related metric with a given prediction window. Similarly to its reactive counterpart, the proactive listener extracts the values from the TrendOccurrenceEvent and evaluates whether there will be the SLAViolation or not. If the predicted value is violating the threshold defined in the SLA then the possible SLAViolation is published further.

In case of predicted SLA violations there exists an optimization in the PRIDE framework which significantly improves the quality of the prediction mechanisms. It is assumed that in order to handle such a predicted violation, the notifications from two different regression periods (c.f. Listing A.7) must occur. It allows to avoid the situation when single peak of a given metric value triggers the adaptation of the whole system. To implement such mechanisms a timed eviction cache which stores the predicted violations is used. Its implementation is provided by the Google Guava library which is a set of extensions and utilities to the standard Java SDK. By default the PRIDE framework
uses the 30 seconds eviction strategy for the predicted SLA violations which proved to be sufficient.

The realization of the symptom matchmaking concept is implemented in the *SymptomMatchmaker* class. There are two main methods which are constantly evaluated during the system uptime, as shown in Listing A.15. First, when the SLAViolation occurs the matchmaker actuates the symptom patterns repository. It is done through the conversion of correlated metric values into a set of *SymptomPatterns*. The set contains the influence values of specific metrics extracted from the PRM model at the time of SLAViolation. Next, all of the matched patterns are constantly evaluated when the correlation values between the metrics change. Thanks to that it is possible to predict whether the current values of the PRM model indicate that there will be the SLAViolation. When such situation occurs, the SymptomMatchmaker notifies the AdaptationManager which in turn triggers the enforcement of the adaptation actions.

The final part of the system is to prevent the SLA violations that occurred or were predicted. The AdaptationManager can be configured to work in one of the four modes: LayeredReactive, LayeredProactive, GlobalReactive and GlobalProactive. Depending on the selected mode the manager is analysing the dependences either between the elements of different layers (Global) or only between the elements of the layer which it is supervising (Layered). Listing A.16 presents the code that is executed when the AdaptationManager handles the SLA violation.

In case the current SLA violation is not related to the layer supervised by the non-global Adaptation Manager nothing is done. Additionally, the manager checks whether it is valid to perform some AdaptationActions for a given violation or not. The PRIDE framework contains a timeout policy which restricts enforcing adaptation actions too many times at a narrow time interval.

If the preliminary checks pass then a correlation threshold is obtained from a specified AdaptationStrategy. Next, the LongTermCorrelationStatistics map is fetched from the CorrelationManager using obtained treshold. Depending on the mode in which the manager is working it either evaluates the possible AdaptationActions in a layered or global manner. All of the prepared actions (which may be related to the modification of services deployment or virtualized resources parameters) are passed to the TransitionEnforcer. It publishes selected actions back to the Execution Environment Part of the PRIDE framework where they are handled by the proper Management Agents.

### 5.3 Summary

This chapter presented the implementation of the Proactive Adaptation Framework architecture (c.f. Chapter 3). The utilized technology stack selected for the purpose of transforming the PIM into the PSM model was based on the characteristics of the service
oriented systems. Thanks to the maturity, commodity as well as enterprise grade of the Java technology it was decided to adopt it as the core of the system implementation. The chapter also included the integration steps that have to be realized in order to enrich the existing service oriented solutions with the capabilities of proactive adaptation enabled by the implemented prototype. The concrete realization of architectural patterns presented in the previous Chapter ensures that the proposed solution could be deployed in parallel to the supervised system in a completely transparent way. Moreover, the selected technological stack ensures that the system can be easily scaled horizontally while ensuring low computation overhead on the deployed hosts.

The introduced implementation exposes several properties which are directly related to the common characteristics of the service oriented environment presented in Section 5.1. The most important properties of the described implementation are as follows:

- the runtime enrichment of supervised environments through low-level system instrumentation;
- the dynamic discovery of dependant systems resources and their automatic inclusion in adaptation process through the declarative specification of adaptation policy;
- the generic data processing and correlation mechanisms that can be reconfigured at runtime.

On the other hand, there are a few drawbacks of the proposed solution. First, it requires manual decomposition of the SLA definition by the system operator in order to map its high-level representation into the layer-specific one. Second, it does not allow for the dynamic adaptation of the prediction window length and thus it cannot be used in cases when the dynamism of the supervised system is changing.

Further, the utilization of the prototype framework in the service oriented system will provide the means for the functional and non-functional evaluation of proposed concepts. For this purpose, it was decided that a dedicated private cloud environment would be set up and the exemplary SOA application would be enriched with the capabilities of the PRIDE framework. The detailed description of the conducted evaluation and its results are presented in the next chapter.
Chapter 6

**System Evaluation and Experimental Results**

In order to verify the correctness of concepts and design aspects introduced in this dissertation it is necessary to propose the evaluation experiments which confront the formulated thesis statement and the high-level design of the proactive adaptation in the multi-layered service oriented systems. The proposed experiments should cover the most important elements of both functional and non-functional aspects. In the context of functional evaluation it is necessary to verify the following aspects. First, whether the proposed concept of the PRM model offers the necessary information for the purpose of the adaptation enforcement. Second, the analysis of the system behaviour in different configurations of adaptive controllers (layered and global manner); the assessment of the capabilities of the online prediction mechanisms and the examination of the impact of the relationship model on the adaptation process. From the perspective of non-functional evaluation the experiments should try to answer the following questions: whether the overhead generated by the proactive adaptation enforcement is acceptable and to which extent the proposed solution could be applied. The evaluation is done through the application of the PRIDE framework in a set of five experiments in which real-life SOA compliant application is supervised in accordance with the specified adaptation strategy.

The structure of this chapter is as follows. The first section presents the overview of the methodology used in all the performed experiments as well as their brief description. In this section an overview of the SOA related application scenario utilized in the experiments is presented. The section is finalized by the detailed description of experimental testbed (including hardware and software aspects). The second section describes both the functional and non-functional parts of the evaluation. The former covers the following experiments: the AC configurations comparison in a layered and global manner; the online prediction aspects in the adaptation enforcement and utilization of the relationship model in the adaptation process. The latter covers the assessment of the resources...
overhead introduced by the deployment and execution of the adaptation mechanisms as well as the verification of the PRIDE framework scalability capabilities.

6.1 Evaluation Methodology

The goal of the presented evaluation is to verify if the concept of multi-layered service oriented systems proactive adaptation supports the formulated thesis statement and can be successfully applied in a real-life application scenario. The evaluation methodology is divided into a few separate experiments (each referring to one of the aforementioned requirements) utilizing one joint case study which exposes different parts of the system capabilities.

The following experiments have been proposed to perform the functional evaluation of the proposed concepts.

Experiment 1 - Utilization of Property Relationship Model in Adaptation Process.
The experiment focuses on the PRM model verification in the context of its correctness and usefulness. First of all, it is evaluated whether the proposed approach for the PRM construction and actuation (c.f. Section 3.3.1) are valid and offer a sufficient information while enforcing the adaptation actions. Next, the experiment verifies the proper classification of relationship between a given QoS-related metrics which could be utilized for a further set up of Adaptive Controllers.

Experiment 2 - Influence of the Adaptive Controllers Communication Scheme on the Adaptation Enforcement Results.
The experiment attempts to compare the results of the adaptation conducted with different Adaptive Controller communication schemes. The test scenarios include four different configurations of the AC which reflect all of the possible communication schemes and elaborate on their influence on the overall adaptation process.

Experiment 3 - Online Prediction Aspects in Adaptation Enforcement.
This experiment tackles the aspects related to the realization of the proactive adaptation and its possible gains compared with the reactive approach. The executed test scenarios elaborate on the behaviour of the adaptation mechanisms in the context of different values of the prediction parameters. Moreover, the experiment also presents how the dynamics of the changes occurring in the system affects the decision quality of the proactive adaptation enforcement.

Experiment 4 - Behaviour of the system under varying SLA.
The experiment focuses on the verification of the system functional capabilities while the agreed SLA contract is varying at runtime. The executed test scenarios
elaborate on the capabilities of the adaptation mechanisms to adapt the supervised environment to match a new definition of the service contract. It also illustrates the runtime reconfiguration capabilities of the proposed solution.

**Experiment 5 - System Overhead and Scalability Aspects**

This experiment evaluates the overhead (computational resources and network bandwidth) caused by the deployment and usage of the proactive adaptation mechanisms introduced by the proposed solution. The assessment is done from the perspective of the increasing number of resources (virtual machines and services) and different scopes of the SLA applied in the system.

The combination of these five experiments allows for the presentation of all the concepts and methods introduced in the dissertation as well as for covering the formulated statement.

**Application Scenario**

The application used in the evaluation scenarios was composed as an OSGi-based SOA application. It is named PBT and was modelled on the basis of well-known example of Web Service Choreography Interface (WSCI). The goal of the application was to offer the functionalities of planning, booking, and reserving airline trips for the end users from the perspective of travelling agent. The high-level view of the adopted business process is presented in Figure 6.1 and represents the business logic flow of the PBT application.

As can be seen, the application consists of six services which could be deployed in a cloud environment. The functionality of each service as well as its nominal response

---

http://www.w3.org/TR/wsci/ accessed on April 2014.
time is presented in Table 6.1. The nominal values of service response time specify the typical amount of time spent while processing the single request. In case when some disturbances are observed in the system (excessive load etc.), these values are higher than normal.

<table>
<thead>
<tr>
<th>Service name</th>
<th>Description</th>
<th>Nominal response time (values assumed for initial experiment)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend</td>
<td>it is a service that is responsible for handling all the requests that are sent through the user interface. It includes the transformation of the data in accordance with the desired format such as XML, JSON or HTML.</td>
<td>80 ms</td>
</tr>
<tr>
<td>Planning</td>
<td>a service that performs operations related to planning trips. Depending on a given trip parameters it returns the offer a several sets of flight legs that can be used.</td>
<td>245 ms</td>
</tr>
<tr>
<td>Booking</td>
<td>a service that realizes operations of reserving the airline tickets in the external subsystem of fly operator. The reservations are stored internally in the PBT and after the payment is done, they are transformed into the airline tickets of a given operator.</td>
<td>235 ms</td>
</tr>
<tr>
<td>Seats</td>
<td>the service responsible for the communication aspects with external fly operators in the area of seats availability. Obtained values are stored in the distributed cache for the purpose of further utilization in the application.</td>
<td>110 ms</td>
</tr>
<tr>
<td>Persistence</td>
<td>it is a service responsible for the realization of all the operations related to persistence. It includes not only saving the data into the database but also updating the values stored in a distributed cache spanned across all the infrastructure.</td>
<td>25 ms</td>
</tr>
<tr>
<td>Notification</td>
<td>this service is used to register for the notification about the payment and/or registration deadlines after the tickets are booked. It also provides the capabilities of notifying about the upcoming trip using the electronic mail, short text messages or automated phone calls.</td>
<td>20 ms</td>
</tr>
</tbody>
</table>

Table 6.1: Application scenario services description

In order to simulate the load in the PBT application a separate module was implemented and called the Custom Load Generator Module (CLGM). The configuration includes among others the number (and probabilities) of service Request per Minute (RPM) as well as the number of sending threads. The PBT application flow as well as the invocations made by the workload module were created in accordance with the itinerary-based routing concept [119] widely used in the enterprise-class applications. The hardware
Chapter 6. System Evaluation and Experimental Results

specification of the environment in which the PBT application was deployed as well as the process of its deployment is described in the next section.

Hardware and Software Configuration

The environment set up for the purpose of the aforementioned experiments was built with the use of the enterprise quality software and hardware.

The hardware layer consisted of Cisco UCS C200 M2 High-Density Rack Server\(^8\) which is a part of Cisco Unified Computing System (UCS) \(^9\). It is mostly used as a platform for technologically advanced production-level data centers. The hardware characteristics of the UCS was the following: CPU Intel Xeon E5639 2.53 GHz with 2 sockets (6 processor cores per socket) and 24 logical processors; 49 GB of RAM memory; TOSHIBA Serial Attached SCSI Disk 560 GB.

The software layer foundation was the VMware vSphere Hypervisor 24 5.1 Update 1 (ESXi)\(^9\) which is a bare-metal hypervisor capable of hosting virtualized environment including not only the virtual machines with various operation systems but also virtual networking capabilities. The ESXi was installed in Cisco UCS as its primary operating system.

All the VMs used during the evaluation scenarios were provided with the 64-bit Linux operating system Ubuntu Desktop 12.04 LTS. The hardware configuration of each VM was the following: CPU - 2 virtual sockets with 2 cores per socket (limited to 1500 MHz); 2 GB of RAM memory; 12 GB of virtual disk space provided from datastore. The naming convention of the VM was following : dzvmN (where N was the number of the VM) and resulted in names like dzvm1, dzvm2, dzvm3, etc. All the VMs were connected into a single network by means of distributed vSwitch using one port group with the egress bandwidth limited to 1 Gbit.

In order to deploy the PBT application a proper technology stack was needed. Because of the fact that the PBT is an OSGi-based application the Fuse ESB (version 4.2.0) implementation was selected. For the purpose of the distributed communication between the PBT services the Oracle Coherence (version 3.1.7) distributed cache was chosen. Selected technologies were deployed to all the VMs and launched with the use of the JVM (Java SE Runtime Environment - build 1.6.0 45-b06). Each of the OSGi containers was given a name in a similar pattern as VM (fuseN).

The PBT application was deployed to the Fuse containers using the build-in features capabilities of Apache Karaf. In order to make the experimental testbed more similar to the production environment the Oracle Coherence cache servers (single JVM process per VM) were initially configured as presented in Figure 6.2

As can be seen, there are three different groups of nodes which host different services. All the nodes that are within the frontend group are preconfigured to host only the services that are used for the purpose of the user interface, which include the *frontend service* introduced in Figure 6.1. The nodes which are part of the backend group host services that are more CPU intensive and time consuming. They include such services as *planning, booking and seats*. Finally, there are the storage group host services that realize the database and persistence operations. The *persistence and notification services* are hosted on the nodes of this category. As presented in Figure 6.1 the services from the frontend group can only communicate with the storage group with the use of backend group. Such configuration enabled to introduce the load-balancing mechanisms improving the overall performance and scalability of the presented solution.

Despite the fact that the introduced hardware and software configuration is the enterprise class, it is still insufficient to maintain a given quality of service without the adaptation enforcement. The next sections of this chapter describe the experiments realized in this area as well as the conclusions that can be reached on their basis.

### 6.2 Functional Evaluation

In this section a functional evaluation of the concepts related to the proactive adaptation of the multi-layered service oriented environment is made. The evaluation is divided into a sequence of three experiments that presents a different aspects tackled in this dissertation which in overall confront the statement of the thesis.
In order to evaluate the functional aspects from the same perspective in all the subsequent experiments a single case study (in the context of presented PBT application scenario) will be used. In this study case it is assumed that the PBT application suffers a highly increased load which leads to the unacceptable response time during the system usage by the end users. The increased load of the system is widely observed in the similar class of the applications during the holidays period, after the announcement of trip discounts, etc. The SLA defined for the PBT application was specified by the administrator in the following way (threshold values specified as approx. twice as much as during nominal system operation):

- Average planning service response time lower than 500ms;
- Average booking service response time lower than 400ms;
- Average seats service response time lower than 250ms;
- CPU consumption of all nodes lower than 80%.

In all experiments presented in this section the initial deployment of the services in the aforementioned infrastructure was as follows: fuse1 - frontend service; fuse2 - booking service; fuse3 - planning service; fuse 4 - seats service; fuse5 - notification service, persistence service. The CLGM was started on an external machine and it generated the request according to the chart presented in Figure 6.3.

![Fig. 6.3: Load generated while conducting experiments in the functional evaluation](image)

As can be seen, the testing scenario took almost 10 minutes and reached the average of approximately 1000 [RPM]. From 2nd to 6th minute of the experiment the system is under the heaviest load because the CLGM generates exactly 2 thousand [RPM]. The presented scenario assumes that all of the generated requests are hitting the frontend service. However, the other services are used depending on the behaviour of the end users. Only 90% of them are using the planning operations which invoke planning service and 60% are making the reservations which utilize the booking service. In accordance with the application scenario (c.f. Figure 6.1) all the invocations of booking service also triggered the invocation of seats, persistence and notification services.
The observed services invocation count (2 seconds moving average) during the system runtime is presented in Figure 6.4. The presented results illustrate that starting from 120th second of the experiment the average response time metric of planning service (rt_srv.planning) is rapidly growing (as presented in Figure 6.5e) and is significantly different from its nominal value. Such a situation causes that during the increased load the PBT application is not responsive and fails the demands of the travel agency clients. The root cause of this problems can be easily traced by the analysis of the resources utilized by the PBT application. As can be seen in Figure 6.6b, the value of the cpu_fuse2 metric (container which hosts the planning service) is hitting almost 100% from the time when the CLG2 generated 800 RPM. In addition, the cpu_fuse3 shown in Figure 6.6c (container which hosts booking service) is also very high and is on the edge of influencing the response time of the hosted services. On the other hand, the rest of containers have more resources that can be used in order to improve the overall system performance.

In order to solve the problem with the system responsiveness it was decided to employ the PRIDE framework to enrich the PBT application with the proactive adaptation capabilities. According to the PRIDE integration process, described in Section 5.1.1, the administrator executed each of the necessary steps. First of all, the PBT containers were enriched with the aspects which enabled the dynamic installation of sensors and effectors. The prepared configuration of sensors included the monitoring of all the system services as well as OS level CPU measurements. The prepared configuration of effectors included the mechanisms for the dynamic instantiation of PBT application services. In order to evaluate which combination of the introduced adaptation mechanisms extensions (provided in the PRIDE framework) is the most suitable and gives the best price to a value ratio a sequence of three experiments was conducted.

To maintain the clarity of experimental results, each section is divided into the following subsections:

- system configuration - describes the configuration of the PRIDE framework including the internal configuration of adaptation mechanisms;
Chapter 6. System Evaluation and Experimental Results

(a) Booking service response time metric

(b) Frontend service response time metric

(c) Notification service response time metric

(d) Persistence service response time metric

(e) Planning service response time metric

(f) Seats service response time metric

Fig. 6.5: Initial values of response time metric values without enforcement of adaptation
Chapter 6. System Evaluation and Experimental Results

(a) Fuse 1 CPU consumption metric

(b) Fuse 2 CPU consumption metric

(c) Fuse 3 CPU consumption metric

(d) Fuse 4 CPU consumption metric

(e) Fuse 5 CPU consumption metric

Fig. 6.6: Initial values of CPU consumption metric values without enforcement of adaptation.
Chapter 6. System Evaluation and Experimental Results

- experiment results visualization - presents the results obtained during the experiments presented from a high-level perspective of a given SLA;

- results discussion - describes the gains (or eventual losses) observed during the test scenario. It explains the specific actions undertaken by the PRIDE system as well as presents the data on which such actions were decided;

- experiment conclusions - summarize and concludes the execution of the experiment.

6.2.1 Experiment 1 - Utilization of Relationship Model in Adaptation Process

The main goal of this experiment is to verify the usefulness of the PRM model utilization while deciding about the actual configuration of the PRIDE framework adaptation mechanisms. The presented results were obtained by enabling the PRIDE correlation measurement during system uptime for a given set of metrics (i.e. values presented in Figure 6.5 and 6.6) which are used to construct the PRM. The detailed structure of the PRM model was previously introduced in Section 3.3.1. During the experiment, none of the adaptation mechanisms was enabled.

Experiment results and discussion

The experiment was conducted with the same load as presented in Figure 6.3. The measurements of the PRM model were done throughout the whole scenario execution. Yet, the most significant results were obtained while the system was working in a nominal way, thus no SLA violation could be observed. As shown in Figure 6.5e and 6.6b, these periods were identified as follows: i) the first two minutes of the experiments ii) the last two minutes of the experiments. In the period in between when the system was not working in a nominal way the obtained results were insignificant.

The obtained results are visualized in a form of matrix, which represents the correlation values computed with the use of the configured correlation technique. The utilization of the Pearson’s coefficient produced the incorrect correlation values because this technique measures the strength of the linear relationship between the variables. Therefore, in case of metrics that do not expose strong linear relationship the Spearman’s rank correlation coefficient should be used.

To ease the way of finding hidden patterns in the analysed matrix it was reordered in accordance with a given clustering algorithm. In this case, hierarchical clustering algorithms were used and the identified clusters of dependant metrics were highlighted. Each row and column specifies a given metric measured in the system. Their interpretation is the following:
- *cpu_NAME* - CPU consumption metric of container NAME (such as fuse1, fuse2, fuse3, etc.)
- *ic_srv_NAME* - invocation count metric of service NAME (such as booking, planning, etc.)
- *rt_srv_NAME* - response time metric of service NAME (such as booking, planning, etc.)

A particular cell in the matrix represents the correlation coefficient between two metrics. The metrics can be correlated either positively or negatively. The former means that in case when one metric changes its value, the other will also change it with the same trend. The positive correlation is represented by the intensity of the blue color or circle size. The latter means that the second metric will change with the opposite trend. The negative correlation is represented by the intensity of the red color or circle size. As it can be seen the elements on the matrix diagonal are perfectly correlated as they point to the same metric. The order of the matrix columns and rows resulted in the execution of the hierarchical clustering algorithm. It performs rows and columns reordering in order to find the clusters of correlated data.

As mentioned previously, there are two phases during the experiment which provide

![Correlation matrix visualization](image)

Fig. 6.7: Experiment 1 - Correlation matrix visualisation computed during nominal system run before occurrence of SLA violation
really interesting results. The former is conducted while observing the growing load in the system which causes the increasing consumption of all the resources (c.f. Figure 6.7). The latter is conducted while the decreasing system resource consumption can be observed (c.f. Figure 6.8). Altogether, the clustering results will show the best structuring of the Adaptive Controllers as well as answer the question whether the clusters are dependant between each other thus leaning into the global adaptation approach.

Figure 6.7 presents the correlation matrix which was computed in the first two minutes of the test scenario, before the occurrence of the SLA violations. It is a temporal representation of the PRM model used by the Adaptive Controller in the first part of the experiment. The internal mechanisms of the PRIDE framework detected two clusters of independent metrics. The first cluster contains the metrics which refer to the service response time metric of the PBT application. The second cluster contains the metrics which refer to both hardware and software resources. Because of the fact, that there exist the dependencies between the metrics related to both hardware and software resources, the preliminary decisions would point to the setup of at least two Adaptive Controllers which will supervise the PBT execution environment. Presented results also show that the invocation count metrics are perfectly correlated with CPU consumption. It means that the increasing value of given services will result in the increase of CPU utilization, and the opposite.

Fig. 6.8: Experiment 1 - Correlation matrix visualisation computed during nominal system run after disappearance of SLA violation
Figure 6.8 presents the correlation matrix which was computed during the last two minutes of the testing scenario. The most beneficial information which could be obtained from this visualisation is that such metric as \textit{rt_srv\_frontend} is negatively correlated with all the metrics related to the invocation count or CPU computation. This occurs because the \textit{frontend service} is the first one while processing the invocations and it is invoked per each request (c.f. Figure 6.1). This in turn, is extremely useful when working on the adaptation strategy because it points out that the lowering of CPU consumption or invocation counts of some services will positively influence the response times of other correlated services.

**Conclusions**

The results presented in this experiment show that the PRM offers significant information which can be used while deciding about the configuration of the adaptation mechanisms. Moreover, the utilization of the same data in the adaptation process itself enables better decision making. In case of the PRIDE framework such information influences the \textit{Adaptive Controllers} while deciding which service should be changed in the dependant system or to what extend the computation resources should be increased.

### 6.2.2 Experiment 2 - Influence of \textit{Adaptive Controllers} Communication Scheme on Adaptation Enforcement Results

The goal of this experiment is to verify which type of the AC communication scheme gives better results when used during the adaptation of exemplary PBT application. On the basis of the results presented in the previous Section 6.2.1 four testing scenarios were executed for the purpose of this experiments. The communication schemes of the \textit{Adaptive Controllers} included the following configurations:

- \( C_{R1} \): one reactive adaptive controller supervising the virtualized resources layer;
- \( C_{R2} \): one reactive adaptive controller supervising the services layer;
- \( C_{R3} \): two reactive adaptive controllers supervising separately the virtualized resources and services layer;
- \( C_{R4} \): one global reactive adaptive controller supervising simultaneously the virtualized resources and services layer.

The adaptation goal defined in this experiment was to maintain the SLA defined in Section 6.2 while the generated load was the same as in Figure 6.3. The adaptation actions which could be executed by the specific Adaptive Controllers are as follows:

- the increase of the CPU resources - virtualized resources layer;
• the addition, removal or shift of service instance from a given container - services layer.

The layered Adaptive Controllers are capable to execute only actions valid for their layer, the global Adaptive Controller may enforce actions on all of the layers.

Experiment results and discussion

In order to ensure that the subsequent evaluation runs did not influence each other the testbed environment was reset each time a given scenario finished. To maintain the clarity of the presentation only selected subset of obtained and processed data will be presented. The results of configurations utilizing single AC in layered manner \((C_{R1}, C_{R1})\) prove that the enforcement of adaptation even in a single layered configuration significantly improves the application performance. However, in such a case, the violations of SLA are still quite common because the supervisory process is realized only in the single layer. It means that in the \(C_{R2}\) configuration there is no adaptation in the services layer while in case of \(C_{R1}\) configuration there is no adaptation in virtualized resources layer. Therefore, these type of AC configurations should be applied when we have the SLA definition that refers only to a single layer of the system or the elements of a given system layers do not influence the behaviour of each other.

Figures 6.9, 6.10, 6.11 and 6.12 present the results of the SLA related metrics (CPU consumption and response time) comparison in case of two configurations \(C_{R3}\) and \(C_{R4}\) - layered (c.f. Figures 6.9 and 6.11) and global (c.f. Figures 6.10 and 6.12) approach to the reactive adaptation. The main difference between these two configurations is the following. In the former, the ACs do not exchange with each other the information about the activities or measurements observed in a given layer. In the latter, the global AC is gathering the contextual information from all the layers and realizes the adaptation enforcement on its basis. The \(C_{R4}\) would be almost equivalent to the scenario where the ACs from \(C_{R3}\) would exchange the context data between them.

In both figures the value of a given measured metric is presented as a red line, applied SLA threshold as a green line while the blue area represents the SLA violations. The vertical markers presented in some graphs represent the adaptation actions enforced by a given AC at specific time. In case of metric related to virtualized resources layer the markers represent the action of CPU addition in a presented VM. In case of service response time the markers represent the action of addition (ADD) or removal (RMV) of an analysed service instance in a given container.

As can be seen, the global adaptation enforcement gives much better results than the layered approach. The amount of time as well as number of incidents when the SLA is violated is significantly lower. The only situation when the SLA violation occurred is the beginning of the second phase of load generation (c.f. Figure 6.3). All subsequent load
Fig. 6.9: Experiment 2 - SLA related CPU consumption metric values utilizing reactive ACs in layers independent approach
Fig. 6.10: Experiment 2 - SLA related CPU consumption metric values utilizing reactive ACs in global approach
Fig. 6.11: Experiment 2 - SLA related response time metric values utilizing reactive ACs in layers independent approach

Fig. 6.12: Experiment 2 - SLA related response time metric values utilizing reactive ACs in global approach
changes does not influence the overall performance of the system at all. Unlike $C_{R4}$, the configuration $C_{R3}$ did not find the way to prevent more violations of a given SLA.

To explain why it happened it is crucial to analyse the exact decisions taken by the ACs in both cases. As could be predicted, the overall number of the adaptation decisions enforced in configuration $C_{R3}$ is much higher than in $C_{R4}$ which happens because the ACs are taking actions separately. Moreover, it is not only the matter of difference in the number of adaptation actions but also their precision. After the violation of a given metric the global AC evaluates how such metric can be improved from the perspective of other layers of the system. In this case, when there is a violation of CPU consumption the system not only increases the available number of CPUs but also instantiates new service instances (or moves existing ones) to other nodes. Thanks to that, it is not only possible to quicker remove a given SLA violation but also to dynamically adapt the supervised environment in order to find a better solution in the current situation. However, in both cases the results are significantly better than they would be without the adaptation mechanisms.

**Conclusions**

The results of the experiment show that even non-complex AC configurations offer significant gains in contrast to the scenario without adaptation. Even the deployment of layer independent AC configuration may be sufficient when there is no direct relationship between the system layers. In cases when we already know the dependencies between the layers the deployment of global AC that is aware of them will give much better results. The results have also proven the correctness of the conclusions formulated in Experiment 1 in which the constructed PRM model pointed to the strong dependencies between different system elements which excluded the layer independent realization of adaptation process.

Despite of the fact, that the global approach for adaptation gives better results than its counterpart, it cannot be used in all cases. There may exist some security policies in the system which will restrict the enforcement of the adaptation process from a global perspective. Moreover, in case of the systems which are highly dynamic, yet deployed in the different geographic locations it may be impossible to enforce the centralized approach. In such cases the decisive Adaptive Controller should be located in the immediate vicinity of the critical system parts to minimize the communication overhead.

### 6.2.3 Experiment 3 - Online Prediction Aspects in Adaptation Enforcement

In this experiment different aspects of adaptation mechanisms are evaluated. Its goal is to verify whether (and if so in which configuration) the proactive adaptation con-
cepts should be utilized. The experiment extends the scenarios (as well as the results) presented in the Section 6.2.1. The SLA definition as well as load characteristic is the same as in the previous one. The configurations of the proactive supervisory adaptive controllers are as follows:

\( CP_1 \) - two proactive adaptive controllers supervising separately the virtualized resources and services layer;

\( CP_2 \) - one global proactive adaptive controller supervising simultaneously the virtualized resources and services layer.

Both the \( CP_1 \) and \( CP_2 \) configurations were evaluated with different values of prediction window length. In the first part of this experiment only the best achieved results are presented. In the second part, a broader evaluation of results relying on this parameter is made.

**Experiment results and discussion**

Similarly to the previous experiment only the most important subset of data will be presented. Figures 6.13, 6.14, 6.15 and 6.16 depict the values of the metrics that are related to a specified SLA. In order to be able to positively predict the upcoming violations it is necessary to analyse the dynamism of the supervised system. In case when the changes of the SLA-related metrics occur slowly the prediction window length may be wider, otherwise it has to be narrower. In case of the analysed PBT application the changes occurs very quickly which is implied by the load generated during the experiment. Therefore, initially a few preliminary runs of the experiment were executed in order to fit the prediction window length. This allowed to specify the prediction window length value equal to 5 seconds. Further part of this experiment presents how the system behaves when the prediction window is greater than 5 seconds and what the implication of such a situation are.

The configuration \( CP_1 \) gives slightly better results than its reactive equivalent - \( CR_3 \). However, as can be seen there is a higher number of enforced adaptation actions. Such situation is understandable when the online prediction mechanisms are utilized.

There are two aspects which at first sight may improve the quality of the discussed approach. The first one is related to the decisions quality of adaptation mechanisms (which in all presented experiments were relatively simple and were based only on measured metric values). The research conducted in this thesis proves that the decisions making can be improved by utilizing the relationship between different elements of the system. This aspect will be described in the next section of this chapter. The second one is related to the value of the prediction window length parameter. At first sight, it may seem that higher values of this parameter will improve the performance of the predictive approach. Additional testing scenarios conducted in this experiment prove that...
Fig. 6.13: Experiment 3 - SLA related CPU consumption metric values utilizing proactive ACs in layered approach
Fig. 6.14: Experiment 3 - SLA related CPU consumption metric values utilizing proactive ACs in global approach
Fig. 6.15: Experiment 3 - SLA related response time metric values utilizing proactive ACs in layered approach

Fig. 6.16: Experiment 3 - SLA related response time metric values utilizing proactive ACs in global approach
such assumption is not entirely true. Figure 6.18 depicts the behaviour of the proactive adaptation mechanisms utilizing different prediction window lengths.

As can be seen, the higher the value of this parameter is, the more SLA violations occur and more adaptation actions take place (c.f. Figure 6.19). Such a characteristic is a result of the following aspects: i) the nature of online prediction mechanisms utilized by the PRIDE framework; ii) the characteristic of the PBT application (high dynamisms of modelled experiments).

The realization of the PRIDE framework online prediction mechanisms relies on the trends estimation utilizing the linear regression (c.f. Section 5). Because of that, the predicted values of metric are computed on the basis of its last measured value and the trend computed for a given time period. If the trend is rapidly rising then the predicted value is also raising significantly. In such cases, longer prediction window length causes that the final predicted value is very high, thus the PRIDE system is enforcing adaptation actions to prevent the SLA violation. When the supervised system and/or application shows the signs of high dynamism then such actions occur frequently which introduces a high amount of unnecessary disturbances in the system. Moreover, it may cause that the PRIDE framework will not notice the violations that should be prevented though this will generate more costs than benefits.
Chapter 6. System Evaluation and Experimental Results

Fig. 6.17: Experiment 3 - Proactive adaptation behaviour with different prediction window lengths - CPU consumption metric - logarithmic scale
Chapter 6. System Evaluation and Experimental Results

Fig. 6.18: Experiment 3 - Proactive adaptation behaviour with different prediction window lengths - response time metric
Chapter 6. System Evaluation and Experimental Results

Conclusions

The results presented in this experiment showed that the adaptation enforcement realized in a proactive manner exposes a few specific characteristics. First of all, with the use of such approach it is possible to almost completely avoid the occurrences of SLA violations. However, the utilization of such mechanisms may introduce higher number of adaptation actions that will be executed. This approach will give better results only when the length of the prediction window will be adjusted to the dynamism of the supervised application as well as the specified threshold values of the SLA definition. In case when the values of the SLA related metrics are changing slowly then the length of the prediction window should be bigger. On the contrary, the higher the dynamism of the supervised application is, the smaller the prediction window length should be.

In order to verify whether the proactive approach is worth using, a cost function of SLA violation has to be defined. For this purpose the following definition is assumed, the
cost function of SLA violation at a time $t$ is the sum of all violation values that occurred during the experiment before the given time. The comparison of SLA violation cost is presented in Figure 6.20 with the use of logarithmic scale. As it can be seen, the global proactive approach not only maintained the given SLA without violations the longest but also avoided almost all of them. Unfortunately, most of the multi-layered service oriented systems do not allow for the global supervision for the reasons of safety or problems with distributed deployment. Yet, presented results point out that it may be worth considering as such approach gives much more flexibility during the supervision and maintenance of such systems.

### 6.2.4 Experiment 4 - Behaviour of the system under varying SLA

This experiment evaluates the behaviour of the system either while the specified SLA has changed at runtime or there were some changes in the execution environment of the supervised system. Its goal is to verify whether the adaptation mechanisms will be able to maintain a given SLA or adapt accordingly to its new definition. The experiment is divided into two separate parts (a and b) which tackle both of the aforementioned aspects. In both cases there exists a cluster of containers which hosts the sample PBT application. Each node hosts all of the services introduced by the sample application. The system experiences the constant load of 1000 [RPM] while being supervised by a single global proactive Adaptive Controller.

In the first part of the experiment it was assumed that at some point the SLA definition related to the CPU consumption metric changed. To evaluate the broadest of the possible cases the scenario assumes CPU utilization reduction of a SLA entry as presented in Table 6.2.

In the second part of the experiment the scenario was slightly different. It was assumed that at some point some of the containers should experience a lower load while the others should experience a higher one. At the same time the acceptable response time of analysed serviced was increased. The SLA definition changes are presented in Table 6.3.
### Table 6.2: Experiment 4 - scenario 1 - SLA definition changes

<table>
<thead>
<tr>
<th>Experiment time [s]</th>
<th>Description</th>
<th>SLA definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial stage</td>
<td>(1) booking lower service response time than 400ms, planning service response time lower than 500ms, seats service response time lower than 250ms; (2) CPU utilization for all containers lower than 80%.</td>
</tr>
<tr>
<td>60</td>
<td>Maintenance mode of container fuse1</td>
<td>(1) services response time same as in Initial stage; (2) CPU utilization for fuse1 container lower than 5%; (3) CPU utilization for other containers lower than 80%.</td>
</tr>
<tr>
<td>120</td>
<td>Maintenance mode of container fuse2</td>
<td>(1) services response time same as in Initial stage; (2) CPU utilization for fuse2 container lower than 5%; (3) CPU utilization for other containers lower than 80%.</td>
</tr>
<tr>
<td>180</td>
<td>Maintenance mode of container fuse3</td>
<td>(1) services response time same as in Initial stage; (2) CPU utilization for fuse3 container lower than 5%; (3) CPU utilization for other containers lower than 80%.</td>
</tr>
<tr>
<td>240</td>
<td>Maintenance mode of container fuse4</td>
<td>(1) services response time same as in Initial stage; (2) CPU utilization for fuse4 container lower than 5%; (3) CPU utilization for other containers lower than 80%.</td>
</tr>
<tr>
<td>300</td>
<td>Maintenance mode of container fuse5</td>
<td>(1) services response time same as in Initial stage; (2) CPU utilization for fuse5 container lower than 5%; (3) CPU utilization for other containers lower than 80%.</td>
</tr>
<tr>
<td>360</td>
<td>Final stage</td>
<td>The same as in Initial stage</td>
</tr>
</tbody>
</table>

### Table 6.3: Experiment 4 - scenario 2 - SLA definition changes

<table>
<thead>
<tr>
<th>Experiment time [s]</th>
<th>Description</th>
<th>SLA definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial stage</td>
<td>(1) booking lower service response time than 400ms, planning service response time lower than 500ms, seats service response time lower than 250ms; (2) CPU utilization for all containers lower than 80%.</td>
</tr>
<tr>
<td>60</td>
<td>Traversing the load from containers fuse1 and fuse2 to fuse4 and fuse5 with fuse3 as an intermediate</td>
<td>(1) booking service response time lower than 800ms, planning service response time lower than 1000ms, seats service response time lower than 500ms; (2) CPU utilization for containers fuse1 and fuse2 lower than 15%; (3) CPU utilization for container fuse3 lower than 50%; (4) CPU utilization for containers fuse4 and fuse5 lower than 95%.</td>
</tr>
<tr>
<td>180</td>
<td>Traversing the load from containers fuse4 and fuse5 to fuse1 and fuse2 with fuse3 as an intermediate</td>
<td>(1) booking service response time lower than 800ms, planning service response time lower than 1000ms, seats service response time lower than 500ms; (2) CPU utilization for containers fuse1 and fuse2 lower than 95%; (3) CPU utilization for container fuse3 lower than 50%; (4) CPU utilization for containers fuse4 and fuse5 lower than 15%.</td>
</tr>
<tr>
<td>300</td>
<td>Final stage</td>
<td>The same as in Initial stage</td>
</tr>
</tbody>
</table>
Experiment results and discussion

Figure 6.21 presents the results obtained during the execution of the described experiment. To maintain the clarity of the figures the results do not present the decisions generated by the Adaptive Controller in the context of the analysed services.

As can be seen the adaptation process enforced by the system was capable of adopting to varying changes in the SLA definition. The system was able to maintain the specified SLA and migrate the necessary services when needed. However, there were some violations which were not prevented in the context of both layers (services and virtualized resources). The former violations are the result of the decisions taken by the Adaptive Controller to prevent the latter. There can be seen a sequence of CPU addition modification requests exactly at 60, 120, 180, 240 and 300 second of the experiment. This was exactly when the modification of SLA violation was triggered and the system tried to adapt the whole environment to it. The violations occurred because at this point there were dozens of requests to the process already scheduled on a given node which had to be handled. Afterwards the services was traversed from this node and the new SLA was kept correctly.

Figure 6.22 presents the results obtained during the execution of the described experiment. As shown, the system successfully adapted the supervised environment to the changed one expressed in the SLA. Similarly to the previous case when a given metric SLA entry was lowered there exists a threshold in which the SLA is violated, caused by the scheduled requests to process. In presented figures there is a lot of violations of fuse3 CPU related metric. Those violations are directly related to the nature of the PBTr application. Each request of the booking or planning service is CPU intensive and in case of 1000 RPM the proposed solution could not predict in time the upcoming violation and made some decisions to prevent it (as already concluded in Experiment 3).

Conclusions

The presented experiment introduced the capabilities of the proposed solution in case of varying SLA definitions. In general the proactive capabilities of the adaptation mechanisms prevent the SLA violations occurrence, but in some special cases they may be not sufficient. Especially when the changes in the system occur very fast and the prediction mechanisms are not able to cope with such rapid changes. Nevertheless, the gathered results show that the proposed solution is capable of dealing with runtime changes of agreed SLA contract.
Fig. 6.21: Experiment 4a - SLA related CPU utilization metric values
Fig. 6.22: Experiment 4b - SLA related CPU utilization metric values
6.3 System Overhead and Scalability Aspects

The purpose of this experiment is to evaluate the overhead of computational resources and network bandwidth which will be caused by the deployment and usage of the proposed solution. The first part of the experiment deals with the overhead generated in the execution environment of the supervised system. The second one evaluates the Adaptive Controller CPU consumption in case of the increasing number of managed containers.

Experiment results and discussion

First, it was evaluated what the exact CPU overhead caused by the deployment of instrumentation mechanisms was, which was further followed by the activation of adaptation process. The metric was measured during the deployment of a single application container in the same scenario in Experiment 2 and 3.

![Fig. 6.23: Experiment 5 - CPU overhead caused by deployment of instrumentation mechanisms and activation of adaptation process](image)

The measurements were done in three different scenarios which are as follows:

- **nominal** - CPU consumption observed during the nominal system operation which was not instrumented;
- **monitoring only** - CPU consumption observed during the nominal system operation which was instrumented and each request was processed by sensors, yet no adaptation actions were performed;
- **monitoring and adaptation** - CPU consumption observed during the system operation and enabled adaptation enforcement.

Figure [6.23] presents the results obtained during the experiment. As can be seen, the overhead CPU overhead generated by the system is negligible. Such good results are achieved because the used sensors do not analyse the content of the processed requests as in most cases it is not necessary when computing invocation count or response time.
metrics. If the into the intercepted request would be needed then the overhead would be for sure much higher.

The next aspect of this experiment is related to the network traffic overhead. Similarly to the previous case the test scenario does also evaluate three different cases: nominal, monitoring only and monitoring and adaptation. Moreover, the probing was done separately for incoming and outgoing network traffic. Figure 6.24 presents the results obtained during the test execution which were measured in the execution environment part (instrumented application container) of the PBT application.

As shown, the achieved results are quite impressive. In the context of the incoming network traffic the difference in all three cases is also negligible. This means that the communication mechanisms designed and implemented in the proposed solution are quite efficient and the adaptation enforcement does not impose high inbound traffic utilization. The situation is slightly worse in the context of the outgoing network traffic. In this case the monitoring mechanisms of proposed solution produce about 40B/s per request which are transferred through the network to the proper Adaptive Controller. Once more there is almost no difference between the scenarios with and without the adaptation enforcement.

The last part of this experiment is related to the resource consumption realized by the Adaptive Controller during its operation. During the subsequent executions of the experiment new containers were added to the cluster. The scenario assumes that the number of services deployed in each of the containers is increasing by 50 in 20 second periods (marked as N srv/cnt). The generated load scheme in the system is the same in all the subsequent configurations and it is assumed that each of the deployed services is invoked once per second.

Figure 6.25 presents the results obtained during the experiment. Each of the graphs represents the averaged CPU consumption done by the Adaptive Controller process. The results are smoothed with the use of bezier spline to for the easier analysis of gathered data. As can be seen, the measured CPU consumption is slightly growing while
the number of managed containers is increasing. Although, the number of evaluated container is not big the results allows to state that the current implementation of a single Adaptive Controller will not be able to handle more than several nodes. Yet, even in such scenarios it can easily manage a hundred of resources in each of them.

Conclusions

The results presented in this experiment verified the correctness of the proposed solution design. Very low resource utilization footprint confirms that the proposed concept was implemented efficiently which in turn verifies the proposed architectural decisions. Moreover, such good results allow to state that the described solution could easily be used in high-scale environments.

6.4 Summary

In this chapter a thorough evaluation of concepts related to the proactive adaptation of multi-layered service oriented systems was done, with the use of the PRIDE framework prototype. The primary goal of the evaluation was to verify whether the formulated thesis statement and high-level concepts of the service oriented environments proactive adaptation presented in this dissertation are valid. The evaluation covered five different experiments which encompassed both functional and non-functional aspects of the described concepts. On the basis of the conducted experiments several conclusions can be stated.

The evaluation has proven that the proposed Property Relationship Model introduces
significant improvements while realizing the adaptation process in multi-layered service-oriented systems. It can be used as a knowledge base about the supervised system which provides useful information when determining the relationships between the different system elements (Experiment 1). Moreover, it is a proven source of information when enforcing the adaptation process regardless of the adaptation type - reactive or proactive (Experiment 2 and 3).

The proposed Adaptive Controller architectural concepts (which extend the MAPE-K pattern) also proved to be a valid. Their internal architecture exposes highly customizable deployment capabilities which fit well into the multi-layered environments (Experiment 2). Moreover, the layered architecture of the Adaptive Controller, which is an extension of the AS3 model, enables their reconfiguration at run-time and transparent management of the supervised environments. In turn, the transformation process of high-level declarative adaptation policy into the low-level supervised system actions makes it possible to adapt to varying changes of the SLA definitions at runtime (Experiment 4). This has also proven that the proposed concepts are compliant with the ACI concept.

The design of the proposed architecture assumed that the system should remain operational in the deployments characterized by a high scale. The evaluation has proven that in the area of CPU consumption the architectural design is correct and enables the supervision of hundred of services. In case of the network bandwidth utilization the results are also exceptionally good, which is achieved thanks to the implementation of dedicated optimization mechanisms in the PRIDE framework and minimization of the data exchange on the wire (Experiment 5).

The presented experiments altogether with their results allowed to successfully verify the fulfilment of all expectations stated in the thesis.
Chapter 7

Conclusions

This chapter concludes the contribution of this dissertation in the area of the service oriented systems adaptation as well as it verifies the thesis statement. Additionally, it also describes the potential directions which could be considered in the future work of proposed concepts.

The runtime prediction of the service realization quality and its utilization in the adaptation process is one of the top research challenges in the recent years. Yet, the hierarchical nature of the SOA environments causes it to be much more complex and demanding task. The proper realization of the adaptation process in such systems is not trivial and it requires several research aspects to be solved.

The first issue directly relates to the aspects of knowledge acquisition and the representation of the supervised system, including its structure and dependencies. Then, there is a need for utilization of such knowledge through the formulation of high-level goal which will be used in the process of adaptation enforcement. Next, the abstract Platform Independent Model has to be proposed which will base on such a high-level goal and realize the adaptation process in a predictive manner. Finally, the PIM should be mapped into the Platform Specific Model which allows for imposing the adaptation process in technology-specific domains.

This chapter elaborates to what extend the proposed approach to proactive adaptation of multi-layered service oriented systems copes with the mentioned issues. Moreover, it identifies the possible drawback or aspects not entirely covered by the dissertation which are important especially in the context of future works.
Chapter 7. Conclusions

7.1 Thesis Verification

The thesis statement formulated in this dissertation expects a proposal of the adaptation mechanisms which can be used to enrich the multi-layered service oriented systems. The mechanisms are supposed to offer runtime configurable capabilities which enable the improvement of the service realization quality in a proactive manner with the use of multi-layer dependencies occurring in the supervised system. The fulfilment of the aforementioned statements is a complex and demanding task. To effectively solve the stated problem a few aspects have to be addressed.

First of all, the service oriented systems are highly dynamic environments constituting of multiple layers where the layers of higher level are often overlapping the complexity of the lower ones. Therefore, the desired approach should expose the capabilities of transforming the high-level expectations into the configurations of mechanisms located in the lower system layers. This dissertation proposed the approach complaint with the AS3 Pattern which enables the application of the high-level adaptation policy vertically across the multiple layers of the S3 stack. Such a process ends with the configuration of low-level sensors and effectors which are seamlessly activated in a supervised system in a transparent manner at runtime. Although, the architecture is valid for all layers of the S3 stack, the prototype implementation of the proposed concepts encompassed only operational resources and services layers. To deal with the dynamisms of service oriented system the policy used in the transformation process is consistent with the declarative approach. This means that it expresses indirectly which elements should be the target of adaptation and they dynamically adapt to the changes in the supervised execution environment.

Another important aspect that has to be addressed to fulfil the formulated statement is related to knowledge representation and acquisition. In order to efficiently enforce the adaptation process in such environments it is crucial to acquire the information about the dependencies of the services and other elements executed in the supervised system. Thanks to that it is possible to adapt the decision-making process at runtime in accordance with the changes in the execution environment. This dissertation introduced the Property Relationship Model which reflects the aforementioned concepts utilizing the correlation coefficient techniques. On its basis it is possible to identify what the dependencies between different system elements look like. The correlation coefficient between different system properties supports the process of taking a decision on which actions should be undertaken when a violation of an agreed SLA contract is anticipatory by pointing out what impact a given adaptation action may have when enforced. Furthermore, the data contained in the PRM model makes it extremely useful when deciding about adaptation enforcement in a proactive manner. During the online prediction activities it is crucial to know what the expected output would be while performing the adaptation actions and this is what the PRM model offers. The predictive capabilities of the adaptation mechanisms proposed in this dissertation are also supported by
other mechanisms such as: trends estimation or symptom-based matchmaking which are well-known in the literature.

Finally, the last aspect that has to be addressed is strictly related to the adaptation enforcement in multi-layered distributed environments. Taking into account that the service oriented systems are multi-layered by their nature the decision-making elements should also be capable of hierarchical/multi-layered deployment. Thanks to that, the adaptation process enforcement will fit well into the architecture of the supervised system itself. Moreover, such approach will ensure that the final solution will be scalable and it will maintain operational even in large-scale geographically distributed systems. The proposed approach ensured the aforementioned through the introduction of the Adaptive Controller concept. It is a standalone entity which enforces the adaptation process in accordance with the specified high-level declarative adaptation policy. The controllers are capable of communicating, sharing context information or supervising each other. Therefore, it makes it possible to form a hierarchical deployment of adaptation enforcement elements which can supervise a given system in parallel.

In overall the verification of thesis statement can be concluded in the following points:

- The declarative specification of adaptation policy can be used to drive the configuration of low-level execution mechanisms deployed in the supervised environments. The automatic transformation of declarative configuration into the implementation level aspects;

- The dynamic instrumentation approach enables the runtime enrichment and reconfiguration of monitoring and actuating system elements;

- The multi-layered approach to the goal-driven adaptation process of service-oriented environments seems to be the most suitable solution. The utilization of the model representing relationship knowledge about the supervised system has a positive impact on the overall process and enables easier decomposition of monolith solutions into more convenient layered equivalents;

- The realization of the adaptation process in proactive manner offers a significant added value, yet it has to be fitted into the dynamisms characteristic of the supervised system.

The combination of all the formulated conclusions ensures that the contribution of this dissertation was sufficient for the successful verification of the thesis statement.

7.2 Future Work

The proposed concepts were proven as valid in the context of proactive adaptation of multi-layered systems. However, there are several areas which may be further improved as a future work. These are as follows:
• The SLA mapping procedure between different system layers could be extended. The proposed solution assumes that the SLA is provided by the system expert which previously divided it accordingly. More sophisticated solution could automatically match and recognize which elements of the SLA should be applied in particular layers.

• The prediction window could be dynamically adopted depending on the current dynamic characteristic of the system. Currently, long prediction window length will work as expected in highly dynamic environments, while the short one will work exactly the opposite.

• Current solution has configurable timeout parameter which prevents the system from enforcing the adaptation actions on the same system element too often. It could be extended to evaluate whether a particular adaptation action has already influenced the system behaviour correctly or not.

• The support for other layers of the S3 model (such as business processes) could improve the applicability of the proposed solution. Currently, only the software components (OSGi-based) and virtual resources (VMWare-based) are supported.

• The current architectural design of the proposed solution provides a solid work which could be extended in the direction of an infrastructure compliant with the concept of Application Centric Infrastructure.
Appendix A

PRIDE Configuration and Code Listings

```
public aspect BundleContextAspect {

private BundleContextAspectInterface aspectImpl;

pointcut getService(BundleContext bc, ServiceReference ref) : execution(* getService(ServiceReference))
            && target(bc) && args(ref);

Object around(BundleContext bc, ServiceReference ref): getService(bc, ref) {
    Object ret = proceed(bc, ref);
    if (aspectImpl == null) {
        try {
            MonitoringProxyClassLoader cl = new MonitoringProxyClassLoader();
            Class aspectImplClass = cl.loadClass("pl.edu.agh.asl.sensors.osgi.invocation.logic.BundleContextAspectImpl");
            aspectImpl = (BundleContextAspectInterface) aspectImplClass.newInstance();
            } catch (Exception e) {
                e.printStackTrace();
            }
        } if (aspectImpl != null) {
            return aspectImpl.aroundGetService(bc, ref, ret);
        } else {
            return ret;
        }
    }
}
```

Listing A.1: Aspect Weaving OSGi BundleContext
Appendix A. PRIDE Configuration and Code Listings

1 public aspect BundleContextAspect {
2 3 private BundleContextAspectInterface aspectImpl;
4 5 pointcut getService(BundleContext bc, ServiceReference ref) : execution(∗ getService(ServiceReference))
6   ∧∧ target(bc) ∧∧ args(ref);
7 Object around(BundleContext bc, ServiceReference ref): getService(bc, ref) {
8   Object ret = proceed(bc, ref);
9   if (aspectImpl == null) {
10      try {
11         MonitoringProxyClassLoader cl = new MonitoringProxyClassLoader();
12         Class aspectImplClass = cl.loadClass("pl.edu.soa.agh.as3.sensors.osgi.invocation.logic.
13            BundleContextAspectImpl");
14         aspectImpl = (BundleContextAspectInterface) aspectImplClass.newInstance();
15      } catch (Exception e) {
16         e.printStackTrace();
17      }
18   } if (aspectImpl != null) {
19      return aspectImpl.aroundGetService(bc, ref, ret);
20   } else {
21      return ret;
22   }
23   }
24  }
25 }

Listing A.2: OSGi Interceptor Socket

1 public class GenericMeasurement implements ExecuteResultHandler, TailerListener {
2   private Executor executor;
3   private String fileName;
4   private Tailer tailer;
5   private File file;
6   private MonitoringEvent.MonitoringType measurementType;
7   private CacheConnector connector;
8   ...
9   ...
10   private void initExecutor(String command, String ... args) throws IOException {
11      CommandLine cmdLine = new CommandLine(command);
12      cmdLine.addArguments(args);
13      file = new File(fileName);
14      if (file.exists()){
15         FileUtil.forceDelete(file);
16      }
17      file.createNewFile();
18      ExecuteWatchdog watchdog = new ExecuteWatchdog(ExecuteWatchdog.INFINITE_TIMEOUT);
19      PumpStreamHandler pumpStreamHandler = new PumpStreamHandler(new FileOutputStream(file, true));
20      executor = new DefaultExecutor();
21      executor.setStreamHandler(pumpStreamHandler);
22      executor.setWatchdog(watchdog);
23      executor.execute(cmdLine, this);
24   }
25   ...
26   private void processGenericMeasurement(String line){
27      GenericMeasurement entry = new GenericMeasurement(Statics.SERVICEMIX_NAME, line);
28      MonitoringEvent event = new MonitoringEvent(Statics.MONITORING_CENTER_ADDRESS, measurementType);
29      event.setValue(entry);
30      connector.put(CacheType.MonitoringCache, event.getMonitoringType(), event);
31   }
32   ...
33 }

Listing A.3: Implementation of GenericMeasurement class
Appendix A. PRIDE Configuration and Code Listings

```java
public class VMManager {
    private ServiceInstance serviceInstance;

    public VirtualMachine obtainVirtualMachine(String name) throws RemoteException {
        connectIfNeeded();
        Folder rootFolder = serviceInstance.getRootFolder();
        VirtualMachine vm = (VirtualMachine) new InventoryNavigator(rootFolder).searchManagedEntity("VirtualMachine", name);
        if (vm == null) {
            throw new RemoteException("No such VM: " + name);
        }
        return vm;
    }

    public void reconfigure(VirtualMachine vm, DeviceType device, long value) throws Exception {
        VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();
        switch (device) {
        case CPU:
            vmConfigSpec.setNumCPUs((int) value);
            break;
        case MEM:
            vmConfigSpec.setMemoryMB(value);
            break;
        }
        Task task = vm.reconfigVM_Task(vmConfigSpec);
        String result = task.waitForMe();
        if (result.equals(Task.SUCCESS)) {
            numOfCPUs.put(vm.getName().replaceAll(Statics.VIHOST_VCENTER, Statics.VIHOST_FUSE), (int) value);
        } else {
            throw new RuntimeException("Error: " + result);
        }
    }
}

Listing A.4: Implementation of VMManager class
```
public class ManagementAgent implements ActionListener {
    ...

    private void initializeConfigurationEntry(Configuration entry) {
        switch (entry.getType()) {
            case SERVICE:
                InfrastructureElementConfig element = createFromConfiguration(entry);
                addInfrastructureInstance(element);
                break;
            case TOP:
                case HOST:
                    for (String key : entry.getConfig().keySet()) {
                        initializeConfigurationEntry(entry.getConfig().get(key));
                    }
                break;
        }
    }

    public InfrastructureManagementResult addInfrastructureInstance(InfrastructureElementConfig element) {
        switch (element.getType()) {
            case HOST:
                return InfrastructureManagementResult.NOT_SUPPORTED;
            case SERVICE:
                ServiceRegistration registration = registerOSGiService(element);
                osgiServices.put(element.getUuid(), registration);
                break;
        }
        return InfrastructureManagementResult.SUCCESS;
    }

    private ServiceRegistration registerOSGiService(InfrastructureElementConfig element) {
        OSGIInfrastructureElement infrastructureElement = new OSGIInfrastructureElement(element, connector, false);
        infrastructureElement.initialize();
        installedServices.put(element.getName(), infrastructureElement);
        return context.registerService(InfrastructureElement.class.getName(), infrastructureElement, 
            Util.convertMapToProperties(element.getProps()));
    }

    Listing A.5: Implementation of ManagementAgent class
1 public class CoherenceCacheConnector implements CacheConnector, Serializable {
    private Cluster cluster;

    public void connect() {
        cluster = CacheFactory.ensureCluster();
        initializeCaches();
    }

    private void initializeCaches() {
        if (dispatcherMapListener == null) {
            managementCache = CacheFactory.getCache(CacheType.ManagementCache.name(), classLoader);
            monitoringCache = CacheFactory.getCache(CacheType.MonitoringCache.name(), classLoader);

            dispatcherMapListener = new MapListener() {
                public void entryInserted(MapEvent mapEvent) {
                    processActionEntry(mapEvent);
                }

                public void entryUpdated(MapEvent mapEvent) {
                    processActionEntry(mapEvent);
                }

                public void entryDeleted(MapEvent mapEvent) {
                    processActionEntry(mapEvent);
                }
            };

            addMapListener(dispatcherMapListener, managementCache, Statics.SERVICEMIX_NAME);
            addMapListener(dispatcherMapListener, monitoringCache, Statics.SERVICEMIX_NAME);
        }
    }

    public void addActionListener(BaseActionType type, ActionListener listener) {
        System.out.println("Adding new addActionListener: "+ type + "\t" + listener);
        listeners.get(type).add(listener);
    }

    public void put(CacheType type, Object key, BaseAction value) {
        switch(type) {
            case ManagementCache: cache = managementCache; break;
            case MonitoringCache: cache = monitoringCache; break;
        }

        synchronized(cache) {
            cache.put(key, value);
        }
    }

    public void setClassLoader(ClassLoader classLoader) {
        this.classLoader = classLoader;
    }

    ...}

Listing A.6: Implementation of CoherenceCacheConnector class
Appendix A. PRIDE Configuration and Code Listings

Listing A.7: CEP statement used for linear regression computation

Listing A.8: CEP statement used for average metric value computation

Listing A.9: CEP statements used for trends estimation
Appendix A. PRIDE Configuration and Code Listings

```java
1 public class CorrelationManager {
...
4     public boolean recomputeCorrelation(int numberOfEntries) {
5         int minimalCount = numberOfEntries;
6         for (String s : metrics) {
7             if (metricValues.get(s).size() < minimalCount) {
8                 minimalCount = metricValues.get(s).size();
9         }
10         }
11         correlationsCount = minimalCount;
12         if (minimalCount <= 1 || metrics.size() == 0) {
13             return false;
14         }
15         RealMatrix realMatrix = new Array2DRowRealMatrix(minimalCount, metrics.size());
16         for (int i = 0; i < metrics.size(); i++) {
17             for (int j = 0; j < minimalCount; j++) {
18                 List<Double> list = metricValues.get(metrics.get(i));
19                 realMatrix.addToEntry(j, i, list.get(list.size() - minimalCount + j));
20             }
21         }
22         long currentTimeMarker = Utilis.experimentCurrentTime();
23         PearsonsCorrelation correlation = new PearsonsCorrelation(realMatrix);
24         RealMatrix correlationMatrix = correlation.getCorrelationMatrix();
25         for (int i = 0; i < metrics.size(); i++) {
26             String firstKey = metrics.get(i);
27             for (int j = 0; j < correlationMatrix.getColumn(i).length; j++) {
28                 String secondKey = metrics.get(j);
29                 String mergedKey = firstKey + " : " + secondKey;
30                 if (correlationValues.get(mergedKey) == null) {
31                     correlationValues.put(mergedKey, new ArrayList<Double>());
32                 }
33                 double correlationValue = correlationMatrix.getEntry(j, i);
34                 correlationValues.get(mergedKey).add(correlationValue);
35             }
36         }
37         correlationTimestamps.add(currentTimeMarker);
38         return true;
39     }
...
```

Listing A.10: Implementation of CorrelationManager class
Appendix A. PRIDE Configuration and Code Listings

```
statements.sla.upper_bound
SELECT * FROM AverageMetricStream where avgValue > :value and
metricName = " : metric"

statements.sla.lower_bound
SELECT * from AverageMetricStream where avgValue < :value and
metricName = " : metric"
```

Listing A.11: CEP statements used for measuring SLA violations in a reactive manner

```
public void update(EventBean[] newEvents, EventBean[] oldEvents) {
    if (newEvents != null) {
        for (EventBean eb : newEvents) {
            Double lastAverage = (Double) eb.get("avgValue");
            String metric = (String) eb.get("metricName");
            Long timestamp = (Long) eb.get("timestamp");
            if (!Double.isNaN(lastAverage)) {
                SLAViolation violation = new SLAViolation("REACTIVE", entry, false, timestamp,
                                                       lastAverage);
                notifyAboutSLAViolation(violation);
            }
        }
    }
}
```

Listing A.12: Implementation of SLAReactiveListener class
PRIDEInternalListener listener = new PRIDEInternalListener(adaptationStrategy.getPredictionWindowList().get(i), entry) {
    @Override
    public void notify(Object o) {
        TrendOccurrenceFact fact = (TrendOccurrenceFact) o;
        String metricName = fact.getMetricName();
        double slope = fact.getSlope();
        boolean properSlope = false;
        switch (entry.getType()) {
            case UPPER:
                properSlope = slope > 0;
                break;
            case LOWER:
                properSlope = slope < 0;
                break;
        }
        double yIntercept = fact.getYIntercept();
        double currentValue = correlationManager.fetchLastMetricValue(entry.getMetricName());
        if (currentValue == -1) {
            return;
        }
        long experimentTimestamp = Util.experimentCurrentTime();
        double predictedValue = yIntercept + (slope * (experimentTimestamp + getPredictionWindow()));
        if (metricName.equals(entry.getMetricName()) && properSlope) {
            switch (entry.getType()) {
                case UPPER:
                    if (predictedValue >= entry.getThreshold()) {
                        publishSLAViolation(fact.getCategory(), entry, predictedValue, getPredictionWindow());
                    }
                    break;
                case LOWER:
                    if (predictedValue <= entry.getThreshold()) {
                        publishSLAViolation(fact.getCategory(), entry, predictedValue, getPredictionWindow());
                    }
                    break;
            }
        }
        switch (entry.getType()) {
            case LOWER:
                estimationDowntrendStatement.addListener(listener);
                break;
            case UPPER:
                estimationUptrendStatement.addListener(listener);
                break;
        }
    }
}

Listing A.13: Implementation of SLAProactiveListener class
private LoadingCache<String, SLAViolation> predictedSLAViolations;

private int SLA_PREDICTIONCONFIRMATION_TIMEOUT = 30;

predictedSLAViolations = CacheBuilder.newBuilder()
    .maximumSize(1000)
    .expireAfterWrite(SLA_PREDICTIONCONFIRMATION_TIMEOUT, TimeUnit.SECONDS)
    .build(new CacheLoader<String, SLAViolation>() {
      public SLAViolation load(String key) {
        return null;
      }
    });

Listing A.14: Timed eviction of predicted SLA violations

public class SymptomMatchmaker {
  ...
  private void updateMatchedViolationPatterns(SLAViolation violation) {
    Map<String, Double> correlatedMetrics = correlationManager.fetchCorrelatedMetrics(violation);
    Set<SymptomPattern> patterns = convertMetricsToPatterns(correlatedMetrics);
    String uuid = UUID.randomUUID().toString();
    symptomPatternsRepository.bind(uuid, violation, patterns);
  }

  private void processCorrelationValues(Map<String, Double> correlationValues) {
    Set<SymptomPattern> patterns = convertMetricsToPatterns(correlationValues);
    Set<SLAViolation> matchedViolations = symptomPatternsRepository.fetchBoundedViolations(patterns);
    for (SLAViolation violation : matchedViolations) {
      if (violation.thresholdReached()) {
        adaptationManager.handleMatchedSLAViolation(violation);
      }
    }
  }

  ...
}

Listing A.15: Symptom matchmaking patterns processing

private void handleSLAViolation(SLAViolation violation, AdaptiveControllerType controllerType) {
  ExecutionEnvironmentLayer layer = null;
  String metric = violation.getEntry().getMetricName();
  ...
  if (!adaptationStrategy.getAdaptedLayers().contains(layer)) {
    return;
  }
  if (!transitionEnforcer.checkIfTransitionIsValid(Utilis.extractInfrastructureElement(metric))) {
    return;
  }
  double threshold = adaptationStrategy.getCorrelationThreshold();
  Map<String, LongTermCorrelationStatistics> stats = correlationManager.fetchStatisticsForMetric(violation, metric, true);
  List<TransitionAction> transitions = null;
  switch (controllerType) {
    case LayeredProactive:
      case LayeredReactive:
        transitions = evaluatePossibleTransitions(layer, violation, stats);
        break;
    case GlobalProactive:
      cases = evaluatePossibleTransitionsInGlobalManner(layer, violation, stats);
      break;
    case GlobalReactive:
      ...
  }
  transitionEnforcer.enforceTransitions(transitions);
}

Listing A.16: SLA violations handling by the AdaptationManager
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACI</td>
<td>Application Centric Infrastructure</td>
</tr>
<tr>
<td>MOSES</td>
<td>MOdel-based SElf-adaptation of SOA systems</td>
</tr>
<tr>
<td>SBA</td>
<td>Service-based Application</td>
</tr>
<tr>
<td>TBF</td>
<td>Time Between Failure</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>MOM</td>
<td>Message Oriented Middleware</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SCA</td>
<td>Service Component Architecture</td>
</tr>
<tr>
<td>BC</td>
<td>Binding Component</td>
</tr>
<tr>
<td>SE</td>
<td>Service Engine</td>
</tr>
<tr>
<td>MEP</td>
<td>Message Exchange Pattern</td>
</tr>
<tr>
<td>NMR</td>
<td>Normalized Message Router</td>
</tr>
<tr>
<td>JBI</td>
<td>Java Business Integration</td>
</tr>
<tr>
<td>JCP</td>
<td>Java Community Process</td>
</tr>
<tr>
<td>JSR</td>
<td>Java Specification Request</td>
</tr>
<tr>
<td>IoC</td>
<td>Inversion of Control</td>
</tr>
<tr>
<td>DI</td>
<td>Dependency Injection</td>
</tr>
<tr>
<td>AC</td>
<td>Adaptive Controller</td>
</tr>
<tr>
<td>AS3</td>
<td>Adaptive SOA Solution Stack</td>
</tr>
<tr>
<td>CEP</td>
<td>Complex Event Processing</td>
</tr>
</tbody>
</table>
Acronyms

CLGM  Custom Load Generator Module
ESXi  VMware vSphere Hypervisor 24 5.1 Update 1
PRM   Property Relationship Model
OSGi  Open Services Gateway initiative
OASIS Organization for the Advancement of Structured Information Standards
OpenGroup The Open Group
OMG   Object Management Group
QoS   Quality of Service
ESB   Enterprise Service Bus
S3    SOA Solution Stack
SLA   Service Level Agreement
SOA   Service-Oriented Architecture
MDA   Model-driven Architecture
PBT   Plan and Book Trip
PRIDE Proactive Adaptation Framework
PIM   Platform Independent Model
PSM   Platform Specific Model
RPM   Request per Minute
RA    Reference Architecture
UCS   Unified Computing System
WSCI  Web Service Choreography Interface
VM    Virtual Machine
AOP   Aspect Oriented Programming
SaaS  Software as a Service
IaaS  Infrastructure as a Service
PaaS  Platform as a Service
API   Application Programming Interface
Bibliography


