Faculty of Computer Science, Electronics and Telecommunications

GOAL-DRIVEN ADAPTIVE MONITORING OF DYNAMIC SERVICE ORIENTED SYSTEMS

Ph.D. Dissertation

Marek Psiuk

Supervisor:
Prof. Krzysztof Zieliński

Kraków, 2013
Adaptacyjne monitorowanie dynamicznych systemów zorientowanych na usługi

Rozprawa doktorska

Marek Psiuk

Promotor:
Prof. dr hab. inż. Krzysztof Zieliński

Kraków, 2013
ACKNOWLEDGMENTS

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

I also thank the members of the Distributed Systems Research Group at the Computer Science Department of AGH University of Science and Technology, especially Daniel Żmuda for his valuable remarks and the time spent on fruitful discussions.

Finally, I express a heartfelt gratitude to my wife, Agata, for her patience, understating and support without which I would not be able to successfully complete 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

Acknowlegements i  
List of Tables v  
List of Figures vi  
Listings viii  
List of Algorithms ix  

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

2 Background and Related Work 6  
   2.1 Model of Service Oriented Architecture 7  
      2.1.1 Review of Approaches to SOA Standardization 7  
      2.1.2 Selected Reference Architecture of SOA 9  
   2.2 SOA Environments 11  
   2.3 Adaptability 14  
      2.3.1 Important Adaptability Concepts 14  
      2.3.2 Adaptability Properties 16  
      2.3.3 Control Loop 17  
      2.3.4 Goal-driven Control Loop Management 18  
      2.3.5 Adaptability Challenges 19  
   2.4 Research Related to Adaptive Monitoring 21  
      2.4.1 Anomaly Detection and Diagnosis 21  
      2.4.2 Comprehensive Approaches to Adaptive Monitoring 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.3</td>
<td>Aspect of Goal Orientation</td>
<td>28</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Prior Work Limitations</td>
<td>29</td>
</tr>
<tr>
<td>2.5</td>
<td>Summary</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Concept of Adaptive Monitoring</td>
<td>31</td>
</tr>
<tr>
<td>3.1</td>
<td>Aspects of SOA Dynamics</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>Abstraction of Adaptive Monitoring</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>SOA System Meta-model</td>
<td>38</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Topology Plane</td>
<td>38</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Causality Plane</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>Measurement and Steering Meta-models</td>
<td>44</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Measurement Aspects</td>
<td>44</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Steering Aspects</td>
<td>49</td>
</tr>
<tr>
<td>3.5</td>
<td>Summary</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>Monitoring Process Realization</td>
<td>56</td>
</tr>
<tr>
<td>4.1</td>
<td>Monitoring Goal Strategy</td>
<td>57</td>
</tr>
<tr>
<td>4.2</td>
<td>Control Loop Anatomy</td>
<td>60</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Motivation</td>
<td>60</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Realization</td>
<td>61</td>
</tr>
<tr>
<td>4.3</td>
<td>Control Loop Algorithms</td>
<td>64</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Nominals Identification</td>
<td>64</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Sentinels Selection</td>
<td>66</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Adaptive Drill-down</td>
<td>70</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary</td>
<td>76</td>
</tr>
<tr>
<td>5</td>
<td>Dynamic Adaptive Monitoring Framework</td>
<td>78</td>
</tr>
<tr>
<td>5.1</td>
<td>Framework Requirements</td>
<td>79</td>
</tr>
<tr>
<td>5.2</td>
<td>SOA System Enrichment</td>
<td>80</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Selection of the Instrumentation Approach</td>
<td>80</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Interceptor Socket Concept</td>
<td>81</td>
</tr>
<tr>
<td>5.3</td>
<td>Design of Monitoring Mechanisms</td>
<td>85</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Topology Discovery</td>
<td>85</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Causality Identification</td>
<td>86</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Measurement Acquisition</td>
<td>88</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Core Logic of the Measurement Interceptor</td>
<td>89</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Ranges Identification and Reporting</td>
<td>91</td>
</tr>
<tr>
<td>5.4</td>
<td>Framework Architecture</td>
<td>94</td>
</tr>
<tr>
<td>5.5</td>
<td>Summary</td>
<td>98</td>
</tr>
<tr>
<td>6</td>
<td>Prototype Implementation</td>
<td>99</td>
</tr>
<tr>
<td>6.1</td>
<td>Technology for Prototype Realization</td>
<td>100</td>
</tr>
<tr>
<td>6.1.1</td>
<td>SOA Dynamics in OSGi</td>
<td>100</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Mapping OSGi to SOA System Meta-model</td>
<td>102</td>
</tr>
<tr>
<td>6.2</td>
<td>Implementation Details</td>
<td>103</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Communication Backbone</td>
<td>103</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Realization of Monitored Service Container</td>
<td>104</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Realization of Monitoring Center</td>
<td>106</td>
</tr>
<tr>
<td>6.3</td>
<td>Summary</td>
<td>109</td>
</tr>
<tr>
<td>7</td>
<td>Evaluation</td>
<td>111</td>
</tr>
<tr>
<td>7.1</td>
<td>Evaluation Overview</td>
<td>112</td>
</tr>
<tr>
<td>7.2</td>
<td>Experimental Environment</td>
<td>113</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Application Scenario</td>
<td>113</td>
</tr>
<tr>
<td>7.2.2</td>
<td>System Setup</td>
<td>116</td>
</tr>
<tr>
<td>7.3</td>
<td>Functional Evaluation</td>
<td>119</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Container Enrichment and Discovery of SOA System Topology</td>
<td>119</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Goal-orientation Addressing Aspects of Dynamics</td>
<td>122</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Drill-down upon Single and Multiple Anomalies</td>
<td>130</td>
</tr>
<tr>
<td>7.4</td>
<td>Non-functional Evaluation</td>
<td>133</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Monitoring Overhead</td>
<td>133</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Scalability Aspects</td>
<td>139</td>
</tr>
<tr>
<td>7.5</td>
<td>Summary</td>
<td>145</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions</td>
<td>147</td>
</tr>
<tr>
<td>8.1</td>
<td>Thesis Verification</td>
<td>148</td>
</tr>
<tr>
<td>8.2</td>
<td>Future Work</td>
<td>149</td>
</tr>
<tr>
<td>A</td>
<td>Extended Evaluation Results: Experiment II</td>
<td>151</td>
</tr>
<tr>
<td>B</td>
<td>Extended Evaluation Results: Experiment III</td>
<td>156</td>
</tr>
<tr>
<td>Acronyms</td>
<td></td>
<td>162</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>165</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>BN probabilities, when all measurements are activated</td>
<td>52</td>
</tr>
<tr>
<td>3.2</td>
<td>BN probabilities, when no measurement is active</td>
<td>53</td>
</tr>
<tr>
<td>3.3</td>
<td>BN probabilities, when the mdirect and mcums of service $S_2$ are activated</td>
<td>54</td>
</tr>
<tr>
<td>3.4</td>
<td>BN probabilities, when the mdirect and mcums of service $S_1$ are activated</td>
<td>54</td>
</tr>
<tr>
<td>4.1</td>
<td>BN probabilities, when $SS = {S_1}$</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>BN probabilities, when $SS = {S_1, S_2}$</td>
<td>70</td>
</tr>
<tr>
<td>4.3</td>
<td>BN probabilities, when $SS = {S_1}$ and $Suspicious = {S_2, S_3, S_4}$</td>
<td>75</td>
</tr>
<tr>
<td>4.4</td>
<td>BN probabilities, when $SS = {S_1}$, $DS_{best} = {S_2}$ and $Suspicious = {S_4}$</td>
<td>76</td>
</tr>
<tr>
<td>5.1</td>
<td>Mapping interceptor pattern to the SOA system meta-model</td>
<td>82</td>
</tr>
<tr>
<td>6.1</td>
<td>Mapping of OSGi to topology plane of SOA system meta-model</td>
<td>102</td>
</tr>
<tr>
<td>6.2</td>
<td>Mapping of OSGi to causality plane of SOA system meta-model</td>
<td>103</td>
</tr>
<tr>
<td>7.1</td>
<td>Mapping of the evaluation experiments into the expectations</td>
<td>113</td>
</tr>
<tr>
<td>A.1</td>
<td>Measurements received by $M_{HQ}$ in experiment II</td>
<td>152</td>
</tr>
<tr>
<td>A.2</td>
<td>Changes in measurements received by $M_{HQ}$ in experiment II</td>
<td>153</td>
</tr>
<tr>
<td>A.3</td>
<td>Initial state of BN in experiment II</td>
<td>154</td>
</tr>
<tr>
<td>A.4</td>
<td>State of BN in experiment II after selection of sentinels</td>
<td>155</td>
</tr>
<tr>
<td>B.1</td>
<td>Initial state of BN in experiment III</td>
<td>156</td>
</tr>
<tr>
<td>B.2</td>
<td>Nominal metric ranges in experiment III after Phase I of the control loop</td>
<td>157</td>
</tr>
<tr>
<td>B.3</td>
<td>State of BN in experiment III after selection of sentinels</td>
<td>158</td>
</tr>
<tr>
<td>B.4</td>
<td>State of BN in experiment III after finishing drill-down</td>
<td>160</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Positioning of SOA specifications</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Concept map presenting the essential elements of SOA-RA</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Hierarchy of adaptability properties</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>MAPE-K loop proposed by autonomic computing</td>
<td>18</td>
</tr>
<tr>
<td>2.5</td>
<td>Taxonomy of Self-Adaptation</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>Concept map presenting the main aspects of the SOA dynamics</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>High-level concept of adaptive monitoring</td>
<td>35</td>
</tr>
<tr>
<td>3.3</td>
<td>SOA system meta-model</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>Sample SOA application scenario</td>
<td>39</td>
</tr>
<tr>
<td>3.5</td>
<td>Topology model of the sample SOA application</td>
<td>40</td>
</tr>
<tr>
<td>3.6</td>
<td>Causality model of the sample application: flow invocations of service $S_1$</td>
<td>42</td>
</tr>
<tr>
<td>3.7</td>
<td>Causality model of the sample application: flow invocations of service $S_2$</td>
<td>43</td>
</tr>
<tr>
<td>3.8</td>
<td>Causality model of the sample application: flow invocations of service $S_3$</td>
<td>43</td>
</tr>
<tr>
<td>3.9</td>
<td>Causality model of the sample application: flow invocations of service $S_4$</td>
<td>43</td>
</tr>
<tr>
<td>3.10</td>
<td>Measurement and steering meta-models</td>
<td>44</td>
</tr>
<tr>
<td>3.11</td>
<td>Relations between the measurement compounds and the metric ranges</td>
<td>45</td>
</tr>
<tr>
<td>3.12</td>
<td>Measurement model in the sample application scenario</td>
<td>48</td>
</tr>
<tr>
<td>3.13</td>
<td>Bayesian network created for the sample application scenario</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>Structure of the monitoring goal strategy</td>
<td>58</td>
</tr>
<tr>
<td>4.2</td>
<td>Control loop in the context of monitoring strategy directives</td>
<td>61</td>
</tr>
<tr>
<td>4.3</td>
<td>Measurement model with monitoring restricted to the blue service flow</td>
<td>66</td>
</tr>
<tr>
<td>5.1</td>
<td>Interaction between the interceptor socket and interceptor services</td>
<td>83</td>
</tr>
<tr>
<td>5.2</td>
<td>Interception performed by the i-socket</td>
<td>84</td>
</tr>
<tr>
<td>5.3</td>
<td>Architecture of DAMON framework</td>
<td>95</td>
</tr>
<tr>
<td>7.1</td>
<td>CCP application</td>
<td>114</td>
</tr>
<tr>
<td>7.2</td>
<td>Structure of virtualized environment</td>
<td>117</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3</td>
<td>Logical brokers topology in the virtualized environment</td>
<td>118</td>
</tr>
<tr>
<td>7.4</td>
<td>Modified fragment of the CCP application</td>
<td>125</td>
</tr>
<tr>
<td>7.5</td>
<td>Structure of the CCP measurement model</td>
<td>126</td>
</tr>
<tr>
<td>7.6</td>
<td>Sentinels selected in the monitored fragment of the CCP application</td>
<td>127</td>
</tr>
<tr>
<td>7.7</td>
<td>Sentinels selected in CCP after the bundle deployment</td>
<td>128</td>
</tr>
<tr>
<td>7.8</td>
<td>Adaptive drill-down triggered by a single anomaly</td>
<td>132</td>
</tr>
<tr>
<td>7.9</td>
<td>Adaptive drill-down triggered by a multiple anomalies</td>
<td>132</td>
</tr>
<tr>
<td>7.10</td>
<td>Combined egress throughput of four VMs hosting the CCP application</td>
<td>135</td>
</tr>
<tr>
<td>7.11</td>
<td>Egress throughput influenced by manual selectivity changes</td>
<td>136</td>
</tr>
<tr>
<td>7.12</td>
<td>Throughput influenced by the drill-down caused by single anomaly</td>
<td>137</td>
</tr>
<tr>
<td>7.13</td>
<td>Throughput influenced by the drill-down caused by multiple anomalies</td>
<td>138</td>
</tr>
<tr>
<td>7.14</td>
<td>Number of the sentinels depending on the size of BN</td>
<td>140</td>
</tr>
<tr>
<td>7.15</td>
<td>BN inferencing process in case of a single monitoring goal</td>
<td>141</td>
</tr>
<tr>
<td>7.16</td>
<td>BN inferencing process in case of a decomposed monitoring goal</td>
<td>141</td>
</tr>
<tr>
<td>7.17</td>
<td>Bandwidth consumed by the monitoring process</td>
<td>142</td>
</tr>
<tr>
<td>7.18</td>
<td>DAMON operation including anomaly incidents in single service</td>
<td>143</td>
</tr>
<tr>
<td>7.19</td>
<td>DAMON operation including anomaly incidents in multiple services</td>
<td>143</td>
</tr>
<tr>
<td>7.20</td>
<td>Amount of data sent in the federation during DAMON operation</td>
<td>144</td>
</tr>
<tr>
<td>A.1</td>
<td>Active measurements of CCP application in experiment II</td>
<td>151</td>
</tr>
<tr>
<td>A.2</td>
<td>Changes in measurements of CCP application in experiment II</td>
<td>153</td>
</tr>
<tr>
<td>B.1</td>
<td>BN states in adaptive drill-down triggered by single anomaly</td>
<td>159</td>
</tr>
<tr>
<td>B.2</td>
<td>BN states in adaptive drill-down triggered by multiples anomalies</td>
<td>161</td>
</tr>
</tbody>
</table>
LISTINGS

6.1 Aspect intercepting OSGi BundleContext . . . . . . . . . . . . . . . . . . 105
6.2 Logic of interceptor socket implemented as Java Proxy . . . . . . . . . . 107
6.3 Creation of monitoring subscription’s MsDefinition . . . . . . . . . . . . 108
6.4 Registration of monitoring subscription . . . . . . . . . . . . . . . . . . 108
7.1 Monitoring goal strategy, named $MG_{TD}$ . . . . . . . . . . . . . . . . . . 121
7.2 Monitoring strategies for the CCP application used in experiment II . . 123
7.3 Monitoring goal strategy, named $MG_{DP}$ . . . . . . . . . . . . . . . . . . 130
# List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acquisition of nominal metric ranges</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>Selection of sentinels in the stale period</td>
<td>68</td>
</tr>
<tr>
<td>3</td>
<td>Adaptive drill-down</td>
<td>72</td>
</tr>
<tr>
<td>4</td>
<td>Detection of causality performed by the flow sniffer</td>
<td>88</td>
</tr>
<tr>
<td>5</td>
<td>M-interceptor core logic</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>Identification and reporting the measurement exact range and overhead</td>
<td>93</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

The rapid evolution of software systems has always been driven by the needs of business. To foster innovation, different subsystems were integrated together to automate the organization’s business processes. Since the minimization of costs was the primary priority, at first the subsystems were integrated without appropriate architectural approach resulting in a creation of so-called data silos and a realization of spaghetti integration anti-pattern. The advent of Service Oriented Architecture (SOA) allowed for removing these problems by decomposing the data silos into services which were enclosing disjoint parts of a business logic. The logic of services could be easily reused inside a given subsystem as well as by other subsystems that were integrated together by means of Enterprise Service Bus (ESB) – the integration platform used in the SOA systems.

The paradigm of the service orientation was widely used for creating systems of increasing complexity and scale. Since systems had no longer a monolithic silo-based nature, an efficient management of such systems was identified as an important challenge. In order to enable a feature-rich management, there is a need for the comprehensive monitoring solution, which is capable of providing the information required for performing various management tasks.

In the pre-SOA days, the monitoring was mostly configured in a static manner and could easily cover all system elements. This ensured that all information needed for management was always available. The scale, complexity and growing dynamism of the service oriented systems does not allow for simultaneous monitoring of all system elements without introducing the noticeable monitoring overhead. Therefore, an important challenge arises in the field of the SOA monitoring. There is a need for a solution, which enables on-demand adjustment of the monitoring process in accordance with the demands of system users and the changing conditions of the execution environment. This dissertation addresses the identified need by proposing the concept of the goal-driven adaptive monitoring, capable of performing a dynamic management of the monitoring process.
Chapter 1. Introduction

1.1 Motivation and Thesis Statement

Applying service orientation principles to the existing IT system or building the SOA system from scratch always involves a significant cost of entry. The identification of the business features, their dependencies and decomposing them into a well-designed service inventory [1] is not an easy task. However, thanks to the service composition, such initial investment is returned by lowering the costs of the future system modification. The deployed and always available services from the inventory can be easily recomposed to follow the changing business demands. So definitely, the service composition, performed most often at run-time, introduces a level of dynamism, which has to be addressed by the monitoring solutions. Fortunately, the atomic services in the inventory are rather stable and the service components (containing the logic) are mostly exposing at run-time the same list of the services, which was defined at the design time. These are important assumptions from which a foundation of the monitoring mechanisms being able to follow service composition can be built.

The challenge for the monitoring systems results from the evolution of the initial service orientation being focused on the dynamic composition. The evolution was caused by demands, which have manifested themselves in the real-life scenarios. Three major categories of such demands can be identified. They are as follows:

- **Dynamic changes of user requirements**, caused not only by simple refinements of higher business processes but also by an involvement of new user groups, changing the core assumptions of handling user requests and experiencing significant fluctuations in the system load;

- **Increasing scale of the SOA systems** which implies growing number of features enclosed in different service components, growing service inventory and most importantly increasing scope of transitive causal relations between the services in large and distributed environment;

- **Dynamic changes of available resources**, caused by outage, failures and pursuit for higher revenue, which leads to reallocation of the resources used by services to the places in the system that need the most attention at a given moment.

In order to address these demands, research efforts were directed into the area of the adaptive systems [2–4], which proposed the adaptation mechanisms on different layers of SOA: adaptation of infrastructure [5], adaptation of service components [6], adaptation of composite services [7], or even adaptation of business processes [8]. Other research efforts were focused on increasing dynamics of the SOA infrastructure itself [9, 10] and allowing for more changes at run-time. For example, components are allowed to register entirely new services, which were not present at design time [11, 12]. Such enrichment simply changed the basic assumption present in the core SOA. The services in the
inventory are not stable anymore as each of them can be dynamically changed at runtime on multiple levels in both functional and non-functional contexts.

On top of this, the mentioned increase of the scale imposes additional challenges. The number of elements in the system is so high, that monitoring of each single element is simply impossible – not enough storage area to stock all monitoring data, not enough CPU power to process it and not enough network bandwidth to transfer it through the system. The only possible approach is restricting the monitoring process to a subset of system elements, but then a challenge related to the selection of these elements arises. What if critical failures occur in given part of the system and it is not monitored with enough amount of details to detect it? This issue was not so difficult when assumptions about the service inventory allowed for grasping the critical system elements and ensuring that the monitoring process always covers them. The same approach is not possible when the system dynamics are increased – the monitoring process cannot be defined with the use of references to services from the inventory. The referenced service could be removed in the future or its characteristics could change in a way, which makes it unimportant for the monitoring.

In order to address the challenge of the monitoring large, dynamic service oriented systems, this dissertation proposes a concept of the adaptive monitoring which is managed by means of a declarative goal. The thesis statement is as follows:

*The dynamic service oriented systems can be enriched with the adaptable monitoring mechanisms enabling the goal-oriented management of the monitoring process, which handles the SOA dynamics by adjusting the monitoring selectivity and identifying a root cause of the encountered system anomalies.*

The specification of a goal allows for alleviating uncertainty related to the actual services operating in the system at a given moment by referring to the system’s model which is constantly traced at run-time. The declared properties of the monitoring process, i.e. selectivity and root cause identification, are interpreted in the following way:

**selectivity** – restricting the monitoring to system parts, which are essential according to the specified goal and limiting therefore consumed resources. In order to perform it, there is a need for a continuous tracking of the causal relationships between system elements and identifying places which monitoring is not intrusive but at the same time allows for inferring about a state of other system parts.

**root cause identification** – increasing the scope of the monitoring process in case of detected system anomalies for purpose of finding their initial cause. The adaptation of the monitoring scope takes into account both the system structure and the causal relationships. The logic of the adaptation is grasped with the use of the mentioned declarative goal.
1.2 Scope and Assumptions

In general, the monitoring system can cover wide variety of features, starting from the level of operating system (OS), going through various middleware layers, then proceeding to the services deployed inside the middleware and finally reaching the application composed of available services. The approach to the SOA systems monitoring presented in this dissertation is focused mostly on the level of services but considers a broader context when it comes to the services deployment. The deployment context starts at the level of middleware however, does not include the details of the underlying operating system.

Additionally, the dissertation assumes that the service monitoring can be realized by an interception of the service invocations without the need of tracing the general system configuration and performing time-consuming log file analysis. It is expected that an appropriate selection of the interception logic is capable of sensing both system and business properties related to the service execution.

Finally, the important assumption of the meta-models related to the proposed adaptive monitoring concept is that a service maintains the same non-functional properties without any dependency on the context of execution. For example, when the service contract claims that service logic ensures a specific response time, then such response time is guaranteed for all the invocations of this service even if they are coming from different services or subsystems.

1.3 Thesis Contribution

The main contributions of this dissertation are as follows.

- Critical analysis of the current state of the art related to the subject of the adaptive monitoring. The analysis identifies limitations which are important in the context of the dynamic service oriented systems.

- The abstract concept of the adaptive monitoring, which proposes combination of three different layers, i.e. the SOA system, the measurement and the steering that can be used for the goal-oriented management of the monitoring process. Each layer is described by a separate meta-model, which instance is needed at run-time for describing the current system state, tracking its changes and reasoning about the adjustment of the monitoring scope.

- The structure of the monitoring goal strategy, that can describe the goal in a declarative manner in the context of all three layers of the adaptive monitoring concept.
• The realization of the monitoring process, which execution is controlled with the use of the steering model founded on the theory of Bayesian networks \([\text{BNs}]\). The process supports the dynamic adjustment of monitoring selectivity, identification of system anomalies and finding their initial cause.

• Design of the framework, referred to as \([\text{DAMON}]\) which can be installed in an existing SOA environment to provide it with the features of the adaptive monitoring. The framework covers the monitoring mechanisms and the architectural details needed for combining the adaptive monitoring concept and the monitoring process realization into an implementation-ready specification.

• Prototype implementation of the \([\text{DAMON}]\) framework realized with the use of the OSGi technology. The prototype covers all the designed monitoring mechanisms and implements the sophisticated message oriented communication layer. The communication layer allows mechanisms to communicate with the logic governing the monitoring process.

### 1.4 Dissertation Organization

This dissertation is organized in the following manner. Chapter 2 presents the background aspects related to the dissertation scope. It covers selection of an appropriate SOA standardization, review of the existing SOA environments, positioning of the adaptive monitoring in the context of the adaptability research field and finally identifying limitations of the current approaches to the adaptive monitoring. The content of Chapter 3 firstly discusses the aspects of the SOA dynamics important for the dissertation and then proposes the abstract concept of the adaptive monitoring. Since the concept involves three meta-models, the SOA system, the measurement and the steering, the structure of each meta-model is presented and analyzed in the context of a simple SOA application. The contribution of Chapter 4 is focused on the realization of the monitoring process. The chapter covers the definition of the structure of the monitoring goal strategy and then the construction of the process control loop with the detailed description of each loop phase and the related algorithms. Chapter 5 presents the Dynamic Adaptive Monitoring Framework \([\text{DAMON}]\) with all its architectural details and the design of the monitoring mechanism expected by the previous chapters. Then, Chapter 6 describes the prototype implementation of \([\text{DAMON}]\) realized with the use of the OSGi technology. The implementation includes two main entities of the \([\text{DAMON}]\) architecture, i.e. the monitoring center and the monitored service container. The evaluation of the prototype implementation is presented in the scope of Chapter 7. The evaluation involves a set of five different experiments, which covers both the functional and non-functional dimensions of the \([\text{DAMON}]\) prototype. Finally, Chapter 8 concludes the dissertation contribution and discusses possible directions of the future research.
Chapter 2

BACKGROUND AND RELATED WORK

This dissertation is focused on adaptive monitoring suitable for dynamic service-oriented systems. There are several background aspects that need special attention in this context. First of all, the concept of Service Oriented Architecture itself. In order to ensure a sound contribution of this dissertation on both the conceptual and technical levels, the appropriate model of SOA has to be assumed. Such a model could be selected from a wide variety of approaches to define the SOA available in the literature. Since the scope of this dissertation covers not only abstract modeling and architectural design but also prototype implementation, there is a need for a concrete realization of the assumed SOA model. Another important aspect is related to the research context of adaptive monitoring, which is located in the well-explored domain of autonomous, adaptive and self-adaptive systems. This context should be analyzed to identify the crucial challenges relevant to the scope of this dissertation. Finally, the work directly and indirectly related to the concept of adaptive monitoring should be reviewed to prove the novelty of the proposed contribution. This chapter discusses all indicated background aspects, which altogether provide an important foundation for the dissertation contents presented in the following chapters.

The structure of this chapter is as follows. The first section discusses various SOA standardizations and presents in detail the SOA Reference Architecture (SOA-RA) created by The Open Group [13]. The following section analyzes several SOA environments that are important in the context of prototyping described in Chapter 6. The third section presents a review of the adaptability domain taking into consideration its potential in addressing the complexity of SOA systems and the current challenges related to adaptive monitoring. The forth section focuses strictly on the research related to adaptive monitoring. The content of the forth section is devoted to a critical analysis of recent publications and a discussion of their limitations.
2.1 Model of Service Oriented Architecture

There have been a few initiatives aiming at standardizing different aspects of Service Oriented Architectures. This section reviews them briefly and then selects the most appropriate one for the reference of main \textit{SOA} building blocks important for this dissertation. The selected specification, SOA Reference Architecture (\textit{SOA-RA}) created by The Open Group \cite{13}, is analyzed and its elements relevant to this dissertation are further described.

2.1.1 Review of Approaches to SOA Standardization

Kreger and Estefan \cite{14} provide a survey of SOA open standards and the specifications proposed by organizations such as OASIS Object Management Group (OMG) and The Open Group. The survey includes not only descriptions of the selected standards but also their interrelations and important similarities. The following standards are considered by the survey:

\textbf{The OASIS Reference Model for SOA} \cite{15} – tries to capture the essence of SOA by introducing an abstract conceptual framework and the appropriate vocabulary. The framework proposes a normative reference applicable to diverse SOA implementations in a consistent manner.

\textbf{The Open Group SOA Ontology} \cite{16} – located on the abstraction level similar to the above Reference Model, it focuses on SOA semantics and identifies the important related concepts along with their interrelations. The ontology is expressed with Web Ontology Language (OWL) \cite{17}, which allows SOA tools to automate the processing of semantic information.

\textbf{The OASIS Reference Architecture for SOA} \cite{18} – by presenting three different viewpoints: Service Ecosystem, Realizing SOAs and Owning SOAs, it introduces an abstract reference architecture, which analyzes SOA from the ecosystem/paradigm point of view. The assumed level of abstraction is not enough for a direct implementation of the SOA system.

\textbf{The Open Group SOA Reference Architecture} \cite{13} – presents a blueprint for the enterprise architecture based on the Service Oriented Solution Stack (S3) \cite{19}, which can serve as a template for instantiating SOA during the development or modernization of any IT system. The presented level of details not only allows for understanding the important concepts and applying the appropriate design, but it is also capable of directly supporting the implementation process.

\textbf{The Open Group SOA Governance Framework} \cite{20} – introduces definitions and the appropriate context related to SOA governance covering its relationship with
both business and IT. The assumption of the proposed framework is that it is not intended for direct usage but to be adapted to a given scenario of SOA deployment.

The Open Group Service Integration Maturity Model (OSIMM) [21] – provides an approach for assessing the maturity of a given IT system in the incorporation of service-orientation principles. The approach involves seven maturity levels and seven dimensions useful not only to identify the current status but also to plan future modernizations towards service-orientation.

The OMG SOA Modeling Language (OMG SoaML) [22] – defines extensions for Unified Modelling Language (UML) [23], which can be used by IT architects to incorporate the SOA design into the existing enterprise architectures. The introduced language allows for defining platform-independent SOA models (PIM) that can be further converted into platform-specific models (PSM).

The aforementioned survey [14] analyzes the above specifications taking into consideration two criteria: the level of abstraction and the completeness of coverage. The outcome of this analysis, presented in Figure 2.1, is used for selecting the specification most suitable for this dissertation. OSIMM and SoaML are not taken into account because they consider the aspects of maturity assessment and modeling, which are not the major focus of this dissertation.

The OASIS Reference Model for SOA is maintained on the highest abstraction level, while The Open Group SOA Ontology supplementing the Reference Model is just behind it. The OASIS Reference Architecture presents a more architecture-oriented perspective, therefore it is less abstract than the Reference Model and the Ontology. The OASIS RA is not sufficient to directly support the implementation process, which makes it less concrete than the reference architecture proposed by The Open Group. The Open Group

Fig. 2.1: Positioning of SOA specifications
Chapter 2. Background and Related Work

SOA Governance Framework is the most concrete specification, however it focuses only on one of many SOA aspects, namely governance. When it comes to the completeness of coverage, the SOA RA of The Open Group turns out to be the most comprehensive proposition, which in the survey [14] is referred to as an end-to-end technical reference architecture covering all IT aspects of the SOA solution.

Since this dissertation aims not only at proposing an adaptive monitoring solution of the SOA system on the conceptual level but also at designing and implementing a working prototype, it was decided to choose the most complete standard with the appropriately low abstraction level, i.e. The Open Group SOA RA (hereafter referred to as SOA-RA). It is described in more detail in the following section.

2.1.2 Selected Reference Architecture of SOA

Figure 2.2 contains a concept map [24] which presents the essential elements of SOA-RA and their relationships. Although S3, on which SOA-RA is based, contains five horizontal and four vertical layers, the presented figure grasps only five of them as it provides a sufficient context for the work presented in this dissertation. The remaining S4 layers are the following: horizontal consumer layer – introduces various modules of interaction with the consumer; vertical layers: integration – by routing, mediating and transporting of service requests it integrates the horizontal layers present in the concept map; quality of service – addresses the aspects of QoS in each horizontal layer; information and policy – covers the key data and information-related issues involved in business intelligence [25].

Fig. 2.2: Concept map presenting the essential elements of SOA-RA
Chapter 2. Background and Related Work

In the concept map, each element is assigned to its S3 layer. The relationships between the elements are expressed by means of a regular concept map notation enriched with simplified constructs of UML: association, aggregation and generalization. The focal point of the concept map is the service which offers certain features that the business performs to achieve a business outcome or a milestone. The functionality of the service is offered by registering it into the service registry from which other services can discover it during run-time. The service is exposed by the service component which is the actual implementation of one or more services. The role of the service components layer is to ensure the proper alignment of the implementation with the service contract, while the role of the service layer is to act as the translation between the consumer and the service component. The service registry is responsible for advertising and discovery of the available services and for run-time binding of services. The registry is located in the governance layer which ensures that the elements of SOA solutions adhere to the defined policies, standards and guidelines resulting from some assumed objectives, strategies and regulations.

Services can be orchestrated by business processes which capture the activities needed to accomplish certain business goals. In today’s solutions, business processes play a central role in bridging the gap between the business and IT which is the main purpose of the business process layer. Preparing for the deployment of all mentioned elements (service component, service and business process) involves packaging them into a deployment unit. The deployment unit represents an executable application that can be deployed as a single package (e.g. exe, war, ear) in the target solution environment. There can be different types of deployment units specific for a given layer – the concept map grasps two of them: the service unit and the business unit. After the deployment unit is deployed and started, it is referred to as solution component. The solution environment represents a run-time infrastructure component of any S3 layer. In other words, it is a run-time instance of a given layer’s building block which provides the infrastructure needed to run the artifacts specific to that layer.

The concept map presents two solution environments: the service container and the process engine belonging to the service layer and the business processes layer respectively. Some solution environments are capable of aggregating other environments – e.g. a Java Enterprise Edition (JEE) application server (service components layer) can enclose a Business Process Execution Language (BPEL) process engine (business process layer) and a container for the Web Service (WS) (services layer). Both solution environments and solution components are executed on the solution platform, which provides them with a fundamental run-time. An example of the solution platform could be Java Virtual Machine (JVM) or .NET Framework. It is not grasped in the concept map, but the solution platform is always hosted by the operational run-time hosting environment which is an abstraction of operation systems. Of course, such abstraction can always be virtualized with the use of the infrastructure virtualization. All described elements, marked in red in Figure 2.2, belong to the operational systems layer, which
conceptually can be thought of as the run-time or the deployment-time of the whole SOA solution.

An important concept that needs some additional explanation is the service container. This building block acts as a container by providing the environment with the ability to invoke, run and manage the services (e.g. controlling run-time invocations, controlling the service’s life cycle). The primary responsibility of the service container is to encapsulate the code that implements low-level details of communication with the service. Additional responsibilities are as follows: encapsulating the components that implement the service, state management, binding of service invocations to other layers (in particular, the integration layer and the business process layer), clustering of services, and their distribution to different consumers.

2.2 SOA Environments

In order to implement a SOA system, there is a need for an environment providing the technology stack capable of ensuring the principles of service-orientation. According to the research published by Gartner [30–32] and Forrester [33–35], currently there are many SOA environments provided both as commercial products and free-of-charge open-source projects. The majority of the proposed environments is offered in the form of Enterprise Service Bus (ESB) or some more advanced framework founded on the ESB infrastructure. According to David Chappel [36], the definition of ESB is as follows:

An ESB is a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity.

Such ESB characteristic makes it a natural match for the SOA system backbone. This section performs a review of the most promising propositions on the SOA environments landscape covering not only the actual environment instances but also their more abstract specifications.

Java Business Integration (JBI) [37] is a specification created under Java Community Process (JCP) as Java Specification Request 208 (JSR 208) [38]. The specification proposes a standardized Java-based environment for integrating service-oriented systems in accordance with the ESB approach. A single node of the JBI environment is the container which consists of a Normalized Message Router (NMR) connecting multiple components differentiated into Service Engines (SE) and Binding Components (BC). Unfortunately,

1 “Service Engine (SE)s provide business logic and transformation services to other components and consume such services. SEs can integrate Java-based applications (and other resources) or applications with available Java APIs.” [38]
2 “BCs provide connectivity to services external to a JBI. This can involve communications protocols, or services provided by Enterprise Information Systems (EIS) resources. BCs can integrate applications (and other resources) that use remote access technology that is not available directly in Java.” [38]
JBI did not achieve wide-spread industry adoption due to several reasons: too tight coupling with XML on the API level; focusing more on the container vendor perspective than the application developer perspective; language support limited to Java \cite{39, 40}. In order to solve the problems of JBI, the initiative of JSR 312 \cite{41} aiming at creation of JBI 2.0 specification was undertaken. Unfortunately, the initiative was abandoned in its early stages.

Service Component Architecture (SCA) is another specification (or rather a set of several specifications) which proposes a standardized model for creating service-oriented architectures \cite{42}. The high-level SCA description is as follows: SCA is based on the idea that business function is provided as a series of services, which are assembled together to create solutions that serve a particular business need ... SCA provides a model both for the composition of services and for the creation of service components, including the reuse of existing application function within SCA compositions.” \cite{43}. Since SCA is not limited to a single programming language, it facilitates not only the aspect of enterprise integration but also the functionality of other SOA layers and it has received strong vendor support (e.g. IBM, SAP, BEA). Its industry adoption exceeds the achievements of JBI \cite{44}.

As described in the author’s article \cite{45}, OSGi technology \cite{46} was initially aimed at a specific range of systems such as mobile computing or automotive electronics. However, recent extensions of this technology, standardized by the following specifications: OSGi Service Platform \cite{47}, OSGi Enterprise Specification \cite{48}, enable leveraging it in a broader variety of environments such as web applications and enterprise systems. The basic goal of OSGi is creating a dynamic component-oriented Java platform for applications developed in accordance with service-oriented design principles \cite{1}. The OSGi framework (referred to also as the OSGi container in this dissertation) provides an execution environment for components called bundles. Bundles expose their functionality as services according to a dynamic publish-bind-find model \cite{49}, which allows for adaptive selection of consumed services. Furthermore, the OSGi container provides the bundles with life cycle capabilities so that there is no need to shut down the entire JVM when a particular bundle is deployed, undeployed or modified during run-time.

Oracle SOA Suite is a comprehensive, hot-plugable software for building, deploying and managing SOA systems \cite{50, 51}. The most important benefits offered by this commercial product include consistent tooling, a single deployment and management model, end-to-end security and unified metadata management. Oracle SOA Suite consists of feature-rich components that provide capabilities related to different layers of SOA and different phases of the SOA system life cycle. The most important components are the following: Oracle BPEL Process Manager, Oracle Web Services Manager, Oracle Business Rules, Oracle Business Activity Monitoring, Oracle Enterprise Service Bus and Oracle JDeveloper. The whole product with all its components is a powerful yet quite complex solution.
JBoss Enterprise SOA Platform \[52\], provided by Red Hat, is a standards-based solution built on the foundation of JBoss Enterprise Application Platform (using OSGi technology) and JBoss ESB \[53\]. The Application Platform delivers the essential infrastructure such as Java Message Service (JMS), clustering, web services and allows for the deployment of services. JBoss ESB provides the foundation of integration functionality such as message routing, event listening, service registry and data transformation. Two important components of the platform are orchestration and the rules engine. The first one covers BPEL and workflow capabilities offered by jBPM \[54\], while the latter provides advanced rule services, business rules execution and content-based routing. Additionally, the platform itself is provided with JBoss Developer Studio, which offers an Integrated Development Environment (IDE) for developing, testing and deploying services, rules, business processes and whole SOA applications.

Fuse ESB \[55\] is an open-source, lightweight integration solution providing the essential ESB functionality. The solution is built on the basis of the Apache ServiceMix project \[56\] and maintains strong specifications compliance. The system-level layer is realized with the use of the OSGi technology and higher layers are, among others, capable of supporting JBI. Fuse ESB aggregates several smaller opensource frameworks such as Apache CXF, Apache Camel and Apache ActiveMQ. It allows for achieving a good overall variety of features without introducing too much general complexity.

There are a few more commercial SOA environments that are worth mentioning: webMethods ESB, Tibco ActiveMatrix Service Bus and Sonic ESB. In the Forrester ESB Report \[33\], all three products are categorized as leaders, which means that they have a strong strategy and an attractive offer. The report makes the following remarks about each of these products. webMethods ESB has the largest number of ESB implementations of all vendors included in the report and the customers interviewed by Forrester were pleased with the product’s functionality. The Tibco solution received many strong scores working well both in a standalone environment and as one of the key components of Tibco’s comprehensive integration solution – BusinessWorks. Sonic ESB has a long history of success stories (first release took place in mid-2000s), which results in strong scores across all evaluated areas. In the context of opensource solutions, Forrester gives the leader category to Fuse ESB and WSO2. Since it was the first evaluation of WSO2, the leader category was a significant achievement. According to the report, WSO2 provides a strong, opensource ESB. Even though its customer base is not very extensive (not comparable with the commercial competition) it does not prevent WSO2 from achieving high scores.

All the reviewed SOA environments provide certain monitoring capabilities, however none of them supports goal-oriented adaptive monitoring targeted by this dissertation. Therefore, instead of leveraging the environments in the context of conceptual or architectural work presented in Chapters 3, 4 and 5, they are considered during the selection...
Chapter 2. Background and Related Work

of technology used for the prototype implementation of the dissertation’s contribution presented in Chapter 6.

The discussed SOA environments allow for the construction of large and complex systems, the management of which is not trivial. Their scale and complexity make manual management, performed by human operators, very tedious, time-consuming and error-prone. Therefore, research efforts have been directed into the field of adaptability. The core concept of this field is self-management and it allows the complex system to manage itself in accordance with some high-level policy. A wide perspective on the adaptability research field which identifies the elements most closely related to the subject of adaptive monitoring is presented in the next section.

2.3 Adaptability

Adaptation of the monitoring process is positioned in a broader context of research focusing more comprehensively on the aspects of adaptability. This section presents this broader context and analyzes the current research challenges in the field of adaptive monitoring.

2.3.1 Important Adaptability Concepts

There are several concepts present in the literature which are directly related to adaptability. The most important ones are the following: autonomic computing \cite{57,59}, adaptive software \cite{60,62} and self-adaptive software \cite{63,65}. They are described in the following paragraphs.

Autonomic computing was proposed by Kephart and Chess \cite{57} as a solution for the increasing complexity of multi-layered software systems. As a result of the diversity related to architectural, networking and technological aspects as well as the increasing scale of software systems, manual management of such systems is very difficult and error-prone. The idea behind autonomic computing is that software systems are provided with the capability of self-management – they can manage themselves in accordance with high-level goals defined by the administrators. A self-manageable system has to ensure the following four properties:

**self-configuration** – ability to apply dynamic and automatic reconfiguration by installing, updating, and recomposing software building blocks;

**self-optimization** – ability to manage resource allocation, performance and tuning parameters in accordance with the requirements of the users and the current request load;

**self-healing** – ability to discover, analyze and react to potential errors and disruptions, which involves self-diagnosing \cite{66} and self-repairing \cite{67};

14
**self-protection** – ability to detect security-related attacks and improper usage, execute mitigating actions and report the related information to the appropriate entities.

According to McKinley et al. [60], the emergence of autonomic computing resulted in a dramatic increase of interest in adaptive systems. Software adaptation can be defined as an adjustment of parameters or a modification of the system structure performed in response to situations encountered in the execution environment [60]. The adjustment of parameters is referred to as *parameter adaptation*, while the modification related to the system structure, realized by exchanging some system components with others, is referred to as *compositional adaptation*. A similar approach to adaptive system definition was proposed by Berry et al. [61]. These authors have introduced a definition of Dynamic Adaptive System (DAS), which is as follows: ”DAS is a computer-based system (CBS) that is capable of recognizing that the environment with which it shares an interface has changed and that is capable of changing its behavior to adapt to the changing conditions”.

The concept of self-adaptive software is very much related to adaptive systems. Oreizy et al. [63] define self-adaptive software in the following way: ”Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation”. As summarized in the survey of Salehie and Tahvildari [64], an open-loop system becomes self-adaptive when it is converted to a closed-loop system using feedback. The feedback can come from two sources: *self* – the software system itself with all its layers and related components and *context* – all aspects of the surrounding environment affecting the system’s properties and its behavior.

All three briefly discussed concepts: autonomic computing, adaptive software and self-adaptive software are very similar and there are publications that use some of them interchangeably, e.g. Huebscher and McCann [69]. Possible distinctions between autonomic computing and self-adaptive software are discussed by Salehie and Tahvildari [64]. Firstly, the perspective of self-adaptivity is narrower than the vision of autonomic computing. This could justify classifying self-adaptive systems as a subgroup of the autonomic ones. Secondly, when analyzing a software system decomposed to layers, self-adaptivity is concerned more with the upper layers, i.e. application and middleware, while autonomic computing takes into account all layers to completely remove the need of human intervention in the managed system.

The difference between adaptivity and self-adaptivity is even more subtle. Yang et al. [70] introduce the term *adapt-ready*, which is used for a program enriched with additional features that make it capable of performing adaptation, i.e. make it an adaptive program. In the context of adaptive software, both presented definitions and available publications [60,62,70,71] focus more on the capabilities that allow for adaptation than on the adaptation process itself. In turn, self-adaptivity is about leveraging the
capabilities of adaptive systems to perform the closed-loop adaptation process, which takes into account the feedback from self and context entities. Although the term self-adaptivity is not explicitly used in this dissertation, the presented work is focused both on adaptability capabilities as well as the adaptation process itself.

### 2.3.2 Adaptability Properties

In the context of adaptability, besides the properties of autonomic computing discussed in the previous section, there are other properties proposed in the literature, the so called self-* properties. Figure 2.3 presents the hierarchy of adaptability properties proposed by Salehie and Tahvildari [64], which is discussed in this section. The hierarchy divides properties into three distinct levels: general, major and primitive. The description of each level is as follows:

#### General level

This level focuses on the global properties of the adaptive software. It contains two property groups, namely self-adaptiveness and self-organizing. The group of self-adaptiveness aggregates the following properties: self-managing, self-governing, self-maintenance (proposed in the context of autonomic computing by Kephart and Chess [57]), self-control and self-evaluating. The second group, self-organizing, is focused on the aspect of physical and logical distribution. The self-organizing property is needed in distributed systems that consist of multiple dispersed components communicating with each other without a holistic view of the whole system. The property of self-organizing is mostly applied in the bottom-up approach, while self-adaptiveness, related more to the adaptation process driven by high-level policies, is rather performed in the top-down manner.

#### Major level

This level is related to the properties that were assigned by Kephart and Chess [57] to the concept of self-management, i.e. self-configuring, self-healing, self-optimizing and self-protecting. When a software system possesses these properties, then the tasks performed manually by administrators to configure, heal, optimize and protect the IT infrastructure can be fully automated [59]. It enables the orchestration of such automated tasks in accordance with some assumed goals into a fully self-manageable system capable of functioning without human intervention.

#### Primitive level

This level is concerned with the properties that provide the prerequisites for the previous level. To ensure self-manageability, the system has to be aware of its internal state (the self-awareness property) and the current external operating conditions (the context-awareness property) [72] [73]. Self-awareness is based on the property of self-monitoring which reflects what is monitored [64]. Additionally, self-monitoring realization is expected to provide information about both consumed and available resources, system components, their performance characteristics and related statuses.
The domain of adaptive monitoring focuses on changing the scope of the monitoring process, i.e. the monitoring subject, according to a particular expectation, e.g. minimization of the monitoring overhead, avoiding the acquisition of unnecessary information or adjusting monitoring selectivity in the face of encountered problems. The analysis of the presented adaptability properties hierarchy shows that the domain of adaptive monitoring is most closely related to self-awareness and its inherent self-monitoring, which is responsible for deciding what system fragment should be monitored.

### 2.3.3 Control Loop

The adaptation process is managed by an entity referred to in the literature as the *control loop* or the *adaptation loop*. There are several approaches to define the control loop, but all of them share certain similarities. Oreizy et al. [63] refer to the control loop as *adaptation management* and indicate that it consists of the following actions: collecting observations, evaluating observations, planning changes, deploying change descriptions and enacting changes. Dobson et al. [74] use the term *autonomic control loop* and state that it consists of collecting, analyzing, deciding and acting. The Architectural Blueprint [59] proposed by IBM introduces a loop divided into phases such as monitoring, analyzing, planning and executing, with the assumption that all phases share some common knowledge. The loop is presented in Figure 2.4 and is referred to as the Monitor Analyze Plan Execute - Knowledge (MAPE-K) loop. The MAPE-K loop assumes that the subject of adaptation is some abstract managed resource that can be monitored and influenced with the use of sensors and effectors respectively. Sensors are used to collect data from the managed resource, while effectors allow for applying changes to the resource according to the loop decisions. The approach of Salehie and Tahvildari [64] is similar to the ones proposed by Dobson et al. [74] and IBM [59]. These authors identify...
the following phases of the adaptation loop: monitoring, detecting, deciding, acting and they also use the concepts of sensors and effectors interacting with the adapted system.

The approach assumed in this dissertation is most closely related to the perspective outlined by IBM’s [MAPE-K] loop. The managed resource is simply a monitoring process that is additionally monitored, analyzed, planned and appropriately executed in order to transform it into a truly adaptive entity.

![MAPE-K loop](image)

**Fig. 2.4:** MAPE-K loop proposed by autonomic computing [59]

### 2.3.4 Goal-driven Control Loop Management

The concept of control loop assumes that the loop execution is governed by some high-level policies or strategies expressing the goals of adaptation. The Architectural Blueprint [59] of IBM proposes the following definition of the policy:

*A policy consists of a set of behavioral constraints or preferences that influence the decisions made by an autonomic manager.*

The assumption is that the adaptation policy contains some knowledge allowing for determining when and what changes should be applied to the adapted system. The policy representation has to be standardized to ensure its common understanding. It is needed in order to share the policy definitions among different components of the distributed system and to ensure that component execution complies with the expectations of the policies. In the context of self-adaptive systems, the policy-based approach has been addressed both by the research contributions [75, 76] as well as more practical frameworks, e.g. the StarMX framework for Java enterprise environments [77].

There are three types of policies present in the literature [78]:

- **action policy** – specifies the actions that should be taken in a given state of the system.
  - The action is often expressed in the form of event-condition-action (ECA) rules.
Chapter 2. Background and Related Work

There is no explicit objective in the action policy. The resulting system state simply depends on the execution of each matching ECA rule. In order to create the action policy, a thorough knowledge about the system structure and behavior is needed.

**utility policy** – is based on a function that for a given system state assigns a single real value to each possible resulting state. The function directly encodes the trade-offs between different states, therefore any conflicts that may occur in the action policy are avoided here. The function is used for computing the optimal action in the current system state by maximizing the utility function.

**goal policy** – specifies the desired system state or the criteria for a group of states. This type of policy is focused on what we want and does not consider the means for achieving it. Thus, it is more abstract and, at the same time, more flexible. The important requirement of the goal policy is the availability of sophisticated system models. These models are used in the planning phase of the control loop for choosing the most appropriate action in the context of the current system state and the defined goal.

This dissertation assumes the usage of the goal policy that is leveraged for expressing the scope of the system which should be subjected to monitoring and the guidelines for performing adaptation of the monitoring process.

### 2.3.5 Adaptability Challenges

This section discusses the challenges identified by Salehi and Tahvildari [64] in the context of their taxonomy of self-adaptation. The taxonomy is presented in Figure 2.5. The concept of self-adaptation is decomposed into the following four facets: (i) **object to adapt** – is concerned with the adaptation subject and location; (ii) **realization issues** – takes into account the means by which adaptation is applied; (iii) **temporal characteristics** – deals with the issue regarding when the managed resource is changed; (iv) **interaction concerns** – covers the possible interactions with other self-adaptive systems.

Although each approach to adaptation could be classified with the use of all proposed facets, taking into account the scope of this dissertation only the sub-facet of *continuous / adaptive monitoring* is analyzed. This sub-facet is defined in the following way: ”Continuous/Adaptive Monitoring sub-facet captures whether the monitoring process (and consequently sensing) is continually collecting and processing data vs. being adaptive in the sense that it monitors a few selected features, and in the case of finding an anomaly, aims at collecting more data. This decision affects the cost of the monitoring and detection time.” [64]. Such definition is close to the interpretation of adaptive monitoring assumed in this dissertation.
The work of Salehie and Tahvildari [64] reviews 16 different research projects, using as criteria all facets and sub-facets of the discussed taxonomy. In the continuous / adaptive monitoring sub-facet, 14 projects were identified as having regular continuous monitoring and only two projects, Rainbow [79] and KX [80, 81] were discovered as the ones facilitating semi-adaptive monitoring. This proves that there is much space for improving the current research state of the adaptive monitoring domain. The formulated conclusion is reflected in the challenges proposed by the authors [64]. The proposed research challenges are classified into the following categories: self-* properties, adaptation processes, engineering issues and interaction. The category of adaptation processes contains some challenges, which are particularly relevant to the contents of this dissertation. These selected challenges are the following:

**Monitoring challenge** – One of the important challenges related to monitoring is taking into account the cost of sensing. It can be common that sensors collect information that is not needed for the adaptation process. In such cases they only cause unnecessary resource consumption. However, when the system deviates from a particular state defined as "normal", it may be needed to increase the amount of the gathered data (increase self-awareness), especially in the system surroundings in which the deviation was detected. Oreizy et al. [63] claim that the formulated issue is especially important in self-adaptive software. The planning phase of the...
control loop should try to select only these sensors that are relevant to the current state of the adaptation process.

**Detecting challenge** – Another challenge, closely related to the previous one, is concerned with proposing an approach which ensures that the difference between "normal" and "abnormal" system operation can be easily detected. There are various propositions based on both static and dynamic system analysis, however they are neither complete nor sufficient. Since differentiating between "normal" and "abnormal" system states should be an input to the adaptive monitoring process, a sound solution for this challenge is a prerequisite for addressing the challenge of monitoring.

This dissertation focuses on proposing and then validating a solution for both discussed challenges.

### 2.4 Research Related to Adaptive Monitoring

This section performs a critical review of the approaches that are closely related to the main scope of this dissertation. The presented dissertation aims at proposing adaptive monitoring which provides not only a goal-based monitoring reconfiguration but also a diagnosis of a potential problem, i.e. the identification of its root cause. Thus, the first subsection is devoted to a review of works related to anomaly detection and diagnosis, the second subsection focuses strictly on comprehensive solutions providing all compounds required by adaptive monitoring solutions and the third subsection highlights the aspect of goal orientation in the works related to adaptive monitoring. The last subsection summarizes the limitations of the discussed approaches.

#### 2.4.1 Anomaly Detection and Diagnosis

Magalhaes and Silva [82] propose an approach to detecting performance anomalies in web-based applications with the use of statistical techniques similar to the ones used by Munawar et al. [83–85]. The authors assume that a change of the performance characteristic can have at least one of the following causes: (i) a workload variation, meaning that the system is not sufficiently scalable; (ii) some internal problem that needs to be analyzed and recovered by some additional actions and (iii) an update/upgrade in the application, system or configuration. The paper discusses an implementation that supports situations involving the first cause. The proposed solution realizes instrumentation with the use of the AspectJ Aspect Oriented Programming (AOP) framework for Java [86] and some additional command line tools. The AspectJ aspects are added to the application server during its start-up by means of Load-Time Weaving (LTW) [87]. The whole instrumentation covers multiple monitoring parameters such as CPU, amount of
available memory, number of running threads, numbers of bytes sent/received to a disc or a network. The implementation leverages the techniques of the Pearson correlation coefficient and the Dynamic-Time Warping (DTW) algorithm, which are used in two core components: the Performance Analyzer and the Anomaly Detector. The Performance Analyzer checks the Pearson correlation and the DTW distance between the number of concurrent requests and the response time. When an anomaly happens and these two parameters are correlated, the Anomaly Detector carries out an additional Pearson correlation between the number of concurrent users per a period of time and the collected monitoring parameters. This allows to identify the parameter most likely related to the anomaly identified by the Performance Analyzer.

Yue Zhang et al. [88–90] propose a novel accountability middleware, referred to as Llama, which supports dependable SOA monitoring, diagnosis and reconfiguration. The middleware implements its functionality by mapping a SOA business process to a Bayesian network. The business process is modeled by Directed acyclic graph (DAG), which makes the mapping relatively simple. After mapping is finished, the output of each service is a node of the BN which is influenced by the service operation (another node) and the outputs of all consumed services. In the exemplified scenarios, each BN node can have two states: (i) the node functions correctly and meets its Service Level Agreement (SLA); (ii) some failure happened and the node does not function correctly. It is required that the appropriate prior node probabilities and conditional probability tables (CPTs) are provided to the BN either manually by domain experts or automatically on the basis of some historical monitoring data. The created BN model allows for achieving two important features relevant to the concept of adaptive monitoring: diagnosing the root cause of a failure (both single and multiple failure points are considered) and selecting a subset of services for monitoring to minimize the monitoring overhead and the uncertainty about the business process. The diagnosis is performed by iteratively selecting of the most probable failure cause and investigating its log files. Iterations are performed until the threshold representing the probability failure is reached. Services selection (referred to as evidence channels selection) is based on the concept of the Shannon entropy [91] and is performed by mapping the selection problem to three variations of facility locations problems: k-median, set covering and uncapacitated facility location (UFL). The proposed Llama middleware is realized as an extension of ESB (referred to as ASB – accountability service bus). Its prototype implementation was based on Mule ESB.

The used instrumentation assumes the interception of service invocations, which in case of Mule ESB were achieved by means of its interception framework.

The discussed Llama middleware proposed by Yue Zhang et al. has both the appropriate architecture and powerful inferencing capabilities of BN, leveraged for diagnosis and services selection. However, from the perspective of SOA-oriented adaptive monitoring several issues can be identified. Firstly, the assumed model of the business process – DAG – does not take into account different service communication patterns covering both

synchronous and asynchronous invocations. Secondly, the BN model assumes only two node states. It could impose difficulties when for example a small performance deviation in several services occurs. It is not clear whether such a small deviation should be represented by a service failure or not. Additionally, the specification of the nodes prior probabilities and CPTs could be quite cumbersome and, in the case of dynamic system changes, rather problematic to maintain over time. Thirdly, it is assumed that the model of the business process is known (e.g. it was extracted during service planning). In dynamic systems, when the causal relations between service change rather often, such assumption is not very realistic.

Agarwal et al. [92] propose an SLA-based rapid root cause identification achieved on the basis of dependency graphs and dynamic run-time performance characteristics. The identification is realized by comparing the current performance (measured as average response time) of all components comprised by an end-to-end transaction with thresholds established earlier in an automated way. Although the contribution does not cover a complete solution for adaptive monitoring, the proposed approach could be used as a part of the adaptive monitoring control loop. Authors assume that a transaction can be in two states. When, as a whole, the transaction complies with the SLA its state is good. Otherwise the state is referred to as bad. The thresholds are established by classifying the measured response times of the component as being bad or good. The measurement of a given component is classified as bad when any component’s parent transaction is in the bad state. Otherwise the measurement is classified as good. All good measurements are averaged to create the good behavior model, which serves as the component’s threshold. A respective model is created for bad measurements. When an SLA of some transaction is violated, then root cause identification is performed by ranking all components comprised by the transaction tree in terms of the difference between their good and bad behavior models. The created ranking is then appropriately processed with the assumption that a large difference in a given component is caused by large differences in its children. Unfortunately, such assumption is true only in the cases of single fault and synchronous invocations, which does not make the approach directly usable in the context of SOA systems.

Rui Zhang and Alan Bivens [93] propose an interesting approach to performance problem localization in service-oriented systems. The authors use BN to model the relation between the end-to-end response time of the composite service and the response times of its atomic services. For this purpose, the concept of Response Time Bayesian network (RTBN) is introduced. RTBN consists of a set of variables representing the response time of atomic services connected to one variable representing the end-to-end response time. The Conditional Probability Table (CPT) of the end-to-end response time variable is defined using a deterministic function directly obtained from the workflow information. In another publication [94], the authors state that the variables used in RTBN can be either continuous or discrete depending on the requirements of the application. The authors also mention that RTBN can be potentially mapped to metrics other than the
response time. The proposed RTBN concept solves the issue of CPT definition (most of the time learned from historical data or provided manually), however the simplified two-level BN model, similarly to the model of Rish et al. [95], does not allow to grasp hierarchical dependencies between complex services involving several composition levels.

The work related to the scope of this dissertation covers also projects aiming at some aspects of self-adaptation that are partially related to adaptive monitoring. Two such projects were mentioned in Section 2.3.5, namely Rainbow [79] and KX [80, 81].

Rainbow is a framework which leverages software architectures and reusable infrastructure to support self-adaptation of software systems. The goal is achieved by proposing the properties of interest and the mechanisms supporting dynamic modification, which can be tailored to the specific system. These mechanisms allow the self-adaptation manager to select the system aspects that will be modeled, monitored and processed by the adaptation logic. The mechanisms providing measurements are implemented as probes that are capable of observing various system states. The probes can provide monitoring data either in the push model or in the pull model allowing for querying the probes on-demand. The functionality of probes is complemented by resource discovery, which, similarly to probes, provides information in the pull model. The information gathered by the probes is aggregated by components referred to as gauges, which use the aggregated data to update the properties of the adaptation model. The aspects of adaptive monitoring are visible in the design of gauge and probe APIs, which not only supports on-demand data retrieval but also lets developers to plug in different gauge and probe implementations according to the adaptation loop requirements.

The name of the KX project is an abbreviation of Kinesthetics eXtreme, an implementation of external monitoring infrastructure, which can be easily integrated into distributed legacy systems in order to provide them with autonomic properties of self-management (cf Section 2.3.1). KX also uses the concepts of probes and gauges, the functionality of which is similar to the one proposed by Rainbow. The difference lies in the general architecture. KX is a lightweight, decentralized collection of loosely-coupled components communicating by means of a publish-subscribe event system. One of the KX components, the Event Distiller, is particularly interesting in the context of adaptive monitoring. The Event Distiller is responsible for detecting causality among the events coming from gauges. The detection is performed with the use of nondeterministic state engines that apply a time-based pattern matching to event sequences. The detected causality information allows for realization of root cause identification in the case of non-trivial cascading problems.

There are several other research initiatives that are worth mentioning in the context of anomaly detection and diagnosis. Chen et al. [96] propose the Pinpoint framework allowing for problem determination in large, dynamic systems. The framework traces end-to-end requests and performs the appropriate clustering analysis. The approach ensures that no a priori application-level knowledge is needed to use the framework. Sambasivan
et al. [97] present the Spectroscope tool, which implements algorithms for identifying and ranking structural flow changes and performance deviations in a distributed system. Instead of focusing on anomaly identification, the Spectroscope compares whole groups of requests and classifies the discovered differences into the response-time and structural mutations. Attariyan et al. [98] present another interesting tool called X-ray. The functioning of the X-ray is divided into two phases. Firstly, the tool records the execution of the production system and acquires performance data. Secondly, in an off-line manner, the X-ray replays the recorded execution and performs a dynamic information flow analysis in order to identify the root cause of performance problems.

### 2.4.2 Comprehensive Approaches to Adaptive Monitoring

This section reviews comprehensive research approaches that involve the complete set of elements combined into a working adaptive monitoring solution.

In the contributions of Munawar and Ward [83] and Munawar et al. [84], extended further in Munawar’s PhD Thesis [85], an interesting approach to adaptive monitoring for component-based systems is proposed. The approach uses statistical techniques to identify relationships in the monitoring data which are then used to check the system state and, in case of failure, to diagnose its cause. No knowledge about the underlying system is assumed, therefore the system model is created with the use of linear regression applied to pairs of metrics identified in the system. The model contains only the information about the monitoring metrics and it does not involve any dependencies between the system components. The adaptation logic of the monitoring process is divided into the following phases: (i) creation of system model; (ii) metric selection – both manual and automatic selection of the most appropriate metric set are considered (taking into consideration the amount of information and the overhead); (iii) minimal monitoring – selected metrics are monitored and the gathered data is confronted with the established system model; (iv) detailed monitoring – it is triggered when the data from minimal monitoring differs too much from the model, which results in the activation of all system metrics for the purpose of problem diagnosis.

The high-level adaptation scheme proposed by Munawar et al. is similar to the one presented in this dissertation, however the realization details are different. The following drawbacks can be indicated. Firstly, lack of component dependencies discovery can cause problems in dynamic systems considered in this work. A change in the deployment of the components could be easily mistaken with a system failure and could unnecessarily trigger detailed monitoring. Secondly, detailed monitoring triggers monitoring of all metrics which can result in a significant overhead. This problem is pointed in [85] and the solution of adding an additional intermediate level is discussed. It still would result in three static levels that are not dynamically adapted to the current system state. Thirdly, the approach simply assumes that the monitoring facilities provided by the execution environment are sufficient and does not propose any additional monitoring
mechanisms that could be integrated with the adaptation logic.

Another comprehensive adaptive monitoring approach is proposed in the following publications authored and co-authored by Ehlers: [99–102]. The authors follow the vision of autonomic computing and adopt the MAPE-K loop for providing a self-adaptive monitoring solution capable of investigating performance anomalies. The whole approach is implemented as part of the Kieker monitoring and analysis framework. The implementation is targeted to JEE applications and uses AOP to crosscut the chosen JEE API (e.g. servlets or Enterprise Java Beans) and also regular business operations. Topological and causal system aspects are discovered by intercepting the first occurrence of each operation invocation and remembering the caller, the callee and the appropriate context.

The architecture of the proposed solution is well-designed. The capabilities of the instrumentation and collection layers and the algorithms of anomaly rating are sufficient for sound adaptive monitoring. However, inferencing related to the analysis of the calculated anomaly scores is not very sophisticated. In the exemplified scenarios, the activation of a given monitoring probe is dependent on the anomaly score of its parents, i.e. callers. It does not leverage the full potential of the information provided by the combination of the system structure (topology), the call graph (causality) and the anomaly scoring, which could allow to directly pinpoint anomalous operations located deeper in the application without activating the probes on all recursive parents. The extent to which this information can be leveraged is shown in the following papers [103, 104]. Their authors present different approaches to failure diagnosis based on the partial information about anomalies and system dependencies. The goal of both contributions is to rank the system elements by the probability that a given element is responsible for the anomaly. Unfortunately, such a ranking focusing on root cause diagnosis is not sufficient for the sophisticated control loop of adaptive monitoring in cases where there are multiple sources of anomaly. In such cases an additional logic processing the ranking information combined with the system dependencies and advising the next steps of adaptive monitoring is needed.

In the work of Okanovic et al. [105, 106], another approach to the realization of adaptive monitoring for JEE on the basis of the Kieker framework extension is presented. The approach, referred to as DProf, is similar to the one proposed by Ehlers et al. [99–102], however there are several differences. Instrumentation is also realized with the use of AOP, but its reconfiguration is performed in a different way. It is achieved by changing the aop.xml descriptor of the archive deployed to the application server. Unfortunately, such operations could potentially lead to problems with the application sessions and the processing of the transactional context. In DProf, the logic of adaptive monitoring takes into account the SLA [107, 108] of the monitored entities. It is assumed that the metric thresholds of all elements related (directly or indirectly in the call tree)
to the SLA are either provided or learned a priori. Reconfiguration of monitoring is performed only when the monitored data differs from the SLA contract. Then, DProf proceeds, step by step, through the subsequent levels of the call tree and activates monitoring there until the primary difference between the SLA and the current state is finally found. Such approach also suffers from limited inferencing capabilities discussed in the previous paragraph. Moreover, the process in which the metric thresholds are learned and updated along with system evolution is not described with sufficient details.

Rish et al. [95] propose the concept of active probing. The concept involves an information-theoretic approach to adaptive probe selection aiming at speeding up the diagnosis process and minimizing the monitoring costs. Initially, a limited set of probes is activated. Whenever a problem is detected, additional probes are activated to narrow down the potential cause of the problem. The information provided by the probes is used for updating the system state modeled with the use of BNs. The probes are activated one by one according to the inferencing process performed over the system model. Each time the inferencing process selects the probe of which activation provides the largest amount of information. In the context of comprehensive adaptive monitoring, active probing lacks the feature of activating several probes at once. This would often be more appropriate for fulfilling some high-level monitoring goal. Moreover, the proposed two-layer structure of the BN model (one level for the system elements and one for the probes) does not allow for grasping the hierarchical dependencies between the system elements which are common to complex SOA systems considered in this dissertation.

There are several other approaches addressing the issues of adaptive monitoring. Keung et al. [109] present adaptive monitoring in the context of grid systems, while Katsaros et al. [110] focus on a solution for cloud environments. Keung et al. propose the implementation of a self-adaptive mechanism to autonomously maximize the performance of the Grid Index Service. Katsaros et al. propose scalable and adaptable monitoring mechanisms targeted for the Platform as a Service (PaaS) class of cloud systems. The adaptation logic covers not only the reconfiguration of monitoring time intervals but also changing more advanced monitoring parameters. Garth et al. [111] leverage genetic algorithms to introduce a novel approach and a supporting framework for run-time optimization of WS monitoring. The optimization analyzes the measured QoS and the evolving system state and continuously adapts a set of WS monitors. The following papers, [112] [113], present another view on adaptive monitoring. Their common assumption is that often the monitoring overhead results from repetitive checking of the same situations in the monitored systems. Therefore, the authors propose to analyze the context of monitoring events and to apply dynamic adaptation, which results in ignoring the redundant monitoring information.


2.4.3 Aspect of Goal Orientation

This section analyzes the approaches to adaptive monitoring from the goal orientation point of view.

Clark et al. [114] present an approach to self-adaptive monitoring of services that focuses on reacting to changes on the level of perceived risk by adjusting monitoring selectivity. It is assumed that the whole system involves different parties, each providing some services and also consuming the services of other parties. The monitoring can be performed in two different modes: active and passive. The active mode requires a separate monitoring service, which causes additional overhead. In each mode, the polling interval related to the monitoring data acquisition can be automatically adjusted. The risk level related to the service provided by a particular party is determined by the use of both local and environment knowledge. The local knowledge includes transaction cost and its history, while the environment knowledge covers the information related to reputation, which can be for example provided by the reputation authority. Each party defines an adaptation policy that specifies which monitoring model and pooling interval corresponds to which level of perceived risk. The presented perspective on adaptive monitoring differs from the one proposed in this dissertation. The scope of the proposed adaptation loop is limited to switching between two static levels and adjusting the polling interval. Additionally, the designed policy structure does not allow for sophisticated declarative description of adaptive monitoring goals in the context of system topology and service causality.

The work authored and co-authored by Ehlers [99–102] proposes an approach based on monitoring rules specified with Object Constraint Language (OCL) [115]. These rules allow for the activation and deactivation of the monitoring probes on the basis of contextual information containing, among others, operation response times, resource utilization and the calculated anomaly scoring. Anomaly is defined as a significant deviation between the measured observation and the previously forecast value based on historic monitoring data. The proposed rules format is most closely related to the action policy formulated with the use of the ECA notation, which, as mentioned in Section 2.3.4, to some extent lacks the potential of a higher-level goal policy.

In the work of Okanovic et al. [105, 106], the monitoring process is driven by an XML document, named DProfSLA, which contains definitions of the system SLA. Additionally, DProfSLA specifies, which call trees of the system should be monitored. When monitoring in the context of a given DProfSLA document is started, only the top level of each specified call tree is initially covered. Reconfiguration of monitoring is performed when the monitored data differs from the specified SLA indicators. The drawback of the proposed approach is the fact that the DProfSLA has no influence on the way in which the reconfiguration is performed. The document is restricted only to the specification of the elements that should be covered by the monitoring process and the expected SLA.
2.4.4 Prior Work Limitations

The previous two sections presented multiple approaches for the realization of both anomaly diagnosis and adaptive monitoring itself. Each discussed work had certain limitations. A summary of all these limitations is as follows:

- The models used for system representation do not take into account different communication patterns of SOA systems. Loose-coupling, an important principle of service-orientation, is often achieved by means of asynchronous interactions. The model used for adaptive monitoring of SOA systems should consider both synchronous and asynchronous operations.

- Often, before using the proposed models, a more detailed "calibration" of the model parameters is needed (e.g. providing CPTs in the Bayesian network). This could be performed by domain experts who configure the model according to their expertise. Unfortunately, it is a tedious and error-prone process, not appropriate for dynamic systems in which model recalibration could be needed quite often.

- In dynamic systems, the adaptation logic of the monitoring process needs detailed information about the underlying system. Such information should be continuously updated at run-time by the appropriate discovery mechanisms. Most of the proposed approaches do not cover both important aspects of the underlying system: structural information (referred to as topology) and the information about causal relationships.

- The proposed approaches involve using some goal-oriented policies for managing the control loop. However, they are either not very sophisticated, i.e. they miss information specifying what system parts should be covered by continuous topological and causal discovery, or they specify the scope of monitoring without any high-level guidelines for the realization of monitoring adaptation (e.g. how aggressive the adaptation should be in identifying the sources of anomalies).

- Some proposed system models are promising, but they lack the capability of representing hierarchical dependencies (both topological and causal) between the system elements. Such hierarchical dependencies of services are common in SOA systems.

- The inferencing capabilities of the proposed adaptive monitoring approaches are either not very sophisticated or too much focused on suggesting the nearest most probable anomaly cause. Thorough adaptive monitoring should consider multiple possible anomaly causes and should leverage inferencing for decreasing the system uncertainty in a manner most appropriate to the current system state and the declared monitoring goal.

- Often the execution phase of the control loop is limited to several static adaptation levels and sometimes the highest level simply represents monitoring of the whole system.
system. It is not suitable for dynamically changing systems where, in order to minimize the overhead, the monitoring should closely follow the current system behavior.

- The discussed works do not cover a thorough approach to instrumentation of SOA systems capable of the following: (i) acquiring all data necessary for topology and causal discovery and (ii) enabling the realization of monitoring process adaptation.

The enlisted limitations show that there is a space for improving the research state of adaptive monitoring on several different levels such as instrumentation, system modeling, inferencing or goal-oriented steering. In the course of the reviewed work, there was not a single research contribution that would be free of all indicated limitations. Therefore, in the scope of this dissertation, a new approach to adaptive monitoring is proposed, which improves the identified state of the art by overcoming all limitations of the prior work.

2.5 Summary

The scope of this dissertation is related to the following essential issues: (i) the SOA anatomy and its aspects of dynamics; (ii) the current challenges in the research field of adaptability; (iii) the approaches to the adaptive monitoring. This chapter presented important background aspects and performed a thorough review of the literature related to these issues.

When it comes to the SOA anatomy, there are several specification attempts, from which SOA-RA proposed by The Open Group is the most relevant one, taking into consideration its abstraction level and the completeness of coverage. However, this specification does not describe aspects of the SOA dynamics with enough details, which is taken into account in the following Chapter.

In the context of the adaptability research field, there are two important challenges, which should be addressed by this dissertation i.e., monitoring challenge and detecting challenge. Both are related to avoiding a continuous monitoring process by focusing only on the information valuable for the user and reacting to abnormal system behavior.

The current approaches to the anomaly diagnosis and the adaptive monitoring have several important limitations. These limitations are overcome by the novel approach to the adaptive monitoring presented in the following chapters. The limitations related to the models covering the structure, behavior and adaptation of the SOA system are addressed in Chapter 3. The issues related to the goal-orientation and inferencing capabilities are considered in Chapter 4. Finally, the lack of thorough instrumentation integrated into an implementable framework is touched by the contribution of Chapter 5.
Why monitoring calls for the aspects of adaptation? This question was already asked in the introduction and the simplified answer is as follows: in order to address larger, more complex and more dynamic systems without causing a noticeable overhead. The adaptive monitoring system is built on the basis of several compounds. The first essential compound is the concept. The goal of this chapter is to define a conceptual framework which establishes some high-level boundaries. The boundaries define what, where and how the aspects of monitoring should be adapted in order to target complex, dynamic, large-scale service oriented systems. The definition takes into account two general issues of adaptive monitoring identified in Section 1.1, i.e. selectivity and root cause identification. The proposed conceptual framework does not try to provide a direct solution for these issues. By using concept maps, meta-modeling and allowing for many actual realizations, the framework enables the fulfillment of the dissertation goals in the subsequent chapters, which present the rest of the adaptive monitoring compounds.

The structure of this chapter is as follows. The first section presents the aspects of the SOA dynamics that are in the scope of this dissertation. It gives the foundation on which abstraction of the adaptive monitoring is proposed in the subsequent section. The abstraction decomposes the monitoring process into three layers: SOA system, measurement and steering. Each layer is provided with a model tailored for the layer’s specifics. To make the abstraction more concrete, the following sections present the structure of each model and assess the fulfillment of the requirement. All introduced models are explained using a simplified application case. It allows confronting proposed approach with a common intuition and by these means, grasping the process in which the proposed layers and their respective models are used. This is valuable for Chapter 4 in which a concrete realization of the monitoring process is defined.
3.1 Aspects of SOA Dynamics

The main building blocks of SOA systems were described in Section 2.1.2 on the basis of Open Group SOA Reference Architecture. Extensiveness and the appropriate abstraction level of SOA-RA allow for grasping some aspects of the Service Oriented Architecture dynamics. However, the crucial concepts are not described thoroughly enough. Therefore, this section presents SOA-RA enrichment which highlights the aspects of the SOA dynamics that are in the scope of this dissertation. The description of SOA-RA presented in Section 2.1.2 involved a wider view on Service Oriented Solution Stack. Since this dissertation is focused mostly on services layer, service components layer and operational systems layer, these layers are the focal point of the identified SOA dynamics.

Fig. 3.1: Concept map presenting the main aspects of the SOA dynamics on the basis of SOA-RA

Figure 3.1 contains a concept map which presents the aspects of the SOA dynamics. The concepts related to the dynamics were marked in the figure in orange. Some fragments of SOA-RA were excluded from the figure to preserve clarity. The figure grasps the following three main aspects related to the SOA dynamics:

(i) dynamics of services registration and discovery;
(ii) dynamics of service components deployment;
(iii) adaptability of services, service components and operational systems.

They are described in the following paragraphs.

The essential characteristic of the first aspect (i) is that the service component is able to perform a dynamic interaction with the service registry. In the SOA-RA only services interact with the service registry and such interaction was performed mostly upon the deployment of the service unit. The introduced dynamic interaction is always related
Chapter 3. Concept of Adaptive Monitoring

to services exposed by the component and can involve its registration, unregistration as well as modifications of the service contract. In particular, a modification of the service contract introduces a significant difference in comparison to the previous situation where the contract was established at the design time, then added to the service inventory and finally registered at run-time in the service registry. Besides performing service registration, the service component is also capable of discovering services which were registered by other service components. Such discovery can be performed actively, by querying the service registry for some service contracts, or passively, by subscribing in the registry for the contracts in which the discovery is interested. Then, the service registry notifies the subscribed service component that the required contract was registered, unregistered or simply changed.

Service discovery can influence causal relationships between the services. When service A invokes service B, then a causal relationship between them is established. The discovery of new services at run-time allows for invoking them and by these means establishing new causal relationships. The dynamic interaction with the service registry mentioned before not only can trigger the creation of new relationships, but it can also change the existing ones (e.g. by means of contract modification). Causal relationships of services are grasped by the service flow that aggregates all services which are either directly or indirectly related in the context of causality. For example, when service A invokes B and B invokes two services: C and D, then all four services constitute a single service flow. The service flow can be perceived as an execution trace of the invocation of some business process. With many historical execution traces from some system, it is possible to find traces of the isomorphic structure. Each of the isomorphic structures represents one service flow. As it is presented in Figure 3.1, service discovery is able to indirectly influence service flows.

The second aspect (ii), grasped in Figure 3.1 is related to the installation of deployment units in the solution environment. As mentioned in Section 2.1.2, an example of the solution environment is the service container. This dissertation assumes that a service unit, which aggregates service components, can be dynamically deployed, undeployed or redeployed to a service container. This feature is the answer to dynamic changes of the resources available in the solution environment (cf. Section 1.1). Changes in availability of these resources can influence dynamic deployment. For example, when a given service container runs out of RAM memory, then some of its service units are redeployed to another container that is less overloaded. Such redeployment changes the system’s structure, which is hereafter referred to as topology.

The third aspect (iii) of the SOA dynamics is related to adaptability, which – as it is presented in Figure 3.1 – can be added to any horizontal layer of S3. It is assumed that in the service layer adaptability is added by enriching the service container, which makes adaptability transparent for services and service components. Added adaptability increases the run-time dynamics of both services and service components. It could
manifest itself in different ways: changing non-functional characteristics of a given ele-
ment, changing the element’s internal structure, changing resources assignment, etc. It
could also be somehow related to aspects (i) and (ii) – i.e. involving changes of causal
relationships and the system topology in the adaptation loop [2] by executing actions
that trigger: registration (unregistration) of some services, deployment (redeployment)
of service units or changes in the service composition influencing the service flows.

3.2 Abstraction of Adaptive Monitoring

This section presents an abstract view on the concept of adaptive monitoring proposed
in this dissertation. The whole concept is built on the foundation of three models: the
SOA system model, the measurement model and the steering model, which are described
in the following paragraphs.

**SOA system model** represents the current state of a dynamic system in the context
of structure and causality. The aspects of the SOA dynamics identified in the pre-
vious section show that there are two important elements which can be subjected
to dynamic changes, namely the system topology and service causal relationships.
Therefore, the SOA system model covers: (i) the current system structure - topol-
ogy and (ii) service flows, which grasp the aspects of causality. Continuous tracking
of the SOA system model provides information about the system elements that can
be potentially covered by the monitoring process.

**Measurement model** serves the following purposes: it identifies measurements which
can be potentially acquired from the system; it represents the values of measure-
ments in the scope of the monitoring process and it provides information about
the overhead incurred by the monitoring process. This model is built on the ba-
sis of the SOA system model and therefore it takes into consideration the system
structure and causality. For example, if a given service is used in the context
of three different flows, then three different measurements related to the cumula-
tive response time can be potentially acquired from the monitoring of the service.
When such a service is actually monitored, three values – one for each flow – will
be covered by the measurement model and a specific monitoring overhead will be
added to all three flows.

**Steering model** is built on the basis of the SOA system model and the measurement
model. Its purpose is providing the information needed to adjust the monitoring
scope – deciding which measurements and which elements from the SOA system
model should be covered by the monitoring process, taking into consideration the
high-level goal defined by the monitoring system user. The provided informa-
tion should cover the following aspects: redundancy of the monitoring process
– identification of measurements which provide indirect information about other
measurements; general system health – identification of measurements which are anomalous in the context of some assumed criteria.

Figure 3.2 presents a complete view on the high-level concept of adaptive monitoring. The presented concept decomposes the monitoring process into three layers. Each layer is related to one of the three models presented earlier. The SOA system layer is responsible for the construction of the SOA system model. The Measurement layer is responsible for the construction of the measurement model and the steering layer constructs and uses the steering model. The anatomy of all layers is similar. Each layer has the following elements: (i) a model updater which constructs and updates the model in the responsibility of a given layer, (ii) some mechanisms which are used for acquiring the data needed by the model updater and (iii) a reconfigurator which allows for influencing the internal logic of a given layer. Since the SOA system layer and measurement layers are focused on gathering the monitoring data, their respective mechanisms (discovery and measurement) are hereafter referred to as monitoring mechanisms.

![Fig. 3.2: High-level concept of adaptive monitoring](image-url)

The whole monitoring process is driven by the monitoring goal provided by the user of the monitoring system (e.g. an administrator) to the steering layer. The monitoring goal is expressed by a declarative description specifying what, where and how should be monitored. The goal can relate to the existing models (SOA system, measurement, steering) to precisely define the user’s intention. The monitoring system is capable of...
pursuing many monitoring goals simultaneously. In such a case the steering layer takes into account all of them when adapting the monitoring scope.

The arrows depicted in Figure 3.2 present the relationships between the concept elements. The SOA system layer sends the constructed SOA system model to the measurement and steering layer. The measurement layer sends the measurement model to the steering layer, which updates the steering model for the purpose of performing an inferring process upon the model. The steering layer communicates with the measurement and discovery layers by means of monitoring subscriptions. An instance of the monitoring subscription is a declarative specification of measurement elements and/or the SOA system model which should be covered by the mechanisms of the respective layer. When the steering layer announces the monitoring subscription, then the measurement and/or discovery layer should reconfigure its mechanisms in compliance with the subscription and should continuously gather the requested data as long as the subscription is announced.

The relationships grasped by the arrows inside each layer are related to the internal information flow. This flow is similar in the measurement and discovery layers. The reconfigurator listens for changes in the announced monitoring subscriptions, checks if the layer mechanisms need an adjustment and performs it only when the adjustment is really needed. For example, when subscription A requests monitoring of the response time in services X and Y, then the measurement layer performs a reconfiguration of its mechanisms and starts gathering data about the response time in these two services. However, when another subscription B is announced and it requests the response time of service Y, then it is simply ignored by the measurement mechanisms reconfigurator because service Y is already being measured. When the mechanisms are reconfigured, then a continuous monitoring process in a given layer is started.

Discovery mechanisms monitor the system fragments specified in the monitoring subscriptions and pass the information about the events related to topology and causality to the SOA system model updater. Measurement mechanisms measure the metrics in the SOA system model elements (in accordance with the monitoring subscription) and pass the measured values to the measurement model updater. Model updaters in both the measurement and discovery layers update the respective model in an on-demand manner. The models are updated only when a significant change is reported by the respective mechanisms. As soon as the model changes, it is sent to the appropriate layer (or layers).

Information flow in the steering layer is slightly different from the flow in the other two layers. The execution of the steering layer logic begins when a description of the monitoring goal is delivered to the layer. The goal is processed by the monitoring scope reconfigurator, which identifies the initial monitoring scope. The appropriate monitoring subscriptions are announced and the lower layers process them to finally respond with the constructed models. The models are sent to the steering model updater, which
analyzes them and performs an on-demand update of the steering model. Only if the steering model has been updated, the updater extracts the information related to the monitoring overhead from the measurement model and notifies the inferencer that it should perform the inferencing process. The inferencing process is performed when the steering model and the information have been extracted from the measurement model. The goal of the performed inference is to decide whether the selectivity of the monitoring process established by the currently active monitoring subscriptions should be changed or not. The inferencing process covers the following aspects:

i) it analyzes the overhead that the monitoring process incurs on the underlying system – if the overhead is too high, then the process selectivity should probably be increased;

ii) it analyzes the redundancy of the monitoring process – if the values of some measurements can be indirectly inferred, then probably they do not have to be monitored;

iii) it verifies the general system health – if some measurements or SOA system model changes are suspicious, then the monitoring process selectivity should probably be increased to identify any potential failures.

When inferencing in the context of all three mentioned aspects, the inferencer always ensures that the considered monitoring scope change complies with the monitoring goals. Fulfilling the monitoring goals is the highest priority of the steering layer logic.

The presented concept is compliant with the vision of autonomic computing discussed in Sections 2.3.1 and 2.3.3. The proposed layers of adaptive monitoring are able to perform all phases of the MAPE-K loop. The monitoring step is performed by measurement mechanisms, discovery mechanisms and model updaters in the two respective layers. The execution step is performed by the measurement mechanisms reconfigurator and the discovery mechanisms reconfigurator. The analysis and planning steps are realized by the steering layer. Analysis is performed by the steering model updater and planning is done in the inferencer and the monitoring scope reconfigurator. The autonomic element gathers knowledge that in the case of the proposed concept is kept in the three introduced models: steering, measurement and SOA system. The managed resource is the monitoring process itself, which is self-monitored and adjusted by the measurement and discovery layers and the scope of which is evaluated and influenced by the steering layer.

The high level of the presented adaptive monitoring abstraction allows for choosing an arbitrary design for each of the three introduced models. This dissertation uses the presented abstraction to build a fully functional proof of the concept. Therefore, for each model a meta-model describing its structure is defined. These meta-models are presented in the following sections.
3.3 SOA System Meta-model

Figure 3.3 presents a meta-model that grasps the structure of the SOA system model by means of a UML class diagram. The meta-model includes the elements of SOA systems identified in Sections 2.1.2 and 3.1, which are important for the monitoring process, and it defines the details of their relationships such as the relationship type and cardinality. For the purpose of the meta-model completeness, some new concepts that have not been defined before are also introduced. Since the SOA system model is related to two different aspects: topology and causality, the meta-model grasps the structures of these aspects in two respective planes. Instances of the two planes are referred to as the topology model and the causality model respectively and, when combined, they represent the SOA system model. The service is the central element, which is shared by both planes. The planes are described in detail in the following sections.

3.3.1 Topology Plane

In the topology plane, the service is associated with a service component and an abstract service. The service represents a single running instance of some abstract service, and therefore a single abstract service can have many services – instances. The abstract service can be interpreted as a contract to the fulfillment of which its instance is committed. For example, in the Web Service technological stack, a WSDL description could be perceived as an abstract service which can be instantiated by the service exposing the functionality in accordance with the WSDL. The service is always exposed by a single service component. However, a given abstract service can be exposed by different service components – a different service instance for each component.

Similarly to SOA Reference Architecture, a set of service components is aggregated by a solution component, which is a run-time representation of some service unit. The
solution component can be deployed directly to a service container, but it can also be aggregated by an *application*. The application is a logical grouping of one or more solution components, which together implement some larger application functionality. Both the solution component and the application can be deployed to a single service container. The relationship between the service unit and the solution component can be compared to the one between the abstract service and the service. The solution component is an actual instance of a given service unit, and therefore a given service unit can be indirectly deployed to multiple service containers by means of different solution components of the service unit. The service container is always executed on a solution platform. A single solution platform, e.g. JVM, can host multiple service containers. However, most commonly, only a single container is launched in a single instance of the solution platform. A set of different service containers, interconnected through the network by some communication mechanisms, form a *federation*, which is the most general element of the topology plane. Containers are connected into a federation in order to increase reliability, e.g. remove the problem of a single point of failure, and to increase performance, e.g. distribute the workload to different containers. Each element of the topology plane can have a life cycle which covers its deployment, configuration and initialization.

In order to better explain both the topology and causality planes, a sample SOA application scenario is presented in Figure 3.4. This scenario will be used in this section for discussing its SOA system model – a particular instance of the SOA system meta-model – and in Section 3.4 for explaining the measurement and steering models. The application scenario involves four services (both synchronous and asynchronous), which are deployed in a specific topology. The services invoke each other in a certain manner resulting in

![Diagram](image-url)
Chapter 3. Concept of Adaptive Monitoring

the creation of some causality links.

The topology of the presented application is as follows. Service components $A$ and $B$ expose one service ($S_1$ and $S_2$ respectively), while service component $C$ exposes two services ($S_3$ and $S_4$). Each of the presented services has its corresponding abstract service, which is excluded from the figure to preserve clarity. Service component $C$ is bundled by the solution component $Y$. Service components $A$ and $B$ are bundled in a single solution component $X$, which is then aggregated by application $X$. The application is actually not needed when only one solution component is involved. In this scenario, it is used only to exemplify the topology plane. Both solution components have their corresponding service units (not presented in the figure). Application $X$ and solution component $Y$ are deployed to two service containers, which are interconnected into a single federation. Each container is executed on a dedicated solution platform – an instance of JVM (not presented in the figure).

The topology model of the sample scenario is presented in Figure 3.5 by means of a UML object diagram. The model covers all elements of the topology plane with at least one object instance for each class from the plane.

Fig. 3.5: Topology model of the sample SOA application

3.3.2 Causality Plane

The purpose of the causality plane is to describe the service flows (defined previously in Section 3.1) from the perspective of a single service. Therefore, in a given service, for each flow spanning this service, a respective class of flow invocations is present. This class represents the fact that a given service is invoked on behalf of the classes flow. A single instance of flow invocations is related to several concrete realizations of service invocation. Service invocation represents an act of either local or distributed communication, which results in the functionality provided by one service being consumed by another service. The invocation type, to which service invocation is related,
states whether invocation is singular (one service invoked) or parallel (many services invoked in parallel) and whether it is synchronous or asynchronous. Flow invocations are always related to at most one instance of initiating, causing, terminating invocation and to as many as possible instances of caused, consumed invocations. Their definitions in the context of a given service (named service X) are provided below.

**initiating** – Invocation of service X performed by another service. Invocation starts the execution of the service X logic.

**terminating** – Either the last asynchronous invocation performed by service X (invocation finishes execution of service X logic) or a response to a synchronous initiating invocation.

**consumed** – Either a synchronous invocation of another service, e.g. service Y, performed by service X, or asynchronous invocation of service X, performed by another service, e.g. service Z, under the condition that service Z invokes service X in its terminating invocation.

**caused** – Non-terminating and non-consumed invocation performed by service X (only asynchronous invocations can belong to this category).

**causing** – The most recent invocation, which either directly or indirectly causes the execution of service X logic and which is invoked by a service the logic of which is not terminated at the time of initiating service X.

In order to simplify the way of referring to the above flow invocations, the following convention is assumed when considered appropriate. If service X invokes service Y, depending on the concrete invocation realization, it could be referred to as follows:

- service X initiates service Y (when X makes an invocation which initiates Y);
- service Y terminates service X, or X terminates on Y, or X terminates by invoking Y (when the invocation is asynchronous and terminating);
- service X terminates service Y (when the invocation is synchronous);
- service X consumes Y (when the invocation is synchronous and consumed);
- service Y consumes X (when the invocation is asynchronous and consumed);
- service Y is caused by X (when the invocation is caused);
- service X is causing service Y (when the invocation is causing).

The last not yet explained element of the causality plane – the **sync type** – represents the fact that, in the context of causality, a service can be either synchronous or asynchronous. In a synchronous service, the initiating invocation is always synchronous and it both initiates and terminates the execution of the service logic. In an asynchronous service, the initiating invocation is always asynchronous and it never terminates the execution of the service logic.
Chapter 3. Concept of Adaptive Monitoring

Analogically to the topology plane, for the purpose of better explanation, the causality plane is now discussed in the context of the sample SOA application scenario (cf. Figure 3.4). The scenario involves four services and three service flows. There are two synchronous services: $S_1$, $S_3$ and two asynchronous ones: $S_2$, $S_4$. The service flows are named red, green and blue and they are marked with the color of arrows and with the appropriate arrow labels. Each arrow represents either asynchronous or synchronous service invocations. Ordering the execution of invocations, in the context of a given flow, is depicted in the arrow labels. In synchronous invocations, the labels have two numbers: ordering the request and ordering the response. For each service, a figure and a description grasping the related flow invocations is presented in the following paragraphs. On each figure, the elements related to the service of the presented flow invocations are marked in gray.

**Service $S_1$** flow invocations are presented in Figure 3.6. This service is initiated from the outside, therefore in all three flows there is no initiating, causing or terminating invocation of $S_1$. In the red flow, there are the following sequential invocations\(^8\) synchronous call of service $S_3$ ($IR_{1,2}$) and asynchronous call of service $S_2$ ($IR_3$ and $IR_4$). Invocation $IR_{1,2}$ is synchronous, therefore it is classified as consumed. Invocation $IR_4$ is a terminating asynchronous invocation performed by $S_2$, therefore it is also classified by $S_1$ as consumed. Invocation $IR_3$ is neither consumed nor terminating, therefore it is classified as caused. In the green flow, the situation is the same as in the red flow, but there is one parallel asynchronous call of services $S_2$ ($IG_{1a} – caused$, $IG_{2a} – consumed$) and $S_4$ ($IG_{1b} – caused$, $IG_{2b} – consumed$). In the blue flow, the situation is different. Service $S_4$ is classified as consumed by $S_1$ because $S_4$ asynchronously invokes $S_1$ in the terminating invocation ($IB_5$). Invocation $IB_1$ is classified as caused because $S_2$ is invoked by $S_1$, but it does not terminate on $S_1$.

![Fig. 3.6: Causality model of the sample application: flow invocations of service $S_1$](image)

**Service $S_2$** flow invocations are presented in Figure 3.7. This service is spanned by all three flows. In the case of the red and green flows the situation is simple. The $S_2$ logic terminates by an asynchronous invocation of $S_1$ ($IR_4, IG_{2a}$). At the same time, service $S_1$ initiates the $S_2$ logic and also causes it – $IR_3$, $IG_{1a}$ ($S_1$ is classified as causing because it is the most recent service calling $S_2$ which is not terminated when $S_2$ is initiated). In the case of the blue flow, service $S_2$ is initiated by service $S_1$ – $IB_1$ (this invocation is

\(^8\)It is required that the causality model retains the information about ordering invocations (e.g. $S_3$ is invoked before $S_2$).
also classified as causing). Then service $S_3$ is synchronously consumed – $IB_{2,3}$. Finally, $S_2$ terminates by an asynchronous invocation of service $S_4$ – $IB_4$.

**Service $S_3$** flow invocations are presented in Figure 3.8. The service is spanned by the red and blue flows. In both flows, $S_3$ simply terminates by a response to synchronous invocations, performed by $S_1$ ($IR_{1,2}$) and $S_2$ ($IB_{2,3}$). These invocations are also initiating and causing for service $S_3$.

**Service $S_4$** flow invocations are presented in Figure 3.9. This service is spanned by the green and blue flows. In the green flow, $S_4$ is initiated by an asynchronous invocation performed by $S_1$ ($IG_{1b}$, this invocation is also classified as causing) and terminated by an asynchronous invocation of $S_1$ ($IG_{2b}$). In the blue flow, $S_4$ is initiated by an asynchronous invocation performed by $S_2$ – $IB_4$. Since this is an asynchronous invocation, which also terminates the $S_2$ logic, it is treated as being consumed by $S_4$. Service $S_4$ terminates by an asynchronous invocation of $S_1$ – $IB_3$. To determine the causing service, the invocations that led to the execution of $S_4$ have to be analyzed. The most recent invocation is $IB_4$, but since the $S_2$ logic is terminated, at the time of initiating $S_4$ it cannot be considered as causing. Then there is invocation $IB_1$ performed by $S_1$ and this one can be considered as causing for $S_4$ because the $S_1$ logic is not terminated when $S_4$ is initiated.
Chapter 3. Concept of Adaptive Monitoring

The presented construction of the causality model is very useful for determining possible measurements in the measurement model and for identifying the measurements that provide indirect information about other measurements in the steering model. The structure of both models is presented in the subsequent section.

3.4 Measurement and Steering Meta-models

The meta-models describing the aspects of measurements and steering are presented in Figure 3.10. Similarly to the SOA system meta-model, the notation of a UML class diagram is used. The elements from the SOA system meta-model are referred to by the measurements structure, which is then used as a foundation to define the steering meta-model. Both meta-models are described in the following paragraphs.

![Fig. 3.10: Measurement and steering meta-models](image)

3.4.1 Measurement Aspects

The central element of the measurement meta-model is the measurement entity itself. Measurement is always related to a particular metric, therefore Figure 3.10 associates measurement with a single metric type. A metric can be related to system aspects, e.g. service response time, consumed memory, consumed processor time, but it can also represent some business aspects, e.g. stock trend in the trading system or average transfer amount in the banking system. It is assumed that the value of a given metric is not sensitive to the context of invocation, i.e. the metric value is not dependent on the service flow in which the service is invoked (e.g. a given service is invoked on behalf of five different flows and always consumes 25 MB of memory).
Measurement can be in two states: active and inactive. When measurement is inactive, then the monitoring process does not cover the metric values related to the measurement. Inactive measurement simply represents a point in the system that can be potentially monitored. When measurement is activated, then the metric values are gathered by the monitoring process and the measurement starts to describe the state of a given metric by claiming that the measured metric values belong to a metric range defined by lower and upper bounds. Activation of the measurement starts incurring an overhead on the monitored system. Quantitative representation of this overhead is an integral part of the measurement entity. The overhead is represented as a normalized amount of resources that were consumed due to the realization of the monitoring process in the context of a given measurement. Activated measurement is meaningful only in a time context established by the measurement period element. The period specifies the time frame (starting at some point in the past and lasting until now) in which all metric values fit the metric range, i.e. are higher than/equal to its lower bound and lower than/equal to the higher bound. The formal definition of an activated measurement (symbol $M$) involves four compounds:

$$M = [LLB, LB, UB, UUB]$$

where:

- $LLB$ – some assumed lowest metric value;
- $LB$ – lower bound of the metric range to which the measured values belong;
- $UB$ – upper bound of the metric range to which the measured values belong;
- $UUB$ – some assumed highest metric value.

These explained four compounds define three metric ranges of the measurement:

- **under range** = $[LLB, LB]$;
- **exact range** = $[LB, UB]$;
- **over range** = $[UB, UUB]$.

A graphical representation of the relations between the measurement compounds and the metric ranges is depicted in Figure 3.11. An exact range defines the actual boundaries for the metric values measured in the measurement period, while under and over ranges define the boundaries into which the measured values may fall when they exceed the exact range. The motivation for describing the measurement with the use of ranges is to
decrease the overhead incurred by the monitoring process. Propagating the information that a given metric belongs to a certain value range consumes less resources (mostly network bandwidth) than propagating each measured value of some metric.

The measurement meta-model includes two types of measurements: measurement direct and measurement cumulative, which are referred to as mdirect (MD symbol) and mcum (MC symbol) respectively. When referring to multiple mdirect or mcum metrics names, the expressions mdirects and mcums are used. The mdirect describes directly the metric value of a given service and it is not related to any service flow. The meta-model shows than one service can have multiple mdirects – one for each metric type. Mdirects of the response time metric (RT symbol) in the sample application scenario presented in Figure 3.4 could be as follows:

\[
MD_{RT}(S_1) = [40ms, 80ms, 120ms, 1120ms]; \\
MD_{RT}(S_2) = [90ms, 180ms, 220ms, 1220ms]; \\
MD_{RT}(S_3) = [135ms, 270ms, 330ms, 1330ms]; \\
MD_{RT}(S_4) = [20ms, 40ms, 60ms, 1060ms].
\]

LLB and UUB were set as follows: LLB was set to the half of LB and UUB was set to the value of UB incremented by 1000ms.

The mcum describes a particular service in the context of some flow. In the meta-model, a service always has a corresponding element of flow measurements which aggregates all mcums related to different metric types. A service can be associated with multiple flow measurements, one for each service flow. The mcum of a given service and flow describes the cumulative metric value of services which are consumed by service X in the context of the assumed flow. For example, when service A synchronously invokes service B, then A is aware of the response time of B, therefore the mcum of A will aggregate the response time of both A and B. In the case of other metrics, e.g. the consumed processor time, service A does not have to be necessarily aware of the processor time consumed by service B. However, in the presented concept of adaptive monitoring it is assumed that measurement mechanisms provided in the measurement layer (cf. Figure 3.2) ensure that the metric value of the measurement in a given service is always passed to the service invoked in the terminating invocation (defined in the SOA system metamodel). This approach guarantees that a given service is always aware of the metric values measured in its consumed services.

The mcum has two compounds: (i) a formula describing the metrics value aggregation; (ii) actual metric ranges of the mcum. When a given mcum is active, then its ranges can be monitored directly in the system. When a given mcum is inactive, then the ranges can be calculated by assigning particular values in the formula, assuming that all measurements needed for the calculation are currently activated. In order to calculate the aggregation formula, the SOA system model is needed. Assuming that the
formula for service X in flow R is calculated, the aggregation formula is defined on the
indirect of X and on the mcums (in flow R) of the services consumed by service X.
The mcums of the consumed services are referred to as consumed measurements and
the relationship between an mcum and its consumed measurements is referred to as
consumed relationship. As presented in the meta-model, one mcum can be associated
with multiple consumed measurements. The exact algorithm to calculate the formula is
of course dependent on the metric type, e.g. aggregation for the response time metric
will be different than for the consumed memory metric, however the evaluation of this
formula is metric type agnostic. The evaluation of the formula can be decomposed into
the evaluation of aggregation operators such as: addition, multiplication, maximization.
The evaluation of an arbitrary aggregation operator (\(OP_{AGG}\)) on a set of measurements
\((M_1, \ldots, M_n)\) is defined as follows:

\[
OP_{AGG}(M_1, \ldots, M_n) = [LLB_{AGG}, LB_{AGG}, UB_{AGG}, UUB_{AGG}]
\]

where:

\[
M_x = [LLB_x, LB_x, UB_x, UUB_x];
\]

\[
LLB_{AGG} = OP_{AGG}(LLB_1, \ldots, LLB_n);
\]

\[
LB_{AGG} = OP_{AGG}(LB_1, \ldots, LB_n);
\]

\[
UB_{AGG} = OP_{AGG}(UB_1, \ldots, UB_n);
\]

\[
UUB_{AGG} = OP_{AGG}(UUB_1, \ldots, UUB_n);
\]

The mcums of the response time metric in the sample application scenario could be as
follows (symbols \(R\), \(G\), \(B\) are used to refer to the red, green and blue flows respectively):

\[
MD_{RT}(S_1) = [40ms, 80ms, 120ms, 1120ms];
\]

\[
MD_{RT}(S_2) = [90ms, 180ms, 220ms, 1220ms];
\]

\[
MD_{RT}(S_3) = [135ms, 270ms, 330ms, 1330ms];
\]

\[
MD_{RT}(S_4) = [20ms, 40ms, 60ms, 1060ms];
\]

mcums in the red flow:

\[
MC_{RT}^{R}(S_2) = MD_{RT}(S_2) = [90ms, 180ms, 220ms, 1220ms];
\]

\[
MC_{RT}^{R}(S_3) = MD_{RT}(S_3) = [135ms, 270ms, 330ms, 1330ms];
\]

\[
MC_{RT}^{R}(S_1) = MC_{RT}^{R}(S_3) + MC_{RT}^{R}(S_2) + MD_{RT}(S_1)
\]

\[
= OP_{+}(MC_{RT}^{R}(S_3), MC_{RT}^{R}(S_2), MD_{RT}(S_1))
\]

\[
= [265ms, 530ms, 670ms, 3670ms].
\]

mcums in the green flow:

\[
MC_{RT}^{G}(S_2) = MD_{RT}(S_2) = [90ms, 180ms, 220ms, 1220ms];
\]

\[
MC_{RT}^{G}(S_4) = MD_{RT}(S_4) = [20ms, 40ms, 60ms, 1060ms];
\]
The mcums in the blue flow:

\[
MC^g_{RT}(S_1) = Max(MC^g_{RT}(S_2), MC^g_{RT}(S_4)) + MD_{RT}(S_1) = OP_+ (OP_{max}(MC^g_{RT}(S_2), MC^g_{RT}(S_4)), MD_{RT}(S_1)) = [130ms, 260ms, 340ms, 2340ms].
\]

A graphical representation of the measurement model and its relationships with services in the sample application scenario, which correspond to the presented equations, is depicted in Figure 3.12. The places of services \(S_2\) and \(S_3\) were exchanged to minimize the number of arrow crossings. Each service has its related measurements: one indirect and zero or many mcums. The mcums that were referring only to one indirect were excluded from the figure to preserve clarity. All measurements are placed in levels representing indirection between the measurements of a given level and the direct measurements. All

Fig. 3.12: Representation of the measurement model and its relationships with services in the sample application scenario presented in Figure 3.4
mdirects are placed on level 0, which represents a direct relation to the metric measured in a given service. The mcums are placed on a level the number of which resulted from incrementing by one the highest level of all consumed measurements. Building the presented relationships does not require the activation of any measurement. The information about the consumed services present in the SOA system model is sufficient for this task. This ensures that such relationships are universal for all metric types. The presented relationships and the algorithm for calculating the aggregation formula of a given metric type are sufficient for determining the formula for each mcum.

3.4.2 Steering Aspects

As stated before, abstraction of adaptive monitoring presented in Section 3.2 allows for choosing an arbitrary realization of the steering meta-model. This dissertation uses the theory of Bayesian networks (also referred to as BNs) to build instances of steering models. According to Jensen et al. [116], a Bayesian network consists of the following.

(i) A set of variables and a set of directed edges between the variables.

(ii) Each variable has a finite set of mutually exclusive states.

(iii) The variables together with the directed edges form an directed acyclic graph (DAG).

(iv) To each variable $A$ with parents $B_1, \ldots, B_n$ a conditional probability table $P(A|B_1, \ldots, B_n)$ is attached.

As stated before, the steering meta-model is built on the foundation of the measurement model. Therefore, in Figure 3.10 there are one-to-one relations between the essential elements of the respective meta-models. Such approach allows for grasping the most important aspects of measurements with the use of Bayesian networks. The inferencing capabilities of Bayesian networks are able to directly provide the important features of the steering model formulated at the beginning of Section 3.2, i.e.:

- identification of measurements which provide indirect information about other measurements;
- identification of measurements which are anomalous in the context of some assumed criteria.

The structure of the steering model which ensures the aforementioned features is now explained. The central element of the steering meta-model is measurement probability, referred to as mprob. Each measurement has its corresponding mprob, which is a variable of a Bayesian network. The states of a given mprob represent the metric ranges of its measurement. Each range is described by the range probability presenting the likelihood
that the mprob is currently in a given state - i.e. the relevant metric values belong to a given range. In other words, the range probabilities of a given mprob represent the probability distribution of its states, which can be calculated in a given BN by performing the inference process. The class of range probability from the meta-model can be potentially extended to represent a certain range status. The following three extensions are introduced:

**Evidenced** – The measurement of mprob is currently active and the monitoring data gives evidence that the evidenced range probability is equal to 100%.

**Nominal** – The measurement of mprob was active in the past but it is not active now. The nominal range is the exact measurement range from the past of which the current likelihood is above some assumed threshold.

**Suspicious** – The measurement of mprob was active in the past but it is not active now. The suspicious range is the under measurement range or the over measurement range from the past of which the current likelihood is above some assumed threshold.

Since the measurement model contains mdirects and mcums, the steering meta-model allows for representing them by means of *mcum probability* and *mdirect probability* which are concrete realizations of the mprob. The mcum probability is hereafter referred to as mcprob (*MCP* symbol), while the mdirect probability is hereafter referred to as mdprob (*MDP* symbol). The consumed measurements are represented in the steering model by means of *mconsumed probability*. The edges of Bayesian network are modeled in accordance with the relations between the mcums and their consumed measurements (i.e. consumed relationships) present in the measurement model. Each such relation is represented in the steering model by a directed edge connecting the mconsumed probability (the parent) with its mcprob (the child).

The mdprobs are variables of a Bayesian network which do not have any parents. Therefore, they do not have the *conditional probability table* (*CPT*). Instead, they contain one dimensional table showing the range probabilities of a given mdprob. The mcprob, contrary to the mdprob, has one or several parents – the respective mconsumed probabilities - which are present in the CPT of the mcprob. The CPT presents quantitative information about the probability of mcprob states conditioned on the configuration of parent variables. A set of mprobs connected with directed edges forms a DAG which represents a single Bayesian network. Since there are different measurements for different metric types, each metric type will also have a separate set of mprobs in the context of a given causality model. In the steering model, each such set of mprobs forms a separate Bayesian network.

So far, we have covered the first three aspects of the Bayesian network structure in the steering model: (i) variables, (ii) variable states, (iii) edges between the variables.
forming a DAG. The last element is calculating CPTs. In the case of the mdprobs, it can be simply assumed that the probability distribution over the considered metric ranges is uniform - i.e. when the respective mdirect is not monitored it is assumed that each range identified in the measurement model is equally possible. Of course, different arbitrary probability distribution can also be assumed when needed. For example, the system administrator knows from his/her experience that a particular logging service has never failed, therefore the probability of the over range of its response time could be set to a low percentage value.

The CPTs of the mcprobs are calculated with the use of sampling over the aggregation formula present in the respective mcum. In a given mcprob, for each combination of parent states the following is performed:

(i) for each parent, a vector of random metric values drawn from the metric range representing the state in the combination is prepared (the size of the vector is equal to the number of samples assumed before starting the sampling process);

(ii) when vectors from all parents are ready, the mcum formula is used to reduce all vectors into one representing the metric values of the mcum;

(iii) each element in the reduced vector is assigned to the mcum range – the number of assignments to a given range divided by the size of the vector is written in the cell of the CPT representing a given combination and a given range.

Fig. 3.13: Bayesian network created for the sample application scenario presented in Figure 3.4

An instance of the steering meta-model – the Bayesian network for the response time metric in the sample SOA application scenario is presented in Figure 3.13. The network structure is exactly the same as the one of the measurement model presented in Figure
This allows for identifying an important property of the proposed meta-models, i.e. the parent-child relationship in the created Bayesian networks is exactly the same as the consumed relationship present in the measurement model.

The potential of Bayesian networks lays in their capability of reasoning under uncertainty. The question is how the steering model can benefit from this potential. In order to answer this question, monitoring of the presented sample application scenario is now considered. The consideration covers monitoring different service sets and shows that the Bayesian networks provide very important information allowing for choosing the services the monitoring of which provides the maximum amount of information about the whole application.

When the monitoring process covers the response time of all four services of the sample application (i.e. all mdirects and all mcums of the response time are activated), then the current exact range of each service is known and, in the steering model, it is certain that each mprob is in the state representing the evidenced exact range. The measurement probabilities in such a case are presented in table 3.1. As it can be seen, there is no uncertainty in such a situation - the information entropy (also referred to as the Shannon entropy [91]), which can be interpreted as the measure of variable uncertainty, equals zero for each measurement probability.

<table>
<thead>
<tr>
<th>mprob under exact over entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCP_RT_RT(S_1)</td>
</tr>
<tr>
<td>MCP_RT_RT(S_2)</td>
</tr>
<tr>
<td>MCP_RT_RT(S_3)</td>
</tr>
<tr>
<td>MCP_RT_RT(S_4)</td>
</tr>
<tr>
<td>MCP_G_RT(S_1)</td>
</tr>
<tr>
<td>MCP_G_RT(S_2)</td>
</tr>
<tr>
<td>MCP_G_RT(S_3)</td>
</tr>
<tr>
<td>MCP_G_RT(S_4)</td>
</tr>
<tr>
<td>MCP_B_RT(S_1)</td>
</tr>
<tr>
<td>MCP_B_RT(S_2)</td>
</tr>
<tr>
<td>MCP_B_RT(S_3)</td>
</tr>
<tr>
<td>MCP_B_RT(S_4)</td>
</tr>
<tr>
<td>MDP_RT(S_1)</td>
</tr>
<tr>
<td>MDP_RT(S_2)</td>
</tr>
<tr>
<td>MDP_RT(S_3)</td>
</tr>
<tr>
<td>MDP_RT(S_4)</td>
</tr>
<tr>
<td>cumulative entropy: 0,00</td>
</tr>
</tbody>
</table>

Table 3.1: Measurement probabilities of the response time metric in the Bayesian network of the sample application scenario presented in Figure 3.13 when all measurements are activated (all mprobs are highlighted in the first column)

Now we will consider a situation in which the monitoring process of the same application scenario is stopped (no measurements are active). Since we do not have any other knowledge, we assume a uniform distribution of probability in the mdprobs. The measurement probabilities in such a case are presented in table 3.2. As expected, the probabilities of all mdprobs are equal. However, the probabilities of the mcpbs are not equal because they are influenced by the mdprobs in accordance with the CPTs of
### Table 3.2: Measurement probabilities of the response time metric in the Bayesian network of the sample application scenario presented in Figure 3.13, when no measurement is active (no mprob is highlighted)

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP_{RT}^B(S_1)$</td>
<td>19.89%</td>
<td>17.97%</td>
<td>62.14%</td>
<td>0.93</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_1)$</td>
<td>17.67%</td>
<td>17.29%</td>
<td>65.04%</td>
<td>0.89</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_2)$</td>
<td>14.32%</td>
<td>11.09%</td>
<td>74.59%</td>
<td>0.74</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_2)$</td>
<td>28.41%</td>
<td>19.18%</td>
<td>52.41%</td>
<td>1.01</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_3)$</td>
<td>20.1%</td>
<td>14.36%</td>
<td>65.54%</td>
<td>0.88</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_4)$</td>
<td>28.41%</td>
<td>19.18%</td>
<td>52.41%</td>
<td>1.01</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_4)$</td>
<td>20.1%</td>
<td>14.36%</td>
<td>65.54%</td>
<td>0.88</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_1)$</td>
<td>33.33%</td>
<td>33.33%</td>
<td>33.33%</td>
<td>1.10</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_2)$</td>
<td>33.33%</td>
<td>33.33%</td>
<td>33.33%</td>
<td>1.10</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_3)$</td>
<td>33.33%</td>
<td>33.33%</td>
<td>33.33%</td>
<td>1.10</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_4)$</td>
<td>33.33%</td>
<td>33.33%</td>
<td>33.33%</td>
<td>1.10</td>
</tr>
</tbody>
</table>

Cumulative entropy: 8.85

Each mprob. As it can be seen, it is very unlikely that the mcums still belong to the exact range. It is more probable that they now belong to the over range. The entropy of each mprob is relatively high (please note that 1,10 is the maximum entropy for a given variable), which represents high uncertainty about the system (it is also confirmed by high cumulative entropy).

The goal of the presented adaptive monitoring concept is twofold: (i) reduction of the uncertainty about the SOA system; (ii) reduction of the monitoring cost - i.e. the overhead incurred by the monitoring process. The probability calculus provided by Bayesian networks allows for estimating how a given decision about changing the monitoring scope will influence the uncertainty, while the measurement model allows for estimating the influence on the monitoring cost. We will now try to decrease the uncertainty about the sample application scenario in two different ways:

9. (i) by activating the response time measurements of service $S_2$ - mprobs presented in Table 3.3 (ii) by activating the response time measurements of service $S_1$ - mprobs presented in Table 3.4. As it can be seen, activating the measurements of service $S_1$ decreases the uncertainty about the system more significantly than activating the measurements of service $S_2$. When monitoring only the measurements of service $S_1$ and receiving the information that its mprob is in the state identified previously as exact, we can be almost sure that all mprobs of the sample application scenario are in their nominal ranges. This is a valuable piece of information that can be directly used by the steering layer. When during the monitoring of service $S_1$ suddenly higher response times are observed, then the Bayesian inferenc-

---

9From the perspective of Bayesian networks, activating a given measurement is interpreted as adding evidence to the network, i.e. setting a given variable (variables) to one of its (their) states.

10The only doubt is about service $S_4$ which has a relatively high probability of under range. However, since the response time of this service is relatively low, such probability distribution could be considered as acceptable.
ing will allow for identifying the mprobs with suspicious states – the mprobs that are most likely responsible for the change in the observed measurement. The presented case exemplifies the way in which the steering model can benefit from the potential of Bayesian networks and proves that Bayesian-based steering is capable of providing two important features recalled at the beginning of this section: (i) identification of measurements which provide indirect information about other measurements; (ii) identification of measurements which are anomalous in the context of some assumed criteria.

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP_{RT}^R(S_1)$</td>
<td>15,10%</td>
<td>54,36%</td>
<td>30,54%</td>
<td>0,98</td>
</tr>
<tr>
<td>$MCP_{RT}^G(S_1)$</td>
<td>11,97%</td>
<td>37,84%</td>
<td>50,19%</td>
<td>0,97</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_1)$</td>
<td>5,78%</td>
<td>41,61%</td>
<td>52,61%</td>
<td>0,87</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_2)$</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_4)$</td>
<td>3,30%</td>
<td>65,10%</td>
<td>31,60%</td>
<td>0,76</td>
</tr>
<tr>
<td>$MDP_{RT}(S_1)$</td>
<td>33,33%</td>
<td>33,33%</td>
<td>33,33%</td>
<td>1,10</td>
</tr>
<tr>
<td>$MDP_{RT}(S_2)$</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>$MDP_{RT}(S_3)$</td>
<td>12,77%</td>
<td>85,69%</td>
<td>1,54%</td>
<td>0,46</td>
</tr>
<tr>
<td>$MDP_{RT}(S_4)$</td>
<td>33,33%</td>
<td>33,33%</td>
<td>33,33%</td>
<td>1,10</td>
</tr>
</tbody>
</table>

cumulative entropy: 6,24

Table 3.3: Measurement probabilities of the response time metric in the Bayesian network of the sample application scenario presented in Figure 3.13 when the mdirect and mcums of service $S_2$ are activated (the related mprobs are highlighted in the first column)

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP_{RT}^R(S_1)$</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>$MCP_{RT}^G(S_1)$</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_1)$</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_2)$</td>
<td>4,65%</td>
<td>95,19%</td>
<td>0,16%</td>
<td>0,20</td>
</tr>
<tr>
<td>$MCP_{RT}^B(S_4)$</td>
<td>3,71%</td>
<td>96,19%</td>
<td>0,10%</td>
<td>0,17</td>
</tr>
<tr>
<td>$MDP_{RT}(S_1)$</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>$MDP_{RT}(S_2)$</td>
<td>6,85%</td>
<td>93,09%</td>
<td>0,06%</td>
<td>0,25</td>
</tr>
<tr>
<td>$MDP_{RT}(S_3)$</td>
<td>8,22%</td>
<td>91,18%</td>
<td>0,60%</td>
<td>0,32</td>
</tr>
<tr>
<td>$MDP_{RT}(S_4)$</td>
<td>46,70%</td>
<td>52,18%</td>
<td>1,12%</td>
<td>0,75</td>
</tr>
</tbody>
</table>

cumulative entropy: 1,69

Table 3.4: Measurement probabilities of the response time metric in Bayesian network of the sample application scenario presented in Figure 3.13 when the mdirect and mcums of service $S_1$ are activated (the related mprobs are highlighted in the first column)
3.5 Summary

The main contribution of this chapter is the high-level concept of the adaptive monitoring. The concept specifies three layers: SOA system, measurement, steering; and five elements: SOA system model, measurement model, steering model, monitoring goal and monitoring subscription. The model of each layer is described with a use of the respective meta-model. The SOA system meta-model introduces planes of topology and causality, which take into account the following aspects of the SOA dynamics: service registration and discovery; service components deployment; adaptability. The measurement meta-model is built around the concept of measurement, which represents either direct or cumulative metric values of given service. While, the steering meta-model allows for mapping the instances of measurement model into Bayesian networks. It enables inferencing which is used for deciding about the monitoring scope adjustments. The following two chapters focus on the elements not extensively considered so far, i.e. the monitoring goal and the monitoring subscription. Chapter 4 defines a concrete realization of the monitoring process and elaborates on controlling the process by means of the monitoring goal. Chapter 5 presents a design of the adaptive monitoring framework in which the monitoring subscription plays a crucial role.
Chapter 4

Monitoring Process Realization

The abstract concept of adaptive monitoring introduced in the previous chapter leaves an open space for defining many possible realizations of the adaptive monitoring process. The process is the second compound, after the concept, which contributes to the construction of the adaptive monitoring system. There are several issues that need to be addressed in the scope of process definition. The first issue is controlling the monitoring process. In this dissertation, it is assumed that the monitoring process is driven by a high-level, declarative goal. Therefore, the structure of goal description has to be defined. The second issue is defining the skeleton of the cycle, often referred to as the control loop, in which adaptation of the monitoring process is performed. The last issue is designing internals of the control loop and their inter-operation. All three issues are addressed in this chapter, which proposes a concrete realization of the adaptive monitoring process. Using as foundation the concept of adaptive monitoring, the proposed contribution directly targets the fulfillment of the thesis statement, i.e. providing a selective monitoring process, capable of identifying the root causes of detected problems.

The structure of this chapter is as follows. The first section recalls the high-level interpretation of the monitoring goal introduced earlier and defines the concept of monitoring goal strategy. This strategy is used for controlling the control loop, which is the heart of the monitoring process. The anatomy of the control loop is presented in the subsequent section. The description of anatomy realization is preceded by a motivation presenting observations of issues in [SOA] systems related to monitoring. The anatomy realization is decomposed into three distinct phases that cooperate to achieve a dynamic selectivity. The third section focuses on the internals of each control loop phase. The internals are described by algorithms presenting the means by which each phase fulfills the expectations formulated by the control loop anatomy.
Chapter 4. Monitoring Process Realization

4.1 Monitoring Goal Strategy

A high-level interpretation of the monitoring goal was already presented in Section 3.2. It is as follows:

The whole monitoring process is driven by the monitoring goal provided by the user of the monitoring system (e.g. an administrator) to the steering layer. The monitoring goal is expressed by a declarative description specifying what, where and how should be monitored. The goal can relate to the existing models (SOA system, measurement, steering) to accurately define the user’s intention.

The aforementioned declarative description of the monitoring goal is referred to as monitoring goal strategy or just monitoring strategy. The detailed structure of the monitoring goal strategy is presented in Figure 4.1 by means of a UML class diagram. Each class related to one of the models presented in the previous chapter is annotated with the appropriate description and marked with the color used previously in the figure presenting the respective model. References to the defined models allow for grasping user’s intention in the context of the following aspects: where should the monitoring be activated – the SOA system model, what should be monitored – the measurement model, how should the monitoring process be adapted over time – the steering model. There is also an important aspect related to the monitoring process timing, i.e. when should the process be started.

The purpose of the monitoring goal strategy is to describe the monitoring process in the context of the aforementioned aspects – where, what, how and when. The "when" aspect is addressed by means of the activation of the monitoring strategy. Finishing the definition of the monitoring strategy is not sufficient to consider it by the monitoring process. Only when the strategy is activated, it starts influencing the monitoring process. Triggering the activation of the monitoring strategy is controlled by the following three classes: time frame, topology condition, causality condition. The time frame simply specifies the period in which the strategy should be active. The topology condition and the causality condition impose a requirement on the topology model and the causality model respectively. This requirement has to be fulfilled prior to the activation of the strategy (e.g. activate the strategy after four different service flows have been discovered in a given system fragment).

The remaining monitoring process aspects: where, what and how are covered by describing the following monitoring activities:

(i) gathering information about the SOA system in terms of topology and causality – covered by tmodel tracker and cmodel tracker classes respectively;

(ii) managing the activation of measurements from the measurement model – covered by measurement activation classes;
Fig. 4.1: Structure of the monitoring goal strategy

(iii) directing the steering layer in the context of selectivity and on-demand root cause identification – covered by steering directives class.

When the monitoring strategy involves the tmodel tracker, the monitoring process gathers the topology of the SOA system and updates it in line with potential changes. The class of topology element refers to any object of the topology plane of the SOA system model. If the tmodel tracker does not aggregate any topology element, then the topology of the entire federation is gathered. If the tmodel tracker aggregates some topology elements (e.g., two selected federation containers), only the topology fragment located inside the selected elements is created and tracked. In the case when the monitoring strategy involves the cmodel tracker, the monitoring process discovers the causality model and – similarly to the tmodel tracker – updates it constantly in accordance with the changes in the system. If the cmodel tracker is associated with some topology elements, then the causality model built by the monitoring process is restricted to the service flows identified in these topology elements. Otherwise, the causality model of the whole federation is tracked.

By default, the cmodel tracker instructs the monitoring process to discover invocations of identified service flows only in the topology elements associated with the cmodel tracker. However, the user is able to extend the tracking process by aggregating the chosen service flows in the cmodel tracker. If some flows are aggregated, then all their invocations are tracked, even when they are beyond the topology elements associated with the cmodel. Similarly to the cmodel tracker, the tmodel tracker also by default
restricts topology discovery to the topology elements of tmodel tracker. There is a possibility of overcoming this restriction by associating the tmodel tracker with some service flows. If this is the case, then the topology model of all elements spanned by the associated flows is tracked.

When monitoring strategies including tmodel and cmodel trackers are activated, the monitoring process creates instances of topology and causality models. Then, the measurement model is automatically created in the measurement layer of the adaptive monitoring concept (cf. Section 3.2). However, by default no measurements are activated. As mentioned earlier, the class of measurement activation allows to manage these activations. The monitoring strategy can have none or many instances of the measurement activation class. If no measurement activations are provided in a given monitoring strategy, no new measurements are activated by the monitoring process. Each measurement activation has one or several measurement descriptors as well as single instances of the expiration period and the minimal period. The expiration period specifies how long the monitored metric values should be considered valid for the calculation of the lower and upper bound of a given exact range. Such expiration timeout prevents rare anomalies from influencing the system state over a longer period of time. The minimal period entity specifies the minimal time frame that has to elapse in order to consider the current exact range as valid. It allows to influence the inertia of the monitoring process, i.e. how fast the process reacts to changes in the system. The measurement descriptors specify which measurement should be activated. Such specification can be provided in the two following ways:

- specifying directly the instances of measurements from the measurement model;
- specifying the measurements indirectly by providing a combination of topology elements, service flows, metric types and measurement types (direct or cumulative). A logical and is performed on all elements of such a combination and all resulting measurements are chosen.

The measurement descriptors also specify the directives for the calculation of the lowest and highest bound in the activated measurements.

The last not yet discussed monitoring activity described by the monitoring strategy is directing the steering layer by means of the steering directives class. If this class is not present in a given monitoring strategy, then the monitoring process initiated by such a strategy does not involve advanced aspects of adaptation (only the SOA system and the measurement models are adapted to the current environment status). When this class is present, it aggregates single instances of three other classes: nominals acquisition, drill-down condition, drill-down driver. Each of these classes influences the realization of the

\[\text{For example, the user can specify the following combination: two chosen applications, two service flows (e.g. red and green), consumed memory metric, direct measurement. It will result in the activation of all indirect consumed memory measurements identified in two applications (present in the topology model) and two service flows (present in the causality model).}\]
control loop. The nominals acquisition is related to the phase in the control loop when
the typical behavior of the monitored SOA system is learned. The drill-down condition
defines a rule the fulfillment of which triggers the process of on-demand root cause
identification, also referred to as on-demand diagnosis. The drill-down driver directs
this process and manages the minimization of the monitoring overhead. Additional
details on each of the three classes are provided in the next section.

4.2 Control Loop Anatomy

The control loop, is the essential element ensuring the adaptivity of the proposed mon-
itoring process. The control loop is directed by the declarative description of the moni-
toring goal – the strategy. This section describes the anatomy of the control loop in the
context of monitoring strategy directives. The monitoring strategy allows to manage selectivity in a fine-grained way, while the control loop focuses on selectivity optimization
and on the approach to on-demand root cause identification.

4.2.1 Motivation

The concept of adaptive monitoring presented in Chapter 3 introduced a generic conceptual
framework for providing the monitoring process with aspects of adaptability. The
monitoring strategy described in the previous section provides the means to manage this
framework in a declarative way. The strategy allows the user to choose elements in each
monitoring layer (SOA system, measurement, steering), which should be encompassed
by the monitoring process. This way, selectivity can be achieved and it can be precisely
controlled by the user. The limitation of such selectivity management is that it is static
– a given strategy, once activated, maintains the same level of selectivity during the
run-time. It is inadequate for dynamic systems addressed by this dissertation. The
introduction of dynamic selectivity by means of the proposed control loop is motivated
by the following observations:

(a) When activation of a given monitoring strategy starts monitoring the process of
a particular SOA system fragment, often many service flows are discovered. This
introduces strong relations between the nodes of Bayesian networks in the steering
model. In such a case, as presented at the end of Section 3.4.2 monitoring a
subset of measurements can result in system uncertainty similar to that in the
case of monitoring all measurements. Monitoring all measurements is just a waste
of resources in such a situation.

(b) Even a dynamic SOA system most of the time behaves in a particular, predictable
way, i.e. metric values in the measurements belong to the same ranges for an ex-
tended period of time. Of course, behavior can change as a result of service recom-
position, a modification of the business logic or some functional or non-functional
failure, however the mean time between the changes is still quite significant. It is therefore reasonable to restrict the monitoring process in between the changes (no unnecessary overhead) and intensify it when a change occurs and should be tracked.

4.2.2 Realization

The anatomy of the control loop is presented in Figure 4.2. The loop was divided into three phases: (I) nominals identification, (II) sentinels selection and (III) adaptive drill-down covering two cycles: tracking cycle and diagnosis cycle. The figure presents interactions between the logic of each phase, the system user and the elements of the monitoring strategy structure (presented as classes). The whole loop is initiated by the user of the monitoring system, which activates the chosen monitoring strategy. It is of course required that this particular strategy contains an instance of the steering directives class. Each phase of the control loop changes the monitoring scope to fulfill the goal of the phase. Changes in the monitoring scope are realized by means of activating and deactivating the measurements from the measurement model. It could be interpreted as a realization of some monitoring scenario, which is most suitable for the current state of the SOA system and the logic performed by the control loop at a given moment.

At the beginning of the first phase, the SOA system model covering the topology and

![Diagram](image-url)
causality planes is discovered in accordance with the monitoring strategy. Then, all measurements that can be potentially activated in the system fragment covered by the topology model are identified. After this, the main logic of Phase I begins. All measurements related to the monitoring goal strategy are activated and the current behavior of the monitored system is learned. The behavior is learned by identifying the nominal metric range of each measurement. A metric range is considered nominal when the metric values fit in the range for an extended period of time. For example, ordering service in an on-line book store has a response time between $100\text{ms}$ and $200\text{ms}$ most of the time of its execution. However, the values of the response time can change when the load on the server is too high or some failure is encountered. An instance of the nominals acquisition class decides when the process of nominal measurements identification is finished.

Finishing the nominals identification triggers the creation of Bayesian networks. A single $\text{BN}$ is created for each set of measurements, which are directly or indirectly related to each other, through the consumed relationship established in the measurement model (cf. Section 3.4.1). It should be noted that the measurements in such a set are related to the same metric. After all $\text{BN}$ are created, Phase I is finished. For each created $\text{BN}$ the logic of Phase II and, if necessary, Phase III is independently performed. Sometimes, simultaneous realization of Phase II and Phase III for multiple $\text{BN}$ or simply multiple monitoring strategies can interfere with each other. This issue is discussed in Chapter 5.

Phase II benefits from the assumption that nominal measurements will not change quickly. There are not any significant changes in the system during this phase, therefore its time frame is referred to as the stale period. In Phase II, the monitoring overhead is minimized by deactivating these measurements that do not significantly influence the uncertainty about the system. It is achieved by selecting a set of sentinels, i.e. services, which measurements are not deactivated, and deactivating all other measurements. The name sentinels was chosen because the sentinel measurements (the measurements of the selected sentinel services) guard the sustainability of the nominal system state. The selection process is directed by the drill-down driver object of the monitoring goal strategy. After the process is finished, the sentinel measurements along with the tmodel and cmodel trackers of the monitoring strategy are constantly checked for any changes relating to either the system model or the measurement model. In case of system model modification, the measurement descriptors of the monitoring strategy are used to check if the final set of measurements matching the descriptors has changed. If so, the control loop goes back to Phase I, where the nominal ranges of any new measurements are identified and then, after the transition to Phase II, sentinels selection is repeated. Such a cycle covering Phase I and Phase II is referred to as the tracking cycle (cf. Figure 4.2), because its purpose is to track changes in the underlying SOA system.

\footnote{The creation of a $\text{BN}$ on the basis of the measurement model was discussed previously in Section 3.4.2.}
measurement model modification, the drill-down condition of the monitoring strategy is used to check if the difference between the measurement model and the steering model is sufficient to trigger the diagnosis process. When the difference is sufficient, transition to Phase III is performed.

The purpose of Phase III is root cause identification of the difference between the measurement model and the steering model. The difference is caused by changing metric values of some measurements from the nominal range to some new range, referred to as anomalous. If all measurements with anomalous ranges (referred to as anomalous measurements) were active in Phase II (their services were chosen as sentinels), then identification is straightforward because all information needed for pointing the root cause is available. However, when some anomalous measurements are not active, the identification is not easy. The easiest approach would be to activate all measurements (just like in Phase I), however this could cause unnecessary monitoring overhead. In order to enable root cause identification without any significant monitoring overhead, the adaptive drill-down procedure is proposed. The adaptive drill-down is based on the previous author’s research related to ESB monitoring [117].

Drill-down uses the inferencing of Bayesian networks to make the most sensible steps in root cause identification, i.e. to activate the most appropriate measurements. The sensibility of steps is determined by the drill-down driver, in particular by its fitness function, which ranks the next possible steps by their contribution to minimizing entropy and the monitoring overhead. When root cause identification is finished, the diagnosis is reported to the user. The user can decide if the monitoring goal strategy should still be active. For example, in the case of some diagnosed failure, continuation of the monitoring process may not be justified. If there is no interaction with the user or if the interaction was present and the user decided that the monitoring strategy should still be active, the monitoring process goes back to Phase II. The identified anomalous ranges become new nominal ranges and sentinels selection is performed again. The cycle covering Phase II and Phase III is the second cycle presented in Figure 4.2. Since its purpose is to identify anomaly root cause, it is referred to as diagnosis cycle. The presented structure of the control loop ensures the aforementioned dynamic selectivity. It is achieved by disabling the measurements in the stale period and activating them on-demand for identifying the changes between the so-called nominal and anomalous system operation.

The whole concept of adaptive monitoring and the control loop in particular is built on the following important assumptions related to the design of monitoring mechanisms 13:

(a) Metric values of a measurement in a given service are always propagated along with the terminating invocation. This approach guarantees that the monitoring mechanisms of a given service are always aware of metric values measured in its consumed services.

13The first assumption was already formulated in Section 3.4.1 and is just repeated here.
(b) Deactivation of a given measurement causes that ranges of metric values are not directly reported anymore by the measurement, where reporting means sending the ranges to some component for further analysis. However, even when the measurement is deactivated, it still complies with the previous assumption, i.e. that the measured metric values are propagated along with the terminating invocation.

(c) When a given measurement is deactivated, acquiring metric values and propagating them along with the terminating invocation does not introduce significant overhead, i.e. the overhead is not noticeable for the monitored system.

Such assumptions allow to benefit (in the sense of overhead reduction) from the measurement deactivation in the stale period and, simultaneously, to sustain the ability of the sentinels to guard the nominal system state. The detailed design of the monitoring mechanisms fulfilling the presented assumptions is presented in Chapter 5.

4.3 Control Loop Algorithms

For each phase in the control loop, an algorithm for performing the crucial phase task was designed. All three algorithms are presented in this section and the important details of the control loop operation as well as the details of the monitoring strategy parameters influencing this operation are explained. The following notation is used in the algorithms pseudo code:

\[\begin{align*}
MG & \quad \text{monitoring goal strategy;} \\
M & \quad \text{measurement;} \\
MP_i & \quad \text{mprob related to measurement } M_i; \\
Dsc & \quad \text{measurement descriptor of } MG; \\
Dsc.Min.per & \quad \text{minimal period related to } Dsc; \\
Dsc.Exp.per & \quad \text{exact range expiration period related to } Dsc. \\
\end{align*}\]

4.3.1 Nominals Identification

The logic of nominal measurement identification, being the crucial task of Phase I, is presented in Algorithm 1. The algorithm works on a given monitoring strategy, its measurement descriptors and the resulting measurements. As the output, the algorithm returns the nominal ranges of all measurements. The main logic of identification is enclosed in a while loop (line 10) performed until all measurements have their nominal ranges. The body of this loop is executed parallely for each measurement which nominals were not yet identified. The loop logic relies on the measurement mechanisms (described in detail in Chapter 5), which, after measurement activation (line 7), are

\footnote{Minimal period and expiration period are assigned to a measurement descriptor indirectly by means of a measurement activation – cf. Section 4.1.}
Algorithm 1 Acquisition of nominal metric ranges

Input: \( MG \) – monitoring strategy; \( \forall x Dsc_x \in MG : \{ M_1, \ldots, M_n \} \in Dsc_x \)

Output: \( \forall x Dsc_x, \forall n (Dsc_x) : nom\_range(M_n) \)  // nominal metric range

1: \( M_{todo} \leftarrow \emptyset \)
2: for each \( Dsc_j \in MG \) do
3:   for each \( M_i \in Dsc_i \) do
4:     add \( M_i \) to set \( M_{todo} \)
5:     \( iter(M_i) \leftarrow 0 \)
6:     \( exact\_range(M_i) \leftarrow null \)
7:     activate \( M_i \)
8:   end for
9: end for
10: while \( M_{todo} \neq \emptyset \) do
11:   for each \( M_i \in M_{todo} \) perform in parallel do
12:     Wait for duration of \( Dsc(M_i).Min\_per \) and detect any changes of \( exact\_range(M_i) \) reported by respective measurement mechanisms
13:     if change of \( exact\_range(M_i) \) is detected then
14:       \( iter(M_i) \leftarrow 0 \)
15:     else
16:       \( iter(M_i) \leftarrow iter(M_i) + 1 \)
17:     end if
18:     if \( iter(M_i) = NomAcq.Iter \) then
19:       \( nom\_range(M_i) \leftarrow exact\_range(M_i) \)
20:       remove \( M_i \) from set \( M_{todo} \)
21:     end if
22:   end for
23: end while
24:

expected to report any changes of the exact range (line 12). If no change of the exact range is detected for the number of iterations specified in the nominals acquisition object (referred to as \( NomAcq.Iter \)), then a given range is considered to be nominal. It is not presented explicitly in the algorithm, but if exact ranges of some measurements are not reported at all by the measurement mechanisms (e.g. because there are no invocations that could be monitored), then such measurements are simply kept activated and excluded from the remaining control loop phases. It is concluded that keeping them activated will not impose any noticeable overhead. If the invocation frequency changes in the future, then such measurements are reassessed by the presented algorithm.

To better explain the algorithm of nominals identification, the results of its execution on the sample application scenario, introduced in Section 3.3.1, are now discussed. Let us assume that the user already knows the topology and causality related to the sample application (because of some other monitoring strategies activated in the past) and now wants to monitor the response time in the blue service flow. Therefore, the user defines the monitoring goal strategy, named \( MG_{Sample} \), with a single measurement activation. The strategy also contains \( tmodel \) and \( cmodel \) trackers, which ensure that the SOA sys-
tem model of the sample application will be continuously tracked. The activation has one measurement descriptor containing a minimal period equal to 120 seconds and the following indirect measurement specification: cover all response time measurements of the blue service flow. In order to influence the control loop, the user provides MG_Sample with the steering directives specifying, among others, that NomAcq.Iter = 3. When the strategy is activated, Phase I of the control loop begins and Algorithm 1 starts its execution. Figure 4.3 presents all measurements covered by MG_Sample. All of those measurements are activated in line 7 of the algorithm. Since the flow is invoked quite often, the first period of 120 seconds is enough to identify stable exact ranges of all measurements. However, since NomAcq.Iter = 3, another two iterations – periods of 120 seconds – are needed to finish the algorithm. The identified nominal ranges are presented in Figure 4.3.

Fig. 4.3: Representation of the measurement model and its relationships with services in the sample application scenario presented in Figure 3.4 when the monitoring strategy covers only the blue service flow. For each measurement a nominal range of the response time is assigned

### 4.3.2 Sentinels Selection

The logic of sentinels selection performed in Phase II of the control loop is presented in Algorithm 2. This algorithm operates on a single Bayesian network and its related measurements. Since the purpose of the main algorithm is to minimize overhead, it was decided that deactivation of single measurement related to one service will not be considered. If a service has two measurements, like for example $S_4$ in Figure 3.4
then deactivation of only one of them (e.g. $MC^B(S_4)$) does not reduce the actual overhead. Therefore, both Algorithm 2 and Algorithm 3 operate on services instead of single measurements: activation or deactivation of a service refers to activation or deactivation of all measurements related to this service in the context of a particular BN. As a consequence, the term sentinel refers to a service. Choosing a service for a sentinel means choosing all its measurements (in the context of the processed BN). The output of Algorithm 2 is a set of sentinels the activation of which fulfills the user’s expectations formulated in the drill-down driver.

The sentinels selection algorithm works in a loop (line 4). There is one iteration for each so-called search level. Level specifies the size of the set containing sentinel services (this set is referred to by the SS symbol). The following parameters can be passed to the discussed algorithm by means of the drill-down driver (referred to as DDD):

- **DDD.Fitness** – a fitness function operating on entropy and overhead. By specifying this function, the user can define which aspect has a higher priority. For example, if overhead is more important, the function could be defined as follows: $DDD.Fitness(entropy, overhead) = 10^6 \times \text{overhead} + \text{entropy}$.

- **DDD.Level.iter** – a value specifying how many searches of the best service combination (SS sets) can be performed on a given search level. Since the number of possible SS sets on a given level could be quite large\(^{14}\), **DDD.Level.iter** is used to heuristically restrict the number of evaluated combinations and by these means to decrease the algorithm’s complexity. The higher is the value, the better, but at the same time the slower is the algorithm.

- **DDD.Best_SS** – number of the best service combinations (in terms of fitness) remembered on a given level, which are used as a basis for generating new combinations on the next level. Remembering the best service combinations in SS sets is a heuristic, which improves the algorithm’s quality without increasing its computational complexity.

- **DDD.Min_suff** – a condition determining if the sentinel set given as the argument is sufficient for the user’s needs. For example, the condition could state as follows: return true when sentinels ensure that the probability of over range in all measurements is not higher than 10%.

As can be seen in Algorithm 2, the loop of the selection logic (line 4) is performed until the **DDD.Min_suff** condition is reached or until the size of the evaluated sentinels set (level) reaches the number of all services. The best SS set on a given level is searched by evaluating the fitness of the most promising (based on the previous levels) service combinations, the number of which is limited by the **DDD.Level.iter** parameter.

\(^{14}\)The exact formula of this number on a given level is as follows: \(\binom{n}{\text{level}}\), where $n$ is the number of services related to a given BN.
Algorithm 2 Selection of sentinels in the stale period

**Input:** \( MG \) - monitoring strategy; a single \( BN \); \( n \) services and \( m \) measurements represented by the provided \( BN \).

**Output:** \( SS \) - set of sentinel services fulfilling \( DDD.Min\_suff \)

1: \( Stack \leftarrow \emptyset \)
2: push \( \emptyset \) on \( Stack \)
3: \( level \leftarrow 0 \)
4: repeat
5: \( Stack\_used \leftarrow \emptyset ; Stack\_level \leftarrow \emptyset ; c \leftarrow 1 \)
6: \( level \leftarrow level + 1 \)
7: repeat
8: if \( Stack \) is empty then
9: break
10: end if
11: remove top \( SS \) from \( Stack \) and push it to \( Stack\_used \)
12: \( SS\_good \leftarrow \) copy of removed \( SS \)
13: \( C^n_{level} \leftarrow \) all possible \( level \) elements combinations, which are not already present in \( Stack\_level \), generated by adding \( \{S_i, \ldots, S_j\} \in \{S_1, \ldots, S_n\} \) to \( SS\_good \)
14: for each \( SS\_level \in C^n_{level} \) do
15: calculate overhead of \( SS\_level \) on basis of measurement model
16: calculate mdirects entropy of \( SS\_level \) on basis of steering model
17: assign fitness to \( SS\_level \) by means of \( DDD.Fitness \) function
18: add \( SS\_level \) to \( Stack\_level \)
19: \( c \leftarrow c + 1 \)
20: if \( c > DDD.Level\_iter \) then
21: break
22: end if
23: end for
24: until \( c > DDD.Level\_iter \)
25: \( Stack \leftarrow inverse(Stack\_used) \)
26: sort \( Stack\_level \) by fitness
27: copy \( DDD.Best\_SS \) number of top elements from \( Stack\_level \) to \( Stack \)
28: until \( (DDD.Min\_suff((top \ of \ Stack) \ returns \ true) \ \vee \ (level = n)) \)

of the drill-down driver. The compounds needed for the fitness evaluation, i.e. overhead and entropy, are calculated on the basis of the measurement and steering model respectively. Entropy is calculated only for direct measurements to represent the actual uncertainty about the services and not the derivative one enclosed in the cumulative measurements. The evaluated combinations are sorted according to the fitness (assigned by \( DDD.Fitness \)). If the best \( SS \) set does not fulfill the \( DDD.Min\_suff \) condition, then the number of best \( SS \) sets specified by \( DDD.Best\_SS \) is retained and the iteration for the next search \( level \) is performed.

To exemplify the operation of sentinels selection, the case presented in Figure 4.3 is analyzed. It is assumed that all nominal measurements were identified, as described in the previous section, and that the user provided the following parameters to the drill-down driver:
Table 4.1: Measurement probabilities of the response time metric in the Bayesian network of the sample application scenario for the situation presented in Figure 4.3, when $SS = \{S_1\}$ (activated mprobs are highlighted in the first column)

$$DDD.Fitness(entropy, overhead) = entropy - \text{overhead is not considered (however the division of the selection loop into levels implies favoring smaller levels first, which results in natural overhead minimization being the second priority after entropy)}. $$

$$DDD.Min.suff - \text{return true when sentinels ensure that the probability of over range in all measurements is not higher than 5\%}. $$

The parameters $DDD.Level_{iter}$ and $DDD.Best_SS$ are not needed in the case of such a small BN. It is also assumed that under and over range of each measurement were calculated in the following way: $LLB$ was set to the half of $LB$ and $UUB$ was set to the value of $UB$ incremented by 1000ms (definitions of $LB$, $UB$, etc. were presented in Section 3.4.1). The execution of Algorithm 2 starts from level 1. All possible $SS$ sets of size 1 are analyzed in the context of entropy. The results, sorted by entropy, are as follows:

- $SS = \{S_1\} - \text{entropy} = 2.69$
- $SS = \{S_4\} - \text{entropy} = 2.73$
- $SS = \{S_2\} - \text{entropy} = 2.83$
- $SS = \{S_3\} - \text{entropy} = 3.30$

The $SS$ set with the lowest entropy (and, in this case, also fitness) is $\{S_1\}$. The measurement probabilities and the related entropy for such $SS$ are presented in Table 4.1.

As can be seen, the measurements $MDP_{RT}(S_2)$, $MDP_{RT}(S_3)$, $MDP_{RT}(S_4)$ have the probability of the over range, higher than the assumed 5\%, therefore they do not fulfill the $DDD.Min.suff$ condition. As a consequence, another algorithm iteration has to be performed.

In the second iteration, all possible $SS$ sets of size 2 are analyzed. The results, sorted by entropy, are as follows:
\[ SS = \{ S_1, S_2 \} \quad \text{entropy} = 1,41; \]
\[ SS = \{ S_1, S_4 \} \quad \text{entropy} = 1,63; \]
\[ SS = \{ S_1, S_3 \} \quad \text{entropy} = 1,68; \]
\[ SS = \{ S_2, S_4 \} \quad \text{entropy} = 1,73; \]
\[ SS = \{ S_3, S_4 \} \quad \text{entropy} = 1,87; \]
\[ SS = \{ S_2, S_3 \} \quad \text{entropy} = 2,20. \]

The \( SS \) set with the lowest entropy (and, in this case, also fitness) is \( SS = \{ S_1, S_2 \} \). The measurement probabilities and the related entropy for this \( SS \) set are presented in Table 4.2. As can be seen, now all measurements fulfill the \( DDD.Min \_suff \) condition, therefore the algorithm finishes by returning \( \{ S_1, S_2 \} \) as a set of services for which the measurements will provide the maximum amount of information when retained as active in the stale period.

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>( MCP_{RT}^B(S_1) )</td>
<td>0,00 %</td>
<td>100,00 %</td>
<td>0,00 %</td>
<td></td>
</tr>
<tr>
<td>( MCP_{RT}^B(S_2) )</td>
<td>0,00 %</td>
<td>100,00 %</td>
<td>0,00 %</td>
<td></td>
</tr>
<tr>
<td>( MCP_{RT}^B(S_4) )</td>
<td>0,36 %</td>
<td>99,34 %</td>
<td>0,29 %</td>
<td></td>
</tr>
<tr>
<td>( MDP_{RT}(S_1) )</td>
<td>0,00 %</td>
<td>100,00 %</td>
<td>0,00 %</td>
<td>0,00</td>
</tr>
<tr>
<td>( MDP_{RT}(S_2) )</td>
<td>0,00 %</td>
<td>100,00 %</td>
<td>0,00 %</td>
<td>0,00</td>
</tr>
<tr>
<td>( MDP_{RT}(S_3) )</td>
<td>18,04 %</td>
<td>78,43 %</td>
<td>3,53 %</td>
<td>0,62</td>
</tr>
<tr>
<td>( MDP_{RT}(S_4) )</td>
<td>46,50 %</td>
<td>50,92 %</td>
<td>2,58 %</td>
<td>0,79</td>
</tr>
<tr>
<td>cumulative mdirect entropy:</td>
<td>1,41</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Measurement probabilities of the response time metric in the Bayesian network of the sample application scenario for the situation presented in Figure 4.3, when \( SS = \{ S_1, S_2 \} \) (activated mprobs are highlighted in the first column)

### 4.3.3 Adaptive Drill-down

Monitoring sentinel measurements is sustained until the detection of anomalous metric ranges, which cause the match of the drill-down condition (referred to by \( DDC \) symbol). This triggers the last phase of the control loop, i.e. the adaptive drill-down. The logic of drill-down is presented in Algorithm 3. This algorithm, similarly to the previous one, operates on a single Bayesian network and its related measurements and ensures that the drill-down condition does not match anymore after the algorithm is finished.

An example of the drill-down condition could be as follows: the condition matches (i.e. returns \text{true}) when the anomalous range of at least one measurement differs by more than 10% from the exact range used in the respective measurement probability\(^{15}\) of the constructed BN.

\(^{15}\)Before the drill-down execution, this exact range is simply the nominal range identified in Phase I. After the first iteration of the main drill-down loop (line \( \text{line 1} \)) the exact measurement ranges can differ from the nominal ones.
The drill-down algorithm operates in a loop as long as the drill-down condition or the
*DDD.Min.suff* condition (checked on all active measurements, i.e. sentinels) is true
(line 1). The loop logic is divided into two main parts: (i) identification of suspicious
services (lines 2-29); (ii) identification of the best drill-down sentinels (lines 30-43).
Suspicious services are the services for which direct measurements are suspected to have
the anomalous metric ranges. The term *best drill-down sentinel* is based on the concept
of sentinels introduced in Algorithm 2. In the context of drill-down, sentinel refers to
a service the activation of which should contribute the most to diagnosing the root
cause of the anomaly matched by the drill-down condition. It should be noted that the
activation of a service (also a term borrowed from the previous algorithm) in fact refers
to the activation of all measurements related to this service in the context of a particular
BN. Identification of drill-down sentinels could be interpreted as extending the set of
sentinels established in the stale period.

Identification of suspicious services is performed by the evaluation of the so-called
*expected_range* and *parentsays_range* (present in the for loop in line 5). It is explained
on the continuation of the example presented in the previous algorithm (see Figure 4.3).
The continuation considers two possible results of Algorithm 2: (i) $SS = \{S_1, S_2\}$; (ii)
$SS = \{S_1\}$. It is assumed that at some point in the stale period there is a failure of
service $S_4$, which causes an increase in its response time by 500ms. This changes the
exact range (identified in Phase I as nominal) of each measurement in the following way
(active measurements are marked in gray and mcum formulas are provided to simplify
the range interpretation):

<table>
<thead>
<tr>
<th>measurement</th>
<th>nominal range</th>
<th>anomalous range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP_{RT}(S_1)$</td>
<td>[570ms, 730ms]</td>
<td>[1070ms, 1230ms]</td>
</tr>
<tr>
<td>$MCP_{RT}(S_2)$</td>
<td>[450ms, 550ms]</td>
<td>n/a</td>
</tr>
<tr>
<td>$MCP_{RT}(S_4)$</td>
<td>[490ms, 610ms]</td>
<td>[990ms, 1110ms]</td>
</tr>
<tr>
<td>$MD_{RT}(S_1)$</td>
<td>[80ms, 120ms]</td>
<td>n/a</td>
</tr>
<tr>
<td>$MD_{RT}(S_2)$</td>
<td>[180ms, 220ms]</td>
<td>n/a</td>
</tr>
<tr>
<td>$MD_{RT}(S_3)$</td>
<td>[270ms, 330ms]</td>
<td>n/a</td>
</tr>
<tr>
<td>$MD_{RT}(S_4)$</td>
<td>[40ms, 60ms]</td>
<td>[540ms, 560ms]</td>
</tr>
</tbody>
</table>

As can be seen, the range of one of the active measurements – $MCP_{RT}(S_1)$ changed
quite significantly (the range boundaries were almost doubled), which triggered the
drill-down process (it is assumed in the example that the drill-down condition matches
when the range difference is higher than 10%). The evaluation of *expected_range* and
*parentsays_range*, performed in lines 3-20 of Algorithm 3 is achieved with a for loop
iterating over measurements divided into levels. The levels represent indirection between
the measurements of a given level and direct measurements.\footnote{The results of evaluation
of over range has to be smaller than 10\%.
\footnote{SS set with only one element is possible if the user has specified in *DDD.Min.suff* that probability
of over range has to be smaller than 10\%.\footnote{The concept of indirection level was introduced in Figure 3.12 presented in Section 3.4.1 and is also depicted in Figure 4.3.}}}
Algorithm 3 Adaptive drill-down  

**Input:** $MG$ - monitoring strategy; a single $BN$; $n$ services and $m$ measurements represented by the provided $BN$.

**Ensure:** When finished, $DDC$ condition does not match the state of the system

1: while $(DDC = true) \lor \left(DDD.\text{Min\_suff}(\text{all sentinels})\right)$ do  
2: $\text{Suspicious} \leftarrow \emptyset$  
3: for $lev = 0 \rightarrow \text{maximal\_level}$ do  
4: // $lev$ represents level of indirection to direct measurements – cf. Figure 4.3  
5: for each $M_i$ on indirection level $lev$ do  
6: if $M_i$ is active then  
7: $\text{expected\_range}(M_i) \leftarrow \text{exact\_range}(M_i)$  
8: if $M_i$ is measurement cumulative then  
9: $\text{parentsays\_range}(M_i) \leftarrow \text{mcum formula on expected\_range of parents}$  
10: end if  
11: else  
12: if $M_i$ is measurement direct then  
13: $\text{expected\_range}(M_i) \leftarrow \text{nominal\_range}(M_i)$  
14: else if $M_i$ is measurement cumulative then  
15: $\text{expected\_range}(M_i) \leftarrow \text{mcum formula on expected\_range of parents}$  
16: $\text{exact\_range}(M_i) \leftarrow \text{expected\_range}(M_i)$  
17: end if  
18: end if  
19: end for  
20: end for  
21: for each active $M_i : \text{parentsays\_range}(M_i) \neq \text{expected\_range}(M_i)$ do  
22: for each $M_c$ child of $M_i : M_c$ is not active do  
23: if $M_c$ is measurement direct then  
24: add service of $M_c$ to $\text{Suspicious}$ set  
25: end if  
26: recursion : perform this for loop again for each non active child of $M_c$  
27: end for  
28: end for  
29: recalculate each CPT referring to measurement which exact range has changed  
30: $\text{DS\_best} \leftarrow \emptyset$  
31: repeat  
32: $\text{DS\_prev} \leftarrow \text{DS\_best}$  
33: for each $S_i$ in $\text{Suspicious} : S_i \notin \text{DS\_prev}$ do  
34: $\text{DS\_level} \leftarrow \text{copy of } \text{DS\_prev}$  
35: add $S_i$ to $\text{DS\_level}$  
36: calculate expected entropy of $\text{DS\_level}$ on $DDD.\text{Most\_prob}$ most probable evidence combinations  
37: calculate overhead of $\text{DS\_level}$  
38: assign $\text{fitness}$ to $\text{DS\_level}$ by means of $DDD.\text{Fitness}$ function  
39: $\text{DS\_best} \leftarrow \text{Max\_fitness}(\text{DS\_best}, \text{DS\_level})$  
40: end for  
41: until $DDD.\text{Velocity}(\text{DS\_best})$ returns true  
42: activate all measurements of services in $\text{DS\_best}$  
43: detect all potential anomalies revealed by activated measurements  
44: end while
of $expected\_range$ and $parentsays\_range$ in the analyzed example are as follows (active measurements are marked in gray):

<table>
<thead>
<tr>
<th>measurement</th>
<th>level</th>
<th>$expected_range$</th>
<th>$parentsays_range$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MC^{B}<em>{RT}(S_1)$ = $MC^{B}</em>{RT}(S_4)$ + $MD_{RT}(S_1)$</td>
<td>3</td>
<td>[1070ms, 1230ms]</td>
<td>[570ms, 730ms]</td>
</tr>
<tr>
<td>$MC^{B}<em>{RT}(S_2)$ = $MD</em>{RT}(S_3)$ + $MD_{RT}(S_2)$</td>
<td>1</td>
<td>[450ms, 550ms]</td>
<td>[450ms, 550ms]</td>
</tr>
<tr>
<td>$MC^{B}<em>{RT}(S_4)$ = $MC^{B}</em>{RT}(S_2)$ + $MD_{RT}(S_4)$</td>
<td>2</td>
<td>[490ms, 610ms]</td>
<td>n/a</td>
</tr>
<tr>
<td>$MD_{RT}(S_1)$</td>
<td>0</td>
<td>[80ms, 120ms]</td>
<td>n/a</td>
</tr>
<tr>
<td>$MD_{RT}(S_2)$</td>
<td>0</td>
<td>[180ms, 220ms]</td>
<td>n/a</td>
</tr>
<tr>
<td>$MD_{RT}(S_3)$</td>
<td>0</td>
<td>[270ms, 330ms]</td>
<td>n/a</td>
</tr>
<tr>
<td>$MD_{RT}(S_4)$</td>
<td>0</td>
<td>[40ms, 60ms]</td>
<td>n/a</td>
</tr>
</tbody>
</table>

When $expected\_range$ and $parentsays\_range$ are evaluated, the algorithm analyzes them, in another for loop, to identify suspicious services – lines 21-28. The for loop iterates over each measurement that $parentsays\_range$ differs from $expected\_range$ and performs a recursion on all non-active children. The results of this for loop in the two considered cases of the $SS$ set are as follows:

- $SS = \{S_1, S_2\}$ – Suspicious = \{S_4\};
- $SS = \{S_1\}$ – Suspicious = \{S_2, S_3, S_4\};

As can be seen, the more services are monitored in the stale period, the easier it is to identify suspicious services in the drill-down process. At the end of suspicious services identification (line 29), the algorithm recalculates all CPTs (Conditional Probability Tables) referring to the measurements in which the exact range has changed in a given iteration of the while loop (either because of an anomalous range detected in some active measurement or because of the assignment of $expected\_range$ in line 16 of the algorithm). Recalculation is performed by simply repeating the sampling algorithm presented in Section 3.4.2.

After the identification of suspicious services is finished, the algorithm begins identifying the best drill-down sentinels (referred to by the $DS_{best}$ symbol). The identification is performed in a repeat loop (lines 31-41). In each iteration of this loop, one of the suspicious services is chosen for the next best drill-down sentinel. The service is chosen by calculating the fitness function for all candidates from the set of suspicious services. The calculation is enclosed in the inner for loop (line 33), where best drill-down sentinel selection is performed with the use of the $Max\_fitness$ operator (line 39), which – from two provided sentinel sets – returns the one with the higher $fitness$ value.

It should be noted that in the case of the drill-down, entropy cannot be calculated in a straightforward way. This is caused by the fact that it can no longer be assumed that all measurements are in a known state (the nominal state in the case of Algorithm 2). Because of this, instead of entropy, the expected entropy value has to be calculated over all possible state combinations of measurements related to the services in the $DS_{level}$ set.
Unfortunately, the number of possible combinations increases exponentially along with the number of measurements. Therefore a heuristic approach is assumed, in which only a given number, specified in the $DDD.Most.prob$ drill-down driver parameter, of the most probable state combinations is taken into account during calculation of expected entropy value. The formula for the expected entropy is as follows:

$$EE = \sum_{i=1}^{n} P(C_i) E(C_i)$$

where:

- $n$ – $DDD.Most.prob$ parameter;
- $C_i$ – state combinations sorted in the descending probability order;
- $P(C_i)$ – probability of a given combination;
- $E(C_i)$ – entropy of a given combination.

The number of iterations in the discussed repeat loop (lines 31-41) is determined by the $DDD.Velocity$ drill-down driver parameter. If only one single iteration of this loop is performed, then only the measurements of a single service will be activated in line 42 of the algorithm. This is the most conservative approach: make only a small step towards root cause identification and see if it was right. In this way, the drill-down takes the most sensible step in each iteration of the while loop (line 1). However, such approach results in a slower execution of the drill-down. After activation of each new measurement, the minimal period (specified in the monitoring goal) has to elapse before the valid monitoring data is acquired. In order to limit the times in which the valid monitoring data is awaited, the user can manipulate the $DDD.Velocity$ parameter to perform more iterations of the repeat loop at once (which can be interpreted as increasing the drill-down velocity). The more iterations of the repeat loop, the more aggressive is the drill-down in pointing sentinels, which could allow for root cause identification. However, higher velocity comes at the cost of precision – decisions about measurement activation may not be as accurate as the ones taken at a lower velocity. The $DDD.Velocity$ parameter can be formulated in several different ways, for example:

- Perform a specified number of repeat loop iterations;
- Keep performing the repeat loop until the probability or the expected entropy of some measurements reaches a specified value;
- Keep performing the repeat loop until the expected entropy is lower than the entropy established by Algorithm 2 in the stale period.

The example described in the context of suspicious services identification is now continued. The case in which $SS = \{S_1, S_2\}$ is not taken into account because it results in $Suspicious = \{S_4\}$. This determines from the beginning that the drill-down will select service $S_4$ as the sentinel. In the case of $SS = \{S_1\}$, there are three suspicious services:
{S₂, S₃, S₄}. Dealing with this situation is now analyzed, with the assumption that the fitness function is the same as in Algorithm 2, i.e. \( \text{DDD.Fitness(entropy, overhead)} = \text{entropy} \). The measurement probabilities before the iterations of the repeat loop (line 31), in the first iteration of the while loop (line 1), are presented in Table 4.3. The anomalous range detected in \( MCP_{RT}^B(S_1) \) (set as the new exact range in the course of suspicious services identification), causes almost uniform increase of the over range probability in mdirects of suspicious services (S₂, S₃, S₄). It is caused by the fact that \( MCP_{RT}^B(S_1) \) provides the information that there is some anomaly but does not give a clue about its possible location – an anomaly in all suspicious services is equally probable.

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>( MCP_{RT}^B(S_1) )</td>
<td>0,00 %</td>
<td>100,00 %</td>
<td>0,00 %</td>
<td></td>
</tr>
<tr>
<td>( MCP_{RT}^B(S_2) )</td>
<td>9,62 %</td>
<td>12,11 %</td>
<td>78,27 %</td>
<td></td>
</tr>
<tr>
<td>( MCP_{RT}^B(S_4) )</td>
<td>0,00 %</td>
<td>0,00 %</td>
<td>100,00 %</td>
<td></td>
</tr>
<tr>
<td>( MDP_{RT}(S_1) )</td>
<td>0,00 %</td>
<td>100,00 %</td>
<td>0,00 %</td>
<td>0,00</td>
</tr>
<tr>
<td>( MDP_{RT}(S_2) )</td>
<td>23,26 %</td>
<td>27,11 %</td>
<td>49,63 %</td>
<td>1,04</td>
</tr>
<tr>
<td>( MDP_{RT}(S_3) )</td>
<td>21,89 %</td>
<td>27,29 %</td>
<td>50,81 %</td>
<td>1,03</td>
</tr>
<tr>
<td>( MDP_{RT}(S_4) )</td>
<td>25,27 %</td>
<td>26,34 %</td>
<td>48,39 %</td>
<td>1,05</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cumulative mdirect entropy: 3,12</td>
</tr>
</tbody>
</table>

Table 4.3: Measurement probabilities of response time metric in Bayesian network of the sample application scenario for situation presented in Figure 4.3, when \( SS = \{S_1\} \) and \( \text{Suspicious} = \{S_2, S_3, S_4\} \) (activated mprobs are highlighted in the first column)

Two cases are considered: (i) the \( \text{DDD.Velocity} \) parameter specifying one iteration of the repeat loop (line 31); (ii) the \( \text{DDD.Velocity} \) parameter specifying two iterations of the repeat loop (line 31). In both cases the \( \text{DDD.Most.prob} \) parameter is considered irrelevant for such a small BN and the \( \text{DDD.Min.suff} \) parameter is assumed to be the same as in the example discussed in the previous section (not allowing over range probability higher than 5%). In the first case each suspicious service is placed in a drill-down sentinel set (\( DS \) symbol) and entropy is calculated. The results of the repeat loop, sorted by the expected entropy, are as follows:

- \( DS = \{S_2\} \) – expected entropy = 1,53;
- \( DS = \{S_3\} \) – expected entropy = 2,01;
- \( DS = \{S_3\} \) – expected entropy = 2,03.

Activating service \( S_2 \) is the best choice. It is expected that the activation of this service will lower the system uncertainty in the most significant way. After the activation of service \( S_2 \) (line 42), it is revealed that this service was not the cause of the anomaly. The measurement probabilities after subsequent identification of suspicious services – in this case, service \( S_1 \) – are presented in Table 4.4. As can be seen, service \( S_4 \) is pointed to be in the over range with 100% probability. The activation of service \( S_2 \) allowed for a straightforward identification of service \( S_4 \) as the root cause of the anomaly detected.
Chapter 4. Monitoring Process Realization

Table 4.4: Measurement probabilities of response time metric in Bayesian network of the sample application scenario for situation presented in Figure 4.3 at second drill-down iteration when $SS = \{S_1\}$, $DS_{best} = \{S_2\}$ and $Suspicious = \{S_4\}$ (activated mprobs are highlighted in the first column) at the end of the stale period.

In the second case of the $DDD.Velocity$ parameter, the results of the repeat loop (with two services in the $DS$ set), sorted by the expected entropy, are as follows:

- $DS = \{S_2, S_4\}$ – expected entropy = 0.66;
- $DS = \{S_2, S_3\}$ – expected entropy = 0.87;

Starting monitoring of services $S_2, S_4$ is the best choice from the entropy minimization point of view. It should be noted that there is no $DS = \{S_3, S_4\}$ case, because service in the second repeat loop iteration is added to the results of the previous iteration, i.e. $DS = \{S_2\}$. Drill-down in the second case of $DDD.Velocity$ parameter is more aggressive – two drill-down sentinels are chosen. However the analysis of the first case has shown that it is sufficient to activate only one service, i.e. $S_2$ to identify $S_4$ as the anomaly root cause. As stated before, drill-down with more repeat loop iterations is faster but at the cost of being less accurate, or in this case introducing unnecessary monitoring overhead (activation of $S_4$ was not needed).

It is important to understand that the drill-down is not limited to diagnosing a single anomaly. Even if multiple anomalies are detected at some point of the stale period, the whole algorithm can perfectly support such a situation. Both identification of suspicious services and identification of best drill-down sentinels are agnostic to the number of anomalies present in the processed Bayesian network.

4.4 Summary

This chapter presented the chosen realization of the monitoring process built on the foundation of the adaptive monitoring concept. The process realization is defined by two major elements: the monitoring goal strategy and the control loop.
The proposed structure of the monitoring goal strategy allows for addressing all layers of the adaptive monitoring concept by specifying where, what and when should be monitoring as well as how the monitoring process should be adapted. The anatomy loop is divided into three phases: nominals identification, sentinels selection and adaptive drill-down. Each phase has a respective algorithm describing the details of the phase logic. The detailed presentation and analysis of all three control loop algorithms allows to state that the algorithms fulfill general expectations formulated in the control loop anatomy – Section 4.2. Algorithm 1 identifies nominal ranges needed in the second phase for the execution of Algorithm 2. Algorithm 2 selects sentinels that minimize the monitoring overhead in the stale period without increasing the system uncertainty in a significant way. Finally, Algorithm 3 performs the adaptive drill-down, which identifies the root cause of anomalies and allows for regaining the balance of the stale period. All three algorithms can be precisely controlled by various parameters specified in the steering directives of the monitoring goal strategy.

The proposed process made certain assumptions about the realization of the monitoring mechanisms and the high-level architecture. These aspects are addressed in the next chapter, which presents the design of the adaptive monitoring framework supporting the concept proposed in the previous chapter and the monitoring process realization – the contribution of this chapter.
Chapter 5

DYNAMIC ADAPTIVE MONITORING FRAMEWORK

After defining the adaptive monitoring concept and proposing a concrete monitoring process realization, there is a need for designing a framework that can be installed in an existing dynamic SOA system for the purpose of providing it with adaptive monitoring features. The framework is expected to comply with the assumptions formulated on the conceptual level (Chapter 3) and on the level of the monitoring process (Chapter 4). In order to fulfill this expectation, several inter-related aspects have to be resolved. The first aspect is choosing the appropriate technique for enriching the underlying SOA system. To provide a sound solution, the selected technique has to ensure that the enrichment can be performed in a non-intrusive manner and that its realization is not dependent on the existing business logic. The second aspect is proposing the construction of mechanisms introduced in the adaptive monitoring concept (Section 3.2), i.e. the mechanisms of discovery and measurement. The last aspect is the architectural view describing the details of the framework components and their interactions. All three identified aspects are directly addressed by Dynamic Adaptive Monitoring Framework, referred to as DAMON, the description of which constitutes the main contribution of this chapter.

The structure of this chapter is as follows. The first section analyzes both functional and non-functional requirements of the framework. The second section discusses the enrichment of the SOA system covering the concept of interceptor socket (a realization of the interceptor pattern in dynamic SOA systems) and the selection of the most suitable instrumentation approach. The third section presents the design of both discovery and measurement mechanisms, covering the aspects of topology, causality and invocation analysis. The forth section presents the architecture of the proposed DAMON framework, which, among others, presents how the monitoring mechanisms are integrated with other framework facilities.
Chapter 5. Dynamic Adaptive Monitoring Framework

5.1 Framework Requirements

Current SOA environments, a subset of which was reviewed in Section 2.2, do not offer any comprehensive features of adaptive monitoring. This issue is addressed by the proposed Dynamic Adaptive Monitoring Framework, the goal of which is to provide the existing environment with the capability of monitoring process adaptation. The framework has to face two types of requirements. First type is related to the contents of the previous chapters. The different layers of adaptive monitoring concept and the realization of the control loop expect certain features such as at run-time mechanisms reconfiguration and sufficient system scalability. The second requirements type is related to general non-functional framework characteristics, which make DAMON attractive to a potential user. All these requirements are summarized in the following list.

R1 – The framework has to implement the monitoring mechanisms according to the expectations of the adaptive monitoring concept (cf. Section 3.2). In the context of the SOA system model layer, topology discovery, elements life cycle tracing and causality identification have to be covered. In turn, in the context of the measurement layer, there is a need to handle the acquisition of metric values, overhead evaluation and calculation of the related metric ranges.

R2 – the monitoring mechanisms have to extract data from the existing SOA environment, therefore the environment has to be appropriately enriched. Adding enrichment and its removal should be performed in a fully automated way. Moreover, to make the enrichment process as seamless as possible, any intrusiveness and coupling with the business logic have to be avoided.

R3 – The construction of the monitoring process control loop presented in Chapter 4 requires the monitoring mechanisms to be reconfigurable at run-time. This is needed for achieving a manageable monitoring selectivity – the essential added value that allows for limiting the overhead incurred by the monitoring process.

R4 – There is need to propagate some additional information along with service invocation. According to the assumptions formulated at the end of Section 4.2.2 the measurement mechanisms should propagate the information required to calculate mcum values. Propagating the information of another type is needed by discovery mechanisms in order to identify invocation realization. The design of the propagation feature has to ensure transparency and limited overhead.

R5 – This dissertation proposes a declarative management of the monitoring process by means of a high-level monitoring goal. The framework has to support the translation between a high-level goal description and a low-level configuration of the monitoring mechanisms.
Chapter 5. Dynamic Adaptive Monitoring Framework

R6 – The framework architecture should support the deployment to a dynamic, distributed environment while at the same time maintaining the acceptable efficiency and scalability. This requirement is concerned both with the monitoring mechanisms directly influencing the environment and with the steering functionality covering the realization of BN inferencing.

The DAMON framework presented in this chapter addresses all the aforementioned requirements. R1 is covered in Section 5.3, where all monitoring mechanisms are described in detail. R2 is addressed by a hybrid approach to SOA environment enrichment discussed in Section 5.2.1. R3 and R4 are considered by the concept of interceptor socket proposed in Section 5.2.2. Finally, R5 and R6 are taken into account in the context of framework architecture presented in Section 5.4. Additionally, a subset of indicated requirements, the fulfillment of which has to be verified in practice, is considered in the framework evaluation presented in Chapter 7.

5.2 SOA System Enrichment

In Adaptive SOA Solution Stack (AS3), proposed by Zielinski et al. [2], realization of the MAPE-K loop [59] assumes that sensors and effectors are added to the managed resource by means of instrumentation. The DAMON framework follows this proven approach and uses instrumentation to enrich the SOA system with the mechanisms necessary for the realization of the monitoring process control loop presented in Chapter 4. The following two sections describe possible instrumentation approaches, the selection of the most suitable one and the concept of interceptor socket allowing for a dynamic acquisition of information from the instrumented elements.

5.2.1 Selection of the Instrumentation Approach

SOA-RA presented in Section 2.1.2 shows that there are at least several elements that could be subjected to instrumentation:

(i) services or a service container in the services layer and probably a service registry in the governance layer;

(ii) service components or service units in the service component layer;

(iii) elements of run-time environment provided by the operational system layer: solution components, solution environments, solution platforms.

The advantages and disadvantages of instrumenting the proposed elements in the context of the SOA dynamics explained in Section 3.1 are now discussed. In propositions (i) and (ii), instrumentation of the elements coupled with the business logic, i.e. services, service
components, and their aggregation – service units, implies that instrumentation can also be, to some extent, dependent on this logic. Such instrumentation would have to be performed partially manually by the developer of services and service components. In the context of the increased SOA dynamics and frequency of changes in the business logic, it is definitely inconvenient. Instrumentation of infrastructural elements such as service container or service registry does not have the problem of business logic coupling, but could require a modification of the source code, which is not always possible. The safest approach would be to instrument the run-time (proposition (iii)), but then influencing the service and service components would need some binding with services and service components layers.

Each discussed instrumentation option has some disadvantages that are not acceptable in the context of requirement R2. Therefore, this dissertation assumes a hybrid approach, which is as follows. Instrumentation is performed on a transition between the service container and the solution environment – the run-time instance of the service container (cf. Section 2.1.2). This approach was developed in the scope of the author’s previous research [118], which assumes that the container is enriched with the instrumentation during the start-up process. Instrumentation is weaved into the run-time binaries with the use of LTW techniques developed by the AOP [119]. The proposed approach does not need access to the source code and ensures that instrumentation is agnostic to the business logic of services. This allows for full automation of the instrumentation as expected in requirement R2.

5.2.2 Interceptor Socket Concept

The instrumentation approach presented in the previous section is capable of fulfilling requirement R2, however current AOP features do not allow for changing aspect weaving at run-time without imposing some additional problems [18]. Therefore, compliance with requirement R3, run-time mechanisms reconfiguration, is not straightforward. The proposed concept of interceptor socket [120] aims at fulfilling R3 by adapting the well-known interceptor pattern [121] to the context of dynamic SOA systems.

According to Schmidt et al. [121], the interceptor pattern allows for a transparent extension of the existing environment by adding interceptors complying with expected interface. When a certain situation occurs, the environment notifies the interceptors with the appropriate information. The pattern consists of the following important compounds:

**Interceptor interface** – defines the signature of hook methods that will be invoked by the dispatcher upon the occurrence of relevant situations;

---

18 This AOP limitation is mentioned in the publications of Ehlers [102] and Okanovic et al. [105], discussed in Section 2.4.2.
**concrete interceptor** – implements hook methods of the interceptor interface to handle relevant situations according to the implemented logic;

**dispatcher** – allows the application to attach and detach concrete interceptors; typically there is a single dispatcher for each interceptor interface;

**context** – provides information about a concrete framework and allows for its reconfiguration;

**application** – the business logic executed inside a concrete framework, which can implement concrete interceptors and register them in dispatchers;

**concrete framework** – provides a generic, extensible environment allowing for the deployment of the application, e.g. ORB, a Web server or an application server.

The aforementioned compounds are mapped to the SOA environment. The mapping is presented in Table 5.1. The most important assumption of this mapping is that the interceptor is provided as a service (it is referred to as i-service). This allows the pattern realization to directly benefit from the following dynamic SOA aspects presented in Section 3.1: (i) deployment of i-services to the SOA environment through service units; (ii) dynamic exposition of i-services performed at run-time; (iii) dynamic discovery of i-services with the use of the discovery provided by the service registry. The main purpose of the interceptor socket (referred to as i-socket) is to intercept service invocations occurring in the service container. The interception is used by the monitoring mechanisms for the acquisition of monitoring data. The presented mapping ensures that the interception process can be easily reconfigured at run-time, which allows for fulfilling requirement R3.

The interceptor socket is an entity that wraps the existing service and intercepts all invocations coming in and out of the service. The creation of an i-socket is realized by leveraging the selected instrumentation approach described in the previous section. Instrumentation is weaved into the logic of the service container responsible for service exposition. Whenever a new service is exposed in the container, instrumentation ensures that a new i-socket is created. The i-socket wraps the actual service and mimics its

<table>
<thead>
<tr>
<th>Interceptor pattern compounds</th>
<th>SOA system meta-model</th>
</tr>
</thead>
<tbody>
<tr>
<td>interceptor interface</td>
<td>abstract interceptor service</td>
</tr>
<tr>
<td>concrete interceptor</td>
<td>interceptor service (i-service)</td>
</tr>
<tr>
<td>dispatcher</td>
<td>interceptor socket (i-socket)</td>
</tr>
<tr>
<td>context</td>
<td>parameters of service invocation</td>
</tr>
<tr>
<td>application</td>
<td>solution component</td>
</tr>
<tr>
<td>concrete framework</td>
<td>service container</td>
</tr>
</tbody>
</table>

Table 5.1: Mapping interceptor pattern compounds to the SOA environment modeled by the SOA system meta-model
behavior. Therefore, whenever the service is invoked, the invocation comes to the i-socket, which after appropriate processing passes the invocation to the wrapped service.

The interceptor socket contains two internal components: service observer and interceptor chain, which allow the i-socket to interact with interceptor services. Such interaction, occurring in the service container, is presented in Figure 5.1. The interaction is based on the mentioned run-time discovery provided by the service registry. When a new i-socket is created, its service observer registers itself in service registry for notifications related to adding and removing i-services. The interceptor service, appropriately packaged in the service unit, can be deployed to the container at any time. Then, the solution component (run-time counterpart of the service unit) can expose the interceptor service. The whole concept of interceptor socket assumes that the exposed interceptor service declares two important pieces of information: (i) the attributes specifying the interests in some i-sockets and (ii) the priority according to which the i-service should be processed. The exposition of the i-service is instantly detected by the service registry. This allows the service registry to notify the service observers of all i-sockets present currently in the container. Then, the service observers evaluate the i-service interests to check if they match the i-socket. If the interests are matching, then the reference of the i-service is added to the interceptor chain according to the priority specified by the i-service.

**Fig. 5.1: Interaction between the interceptor socket and interceptor services in a dynamic SOA environment**
Fig. 5.2: Interception performed by the i-socket which wraps an asynchronous service invoking two synchronous services

The process of service interception is presented in Figure 5.2. Interception is performed by the i-socket which wraps an asynchronous service invoking two synchronous services. As it is depicted in the figure, it is not only initiating and terminating invocations of a given service that are intercepted, but also all other invocations performed by the service. Each intercepted invocation is passed to the interceptor chain, where each i-service handles the invocation in a specific way. Two types of i-services are considered: **agnostic i-service** providing system-level information and **business i-services** providing high-level information related to the details of the business logic.

In order to fulfill requirement **R4**, the interceptor socket provides a feature of invocation flow enrichment. As it is shown in Figure 5.2, invocations going in and out of the interceptor socket can be enriched by the i-socket itself and by the i-services present in its chain. Two types of enrichment are available: (i) **local enrichment**, which is valid only during a single chain transition and never leaves the i-socket; (ii) **exported enrichment**, which is propagated outside the i-socket. The first type is used to communicate between the i-socket and its i-services and between the i-services themselves, the references of which are present in the same chain. The second type is used to communicate between the i-services and the i-sockets of services belonging to the same flow (an element of the causal plan of the **SOA** system model). In the scope of local enrichment, the i-socket provides the i-services with the following information:

- service sync type: synchronous/asynchronous;
- invocation type: singular/parallel and synchronous/asynchronous;
- invocation realization: initiating, terminating;
- invocation target/source: information about the other end of invocation.
On the basis of the information passed from the i-socket, the i-service is able to ignore some invocations (e.g. the i-service is interested only in terminating the invocation and ignores all other invocations). In order to ensure that the actual service contract of a wrapped service is fulfilled, the enrichment is always removed from the invocation before it is passed to the service. After the invocation transits out of service, it is enriched again. Such approach ensures transparency, which is expected in the context of requirement R4.

5.3 Design of Monitoring Mechanisms

Since the SOA system layer involves two separate planes: topology and causality, discovery mechanisms are also divided into mechanisms focusing on topology and focusing on service flows – a crucial element of causality. Both types of discovery mechanisms are presented in the two following sections.

5.3.1 Topology Discovery

The design of topology discovery involves two separate components: topology retriever and topology follower. The purpose of the topology retriever is to extract the current state of topology, when requested. The component is realized as a service unit deployed to all monitored containers. It uses the adapter design pattern [122] to mediate between the topology plane meta-model and the format of topology information exposed natively by a particular SOA run-time. It is assumed that each SOA run-time always provides at least some subset of topological information such as a list of deployed applications, services or components. There is one instance of topology retriever per container of federation. When the component connects to the monitoring center, it informs about the presence of its container.

The purpose of the topology follower is to track topology changes. The component is realized as a set of business i-services that are plugged into the i-sockets of services related to managing the topology. It is assumed that each SOA run-time has to have services which allow for the following: installation of service units, exposition of services, managing the life cycle of the service container and the elements deployed to the container. After the identification of such services, the appropriate business i-services are constructed and deployed to the monitored containers. Each invocation of some topology management service is intercepted and the information about the actual event representing a change of the topology model is extracted. Whenever the event is needed for some monitoring goal strategy, it is dispatched to the monitoring center in an asynchronous manner. Thanks to asynchronous dispatching, the operation of topology management services is not influenced.

The combination of the topology retriever and topology follower components allows for
acquiring complete knowledge about the topology. After the activation of a monitoring strategy requesting some topological information, firstly, in each related service container, the topology retrievers are used for extracting the initial topology state. Then, topology followers are used for tracking all topology changes and keeping the valid representation of the topology state in the topology model.

5.3.2 Causality Identification

The design of causality discovery involves agnostic i-services that are plugged into the i-sockets of all services related to a given monitoring goal strategy. The main purpose of these i-services, referred to as flow sniffers, is detecting any new (not detected before) service flows, identifying invocation details important for the causality plane (not yet identified by the interceptor socket) and reporting all of them to the monitoring center. The interceptor socket already identifies most of the causality information: service sync type, invocation type, invocation realization (but only initiating and terminating) and invocation target/source. The flow sniffer has to identify only the invocation realizations not covered by the i-socket, i.e. the consumed, caused and causing realizations. The definitions of these realizations were introduced in Section 3.3.2, but they are now repeated for the convenience of the reader. A consumed invocation of service $X$ was defined in the following way:

Either a synchronous invocation of another service performed by $X$, or an asynchronous invocation of a service $X$, performed by another service, e.g. service $Z$, under the condition that the invocation is terminating for $Z$.

Therefore, to identify the consumed invocations, the sniffer checks the invocation type (synchronous/asynchronous) and in case an asynchronous invocation is performed by other service, it checks the invocation realization detected by the sniffer of that service, which is expected to be passed by means of the exported enrichment of the invocation (interceptor socket feature). Once the consumed invocations are identified, identification of the caused ones is straightforward because caused invocations are all non-terminating non-consumed invocations performed by a given service. A causing invocation of service $X$ was defined in the following way:

The most recent invocation which either directly or indirectly causes execution of service $X$ logic and which is invoked by a service that logic is not terminated at the time of initiating service $X$.

The definition of causing invocation is recursive, therefore in order to identify it the sniffer enriches (exported enrichment type) all intercepted invocations performed by its service with the currently known causing invocation. If the performed invocation is non-terminating, then the currently known causing invocation is the one that is now executed (service logic is not yet finished). If the performed invocation is terminating,
then the currently known causing invocation is the one passed in the enrichment of the initiating invocation.

When the flow sniffer intercepts invocation on some i-socket, then in order to decide whether the detected invocation creates some new flow or not the sniffer needs two kinds of information: (i) information about the invocations performed so far in the intercepted instance of the service flow; (ii) historical information about service flows detected previously by this sniffer. The first kind of information can only be provided by other flow sniffers that intercepted the earlier flow invocations. The second kind of information can be acquired by introducing a component keeping the local history in a given flow sniffer. In both cases, there is an issue of representing the service flow. A complete representation involving all information about the services covered by the flow could be too large (especially in the case of significantly long flows) to be sent between the i-sockets and to be remembered locally in the flow sniffer. Therefore, it was decided that the flow is represented by a hash of a defined size\[^{19}\] which uniquely identifies the flow. The hash is recalculated by each flow sniffer on the basis of new detected invocations. Such a recalculated hash is passed to the next sniffer in the flow using the exported invocation enrichment of the interceptor socket described in the previous section.

The exact logic of flow detection performed by a single instance of the sniffer for a single group of causally linked invocations is presented in Algorithm 4. The component keeping the local history is referred to as $D$ set and the hash of the service flow passed between the flow sniffers is referred to as $H_{prev}$ (the name of the variable in the exported invocation enrichment). Additionally, it is assumed that two functions are provided by the actual flow sniffer implementation: $Hash$ – a service flow hashing function; $CausalLink$ – a function checking the causal relationship between two invocations. The identification of consumed, caused and causing invocations is performed in line 7 in accordance with the earlier description. The results of the identification together with the information provided by the interceptor socket is added as a local invocation enrichment under variable $LocalCD$ (line 8). In the case of outgoing invocation (invocation performed by the service to which the flow sniffer is attached), an additional exported invocation enrichment is performed. It is ensured that this enrichment covers $H_{prev}$ (line 12) and the causality details exported under variable $ExportedCD$ (line 13). The construction of the algorithm ensures that information about new flows will be sent to the monitoring center only if it has not yet been sent before. The information about new flows, kept in the $Sniffed set$, contains a complete trace of invocations detected by a given flow sniffer.

\[^{19}\]The hash can be created by any of the available well-proven hashing functions such as SHA-1\[^{123}\] or MD-5\[^{124}\].
Algorithm 4 Detection of causality performed by the flow sniffer

Assume: An instance of the flow sniffer algorithm is bound to a single intercepted service;
An algorithm instance is started by \( I_{init} \) – initiating the invocation of the intercepted service; \( D \) – a set containing hashes of flows already detected by this flow sniffer;
\( Hash \) – a hashing function used for unique representation of service flows;
\( CausalLink \) – a function checking if two given invocations have a causal link

Ensure: The information about causality is always propagated to other i-services through external enrichment and complete information about the flow execution in the context of intercepted service is reported on-demand to the monitoring center

1: \( Sniffed \leftarrow \emptyset \) // Temporal set for sniffed invocations
2: \( H_{cur} \leftarrow EEnrich(I_{init}).H_{prev} \) // Getting service flow hash from previous flow sniffer
3: Add \( H_{cur} \) to \( Sniffed \) set
4: for each intercepted invocation \( I : CausalLink(I, I_{init}) = true \land I \neq I_{init} \) do
5: \( H_{cur} \leftarrow Hash(H_{cur}, I) \)
6: Add \( I \) and \( H_{cur} \) to \( Sniffed \) set
7: \( CausalityDetails \leftarrow \) the information provided by the interceptor socket and the results of the consumed, caused and causing identification
8: \( LEnrich(I).LocalCD \leftarrow CausalityDetails \) // Local enrichment
9: \( LEnrich(I).H_{cur} \leftarrow H_{cur} \) // Local enrichment
10: if \( I \) is outgoing invocation then
11: \( EEnrich(I).H_{prev} \leftarrow H_{cur} \) // Ensures that the next sniffer gets \( H_{prev} \)
12: \( EEnrich(I).ExportedCD \leftarrow CausalityDetails \)
13: end if
14: if \( I \) is terminating then
15: if \( H_{cur} \notin D \) then
16: Send \( Sniffed \) set to the monitoring center
17: Add \( H_{cur} \) to \( D \) set
18: end if
19: end if
20: end if
21: end for

5.3.3 Measurement Acquisition

Measurement mechanisms are expected to acquire the data needed for creating an instance of the measurement meta-model, which is built around the concept of the measurement entity. As defined in Section 3.4.1 the purpose of measurement, which is always bound to a particular metric (system or business one) of a given service, is to describe the state of its metric by claiming that metric values belong to a range referred to as the exact range\(^{20}\). Measurement can be in two states: active and inactive. Only when measurement is activated the information about exact range should be sent to the monitoring center, which introduces an overhead to the operation of the measurement’s service.

\(^{20}\)There are also under and over ranges, but they are more related to mapping measurements to the nodes of Bayesian networks.
Chapter 5. Dynamic Adaptive Monitoring Framework

The design of measurement mechanisms is based on the *measurement interceptor* component (referred to as *m-interceptor*), which is realized as an i-service. The purpose of the m-interceptor is to provide the following capabilities for a single service: (i) gathering, processing and propagating the metric values needed to calculate mcum metric values; (ii) identifying the exact ranges of both mdirects and mcums, identifying the overhead and passing all this information to the monitoring center. In order to support the concept of measurement activation, the m-interceptor, similarly to measurement, can also be in two states: active and inactive. By default, after a given m-interceptor is installed but before its activation, it realizes only capability (i). When the m-interceptor is activated, it starts to realize both capabilities (i) and (ii). The logic realizing capability (i) and covering some prerequisites needed to realize capability (ii) is presented in Algorithm 5 (Section 5.3.4), while the main logic related to capability (ii) is presented in Algorithm 6 (Section 5.3.5).

### 5.3.4 Core Logic of the Measurement Interceptor

Algorithm 5 is focused on the appropriate processing of metric values related to cumulative measurements (mcums). As defined in Section 3.4.1, one of the mcum compounds is a formula describing the aggregation of the consumed measurements and the mdirect itself. The mentioned section contains examples of various measurements and evaluations of mcum formulas performed on the following range-based representation of measurement: \( M = [LLB, LB, UB, UUB] \). Algorithm 5 leverages the fact that the mcum formula can be evaluated not only on the range-based data but also on a set of numbers. When each number from such a set represents either the metric value of the consumed measurement or the metric value of the mdirect, the output of the formula evaluation is the metric value of the cumulative measurement. Such evaluation is performed in line 29 of the algorithm.

Similarly to Algorithm 4, Algorithm 5 presents the core m-interceptor logic executed on a single group of causally linked invocations identified with the use of *CausalLink* function. This function performs the identification on the basis of the information about causality extracted from the local invocation enrichment provided by the flow sniffer. It is assumed that a single instance of the m-interceptor handles a single service and a single metric type (referred to as *Metric_id*). The fundamental functionality of the m-interceptor is measuring the metric value of the mdirect (lines 4, 15), which is also necessary to calculate mcum values. Depending on the metric type, e.g. response time, CPU time or some business metric, the respective acquisition mechanism is used by the m-interceptor. The acquisition is updated (line 4) on each invocation processed in the for loop (line 3) to consider only the processing time spent in the intercepted service and not the time spent on waiting for invoked services.

Calculation of mcums is performed with the use of two functions: *CalculateMcumFormula* and *EvaluateMcumFormula*. The first calculates the formula itself (line 25) on the
Algorithm 5 M-interceptor core logic

**Assume:**
- M-interceptor is bound to a given service $S_{id}$ and a given metric $Metric_{id}$;
- M-interceptor has a Formulas map where all calculated mcum formulas are kept;
- An algorithm instance is started by $I_{init}$ — initiating the invocation of the intercepted service;
- CausalLink — a function for checking if two given invocations have a causal link;
- CalculateMcumFormula and EvaluateMcumFormula functions are available;
- Upon activation, the m-interceptor is provided with a $dsc\_map$ map, which maps $M_{id}$ to measurement activation details ($Min\_per$ and $Exp\_per$).

**Ensure:** Gathering, processing and propagating the information needed for cumulative metric calculation is performed. Upon the m-interceptor activation, overhead is estimated and Algorithm 6 is triggered.

1: $Cons_{values} \leftarrow \emptyset$ // Metric values of consumed invocations
2: $Cons_{cd} \leftarrow \emptyset$ // Causality details of consumed invocations
3: for invocation $I = I_{init}$ and then each $I : CausalLink(I, I_{init}) = true$ do
4:   Update the direct metric value acquisition
5:   if m-interceptor is active then
6:     Update the m-interceptor overhead estimation
7:   end if
8:   LocalCD $\leftarrow$ LEnrich($I$).LocalCD // Gets local causality details
9:   if LocalCD states that $I$ is consumed invocation then
10:      // gathering data for mcum calculation
11:      Add EEnrich($I$).Mvalue($Metric_{id}$) to $Cons_{values}$ set
12:      Add EEnrich($I$).ExportedCD to $Cons_{cd}$ set
13:   end if
14:   if LocalCD states that $I$ is terminating invocation then
15:      $M_{value} \leftarrow$ Finish the direct metric value acquisition
16:      if m-interceptor is active then
17:         $M_{overhead} \leftarrow$ Finish the m-interceptor overhead estimation
18:         $M_{id} \leftarrow MD(S_{id})$ // direct measurement id
19:         MInterceptorNotification($M_{id}$, $dsc\_map(M_{id})$, $M_{value}$, $M_{overhead}$)
20:        Improve the m-interceptor overhead estimation
21:   end if
22:   if $Cons_{values} \neq \emptyset$ then // cumulative measurement is encountered
23:      // processing data for mcum calculation
24:      if LEnrich($I$).Hcur $\notin$ Formulas then
25:         new mcum_formula $\leftarrow$ CalculateMcumFormula($Metric_{id}$, LocalCD, Cons$_{cd}$)
26:         Formulas(LEnrich($I$).Hcur) $\leftarrow$ new mcum_formula
27:      end if
28:      mcum_formula $\leftarrow$ Formulas(LEnrich($I$).Hcur)
29:      $M_{value} \leftarrow$ EvaluateMcumFormula(mcum_formula, $M_{value}$, $Cons_{values}$)
30:      if m-interceptor is active then
31:         $M_{id} \leftarrow MC(LEnrich(I).H_{cur})$ // cumulative measurement id
32:         MInterceptorNotification($M_{id}$, $dsc\_map(M_{id})$, $M_{value}$, $M_{overhead}$)
33:        Improve the m-interceptor overhead estimation
34:      end if
35:   end if
36:   // propagating data for mcum calculation
37:   EEnrich($I$).Mvalue($Metric_{id}$) $\leftarrow$ $M_{value}$
38: end if
39: end for
basis of Metric\textsubscript{id}, local causality details (extracted in line 8) and the causality details extracted from the exported enrichment of the consumed invocations (line 12). The latter calculates the final mcum metric value by evaluating the mcum formula (line 29) on the measured mdirect metric value and the mcum values gathered from the consumed services (line 11). To make gathering mcum values from the consumed services possible in line 11, the final metric value \( M_{\text{value}} \) is propagated to the consuming service by means of exported enrichment in line 37. It should be noted that if an service does not consume any other services (checked in line 22), then simply the mdirect metric value is propagated in line 37. To avoid the invocation of \( \text{CalculateMcumFormula} \) each time, when the same group of invocations is intercepted, the calculated mcum formula is identified with the use of the service flow hash (extracted e.g. in lines 24, 26 from the local enrichment provided by the flow sniffer) and remembered in the Formulas map.

The rest of the m-interceptor logic deals with the situation when the m-interceptor is activated, what is checked in lines 5, 16, 30. After activation, the m-interceptor starts to estimate the overhead (lines 6–17) incurred on the intercepted service and starts to notify Algorithm 6 (performed in lines 19, 32 by means of the \( \text{MInterceptorNotification} \) function) about the measured metric values, mdirect in line 19 and mcum in line 32, and about the estimated overhead. The metric values passed to the \( \text{MInterceptorNotification} \) function are identified with some assumed measurement id – \( M_{id} \). In the case of the mdirect (line 18) the \( M_{id} \) is constructed from the service id (some unique identification of the service instance), while the mcum (line 31), similarly to the mcum formula, is identified with the service flow hash (\( H_{\text{cur}} \)) provided by the flow sniffer. Since several fragments of the m-interceptor logic rely on the information provided by the flow sniffer, the sniffer has to always be placed before the m-interceptor in the interceptor chain of the i-socket.

The overhead is represented as the amount of resources, which were consumed (such definition of overhead was introduced in Section 3.4.1) due to the interception of service invocations and reporting data to monitoring center. The design of the m-interceptor logic ensures that only the reporting activity, realized by the \( \text{MInterceptorNotification} \) function, can introduce a significant overhead. All logic of Algorithm 5 besides the invocation of this function, is based on simple conditions, assignments, arithmetic operations, and operations on typical in-memory collections (sets, maps), which do not consume CPU, network bandwidth nor RAM memory in an extensive manner. Therefore, after each invocation of \( \text{MInterceptorNotification} \), the overhead caused by the logic performed in all threads of Algorithm 6 is measured to improve future overhead estimations (lines 20, 33).

### 5.3.5 Ranges Identification and Reporting

The only entry-point to Algorithm 6 is the \( \text{MInterceptorFunction} \) function, which is called only by Algorithm 5. When this function is invoked for a given \( M_{id} \), the first
operation ensures that one instance of each thread provided by the algorithm: Exact Range Validation, Overhead Validation and Expiration runs on behalf of this $M_id$ (line 2). Each of these threads is related to a single instance of the delay queue collection. The delay queue is a common design pattern available e.g. in Oracle JDK since version 1.5 [125]. Oracle JDK describes the delay queue in the following way: "An unbounded blocking queue of Delayed elements, in which an element can only be taken when its delay has expired. The head of the queue is that Delayed element whose delay expired furthest in the past." [125]. Additionally, it is assumed that the tail of the queue is the element, which was most recently put into the queue. Algorithm 6 assumes the following semantics of delay queue methods.

```java
void put(delay, object_1, ..., object_N) – Puts objects 1, ..., N inside a single element in the queue with a delay equal to the first parameter.

object_1, ..., object_N take() – Waits until the head element of the queue is available and its delay has expired, then removes the head from the queue and returns all objects kept inside the head element.

object_1, ..., object_N peek() – If the queue is not empty, then instantly returns all objects kept inside the tail element (even if the delay of the tail element has not yet expired), but does not remove it. If the queue is empty, then null value is returned.
```

When it is ensured that appropriate threads are running on behalf of given $M_id$, the metric value and the overhead value are put to the $expiration\_queue$ with the delay equal to $Dsc.\_Exp\_per^{21}$ (line 3). The metric and overhead values are added to two respective sets (lines 4 and 5), which are then used by the $UpdateExactRangeAndOverhead$ function. This function uses two mentioned sets to calculate the exact metric range (line 9) and the overhead range (line 13). When the exact range or the overhead range changes, it is put to $validation\_exact\_queue$ or $validation\_overhead\_queue$ respectively with the delay equal to $Dsc.\_Min\_per/10^{22}$ (lines 11, 21). These two delay queues are used to avoid sending too many updates to the monitoring center. When one of the mentioned ranges change (or the $upd\_cnt$ is a multiple of 10), then new range is added to respective queue (lines 10 - 16). Only when the delay of given element expires, either the Exact Range Validation or the Overhead Validation thread takes such an element from the respective queue and reports the related range to the monitoring center (lines 21, 22, 27, 28). The expiration of values reported through $M\_Interceptor\_Notification$ is handled by the Expiration Thread. When given values expire in accordance with the $Dsc.\_Exp\_per$ period, they are taken from the $expiration\_queue$ (line 33) and removed from the respective sets (lines 34, 35). Then, the ranges are updated through the $UpdateExactRangeAndOverhead$ function.

---

21 $Dsc.\_Exp\_per$ is the symbol of the exact range expiration period defined in Section 4.3.

22 $Dsc.\_Min\_per$ is the symbol of the minimal period defined in introduction to Section 4.3.
Algorithm 6: Identification and reporting the measurement exact range and overhead

**Assume:** For $M_{id}$ overhead is in $Overheads(M_{id})$ and metric values are in $Values(M_{id})$;
- $validation\_exact\_queue(M_{id})$ – a delay queue for the validation of exact ranges;
- $validation\_overhead\_queue(M_{id})$ – a delay queue for the validation of overhead;
- $expiration\_queue(M_{id})$ – a delay queue for expiration of metric values and overhead

**Ensure:** Changes of the measurement exact range and overhead incurred by m-interceptor activation are identified and reported on-demand to monitoring center

1: function $M\_Interceptor\_Notification(M_{id}, Dsc, M_{value}, M_{overhead})$
2: Ensure that one Exact Range Validation, one Overhead Validation and one Expiration thread is spawned for $M_{id}$
3: expiration\_queue($M_{id}$).put($Dsc.Exp\_per, M_{value}, M_{overhead}$)
4: Add $M_{value}$ to $Values(M_{id})$ set
5: Add $M_{overhead}$ to $Overheads(M_{id})$ set
6: UpdateExactRangeAndOverhead($M_{id}, Dsc$)
7: end function

8: function UpdateExactRangeAndOverhead($M_{id}, Dsc$)
9: $exact_{\ new} \leftarrow [\min\{Values(M_{id})\}, \max\{Values(M_{id})\}]$
10: if $exact_{\ new} \neq validation\_exact\_queue(M_{id}).peek() \lor upd\_cnt\%10 = 0$ then
11: $validation\_exact\_queue(M_{id}).put(Dsc.Min\_per/10, exact_{\ new})$
12: end if
13: $ohead_{\ new} \leftarrow [\min\{Overheads(M_{id})\}, \max\{Overheads(M_{id})\}]$
14: if $ohead_{\ new} \neq validation\_overhead\_queue(M_{id}).peek() \lor upd\_cnt\%10 = 0$ then
15: $validation\_overhead\_queue(M_{id}).put(Dsc.Min\_per/10, ohead_{\ new})$
16: end if
17: $upd\_cnt \leftarrow upd\_cnt + 1$
18: end function

19: thread Exact Range Validation for $M_{id}$
20: while true do
21: $exact\_range(M_{id}) \leftarrow validation\_exact\_queue(M_{id}).take()$
22: Send information to monitoring center about established $exact\_range(M_{id})$
23: end while
24: end thread
25: thread Overhead Validation for $M_{id}$
26: while true do
27: $overhead\_range(M_{id}) \leftarrow validation\_overhead\_queue(M_{id}).take()$
28: Send information to the monitoring center about the established $overhead\_range(M_{id})$
29: end while
30: end thread
31: thread Expiration for measurement $M_{id}$ and $Dsc$
32: while true do
33: $[expired\_value, expired\_overhead] \leftarrow expiration\_queue(M_{id}).take()$
34: Remove expired\_value from $Values(M_{id})$ set
35: Remove expired\_overhead from $Overheads(M_{id})$ set
36: UpdateExactRangeAndOverhead($M_{id}, Dsc$)
37: end while
38: end thread
As presented in the description of Algorithm 5 in the case of the mcum the measurement is identified with the use of the service flow hash \( H_{cur} \) calculated by the flow sniffer. It should be noted that this hash does not yet uniquely identify an mcum from the SOA system model. The same hash can belong to several mcums at the time of interception and only further flow structure determines the actual final mcum. Therefore, after the exact range or the overhead range related to \( M_{id} \) of an mcum is finally reported to the monitoring center, all mcums from the SOA system model related to the reported \( M_{id} \) are updated.

At the end of Section 4.2.2, three assumptions relating to monitoring mechanisms were formulated: (a), (b) and (c). The presented design of both discovery and measurement mechanisms fulfills all of them under the condition that the measurement activation is mapped to the m-interceptor activation. The assumed mapping is as follows: the activating measurement of metric \( M_{id} \) and service \( S_{id} \) is mapped to the activating m-interceptor of metric \( M_{id} \), which intercepts service \( S_{id} \). The disadvantage of such a mapping is that activating a single measurement related to metric \( M_{id} \) and service \( S_{id} \) (e.g. direct measurement) automatically activates all other measurements (e.g. cumulative measurements of three different flows) related to \( M_{id} \) and service \( S_{id} \).

The fulfillment of assumption (a) is ensured by Algorithm 5 which propagates metric value (either mcum or mdirect) with the use of the exported interceptor socket enrichment when the terminating invocation is intercepted. The fulfillment of assumption (b) is also ensured by Algorithm 5 which does not trigger Algorithm 6 unless the m-interceptor is activated and performs metric value propagation even when m-interceptor is deactivated. The fulfillment of assumption (c) – the issue of the overhead – was already mentioned before and is ensured thanks to the construction of Algorithm 5 and Algorithm 6. Significant overhead can be caused only by the latter algorithm, which has to perform some system calls to send the data to the monitoring center.

5.4 Framework Architecture

The architecture of the proposed DAMON framework is presented in Figure 5.3. It combines the contribution of two previous chapters and monitoring mechanisms into a scheme of cooperating components aiming at the realization of the dissertation goals in a scalable and efficient manner. The components and component groups are related to the layers of the adaptive monitoring concept presented in Figure 3.2, which is indicated by using the color selection present in this figure. Some of the components are taken directly from the abstraction described in Section 3.2, i.e. all three model updaters, inferencer, discovery and measurement mechanisms and all three reconfigurators. Therefore, the description of these components is vaguely reminded and then extended in the context of the details provided in Chapter 3 and Chapter 4.

The proposed architecture defines two main entities: the monitoring center (referred to
as $M_{HQ}$) and the monitored service container (referred to as $M_{SC}$). $M_{SC}$ is a service container from the SOA system meta-model, which is enriched with the feature of the interceptor socket by means of the selected instrumentation approach (cf. Section 5.2.1) and which is provided with the mechanisms of discovery and measurement. $M_{HQ}$ is also a service container, but it serves a different purpose. It hosts the components dedicated to gather the monitoring data and to manage the whole monitoring process. The proposed architecture is targeted at monitoring a federation of service containers. All federation containers that are supposed to be monitored have to be instrumented with the interceptor socket and there has to be at least one container, either chosen from the existing ones or added additionally, acting as the monitoring center.

Besides the two main entities and their components, Figure 5.3 presents also flows of information. Monitoring the data gathered from $M_{SC}$ and transferred to $M_{HQ}$ is represented with a regular arrow, while reconfiguration interaction going from $M_{HQ}$ to $M_{SC}$ is represented with a dashed arrow. It is assumed that, between $M_{HQ}$ and $M_{SC}$, the data can be transmitted by means of regular distribution mechanisms used by federation containers to communicate with each other.

As defined in Chapter 4, the whole monitoring process begins when the user activates
some monitoring goal represented by the prepared monitoring strategy. The monitoring strategy is passed to the monitoring scope reconfigurator. Reconfigurator analyzes the cmodel and tmodel trackers (if they are present), extracts their content and passes it to the discovery subscription manager. It also analyzes all measurement activations (if they are present), extracts their content and passes it to measurement subscription manager. The discovery and measurement subscription managers have access to the SOA system model and the measurement model respectively. It allows them to map the high-level monitoring elements of the monitoring strategy to low-level discovery subscriptions and measurement subscriptions, which are then announced to all monitored service containers in the federation. Both subscription types either directly or indirectly specify the monitoring mechanisms that should be activated or queried for a response.

When the discovery and measurement subscriptions are received by the monitored container, they are delivered to the discovery mechanisms reconfigurator and the measurement mechanisms reconfigurator respectively. The discovery reconfigurator invokes directly the topology retriever, reconfigures the scope of the topology follower and installs and/or uninstalls the appropriate flow sniffers. The measurement reconfigurator installs and/or uninstalls the measurement interceptors and activates their necessary subset. When the flow sniffers and the measurement interceptors are installed, they are automatically connected to interceptor chains of relevant interceptor sockets. This allows for starting propagation of the monitoring data. All i-services used in both the discovery and measurement instrumentation are taken from the interceptor registry – a distributed repository from which the code of interceptor services can be downloaded and installed into the monitored service container.

Propagation of the monitoring data starts at the interceptor sockets, which allow for the interception of service invocations on behalf of both the discovery and measurement instrumentation. The components of the discovery instrumentation send the intercepted and retrieved monitoring data directly to the monitoring center. Such data is referred to as SOA system model contribution. The m-interceptors of the measurement instrumentation realize Algorithm 5 while Algorithm 6 is enclosed in a separate component referred to as the range reporter. Therefore, the metric and overhead values calculated by the measurement interceptors are passed to the range reporter for the identification of metric and overhead ranges and further reporting to the monitoring center as measurement model contributions.

In the monitoring center, the SOA system model contribution is received by the SOA system model updater, while the measurement model contribution is received by the measurement model updater. The first updater analyzes the provided topology change events and the information from the topology retriever and updates the current instance of the SOA system model. The updated instance is propagated to the measurement model updater and the steering model updater. The measurement updater analyzes the received SOA system model and by confronting it with the i-services available in
the interceptor registry it establishes a set of measurements that can be potentially activated. Complete information about the measurements and especially the aspects of causality allows the measurement updater to easily map the ranges identified by the range reporters to the appropriate mdirects and mcums. The final outcome of the updater logic – the updated measurement model – is propagated to the steering model updater.

When the control loop of the monitoring process is in Phase I (nominals identification) or Phase III (adaptive drill-down), the steering updater instantly updates the steering model and passes it, together with the measurement model, to the inferencer executing the appropriate control loop algorithms. When the monitoring process is in Phase II (sentinels selection), the steering updater checks the drill-down condition (DDC) and only when it is matched, the update of the steering model and further propagation to the inferencer for the execution of adaptive drill-down is performed. Also when the steering updater detects a change in the SOA system model related to some active monitoring strategy, the appropriate information is passed to the inferencer (in order to perform a tracking cycle of the control loop – cf. Figure 4.2). The inferencer is the heart of the whole architecture, which executes the algorithms described in Section 4.3 and, by these means, manages the adaptation of the monitoring process.

As mentioned in Section 4.2.2, Algorithms 2 and 3 are executed separately for each instance of the identified Bayesian network. Typically, such BNs are not connected to one another, which does not cause any problems. However, sometimes fragments of different BNs can overlap and the execution of the mentioned algorithms for these networks can potentially interfere with each other leading to some conflicting situations. The proposed architecture anticipates such situations and proposes the following simplified approach. After the completion of the control loop Phase I and the creation of Bayesian networks, possible overlaps are identified. For any set of overlapping BNs the user has the possibility to assign priorities for each BN from the set. When a conflicting situation occurs, the BN with the higher priority is preferred over the BN with the lower priority. If the user has not assigned any priorities, then simply the BN with the lower number of nodes is preferred (when the number of nodes is the same, a random BN is preferred). The proposed approach allows for solving all cases in which the algorithms of different BNs formulate contradicting actions.

The presented architecture puts a significant focus on the issues of efficiency and scalability. It is manifested by the following aspects. The first aspect is loose coupling of $M_{HQ}$ and $M_{SC}$. Interactions between these two entities are performed in a message-oriented way through subscriptions and model contributions. Such interactions can be easily realized by asynchronous invocations, which enable achieving high scalability. The second aspect is the possibility of deploying multiple monitoring centers. It allows to decompose the federation into smaller fragments and to realize the monitoring goal independently in each fragment. Such approach limits the parts of the federation where
monitoring overhead is introduced. The third aspect is the construction of monitoring mechanisms, especially flow sniffers and m-interceptors, which directly interact with the intercepted business logic. Flow sniffers report only the first occurrence of a given flow in a given service, therefore even an intensive load of some service flows is not disturbed by the sniffers. M-interceptors do not introduce any significant overhead prior to their activation. After the m-interceptors are activated, the processing of intercepted metric and overhead values is performed asynchronously by means of Algorithm 6 delay queues enclosed in the range reporter component. Combined together, the presented aspects are expected to ensure scalability, which is appropriate for dynamic, large-scale SOA systems targeted by this dissertation.

5.5 Summary

This chapter presented the design of the DAMON framework, which is capable of providing an existing SOA environment with the features of the dynamic adaptive monitoring. At the beginning of the chapter, six requirements were formulated: R1 – diverse monitoring mechanisms; R2 – seamless, non-intrusive enrichment of the SOA environment; R3 – run-time mechanisms reconfiguration allowing for changing monitoring selectivity; R4 – propagation of specific information along with service invocation; R5 – mapping of a high-level monitoring goal to lower layers and R6 – assurance of efficiency and scalability in a dynamic, distributed environment. The requirements coverage is now analyzed.

R1 is addressed by the SOA system discovery mechanisms: topology retriever, topology follower and flow sniffer as well as the measurement mechanism – m-interceptor. The fulfillment of requirement R2 is achieved thanks to a hybrid approach to instrumentation, which assumes that the monitoring enrichment is applied to the service container during its start-up process. Requirements R3 and R4 are covered by the intercept socket which wraps the existing service and allows the i-services to dynamically attach and intercept invocations of the wrapped service. Thanks to that, selectivity can be carefully controlled. The propagation of information along with the invocation is realized on the level of the i-socket and the i-services by means of the local and exported enrichment. Requirement R5 is addressed on the architectural level by the discovery and the measurement subscription managers which map the monitoring strategy to the appropriate subscriptions announced to the respective monitored service containers. Finally, the fulfillment of requirement R6 is possible thanks to: (i) loose coupling of the monitoring center and the monitoring container, (ii) deployment of multiple monitoring centers in the federation and (iii) appropriate construction of the flow sniffers and the m-interceptors. However, a detailed verification of R6 and a general overhead related to requirements R1, R2 and R4 will be practically evaluated in Chapter 7.
Chapter 6

Prototype Implementation

The previous chapters presented three main compounds of the adaptive monitoring system: the concept (Chapter 3), the process (Chapter 4) and the DAMON framework (Chapter 5). In order to prove that the proposed compounds can be combined into a valuable entirety, experimental evaluation has to be performed. However, the necessary experiments can be executed only in a working implementation covering all three mentioned compounds. Therefore there is a need for appropriate prototype implementation. First of all, a particular SOA environment technology for prototype realization has to be selected. The technology has to cover the aspects of the SOA dynamics identified in Section 3.7. Then, the technology internals should be mapped to the proposed SOA system meta-model. Finally, decisions related to the implementation details of all architectural elements have to be made. This chapter presents an evaluation-ready prototype, referred to as the DAMON prototype, which covers all the mentioned aspects: the technology selection, its mapping to the SOA system meta-model and the specification of implementation details.

The structure of this chapter is as follows. The first section begins with the discussion about the selection of a particular environment, suitable for the prototype implementation. The discussion ends with the final choice of the OSGi technology preceded by a sound justification. Then, aspects of the OSGi dynamics are analyzed and technology compounds are mapped to appropriate elements of the topology and causality planes of the SOA system meta-model. Subsequently, the second section presents the implementation details of the whole DAMON prototype. The details cover the communication layer used for sending the acquired monitoring data and dispatching reconfiguration actions directed to the particular monitoring mechanisms. Moreover, the DAMON implementation covers the two main architectural entities introduced by the previous chapter, i.e. monitored service container and monitoring center.
Chapter 6. Prototype Implementation

6.1 Technology for Prototype Realization

The architecture of the adaptive monitoring framework presented in the previous chapter is not bound to any specific technology. Therefore, in order to realize its prototype implementation, a specific technology had to be selected. Section 2.2 presented several SOA environments, both commercial products and open-source initiatives, that could be selected. Two criteria were taken into account during the selection process: universality of prototype built for particular technology and extent to which technology addresses aspects of the SOA dynamics. The analysis of presented environments shows that only the OSGi technology offers a very good support for the SOA dynamics and, at the same time, is standardized by specification ensuring its universality. Thus, OSGi was finally selected for prototype realization. Choosing a standardized technology increases dissertation impact because the created prototype is applicable not only to single product but to whole class of products compliant with the technology’s specification. The following two sections present how OSGi realizes aspects of the SOA dynamics identified in Section 3.1 and how its internals are mapped to the SOA system meta-model proposed in Section 3.3.

6.1.1 SOA Dynamics in OSGi

The following three main aspects related to the SOA dynamics were identified in Section 3.1:

(i) dynamics of services registration and discovery;
(ii) dynamics of service components deployment;
(iii) adaptability of services, service components and operational systems.

This section aims to prove that the OSGi technology has a sound support for the first two aspects and, to some extent, is capable of supporting the third one. As already mentioned in Section 2.2, the basic structural entity introduced in OSGi is a bundle – component, which can be deployed to the OSGi container. The Core OSGi Specification [47] introduces a bundle-oriented model of the OSGi framework, which, among others, defines the following important layers: module, life cycle and services. The first two layers are related to the second aspect (ii) of the SOA dynamics, while the last layer – services is related to the first aspect (i).

The module layer introduces a modularization model for Java which defines rules and management mechanisms for sharing the code between different bundles. The goal of the life cycle layer is providing an API for managing bundles in the module layer. This API defines the structure of the bundle life cycle covering operations which can be potentially applied to a bundle. The operations include starting, stopping, installation, uninstallation and updating of a bundle and can be executed without the need of restarting the whole container. Besides the API focusing on bundles, the life cycle layer provides also
the API for managing the OSGi container itself. In the context of aspect (ii) of the SOA dynamics it is expected that a service unit can be dynamically deployed, undeployed or redeployed in a service container (cf. Section 3.1). If it could be assumed that a service container is mapped to an OSGi container and a service unit is mapped to a bundle, then described module and the life cycle layers fulfill entirely recalled expectation and provide even more sophisticated features such as starting, stopping of bundles and management of the OSGi container.

The service layer provides a dynamic communication model for bundles which is based on the publish-find-bind scheme typical for the service-oriented systems. The model assumes decoupling of the service specification, expressed by java interface annotated with some properties, from the actual implementation – java class implementing the interface. The essential element supporting this model is the OSGi service registry. This registry allows a bundle to register new services, receive notifications about requested services and actively search and acquire services which need to be consumed. All registry interactions can be performed at run-time, which allows bundles to dynamically adapt not only to the current execution environment (mobile, enterprise or web system) but also to other bundles available in the container. Aspect (i) of the SOA dynamics expects the following features: dynamic interaction of the service component with the service registry involving registration, unregistration and modification of the service’s contract; discovering registered services in both active and passive manner. If it could be assumed that certain elements of the OSGi service layer are mapped to the concepts introduced in Section 3.1 in the context of aspect (i), then described functionality of this layer implements both aforementioned features.

The third aspect (iii) of the SOA dynamics – adaptability is supposed to increase the run-time dynamics of both services and service components by changing non-functional characteristics of given element, changing element’s internal structure or changing resource assignment. To certain extent, in OSGi this can be achieved by a combination of features provided by discussed three layers. Changing non-functional characteristic of given service can be achieved on the level of service implementation by subscribing to certain notifications (e.g. better/worse service instance appears/disappears) delivered by the OSGi service registry. Upon receipt of such notification, the service implementation could change the instance of some consumed service. Such change could influence the service non-functional aspects, which is always reflected by the appropriate modification of properties annotating exposed service interface. Finally, changing resource assignment could be realized both on the level of bundles and services by leveraging the life cycle API and potential of the OSGi service registry respectively. Discussed adaptability features are of limited potential, however the latest OSGi Core Specification [47] introduces a mechanism of weaving hook, which offers a means for intercepting bundle class loading and transforming its byte code. To conclude, the combination of all discussed features and mechanisms ensures a substantial support for the realization of adaptable applications with the use of the OSGi technology.
6.1.2 Mapping OSGi to SOA System Meta-model

The SOA system meta-model is the foundation of adaptive monitoring concept presented in Section 3.2. Therefore, the essential prerequisite, for implementing proposed architecture in the OSGi technology is mapping the OSGi concepts to the ones introduced by this meta-model. Some aspects of this mapping were already touched in the previous section. In this section they are extended and organized in a comprehensible tabular form.

Table 6.1 presents the mapping of the OSGi topological aspects to the topology plane of the SOA system meta-model, while Table 6.2 grasps analogical mapping of the causality aspects.

The essential triangle of the topology plane consisting of: service, abstract service and service component (cf. 3.3) is mapped to the central concepts of the OSGi service layer: service instance, service interface (annotated with additional properties during exposition) and service implementation. Both a service unit and a solution component are mapped to an OSGi bundle. However, since a service unit represents executable aggregation of service components, before its actual deployment, it is mapped to a bundle, which is not yet deployed to the OSGi container. The solution component, which refers to a service unit after its deployment, is mapped to the already deployed bundle.

The OSGi bundles can be aggregated with the use of such mechanisms as composites [126] or features [127]. Resulting aggregation of bundles is mapped to the SOA system application. The OSGi container is intuitively mapped to the service container and Java Virtual Machine (JVM), the OSGi execution environment, is mapped to the solution platform. The services in different OSGi containers can communicated with each other by means of the Remote Services mechanism and its more advanced counterpart – Remote Service Admin (RSA) mechanism [48]. The interconnected group of containers is mapped to the SOA system federation.

<table>
<thead>
<tr>
<th>OSGi topology aspects</th>
<th>SOA system meta-model</th>
</tr>
</thead>
<tbody>
<tr>
<td>service instance</td>
<td>service</td>
</tr>
<tr>
<td>service interface</td>
<td>abstract service</td>
</tr>
<tr>
<td>service implementation</td>
<td>service component</td>
</tr>
<tr>
<td>bundle archive NOT installed in the container</td>
<td>service unit</td>
</tr>
<tr>
<td>bundle installed in the container</td>
<td>solution component</td>
</tr>
<tr>
<td>aggregation of bundles</td>
<td>application</td>
</tr>
<tr>
<td>OSGi container</td>
<td>service container</td>
</tr>
<tr>
<td>Java Virtual Machine</td>
<td>solution platform</td>
</tr>
<tr>
<td>interconnected group of containers</td>
<td>federation</td>
</tr>
</tbody>
</table>

Table 6.1: Mapping of OSGi to topology plane of SOA system meta-model
OSGi causality aspects | SOA system meta-model
--- | ---
An invocation of the service instance method performed by the implementation of another service. | service invocation
The synchronous invocations are directly supported, while the asynchronous ones can be provided by means of the Event Admin Architecture [48]. Both singular and parallel invocations are possible. The parallel ones can occur when originating service spawns a few threads making concurrent invocations. | invocation type
Both synchronous and asynchronous services are available (the asynchronous services can be provided through the Event Admin Architecture [48]). | sync type
Invocation X has a causal relation to invocation Y if invocation Y is performed by a thread, which received invocation X in a synchronous or asynchronous way. | invocations causality
A set of the causally related service instance invocations. | service flow

Table 6.2: Mapping of OSGi to causality plane of SOA system meta-model

Mapping of the causality aspects is performed in accordance with the topological mapping. In the realized prototype implementation, general invocations causality is determined by the analysis of threads in which service invocations are performed. Although, it does not ensure covering all possible cases of causal relationships, it is considered sufficient for the purpose of the prototype.

6.2 Implementation Details

This section presents the details of the DAMON framework prototype implementation that is based on the OSGi technology. The implementation was performed in accordance with the mapping presented in the previous section and all algorithms presented in Chapters 4 and 5. The architecture of the DAMON framework (cf. Section 5.4) identified two main entities: monitored service container \( (M_{SC}) \) and monitoring center \( (M_{HQ}) \). This section firstly covers the issue of communication between these entities and then describes the details related to their implementation.

6.2.1 Communication Backbone

As discussed in Section 5.4, the DAMON architecture assumes that \( M_{HQ} \) and \( M_{SC} \) can communicate by means of regular distribution mechanisms of the underlying technology. In case of OSGi, Remote Service Admin (RSA) is such mechanism (cf. Section 6.1.2). The DAMON prototype uses the RSA implementation proposed in the scope of the author’s article [45]. The proposed RSA implementation, enclosed in a bundle referred to as d-provider, is based on Message Oriented Middleware (MOM) and leverages the concept of a network of the JMS brokers. The network of brokers allows distant
JMS providers to communicate with each other by means of distributed queues and distributed topics (cf. [15]). In the DAMON prototype, the ActiveMQ [128] JMS provider is embedded into each OSGi container. Connecting these providers into the network of brokers and deploying the d-provider bundle in each container results in creation of the OSGi containers federation, which provides the following essential RSA distribution features:

**service export** – registering service by the regular OSGi BundleContext API with a specific additional properties results in exporting this service to all federation containers matching the specified properties;

**service import** – consuming service by means of the ServiceTracker mechanism results in importing best matching consumed service from a distant federation container in which the service was exported.

Invocation of the imported service results in a remote call, which is delivered to the respective exported service. After invocation completes, the response is transmitted back. Both synchronous and asynchronous invocation models are supported.

The presented distribution features were used for implementation of the following capabilities of the DAMON prototype: (i) \(M_{HQ}\) announces monitoring subscription to a desired set of \(M_{SC}\) containers; (ii) \(M_{SC}\) reports monitoring data to \(M_{HQ}\). It was achieved by implementing monitoring subscription as a service. The monitoring subscription service is exported in the \(M_{HQ}\) and, by means of RSA mechanism, is imported into all relevant \(M_{SC}\) containers. Thanks to that, the dynamics of subscription registration and all issues related to loosing network connectivity between some federation containers are automatically solved by RSA.

Aside from leveraging RSA features in the context of monitoring subscriptions, d-provider bundle provides useful management capabilities. By default, an OSGi container can be managed with the use of the Java Management Extensions (JMX) technology [129]. D-provider uses services implemented in the context of the JMX management and exports them by means of RSA to the whole federation. Thanks to that, such operations as installation of bundles, changing their states or performing a container shutdown can be performed remotely from any point of the federation.

### 6.2.2 Realization of Monitored Service Container

The implementation of the monitored service container covers three main areas: essential instrumentation, discovery and measurement. Since the essential instrumentation has to be most tightly integrated with the underlying technology, its implementation for OSGi was the most challenging. In order to implement the concept of interceptor socket, an AOP aspect was implemented with the use of AspectJ [86]. This aspect
public aspect BundleContextAspect {

  private BundleContextAspectIface 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 proxyClassLoader = new
          MonitoringProxyClassLoader();
        Class aspectImplClass = proxyClassLoader
          .loadClass("pl.edu.soa.agh.as3.sensors.osgi.invocation.logic.
            BundleContextAspectImpl");
        aspectImpl = (BundleContextAspectIface) aspectImplClass
          .newInstance();
        proxyClassLoader.setAspectIface(aspectImpl);
      } catch (Exception e) {
        e.printStackTrace();
      }
    } if (aspectImpl != null) {
      return aspectImpl.aroundGetService(bc, ref, ret);
    } else {
      return ret;
    }
  }
}

Listing 6.1: Aspect intercepting OSGi BundleContext

is presented in Listing 6.1. As can be seen, the aspect intercepts invocations of the
BundleContext.getService method. In OSGi, this method is executed each time a ref-
erence to a service is acquired. Since class loading in OSGi is more advanced than in
regular Java application, a separate class loader (MonitoringProxyClassLoader) ensures
that the bundle which invocation was intercepted can load the interceptor socket code
base. Presented aspect loads BundleContextAspectImpl class, which creates instance
of the interceptor socket by the invocation of the aroundGetService method in line 22.
A separate i-socket is created for each acquired service reference, which ensures that
mapping between service consumer and service provider can be easily extracted.

The interceptor socket itself is implemented with the use of Java Proxy [130]. The
i-socket implementation imitates the consumed service and executes a specific moni-
toring logic on each service invocation. This logic is presented in Listing 6.2. It per-
forms evaluation of two separated interceptor chains: business i-services and agnostic
i-services (in this particular order) in three cases: before service invocation (calling
the serviceInvokeStarted interceptor method), after service invocation (calling the
serviceInvokeFinished interceptor method) and when service exception is thrown (calling
the serviceInvokeError interceptor method). Both chains are retrieved from the respec-
tive tracker components, which are implemented by means of the OSGi ServiceTracker
mechanism [47]. Thanks to this mechanism, each new interceptor service appearing in
the container is automatically analyzed by all i-sockets and if the i-service properties
are matching given i-socket, the i-service is added to respective chain.

The discovery features were implemented in two separate bundles: topology agent and osgi-sniffer. The topology agent implements the components of a topology retriever, a topology follower and a discovery mechanisms reconfigurator. The topology retriever is implemented by accessing the BundleContext interface, which allows retrieving information about all bundles including their life cycle states. The topology follower is implemented with the use of the BundleListener and the ServiceListener OSGi mechanisms. These mechanisms allow a bundle to register a listener for any events related to the topology changes. Thanks to that, there is no need for implementing the topology follower as a business i-service. The osgi-sniffer implements the flow sniffer component in accordance with its specification, i.e. as an agnostic i-service which is plugged into the OSGi interceptor socket. The sniffer implements Algorithm 4 and ensures that the causality details related to flows are appropriately propagated with the use of local and exported enrichment.

The implementation of the measurement is enclosed in the following bundles: measurement agent and osgi-minterceptors. The measurement agent implements the components of the measurement mechanisms reconfigurator and the range reporter, while the osgi-minterceptors contains the implementation of the measurement interceptors. It was decided that the implementation of two agnostic measurement interceptors will be sufficient for the purpose of the prototype evaluation. The i-services acquiring the following metrics were implemented: invocation count, processing time. The implementation of other metrics, such as the CPU usage or the RAM memory consumption, would not impose problems as its feasibility in OSGi was already proven. The range reporter component was implemented with the use of the Complex Event Processing (CEP) paradigm and the delay queue collection provided by JDK. There are several CEP engines currently available from which Esper was selected for the prototype implementation. The usage of CEP engine greatly simplifies the logic of metric and overhead range calculation (Algorithm 6) in the sliding time window of the exact range expiration period.

The measurement mechanisms reconfigurator and the discovery mechanisms reconfigurator, enclosed in the measurement agent and the topology agent bundles respectively, were implemented with the use of ServiceTracker which tracks the availability of monitoring subscription services. When a new monitoring subscription service relevant for given agent appears, the subscription content is translated into appropriate configuration of underlying monitoring mechanisms.

### 6.2.3 Realization of Monitoring Center

$M_{HQ}$ consists of three coordinators: discovery, measurement and steering (cf. Figure 5.3). All three coordinators and their related components were implemented in a single
bundle named *damon-center*. The components of discovery and the measurement coordinators, except from the realization of their expected logic (managing of monitoring subscriptions and updating respective models), also communicate with all $M_{SC}$, which are covered by activated monitoring goals. The representation of the monitoring goal strategy was implemented with the use of the **XML** language. The examples of various monitoring strategies are thoroughly discussed in the course of functional evaluation presented in Section 7.3.

The details of the monitoring subscription, similarly to the monitoring goal, are also represented in the **XML** format by means of the *MsDefinition* entity. The **XML** representation of MsDefinition was mapped to Java with the use of the JAXB framework [138], which allows for convenient MsDefinition creation. The creation of two sample MsDefinitions: the measurement subscription and the discovery subscription is presented in

```java
public class OsgiInvocationInterceptorSocket implements InvocationHandler {

  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    String id = UUID.randomUUID().toString();
    Enrichment localEnrichment = new LocalEnrichment();
    // Identification of causality details
    
    // Evaluation of agnostic and business interceptors chains and 
    // invoking serviceInvokeStarted method on each interceptor
    new BusinessIntrsEvaluator(businessIntrsTracker) {
      public InterceptorEffect evaluateInterceptor(Interceptor bIntr) {
        return ((BusinessOsgiInterceptor) bIntr).serviceInvokeStarted(
          id, originalSrvRef, consBundleName, provBundleName,
          originalService, m, args, localEnrichment, exportedEnrichment);
      }
    }.evaluateChain();
    Object[] intrs = agnosticIntrsTracker.getServices();
    if (intrs != null) {
      for (Object intr : intrs) {
        AgnosticOsgiInterceptor i = (AgnosticOsgiInterceptor) intr;
        String intercepterClassName = i.getClass().getName();
        i.serviceInvokeStarted(id, originalSrvRef, consBundleName,
          provBundleName, m.getName(), localEnrichment, exportedEnrichment);
      }
    }
    // Invoking the actual service
    Object r = null;
    try {
      r = m.invoke(targetService, args);
    } catch (InvocationTargetException e) {
      throw e.getCause();
    }
    // Evaluation of agnostic and business interceptors chains and 
    // invoking serviceInvokeStarted method on each interceptor
    return r;
  }
}
```

Listing 6.2: Logic of interceptor socket implemented as Java Proxy
Chapter 6. Prototype Implementation

Listing 6.3 The first subscription requests the monitoring of the processing time metric (with appropriate \texttt{Min.per} and \texttt{Exp.per} parameters) in the \texttt{SampleService} service registered by a bundle which name matches the following regular expression \texttt{.*Sample-Bundle.*}. The service can be registered in any container and realized by any service implementation (regular expressions matching all strings – \texttt{".*"} are used). The latter subscription requests the monitoring of the topology changes in the whole federation (\texttt{".*"} regular expression is used).

\begin{verbatim}
1 MsDefinition measurementSubDef = new MeasurementSubscription()
  .withName("SampleMeasurementSubscription")
  .withMetricInstance(new ProcessingTimeMetric())
  .withMinPer(5000)
  .withExpPer(40000)
  .withTargetTopology(new TsOsgi())
  .withSubscriptionOsgiEntry(new SubscriptionOsgiEntry()
    .withContainer(".*")
    .withBundle(".*SampleBundle.*")
    .withImpl(".*")
    .withService("SampleService")
  )

2 MsDefinition topologySubDef = new DiscoverySubscription()
  .withTargetTopology(new TsOsgi())
  .withSubscriptionOsgiEntry(new SubscriptionOsgiEntry()
    .withContainer(".*")
    .withBundle(".*")
    .withImpl(".*")
    .withService(".*")
  )
\end{verbatim}

Listing 6.3: Creation of monitoring subscription’s MsDefinition

\begin{verbatim}
1 public class ManagedSubscriptionSupport implements ManagedSubscription {

2 private BundleContext bc;
3 private ServiceRegistration sr;
4 private MsDefinition msDef;
5 public ManagedSubscriptionSupport(MsDefinition msDef, BundleContext bc) {

6   this.msDef = msDef;
7   this.bc = bc;
8 }

9 public void registerMsSubscriber() {
10   sr = bc.registerService(ManagedSubscription.class.getName(), this, null);
11 }

12 public void unregisterMsSubscriber() {
13   sr.unregister();
14   sr = null;
15 }

16 public MsInstance getMs() {
17   return msDef;
18 }

19 public abstract void passMonitoringEvent(CommonBaseEventType event) {
20   // Handling reported monitoring data
21   ...
22 }

Listing 6.4: Registration of monitoring subscription
\end{verbatim}
The class which supports the monitoring subscription registration is presented in Listing 6.4. When its `registerMsSubscriber` method is invoked, a new ManagedSubscription service is registered in the container, what is noticed by either discovery or management subscription manager. Then relevant subscription managers register the monitoring subscription service, which is exported to remote containers. The appropriate mechanism reconfigurator (either discovery or measurement one) in the remote containers detects the import of the remote monitoring subscription, retrieves its MsDefinition and processes it appropriately. When `unregisterMsSubscriber` is invoked, ManagedSubscription is unregistered. This triggers the unregistration of the monitoring subscription service, which after propagation by means of RSA causes the reconfiguration of the monitoring mechanisms in the relevant $M_{SC}$ containers. All monitoring data is returned to $M_{HQ}$ by means of the `passMonitoringEvent` method, where the data is appropriately processed. The data is transmitted in the Common Base Event (CBE) format [139], which ensures universality and extendability.

The interceptor registry was implemented as the `i-registry` bundle providing a distributed interceptor repository scattered through the federation. The implementation ensures the appropriate redundancy, therefore, when one or several containers are disconnected from the federation, the registry is capable of continuing its proper functioning.

All algorithms of the control loop were implemented in the inference component – the central part of the steering coordinator. The fragments of algorithms related to BN inferencing were realized with the use of UnBBayes [140, 141] – a plugin framework for the probabilistic graph models, which among others, supports Bayesian networks. Since the data model used by UnBBayes differs from the steering model, an appropriate mediator component was implemented. Designed mediation API introduces an abstraction layer which ensures that the underlying BN framework can be easily replaced in the future. The inferencing engine of UnBBayes is configured to use the Junction Tree algorithm [116], which is one of the most commonly used algorithms for exact BN inferencing.

6.3 Summary

This chapter presented the implementation, referred to as the DAMON prototype, created for the purpose of evaluating the dissertation contribution. This prototype complies with the proposed framework architecture (Chapter 5) and allows for the realization of the monitoring process (Chapter 4) in accordance with the adaptive monitoring concept (Chapter 3). The analysis of the SOA environments reviewed in Section 2.2 shows that OSGi is the most appropriate technology for the prototype realization taking into consideration the identified aspects of dynamics and the technology universality. The presented mapping of the the OSGi technology internals to the SOA system meta-model introduced in Chapter 3 ensures that this meta-model can be useful in the real-life SOA.
deployments. The proposed OSGi-based DAMON implementation fulfills the primary goal of the prototype, i.e. enabling realization of the evaluation considering both functional and non-functional dimensions. Such evaluation, in which the DAMON prototype is deployed in simulated environment of a banking institution, is presented in the next chapter.
Chapter 7

EVALUATION

The proposed adaptive monitoring solution calls for a sound evaluation that confronts various expectations formulated in the context of the thesis statement and the high-level design of the DAMON framework. In order to ensure the evaluation soundness, both functional and non-functional dimensions of the dissertation’s contribution should be covered. In the scope of functional dimension, there is a need for verification of all proposed monitoring mechanisms, i.e. topology discovery, causality identification and measurement acquisition. Additionally, the functional dimension should include various components of monitoring center, which cooperate with mechanisms to provide the goal orientation of the monitoring process. The non-functional dimension should try to answer the two important questions: does the overhead introduced by the DAMON prototype is acceptable and does the scalability of the solution is sufficient for large-scale service oriented systems. This chapter presents the DAMON prototype evaluation, which covers all identified functional and non-functional aspects by executing five selected experiments in the environment of real-life SOA application.

The structure of this chapter is as follows. The first section presents the overview of the evaluation by enlisting all performed experiments and by formulating relation between each experiment and the expectations related to both conceptual and technical levels. The second section describes the experimental environment in which the evaluation was performed. The description covers the scenario of the SOA application, named Credit Card Process (CCP), as well as the software and hardware setup used for the application deployment. The subsequent section presents the functional part of the evaluation, which covers the following experiments executed in the CCP environment: container enrichment and topology discovery; addressing aspects of the SOA dynamics by means of the goal-orientation and the realization of the drill-down process triggered by different system anomalies. Finally, the forth subsection contains the non-functional part of the evaluation, which describes the assessment of the monitoring overhead and the verification of the important scalability aspects.
Chapter 7. Evaluation

7.1 Evaluation Overview

The goal of the presented evaluation is to verify if the DAMON framework can be successfully applied to a real-life application scenario and then ensure the properties formulated in the context of the thesis statement, i.e. (properties are named T1 – T3):

T1 – goal-oriented declarative management of the adaptable monitoring process capable of handling aspects of dynamics;

T2 – dynamic adjustment of monitoring selectivity realized by restricting scope of monitoring process in a way which decreases monitoring costs without losing fulfillment of the declared monitoring goal;

T3 – on-demand root cause identification triggered by anomalies detected in the system and achieved by appropriate dynamic adaptation of monitoring scope.

Additionally, the evaluation aims at checking compliance with requirements of the DAMON framework (formulated in Chapter 5), which naturally extend listed T* properties. These requirements were as follows: R1 – diverse monitoring mechanisms; R2 – seamless, non-intrusive enrichment of the SOA environment; R3 – run-time mechanisms reconfiguration allowing for changing monitoring selectivity; R4 – propagation of specific information along with service invocation; R5 – mapping of high-level monitoring goal to lower layers and R6 – assurance of efficiency and scalability in dynamic, distributed environment.

In order to cover all aforementioned expectations, evaluation is constituted of five experiments. All experiments, are carried out in a virtualized environment simulating a private cloud [142] of some banking institution. The environment is hosting the application, Credit Card Process (CCP), which is used as a monitoring subject across different evaluation experiments. It is assumed that resources such as CPU and network throughput, provided by the simulated private cloud environment are appropriately billed for each application. Therefore in such situation, it is particularly important that monitoring process does not consume significant amount of resources. Results discussed in each experiment, if not stated otherwise, were calculated as an average of ten repeated executions, from which the lowest and the highest value was discarded.

Evaluation experiments are divided into two parts: functional and non-functional. The first part is constituted of the following three experiments:

I. Container enrichment and topology discovery;

II. Goal-orientation addressing aspects of dynamics;

III. Drill-down upon single and multiple anomalies;
Chapter 7. Evaluation

Table 7.1: Mapping of the evaluation experiments into the expectations of the thesis statement and the DAMON framework requirements

<table>
<thead>
<tr>
<th>Experiment</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>II</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>III</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IV</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>V</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

The functional evaluation tries to verify the general correctness of framework operation and shows how it behaves in different situations occurring in a dynamic SOA system. The second evaluation part covers two important non-functional issues of the DAMON framework, each evaluated in a separate experiment. These experiments, are as follows:

IV. Monitoring overhead assessment;

V. Verification of scalability aspects.

During evaluation, experiments were performed one after the other. Therefore, results of a given experiment (e.g. topology model or measurement model) can be available in the subsequent experiments.

The combination of the five proposed experiments allows for covering all formulated expectations. It is visible in Table 7.1 which maps the experiments to the expectations.

This chapter contains the essential results of all experiments. However, in case of experiments II and III, the extended results (representing the models structures and the states of BNs), due to their extensive size, were moved to the appendices.

7.2 Experimental Environment

This section describes the environment in which experiments on the DAMON framework were carried out. The description covers the application scenario used across different experiments as well as general system setup concerning hardware and software level details.

7.2.1 Application Scenario

The OSGi-based SOA application considered during the evaluation is presented in Figure 7.1. The application is named Credit Card Process (CCP). The goal of the application is to automate the process of issuing credit cards in a bank. It consists of seven abstract
services and either one or several instances of each abstract service. During its normal operation, CCP involves three different flows referred to as red, green and blue. As depicted in Figure 7.1 each flow, resulting from general logic of CCP application, covers a subset of service instances belonging to given abstract service. The functionality of each abstract service and its related instances is as follows.
Credit Card Issuing (CCI) – abstract service, which has only single synchronous instance. It receives request for issuing a credit card from some external bank infrastructure covering different communication channels: regular face-to-face services, phone services and also internet banking. CCI services consumes instances of OP services. The choice of actual instance of the OP service depends on the communication channel and the requested credit card limit. For example, when the request involves a credit card with higher limit (above 10,000 PLN), a second card with lower limit (below 2,000 PLN) is offered automatically as a bonus. In such case, the red flow applies: firstly OP3 is called (higher limit) and then OP1 (lower limit).

Offer Preparation (OP) – abstract service responsible for managing preparation of credit card offer. There are four different instances of this service (three synchronous and one asynchronous). Each of them uses less or more extensively services TH and consequently CH. The extensiveness depends on the requested credit card limit. OP1 is concerned with the lowest credit limits (below 2,000 PLN), therefore, the TH and CH services are omitted, OP2 is concerned with limits between 2,000 and 5,000 PLN, OP4 between 5,000 and 10,000 PLN, while OP2 covers only the highest limits - above 10,000 PLN.

Tax History Retrieval (TH) – abstract service allowing for gathering the tax history of credit card requester. There are three different synchronous instances of this service. Each is contacting a different credit agency, which provides information with different quality, context and consequently price (each invocation of agency service is billed separately). TH2 is the most expensive, TH3 is more economic, while TH1 is the most economic.

Credit History Retrieval (CH) – abstract service allowing for gathering the credit history of credit card requester. There are three synchronous instances of this service. Similarly to the TH services, each CH service contacts external credit agency, therefore it differs in terms of quality, extensiveness and price. CH3 is the most expensive, CH1 is more economic, while CH2 is the most economic.

Offer Calculation (OC) – abstract service responsible for calculating credit card offer for particular request taking into account credit history and tax history (if they are available). There are two asynchronous instances of this service. OC1 calculates offer for high credit limits (above 10,000 PLN), which covers also a bonus credit card, while OC2 calculates all other offers.

Offer Verification (OVR) – abstract service which has only single asynchronous instance. The service performs verification of offers calculated by services OC1 and OC2. All service flows are involving the OVR service. After successful verification, the service is passing the offer to persistence layer. The choice of actual instance of the service depends on credit card limit.
Persistence \( (P) \) – abstract service of persistence layer, which is responsible for storing the prepared offers in a database. There are three asynchronous instances of this service. Each service covers different credit limit: P1 – limit below 2,000 PLN and above 10,000 PLN; P2 – limit between 2,000 and 5,000 PLN and P3 – limit between 5,000 and 10,000 PLN. When the offer is persisted the service, the instance notifies the CCI service that the offer can be returned to requester.

All services of CCP application have some nominal response time, which is presented in Figure 7.1 Since CCP is implemented in OSGi, each abstract service is represented by a separate service interface and each service instance has a corresponding class, which implements it. All classes are packaged into four distinct bundles: credit-provider, credit-tax, credit-calc and credit-db. Hardware and software configuration of system in which CCP is deployed is described in the next section.

7.2.2 System Setup

To ensure appropriate quality of the DAMON evaluation the simulated private cloud environment for the CCP application deployment was built with the use of the hardware and software of enterprise quality. The hardware layer was realized by Cisco UCS C200 M2 High-Density Rack Server\(^{23}\) belonging to the family of Cisco Unified Computing System (UCS)\(^{143}\) – server platform for data centers often used in production-level solutions. While the software layer was founded on the VMware vSphere Hypervisor (ESXi)\(^{24}\) 5.1 Update 1 (Build 1065491) – bare-metal hypervisor capable of hosting virtualized environment consisting of virtual machines with various operation systems. ESXi was installed on Cisco UCS C200 M2 (later simply referred to as UCS) as its primary operating system. UCS had the following hardware parameters: 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 (referred to as datastore). ESXi Hypervisor was additionally enriched with the component of VMware vCenter Server\(^{25}\) 5.1.0 (Build 1123961), which constitutes a centralized platform for managing the whole virtualized environment.

For the purpose of the evaluation, six different Virual Machines (VM) were created. All VM were provided with the Linux operating system Ubuntu Desktop 12.04 LTS. The following hardware parameters were configured in each VM: CPU - 2 virtual sockets with 2 cores per socket limited to 1500 MHz; 4 GB of RAM memory; 12 GB of virtual disk provided from datastore. VM were given the following names: mpvm1, mpvm2, mpvm3, mpvm4, mpvm5 and mpvm6. All VM were connected into a single network by means of distributed vSwitch on which three port groups were configured: BackEnd

Chapter 7. Evaluation

Fig. 7.2: Screenshot from VMware vCenter presenting structure of virtualized environment with egress bandwidth limited to 1 Gbit; FrontEnd with egress bandwidth limited to 100 Mbits and External with both ingress and egress bandwidth limited to 4 Mbits. The structure of the configured environment is presented in Figure 7.2. As can be seen, mpvm3 and mpvm4 were assigned to the BackEnd port group, mpvm6 was assigned to the External port group, while the rest of VMs were assigned to the FrontEnd port group. Such configuration simulates the bank environment suitable for deployment of the CCP application. BackEnd VMs can be used for providing more intensive processing (offer calculation) and persistence, while FrontEnd VMs can host less intensive processing (tax/credit history retrieval) and the main interface to CCP. The external port group is capable of simulating connection with some external bank infrastructure (e.g. another department) available through the WAN network link. All VMs acquire needed storage from a datastore and all of them are connected to the hosting operating system of ESXi, which in the figure is referred to by its IP address: 172.17.88.20. The connection to ESXi ensures that VMs can be accessed from the network connected to physical Network Interface Controller (NIC) of UCS.

Since the CCP application is designed for the OSGi-based SOA environment, a solution providing such environment is required. For this purpose, among the SOA environments reviewed in Section 2.2, Fuse ESB was selected. Chosen environment not only ensures a compliance with OSGi but is also, by default, provided with the ActiveMQ JMS broker required by the DAMON framework. Version 4.2.0 of Fuse ESB was deployed to all six VMs and launched with the use of the JVM Java SE Runtime Environment (build 1.6.0_45-b06). Each container was given a name fuseN, where N was representing the number of VMs.
The subsequent steps required for preparing experimental environment were as follows:

(i) connecting the ActiveMQ brokers of all Fuse containers into a network of brokers;

(ii) deploying the CCP application to created VMs. Certainly, there are many possible schemes of connecting the JMS provider instances into a network of brokers and deploying given application to a virtualized environment. This more general problem is out of the scope of this dissertation. For the purpose of the evaluation, it is assumed that the simulated private cloud environment has strict security policy (such situation is common in banking institutions). The policy requires that each functional part of an application should be located in a separate VM, which is placed in a dedicated Virtual LAN (VLAN) \[144\]. Thus, VMs cannot freely communicate with each other. Given VM has access only to these VMs, which are allowed by the policy. Such constraint resulted in a specific brokers topology, deployment of CCP application and planned location of the monitoring center. All these aspects are grasped in Figure 7.3. It should be noted that in the network of brokers, when one broker is sending data to the other one, then data is transmitted through all brokers on the shortest path connecting the communicating brokers. Thus, in the chosen topology, when, for example, fuse4 sends some data to fuse5, it is transmitted through fuse3, fuse2, fuse1 and finally reaches fuse5. Bundles of CCP application were deployed to containers fuse1 – fuse4. A consumer application, enabling testing of CCP was deployed as the bundle credit-consumer to the container fuse6. The application can generate both sequential and concurrent workloads of the CCP spanning selected service flows. The container fuse5 was reserved for deployment of monitoring center (later, referred to with the use of the \( M_{HQ} \) symbol).
Chapter 7. Evaluation

7.3 Functional Evaluation

This section presents experiments aiming at the verification of the functionality offered by the DAMON framework. Since in Section 7.4 there is another set of experiments related to non-functional framework dimension, issues such as the monitoring overhead and the aspects of scalability are not considered during experiments of the functional evaluation.

7.3.1 Container Enrichment and Discovery of SOA System Topology

This experiment attempts to verify essential functionality of the DAMON framework, i.e. enriching the existing SOA environment and creating the topology part of the SOA system model (cf. Section 3.2). Therefore, the scope of this experiment covers the following two aspects: (i) deployment of the DAMON framework to the virtualized infrastructure presented in Section 7.2; (ii) evaluation of the monitoring strategy expected to gather the topology of the CCP application.

**DAMON Deployment**

In order to deploy the DAMON framework to create the experimental environment, the following actions were performed:

1. deployment of the d-provider bundle to all OSGi containers with the use of the JMX technology;
2. deploying the damon-center bundle to fuse5 with the use of the JMX technology;
3. shutting down the execution of containers fuse1 – fuse4 with the use of the d-provider management capabilities;
4. adding the DAMON essential instrumentation to containers fuse1 – fuse4 by the execution of prepared scripts, which perform the installation of the interceptor socket JAR files and the modification of the containers launch configuration;
5. starting back containers fuse1 – fuse4;
6. deploying i-registry, topology agent and measurement agent bundles to fuse1 – fuse4 with the use of the d-provider management capabilities;
7. registering interceptors provided by the osgi-sniffer and the osgi-minterceptors bundles in the interceptor registry.

After the execution of action 1, the JMS brokers of all six fuse containers started exchanging messages informing about their availability. When period of 48 ms has elapsed, processing of all messages was finished and all OSGi containers were ready for the operation in a single federation. After the execution of action 2, damon-center was notified about the availability of the remaining five containers in the federation. During the execution of action 3, the management services of all containers exported by d-provider
Chapter 7. Evaluation

were imported to damon-center. Thanks to that, containers fuse1 – fuse4 could be easily shut down. Actions 4 and 5 had to be performed semi-automatically on VMs mpvm1 – mpvm4. The container start-up time was measured before and after the interceptor socket enrichment. The enrichment caused insignificant delay of 0.34 s. After the execution of actions 6 and 7 damon-center received appropriate notifications that the relevant services belonging to deployed bundles are available for import, likewise remaining containers received information about availability of the damon-center services. It has proven that the federation is functioning correctly and that the DAMON framework is ready for operation.

Topology Discovery

The topology discovery was evaluated by means of the monitoring goal strategy $MG_{TD}$ presented in Listing 7.1. The strategy specifies that it is valid only in a particular time frame (given 10 days in April) under the provided topology condition. The condition states that in order to activate a strategy, there has to be at least one bundle named credit-provider in the whole federation. Additionally, with the use of the Constraint tag, the condition states that the strategy should be deactivated, if the credit-provider bundle is available on other node than fuse1 (this could mean some deployment error, which should be corrected before starting actual monitoring). The presented strategy contains only a tmodel tracker element. A cmodel tracker, measurement activations and steering directives are not available, therefore it is expected that the strategy activation will result in the topology discovery only. The first topology element of specified tmodel tracker states that all bundles, in containers fuse1 – fuse3, conforming to the $credit\cdot.*$ regular expression and all their inner elements (recursive attribute set to true), i.e. service interfaces, implementations and instances should be discovered. The second topology element states that JVM in containers fuse1 – fuse4 should also be discovered. During the evaluation of the topology discovery the following actions were performed:

1. activating $MG_{TD}$ monitoring strategy;
2. installing additional bundles credit-dummy and foo-dummy (both contain only one empty class) to containers fuse1, fuse2 and fuse4;
3. changing several times the state of credit-dummy and foo-dummy bundles in container fuse2, (from ACTIVE to RESOLVED state [47] and the other way around);
4. undeploying credit-tax bundle from container fuse2 and deploying it to container fuse1;
5. simulating containers failure by turning off fuse4 and fuse3;
6. deploying another credit-provider bundle to container fuse2.

After the execution of action 1, an appropriate monitoring subscriptions were announced in the federation and topology retriever at each container started reporting the required state of topology. The data from containers was reaching $MHQ$ in the following order: fuse1, fuse2, fuse3 and fuse4, which complies with the general topology of the CCP.
Listing 7.1: Monitoring goal strategy, named \( MG_{TD} \), declaring the topology discovery in the fragment of the \( CCP \) application. The strategy has specific time frame and has to be triggered by given topology condition

application (data from fuse4 has to travel the longest distance in the network of brokers). The whole operation of the topology retrieval measured from the moment of the \( MG_{TD} \) activation lasted 189 ms. The data from container fuse4 encompassed only the information about JVM while the data from the remaining containers encompassed the information about JVM together with the \( CCP \) bundles and their services. It complies with the contents of the tmodel tracker. Until the execution of action 2, by reason of the fact that there were no topological changes, no further data was sent to \( MHQ \).

Upon the completion of action 2, \( MHQ \) received notification about the installation of a bundle credit-dummy in containers fuse1 and fuse2. The installation of a bundle foo-dummy and changes in container fuse4 were ignored in accordance with the \( MG_{TD} \) tmodel tracker. As a result of action 3, \( MHQ \) received notification about the topology change, each time the state of bundle credit-dummy was changed. Each notification was delivered after 11 ms (average value) from occurrence of given bundle state change. What is important, no slow down was observed when action 3 was executed with and without DAMON being deployed. This is caused by an asynchronous dispatching of the topological events (cf. Section 5.3.1). The execution of action 4 resulted in appropriate updates to the topology plane of the created SOA system model - first removing credit-tax bundle from fuse2 and then adding it to fuse1. After the completion of action 5, \( MHQ \) needed 164 ms to detect that containers fuse4 and fuse3 were removed from the federation. Finally, the execution of action 6 resulted in fulfilling constraint of the \( MG_{TD} \) topology condition resulting in the monitoring goal deactivation.
Conclusions

This experiment was touching the following expectations: \( T1, R1, R2, R5 \). The goal-oriented management of the monitoring process (\( T1 \)) and the monitoring mechanisms related to the topology plane of the SOA system layer (\( R1 \)) were verified in the second part of the experiment, where monitoring the goal containing tmodel tracker allowed not only to retrieve the requested initial topology, but also, to track its changes with a minimal delay. The correctness of the topology updates and their insignificant influence on the system functioning contributed to the verification of the mapping monitoring goal to the monitoring subscriptions and to the topology discovery mechanisms (\( R5 \)). The ease of the environment enrichment (\( R2 \)) was verified in the first part of the experiment, where appropriate elements of the DAMON framework were effortlessly deployed without a need of changing the CCP application. It was possible thanks to: the usage of the JMX technology, leveraging the d-provider management capabilities and the execution of scripts which automated container modification.

7.3.2 Goal-orientation Addressing Aspects of Dynamics

This experiment focuses on situations when aspects of the SOA dynamics (cf. Section 3.1) are manifesting themselves in the monitored application. Therefore, the following three cases revealing the potential of a declarative, goal-oriented DAMON approach are considered: (i) identification of the causality model followed by a causal-driven discovery of the topology model; (ii) measuring response time in a fragment of the CCP application without any steering directives; (iii) measuring response time in the same fragment of CCP with additional support from the steering layer. During the evaluation of each case, the dynamic environment changes are simulated and the DAMON reaction is analyzed.

In order to evaluate the proposed cases, four monitoring strategies, referred to as \( MG_{DN1}, MG_{DN2}, MG_{DN3} \) and \( MG_{DN4} \), were prepared. The strategies are presented in Listing 7.2. \( MG_{DN1} \) requires identification of the CCP causality in all Fuse containers. \( MG_{DN2} \) expects a discovery of the topology in the two specific flows of the causality model. The strategy \( MG_{DN3} \) uses both direct and indirect specification of the measurement descriptor to monitor: all OP services, all services of bundle credit-calc and all services spanned by the blue service flow. This strategy does not have any steering directives, therefore all services are expected to be monitored all the time. This is changed in strategy \( MG_{DN4} \), being an extended version of \( MG_{DN3} \), which contains steering configuration allowing for testing the following aspects of the control loop (cf. Section 4.2): loop Phase I – nominals identification; loop Phase II – sentinels selection and finally the loop’s tracking cycle between these two phases. This experiment does not cover Phase III of the loop nor the diagnosis cycle, which are considered in detail in the next two experiments.
Chapter 7. Evaluation

Listing 7.2: Monitoring strategies for the CCP application used in experiment II: strategy 1 (named $MG_{DN1}$) – identification of causality; strategy 2 (named $MG_{DN2}$) – flow-based topology discovery; strategy 3, (named $MG_{DN3}$) – response time measurement; strategy 4 (named $MG_{DN4}$) – extension of strategy which adds steering directives

This experiment continues the previous one making the following assumptions: the system topology was reverted to its original state; the monitoring strategy ($MG_{TD}$ presented in Listing 7.1) is still active.

During the realization of this experiment the following actions were performed:

1. launching credit-consumer application, which starts sixty threads (twenty for each service flow) continuously generating invocations of the CCP service;
2. activating the $MG_{DN1}$ strategy and checking the obtained causality model;
3. using the causality model to create the $MG_{DN2}$ strategy, then activating it;
4. adding new Tax and Credit History services, TH4 and CH4, which offer attractive quality with reasonable response time;
5. simulating changing availability of TH4 and CH4 services by their deployment and undeployment in periods of 60 seconds;
6. activating the $MG_{DN3}$ strategy, then repeating action 4 to simulate dynamic changes;
7. extending the $MG_{DN3}$ strategy with steering directives and activating resulting $MG_{DN4}$ strategy, then repeating action 4 to simulate dynamic changes.

Causality Identification and Extended Topology Discovery

Before the execution of experiment actions, the topology model gathered by $MG_{TD}$, i.e. elements of the CCP application located in containers fuse1, fuse2, fuse3, was already available in the monitoring center ($M_{HQ}$). The completion of action 1 resulted in a continuous execution of all three flows without incurring any monitoring overhead (no strategy influencing the invocations has been yet activated). Upon the execution of action 2, the whole causality model of CCP was gathered after period of 265 ms. This period was needed for a propagation of the $MG_{DN1}$ activation and then for waiting until all services of CCP are invoked in all flows. It stems from the fact that the causality model is detected only when actual invocations are made. If the consumer application started in action 1 did not cover all service flows, then the missed ones would be simply left unidentified. Thanks to the network traffic monitoring, it was verified that the causality information was sent to $M_{HQ}$ only during first time given invocation was intercepted by osgi-sniffer. All subsequent invocations were ignored, what was expected behavior ensuring that all details of the causality are identified without consuming unnecessary resources (network bandwidth).

The identified causality model represented correctly the design of the CCP application presented in Figure 7.1, however three detected flows were not covering bundle (credit-db) and services (persistence layer) deployed to container fuse4. Since these elements were not covered by the $MG_{TD}$ strategy, then they were missing from the topology model and as a consequence the proper identification of their causality was not possible. When it comes to container fuse4, the causality model contained only the information that asynchronous invocations of all three flows (red, green, blue) are going in and out of this container. The execution of action 3 allowed for the alleviating this issue. The tmodel tracker of $MG_{DN2}$ refers to the two already identified flows: the green one and the blue one, and expects that all topology elements spanned by these flows are discovered. Thanks to that, a credit-db bundle and its services P1 (flow blue), P2 (flow green), P3 (flow blue) were detected after period of 186 ms elapsed from the moment of the $MG_{DN2}$ activation.

The execution of action 4, not only changed the topology of the CCP application but
also influenced two service flows. The modified application fragment is presented in Figure 7.4. The two services, TH4 and CH4, were deployed in a new bundle named credit-tax-new. The availability of these services resulted in the following flow changes: (i) since CH4 offers similar quality to CH3 but at a lower price, services TH2 (red flow) and TH3 (blue flow) switched from CH3 to CH4; (ii) availability of TH4 caused recomposition of OP4, which treated TH4 as yet another source for good estimation of the customers tax and credit history; (iii) the previous change resulted in the invocation of blue flow between TH4 and CH4. Since the $MG_{TD}$ strategy is still active and its tmodel tracker matches the name of credit-tax-new bundle, the topology events about deployment of this bundle and registration of its services were delivered to $M_{HQ}$ after 12 ms. The information about flow changes were also delivered to $M_{HQ}$ upon the first invocation of modified flow part, which was 206 ms on the average. The reported flow changes represented correctly the actual modifications grasped in Figure 7.4. Upon the completion of action 5, the reaction of the DAMON framework was similar to the one observed after action 4. Each deployment/undeployment of the credit-tax-new bundle resulted in an immediate notification about the topology change (12 ms) and an appropriate notification about the flow changes, as soon as services OP4, TH2, TH3, TH4, CH3, CH4 were invoked (212 ms on the average).

Fig. 7.4: Modified fragment of the CCP application after the deployment of bundle credit-tax-new, which contained service TH4 and CH4. Parts of CCP, which are not related to the modification are either excluded from the figure or faded out.

**Response Time Measurement**

The immediate result of the $MG_{DN3}$ strategy activation in the scope of action 6 was calculation of the measurement model structure for the whole identified causality model. This calculation was performed before processing the measurement activation of $MG_{DN3}$.
The resulted calculation output is presented in Figure 7.5. Structure correctly represented the causality model. Then $M_{HQ}$ started to process the measurement activation starting from direct specification of measurement descriptor and proceeding to the indirect specification. Finally, an appropriate monitoring subscriptions were announced in the containers federation. The minimal period parameter of the $MG_{DN3}$ measurement descriptor was set to 10000 ms, thus after 10133 ms measurement model was updated with information about the appropriate measurements. Not all measurements of the CCP application were reported, only the active measurements, grasped in Figure A.1 were covered. This is compliant with the direct and indirect measurements specified in the $MG_{DN3}$ covering: all the OP services, all services of the bundle credit-calc and all services spanned by the blue service flow. The results of the reported measurements along with the mcum formulas are presented in Table A.1. As can be seen, the measured exact ranges are appropriate for an average service response time declared in Figure 7.1.

$^{26}$The extended experiment results, due to their size, were moved to Appendix A.

Fig. 7.5: Structure of the CCP measurement model
Furthermore, the calculation of \( LLB \) and \( UUB \) (cf. Section 3.4.1) is compliant with the specification of the \( MG_{DN3} \) measurement descriptor.

Upon the execution of the dynamic changes related to the deploying/undeploying credit-tax-new bundle in the scope of action 6, the structure of measurement model has changed. The changes of measurement model, after the deployment of the credit-tax-new bundle, are grasped in Figure A.2 and the actual values of changed measurements are presented in Table A.2. The identified transformation of the measurement model is compliant with previously discussed changes of the topology and the causality models. After each deployment/undeployment of the credit-tax-new bundle, the measurement model was updated after 10129 ms on the average. The model state was appropriately changing between the state grasped in Figure A.1 and the one grasped in Figure A.2. Table A.2.

**Response Time Measurement with Steering**

After the activation of the \( MG_{DN4} \) strategy in the scope of action 7, the execution of the control loop has started. The topology and causality of the measurements covered by \( MG_{DN4} \) were already identified during previous actions, therefore, the loop execution directly started Phase I – nominals identification. In this phase, \( MHQ \) has executed

![Diagram](image_url)

Fig. 7.6: Sentinels selected in the monitored fragment of the CCP application
Algorithm 1 which after 2 iterations (because \textit{NomAcq.Iter} equaled 2 in \textit{MG}_{DN4}) identified nominal ranges of the active measurements (total duration of 20185 ms). The identified ranges were the same as ranges presented in Table A.1. Then, BN was created. Its structure was the same as the structure of the measurement model presented in Figure A.1 however instead of each \textit{mdirect} an \textit{mdprob} was presented and instead of each \textit{mcum} an \textit{mcprob} was present.

The initial state of the BN network is presented in Table A.3 (as can be seen, an uniform distribution of the \textit{mdprobs} probability was assumed). However, since all measurements related to this BN are activated, it is immediately reflected in the BN by evidencing all its \textit{mprobs} as being in the nominal state. When the creation of the BN was finished, the execution of Phase II (sentinels selection) has started. The services selected as sentinels are presented in Figure 7.6. The state of the BN after reducing the list of the activate measurements to the selected sentinels is presented in Table A.4. As can be seen, resulting over range probabilities are complying with DDD.Min\_\textit{suff} specified in \textit{MG}_{DN4}. The process of the sentinels selection took 456 ms.

After the selection and the activation of the sentinels, the control loop entered, so-called, stale period, where there were no changes in the measurement model. The stale period, has been lasting until the dynamic changes related to the deploying/undeploying credit-tax-new bundle in the scope of action 7 started.

Fig. 7.7: Sentinels selected in the monitored fragment of the CCP application after the deployment of the credit-tax-new bundle
Each deployment of the credit-tax-new bundle resulted in changing measurements listed in Table A.2, therefore the control loop was initiating tracking cycle and all modified measurements (beside CH3 which was excluded from blue flow) were again processed in Phase 1 of the loop. After 20153 ms (2 iterations of Algorithm 1), nominal ranges of the modified (and new) measurements were identified. Then, BN was transformed into a structure of the measurement model presented in Figure A.2 and loop entered Phase 2. The services selected as the sentinels are presented in Figure 7.7. As can be seen, service TH4 was identified as better sentinel than TH3, probably because of its lower response time – MCP^B(TH4) provides more information about CH4 than MCP^B(TH3). The timing of the sentinels selection process was similar as before (around 450 ms).

Likewise, each undeployment of the credit-tax-new bundle resulted in another tracking cycle of the control loop, in which a nominal state of the modified measurements was reassessed. BN was transformed into the previous state and the previous set of sentinels was selected. This proved that the DAMON prototype correctly reacts to the dynamic changes in the monitored environment.

Conclusions

This thorough experiment was addressing the following expectations: T1, T2, R1, R3, R4 and R5. The monitoring mechanisms (R1) and their run-time reconfiguration (R3) covering all models of the DAMON framework (topology, causality, measurement) were thoroughly verified by gathering and correctly updating these models along with the execution of the experiment actions. The subsequent feature, needed for proper mechanisms reconfiguration, was mapping high-level goal to the lower layers (R5). It was each time successfully verified that one of the four considered monitoring strategies has been activated. The propagation of the causality and the measurement information along with the service invocation (R4) was checked during the identification of causality and the acquisition of measurements (strategies MG_{DN2} and MG_{DN3} respectively). The aspects of selectivity (T2 and also R3) were evaluated in conjunction with the strategy MG_{DN4}, which allowed for restricting monitoring process to a set of necessary sentinels. Most importantly, the goal-oriented management of the monitoring process handling aspects of dynamics (T1) was deeply verified in each covered monitoring strategy. The activation of each goal was followed by dynamic changes related to the following categories of the SOA dynamics (cf. Section 3.1): (i) dynamics of the service registration and discovery; (ii) dynamics of the service components deployment. Each time, a monitoring strategy which was specified and activated only once, allowed for creating the related model and tracking its evolution in accordance with occurring dynamic changes. Thanks to that the monitoring goals were constantly fulfilled and the administrator was relieved from reacting, on his/her own, to changes in the environment.
7.3.3 Drill-down upon Single and Multiple Anomalies

The scope of this experiment covers evaluation of the adaptive drill-down algorithm (cf. Section 4.3.3) applying it to the two following cases: (i) occurrence of a single anomaly in one service; (ii) simultaneous occurrence of multiple anomalies in several services. The experiment uses the same consumer workload as the one present in experiment II. The monitoring strategy used in the course of this experiment, referred to as $MG_{DD}$, is presented in Listing 7.3. It is assumed that the strategies from the previous experiments identifying the topology ($MG_{TD}$, $MG_{DN2}$) and the causality ($MG_{DN1}$) of the whole CCP application are still activated, therefore $MG_{DD}$ contains only specification related to the measurement and the steering. The configuration of the steering is similar to the one presented in the $MG_{DN4}$ strategy but declared measurement descriptor covers all measurements of CCP.

During the experiment, the following actions were performed:

1. activating the $MG_{DD}$ strategy and checking measurement model, the created $BN$ and selected sentinels;
2. simulating single anomaly in service CH3 by adding 200 ms overhead to its execution;
3. changing the $DDD.Velocity$ of $MG_{DD}$ strategy from 1 to 3;
4. simulating multiple anomalies in the following services: OP1 (235 ms overhead), CH1 (185 ms overhead) and TH3 (160 ms overhead).

The execution of action 1 resulted in the identification of the measurement model covering all the CCP services. The structure of resulting model was the same as the one

```
<damon:MonitoringStrategy name="DrillDownCCP">
  <MeasurementActivation expirationPeriodMs="40000" minimalPeriodMs="10000">
    <MeasurementDescriptor>
      <Indirect metric="RESPONSE\_TIME">
        <TopologyElement>
          <TopologyElement type="bundle" namePattern="credit\-.*" flowPattern=".*"/>
        </TopologyElement>
      </Indirect>
    </MeasurementDescriptor>
  </MeasurementActivation>
  <SteeringDirectives>
    <NominalsAcquisition iterations="2"/>
    <DrillDownCondition>
      Mexact\_range:LB \&lt; Mnom\_range:LB * 0.85 ||
      Mexact\_range:UB \&gt; Mnom\_range:UB * 1.15
    </DrillDownCondition>
    <DrillDownDriver Best\_SS="6" Level_iter="2000" Most\_prob="10">
      <Fitness> entropy </Fitness>
      <Min\_suff> MP\_over\_range kit: 0.12 </Min\_suff>
      <Velocity type="direct"> 1 </Velocity>
    </DrillDownDriver>
  </SteeringDirectives>
</damon:MonitoringStrategy>
```

Listing 7.3: Monitoring goal strategy, named $MG_{DD}$ used for evaluation of adaptive drill-down

1. activating the $MG_{DD}$ strategy and checking measurement model, the created $BN$ and selected sentinels;
2. simulating single anomaly in service CH3 by adding 200 ms overhead to its execution;
3. changing the $DDD.Velocity$ of $MG_{DD}$ strategy from 1 to 3;
4. simulating multiple anomalies in the following services: OP1 (235 ms overhead), CH1 (185 ms overhead) and TH3 (160 ms overhead).
presented in Figure 7.5. Then Phase I of the control loop has started and after 20206 ms (because in $MG_{DD}$, $NomAcq.Iter$ equaled 2 and minimal period was set to 10000 ms) nominal ranges of all the CCP services were successfully identified. The actual ranges values are presented in Table B.2. Finishing Phase I triggered creation of BN, the structure of which was the same as the one of the measurement model (cf. Figure 7.5). The initial state of BN is presented in Table B.1. In case of mdprobs, the uniform probability distribution was assumed. Then, the execution of the control loop proceeded to Phase II, where sentinels were selected. A set of chosen sentinels is visible in the first iteration of drill-down presented in Figure 7.8. The state of BN after sentinels selection is depicted in Table B.3. As can be seen, similarly to previous experiment, resulting over range probabilities are complying with $DDD.Min_{suff}$ specified in $MG_{DD}$. The process of the sentinels selection took 482 ms and after its completion the control loop entered the stale period. The stale period has lasted until the execution of action 2.

The anomalous response time of CH3 services was observed in changes of the exact ranges at sentinels TH2 and CCI. Changing TH2, CCI exact ranges resulted in breaking the drill-down condition and, as a consequence, started the diagnosis cycle of the control loop. The cycle has started by execution of the loop Phase III, i.e. the adaptive drill-down. Each iteration of the drill-down (cf. Algorithm 3) contains the selection of the suspicious services and then choosing a specific number of these services for the measurement activation. The number of chosen services depends on the $DDD.Velocity$ parameter specified in the monitoring goal. Figure 7.8 presents each iteration of the drill-down execution and highlights suspicious services as well as the ones, which are monitored. The states of BN at each iteration are presented as tables in Figure B.1. If it had been obvious that there is only single anomaly in the system, then measurements of CH3 could be activated in the first iteration. However the drill-down algorithm assumes multiple anomaly sources and in each iteration, instead of pursuing the most probable cause, minimizes the uncertainty about the whole BN. Therefore, in the second and third iteration services P3 and TH3 were chosen respectively. It revealed that CH3 is certainly the reason of the system anomaly. The final state of BN after activation of CH3 measurement is grasped in iteration 4 presented in Figure B.1. Durations of the drill-down iterations were as follows: 323 ms, 156 ms, 57 ms, 24 ms. It can be seen that the duration was depending on the amount of the suspicious services. In between iterations, there were periods of 10154 ms in which ranges of monitored services were acquired.

After the drill-down was finished, the modified characteristic of CH3 was reported to the user. Since the user did not make any interaction, the diagnosis cycle has assumed that the change of CH3 service is a natural evolution and the cycle execution proceeded to Phase II. The changed characteristic of the CH3 response time influenced the subsequent sentinels selection presented in Figure 7.8 (services CCI, TH1, TH3, OVR were chosen). The state of BN after this sentinels selection is depicted in Table B.4.

---

27 The extended experiment results, due to their size, were moved to Appendix B.
Before the execution of actions 3 and 4, the response time of the CH3 service was reverted to its initial state. Therefore, the second drill-down started upon the completion of these actions was based on the previous set of sentinels: CCI, OP2, TH2, P1. The iterations of the second adaptive drill-down are depicted in Figure 7.9 while states of BN after each iteration are presented as tables in Figure 7.2. As can be seen, the occurrence of multiple anomalies resulted in identifying many services as suspicious (iteration 1). However, this was very quickly changed by the aggressive approach of

![Fig. 7.8: States of the monitored services in each iteration of the adaptive drill-down execution triggered by a single anomaly in service CH3 of the CCP application](image)

![Fig. 7.9: States of monitored services in each iteration of the adaptive drill-down execution triggered by multiple anomalies in services: OP1, CH1 and TH3 of CCP application. Legend is presented in Figure 7.8](image)
the drill-down (\textit{DDD.\textit{Velocity}} set to 3) resulting in an activation of the measurements in three crucial services: TH1, TH3 and OVR (iteration 2). Monitoring these services allowed for decreasing the cumulative indirect entropy of whole \textit{BN} from 11.45 to 5.28 (cf. Figure B.2). The next iteration, completely identified all anomalous services and completed the drill-down. Durations of the drill-down iterations were as follows: 597 ms, 345 ms, 46 ms (again relation to number of the suspicious services is visible). In between iterations, there were the same periods as in the previous drill-down execution. If the drill-down had not been so aggressive (\textit{DDD.\textit{Velocity}} set to 1), 6 algorithm iterations would be needed. After the characteristic of services OP1, CH1 and TH3 was identified as a system evolution (no interaction with the user), the next sentinels were chosen. It turned out that they are the same as the initial sentinels set (cf. Figure 7.9).

**Conclusions**

Experiment III was touching the following expectations: \textbf{T1}, \textbf{T2}, \textbf{T3}, \textbf{R1}, \textbf{R3}, \textbf{R4} and \textbf{R5}. All expectations apart from \textbf{T2}, \textbf{T3} were successfully addressed to the similar extent as in experiment II, but this time the monitoring mechanisms (\textbf{R1}, \textbf{R3}, \textbf{R4}) and the goal-orientation approach (\textbf{T1}, \textbf{R5}) were evaluated in the context of the dynamic changes related to non-functional aspect of services (different service response times), which could be classified to the third category of the \textit{SOA} dynamics identified in Section 3.1 (adaptability of services, service components and operational systems). A dynamic reconfiguration of the monitoring selectivity (\textbf{T2}) was intensively evaluated during the realization of the adaptive drill-down, which was able to successfully identify the root cause of system performance deviation (\textbf{T3}) in case of single anomaly as well as in case of several simultaneously occurring anomalies. An analysis of the drill-down iterations ensured that the selection of services, which monitoring is activated in subsequent phases, is performed in accordance with the goal’s fitness function requiring minimization of the system entropy. Moreover, the performance of the inferencing process (measured durations were no greater than 600 ms) was sufficient for effective functioning of the \textit{DAMON} framework.

### 7.4 Non-functional Evaluation

This section contains two experiments aiming at evaluation of the monitoring overhead and the aspects of the \textit{DAMON} framework scalability, which are crucial for fulfilling its expectations.

#### 7.4.1 Monitoring Overhead

This experiment covers the assessment of the monitoring overhead introduced by the \textit{DAMON} framework. The assessment is performed in the environment of the \textit{CCP} appli-
cation, where different scenarios are enacted. In each scenario, metrics related to the monitoring overhead are measured. The following metrics are considered: CPU usage, network bandwidth consumption and service response time. The overhead assessments cover all the monitoring mechanisms described in Chapter 5, i.e.: (i) topology discovery; (ii) causality identification (flow sniffer); (iii) m-interceptor: processing and propagating information for the mcum calculation and (iv) m-interceptor: identification and reporting measurement data. The experiment is divided into two parts: 

- overhead identification – measuring selected metrics related to the overhead in case of unmonitored system and in cases, where each chosen DAMON mechanism is activated;
- selectivity influence analysis – measuring overhead when selectivity of the monitoring process is changing.

**Overhead Identification**

During this part of the experiment the following actions were performed:

1. comparing overhead metrics before deployment of the DAMON framework and after its deployment but with the assumption that no monitoring goal is activated;
2. measuring overhead metrics in case of the topology retrieval and tracking;
3. measuring overhead metrics related to the causality identification and the m-interceptor operation.

In the scope of action 1, the workload involving sixty threads (twenty for each service flow) was generated. Values of the following metrics: CPU usage, network bandwidth consumption and service response time were gathered. It was revealed that a bare installation of the DAMON framework, without any monitoring goal turned on, does not influence the measured metric values. This proves that the mechanism of the interceptor socket, which passively waits for the monitoring goal activation, does not deteriorate the normal system operation.

The execution of action 2 have shown that both gathering of the initial system topology and tracking its changes, thanks to asynchronous dispatching of topology events, have no influence on the CPU consumption and the duration of the operations that change the topology. Only the network throughput was increased upon gathering the initial topology (fetching around 15 kB from each fuse container) and tracking its changes. The topology changes were simulated by executing action 5 of experiment II (deployment and undeployment of the credit-tax-new bundle). The size of each topology event received by $M_{HQ}$ was around 0.35 kB. The amount of the consumed bandwidth is proportional to the size of the federation (in the case of topology retrieval) and to the number of the topology changes (in the case of topology tracking). However, since the topology retrieval occurs rarely (only upon the activation of the monitoring goal) and the frequency of the topology changes (even in a really dynamic systems), is not high enough for causing significant bandwidth consumption (given the size of the topology event), both mechanisms do not introduce a significant overhead.
During the execution of action 3, the overhead metrics were measured during the following four phases (each phase lasted for 30 s): (i) no monitoring mechanism activated; (ii) activation of flow sniffers; (iii) activation of the m-interceptor mcum calculation and its propagation; (iv) activation of the m-interceptor reporting. The activation of the mechanisms in phases (ii) – (iv) covered all services of the CCP application. Values of the CPU usage and the service response time have not been changing between the phases apart from the regular fluctuations resulting from a normal execution of CCP. The only difference was identified in the metric of the consumed network bandwidth. The results of the measuring egress network throughput combined from all VMs hosting CCP are presented in Figure 7.10. It is visible, that the activation of the flow sniffers caused an increased bandwidth consumption in the beginning of phase (ii). It stems from the construction of the flow sniffer algorithm (cf. Algorithm 4), which reports a flow information only on the first of a given flow invocation. It is also apparent that a propagation of the mcum values, in phase (iii), does not generate significant throughput. It is possible thanks to the limited size of the propagated mcum values. The activation of the m-interceptors reporting in all the CCP services causes significant bandwidth consumption. In the case of the private cloud environment of CCP where usage of each resource is meticulously billed, the DAMON techniques aiming at decreasing the monitoring scope are strongly demanded.

![Combined egress throughput of all four VMs hosting the CCP application in four different phases. Phases 2-4 involve activation of a given monitoring mechanism for all services of CCP](image)

**Selectivity Influence Analysis**

The overhead identification proved that the network throughput is the only resource, which is significantly consumed by the DAMON framework. Therefore, this part of the experiment measures the throughput consumption in the scope of the following actions:
Chapter 7. Evaluation

1. manual selectivity changes performed by a gradual activation of the monitoring goals (results in Figure 7.11);
2. execution of the drill-down scenario described in action 2 (single anomaly) of the previous experiment (results in Figure 7.12);
3. execution of the drill-down scenario described in action 4 (multiple anomalies) of the previous experiment (results in Figure 7.13);

![Diagram of monitoring states](image)

**Fig. 7.11:** Egress throughput in all VMs hosting the CCP application influenced by manual selectivity changes. The current monitoring state of CCP is presented with the use of notation from Figure 7.8

136
Results acquired after the execution of action 1 are presented in Figure 7.11. As can be seen, the activation of each monitoring goal increases the throughput consumption. Reporting measurements in given VM affects not only the throughput consumed by this VM, but also the bandwidth consumed by all VMs being on the shortest path from given VM to $M_{HQ}$ (cf. the topology of the network of brokers in Figure 7.3). The results of the drill-down process executed in the scope of action 2 are presented in Figure 7.12. The difference in the bandwidth consumption between the phase of the nominals identification and the stale period is substantial. This proves that the drill-down approach can successfully limit the monitoring overhead. The differences between each drill-down iteration are not significant because only one additional service is covered.

Fig. 7.12: Throughput influenced by the drill-down caused by single anomaly
Fig. 7.13: Throughput influenced by the drill-down caused by multiple anomalies in each iteration. This proves that the drill-down is able to locate the anomaly root-cause without causing the overhead similar to the one of the nominals identification phase, where all services are monitored. In the last phase, presented in Figure 7.12, it is visible, that the new set of the sentinels introduces higher overhead than the previous one. This is caused by the fact that in the activated monitoring goal ($MG_{DD}$ presented in Listing 7.3), the fitness function relies on the entropy and not the overhead. As can be seen in Figure 7.13, the drill-down executed in the scope of action 3 is more aggressive (3 services monitored in each iteration). The aggressive drill-down approach introduces
higher overhead but results in quicker identification of anomalous services.

Conclusions

Experiment IV was touching the following expectations: T2, R1, R4 and R6. The performed overhead assessment covered all mechanisms of the DAMON framework (R1) and proved that all considered metrics, beside network throughput, are not significantly influenced by the monitoring process. The evaluation of the network bandwidth consumption caused by the flow sniffers and the m-interceptors (without reporting turned on) ensures that the propagation of the information along with the service invocation (R4) does not result in a visible increase of throughput. The second part of the experiment shows that an appropriate management of the monitoring selectivity (T2) allows for minimizing the caused overhead in different phases of the control loop. The experiment as a whole shows that the overhead introduced by the DAMON framework is acceptable and therefore, does not prevent achieving the appropriate scalability (R6).

7.4.2 Scalability Aspects

This experiment assesses the following aspects of the DAMON framework scalability:

- relation between the size of Bayesian network and the number of the sentinels selected in the beginning of the stale period;
- relation between the size of Bayesian network and the duration of inferencing process. The inferencing process is performed in Phases II and III of the control loop and its scalability is important for the overall efficiency of DAMON;
- dependence of the monitoring overhead (expressed by consumed bandwidth) on two factors: the size of the monitored containers federation and the number of deployed M_HQ instances;
- relation between the number of anomaly incidents occurring in the application and the monitoring overhead expressed by the additional amount of data sent in the federation.

The importance of the relation between the size of $BN$ and the number of sentinels can be concluded from the previous experiments – the number of monitored services has a direct influence on the incurred overhead. Since the stale period is typically, the longest phase of the control loop, the low sentinels number is important. The discussed relation is grasped in Figure 7.14. The figure contains six different series representing different numbers of service flows present in the causality model. As can be seen, all series can be approximated by the linear functions. This ensures that even systems of significant scale can be covered with relatively small number of the sentinels. Additionally, the results
Chapter 7. Evaluation

Fig. 7.14: Number of the sentinels depending on the size of BN

presented in the figure show that the higher number of flows, the less sentinels are needed for guarding the nominal system state. It is caused by the fact, that the higher number of the flows results in more indirect information passed between measurements. Thus, the monitoring of the single service allows for inferencing about the higher number of other services.

The previous experiments proved that the steering model realized with the use of Bayesian networks is very useful in dynamic service-oriented systems. Unfortunately, in the context of a computational complexity, the general problem of the BN inferencing is classified as NP-hard [116]. Therefore, the time needed to perform the inferencing process may be exponentially dependent on the amount of nodes in the BN. In the steering layer, the node of BN represents a single measurement from the measurement model. Therefore, the performance of the algorithms executed by the inferencer component of the steering layer is directly dependent on the number of measurements covered by the monitoring goal. Figure 7.15 presents this dependence in the context of three different numbers of the service flows. As can be seen, all series can be approximated by the exponential function, which makes the inferencing process not scalable. However, the potential of the goal-driven management of the monitoring process can be easily leveraged to decompose the inferencing process. Figure 7.16 presents the same relation as Figure 7.15 but now instead of single monitoring goal, there are several goals. Each of the goals does not cover more than 30 measurements. As can be seen, such approach makes duration of inferencing process linearly dependent on the number of measurements, which ensures the appropriate scalability. The current implementation of DAMON prototype does not cover automation of goal decomposition. This improvement is planned to be achieved in the scope of the future work.
Chapter 7. Evaluation

![Duration of BN inferencing process](image1)

**Fig. 7.15**: Duration of **BN** inferencing process when the single monitoring goal covers all measurements. The first two series are grasped in the separate chart to highlight the exponential growth.

![Duration of BN inferencing process](image2)

**Fig. 7.16**: Duration of **BN** inferencing process when measurements are covered by separate monitoring goals – one goal covers no more than 30 measurements.

The previous experiment has shown that the consumption of network bandwidth may significantly increase along with the number of services covered by the monitoring process. The throughput generated by the realization of the monitoring process is dependent on the two factors: the amount of monitored services and the distance, expressed in the number of the **JMS** brokers, between monitored services and the monitoring center. A design of the **DAMON** framework allows for adding new **M_HQ** nodes to the existing containers federation. Figure 7.17 presents how consumed bandwidth depends on the...
number of nodes in the federation and the number of monitoring center instances. The presented dependence assumes that each node (OSGi container) is sending the same monitoring workload to $M_{HQ}$ and that containers are joined in a chain with $M_{HQ}$ connected at the end of it (the worst-case topology). In such situation, the amount of the consumed bandwidth is quadratically dependent on the number of containers. As can be seen, a deployment of the new monitoring center can significantly decrease the consumed bandwidth even in case of high number of the containers.

So far, the monitoring overhead was represented by the additional throughput caused by the monitoring process. However, this was only an indirect overhead representation. The actual overhead is equal to throughput multiplied by the execution time, i.e. the additional amount of data, which is sent in the whole federation because of the monitoring process realization. This indicator influences directly the costs of the CCP application execution in the assumed private cloud environment (cf. Section 7.2.2), where each consumed resource is appropriately billed. The additional amount of data is directly dependent on the number of anomaly incidents, which are occurring in the application during given time frame. In order to analyze this dependence, the drill-down scenarios presented in Figures 7.12, 7.13 were executed separately in the time frame of one hour. During each scenario execution, five anomaly incidents (being the same as the ones presented in Figures 7.12, 7.13) were simulated at random moments. The results of the scenarios executions are presented in Figures 7.18, 7.19. The area in blue, referred to as the DAMON overhead, represents the additional amount of data sent in the federation because of DAMON. As can be seen, most of the time the overhead is caused by the stale period. The overhead of the drill-down is incurred only when anomaly incidents are occurring. Then, the incurred overhead is greater in case of the drill-down velocity equal to three (cf. Figure 7.19) than in case of the drill-down velocity equal to one (cf.
Fig. 7.18: Operation of the DAMON framework in one hour time frame, in which five anomaly incidents occurred at single CCP service – CH3. During each incident, a scheme of the drill-down was the same as the one presented in Figure 7.12.

Fig. 7.19: Operation of the DAMON framework in one hour time frame, in which five anomaly incidents occurred at multiple CCP services – OP1, CH1 and TH3. During each incident, a scheme of the drill-down was the same as the one presented in Figure 7.13.
Chapter 7. Evaluation

Figure 7.18. However, in the latter case, the drill-down execution is longer.\(^{28}\) Another important observation is that the overhead caused by Phase I of the control loop (nominals identification) is incurred only at the beginning of the monitoring process. Later on, the activation of all the measurements would be needed only in case where an anomaly incident occurs at all the application services, which is unlikely. Beside the **DAMON** overhead, both discussed figures are grasping the monitoring gain (area in red), which is the difference between the amount of data sent during full monitoring (all services are constantly monitored) and the amount of data sent upon adaptive monitoring of **DAMON**. As can be seen the overhead is relatively small in comparison to the gain.

In order to identify the relation between the number of anomaly incidents and the total sent data, the scenarios, which results are presented in Figure 7.18 and Figure 7.19 were executed with different anomaly incidents frequencies. The results are presented in Figure 7.20. The monitoring overhead is the difference between the given amount of the sent data and the amount sent when no monitoring is enabled. As can be observed, in the case of a single anomaly, the overhead is slightly smaller. This is caused by smaller value of the drill-down velocity. The results prove that, along with increasing number of anomaly incidents, the overhead does not increase significantly. Even, in the case of sixty incidents (one incident per minute) the overhead is significantly lower than the one in the case of full monitoring. This proves that the **DAMON** scalability is suitable for the **SOA** systems with an increased level of dynamics.

\(^{28}\)This is caused by the additional iteration noticeable upon comparing Figure 7.12 and Figure 7.13.

![Fig. 7.20: Amount of data sent in the federation during one hour of **DAMON** operation depending on the number of anomaly incidents occurring in the **CCP** application](image)

Fig. 7.20: Amount of data sent in the federation during one hour of **DAMON** operation depending on the number of anomaly incidents occurring in the **CCP** application.
Conclusions

Experiment V has covered important aspects of the scalability which were required by expectation R6. It was revealed that decreasing the monitoring overhead achieved by selecting a set of sentinels scales well, taking into consideration the size of BN (number of measurements), as well as its complexity (number of service flows). Additionally, the experiment has proved that the problem of the exponential duration of the BN inferencing can be easily solved by the proper management of the monitoring goals, which allows for achieving linear dependence between the duration and the BN size. It was also verified that the consumption of the network bandwidth caused by the realization of the monitoring process can be carefully controlled by the appropriate deployment of the monitoring center instances. Finally, it was proven that the DAMON monitoring overhead is relatively small in comparison to its gain and that in case of more dynamic systems, the overhead does not increase in a significant way.

7.5 Summary

This chapter presented thorough evaluation of the DAMON prototype. The primary purpose of the evaluation was to verify the three essential prototype properties, i.e. T1 – goal-oriented management of the monitoring process; T2 – dynamic adjustment of the monitoring selectivity; T3 – on-demand root cause identification. Additionally, the evaluation was aiming at assessing the fulfillment of six features required from the DAMON framework, namely: R1 – diverse monitoring mechanisms; R2 – seamless, non-intrusive enrichment of the SOA environment; R3 – run-time mechanisms reconfiguration allowing for changing the monitoring selectivity; R4 – propagation of specific information along with the service invocation; R5 – mapping of the high-level monitoring goal to lower layers and R6 – assurance of efficiency and scalability in a dynamic, distributed environment.

The evaluation covered five different experiments. The first three were focused on the general prototype functionality, while the latter two assessed the non-functional aspects related to the monitoring overhead and the scalability. Experiment I has proven that the SOA environment can be effortlessly enriched with the DAMON prototype (R2) and that topology discovery and tracking is realized in accordance with the defined monitoring goal (R5). Experiment II revealed that the goal-oriented management of the monitoring process (T1) is capable of handling aspects of the SOA dynamics in the context of: causality identification, extended topology discovery and measurement acquisition (R1, R4). Experiment III covered the execution of the drill-down process, which allowed for successful verification of changing the monitoring selectivity (T2, R3) and ensured correctness of root cause identification in the case of both single and multiple anomalies (T3). Experiment IV presented the monitoring overhead assessment, which ensured that the incurred monitoring overhead is acceptable (R6) and that it can be
carefully controlled by the adjustment of the monitoring selectivity \( (T2, R3) \). Finally, experiment V has proven that the scalability of the steering layer and the throughput management (possible thanks to deployment of multiple monitoring center instances) and the overhead incurred at higher anomaly frequencies are sufficient for targeting dynamic large-scale SOA systems \( (R6) \). Combined together, all experiments were able to successfully verify the fulfillment of each identified expectation.
Chapter 8

CONCLUSIONS

The contribution of the previous chapters covered a complete solution of the goal-driven adaptive monitoring targeted for the dynamic service oriented systems. This chapter summarizes the contribution, verifies the thesis statement, and discusses the potential directions of the future work.

The constant evolution of the SOA systems has a significant influence on the systems dynamics and the complexity. The dynamic changes of the user requirements and the available resources, as well as the increasing systems scale require restricting the scope of the monitoring process. However, selecting elements of a complex system which could be excluded from a monitoring, in the context of specific constraints, is not a trivial task. There are several research issues which need to be resolved.

The first issue consist in gathering and representing the knowledge about the monitored system structure and the dynamic relations between its elements. Then, there is a need for grasping the mentioned constraints and the high-level goal which should be considered during the realization of the monitoring process. Subsequently, the general scheme used for adjusting the scope of the monitoring process has to be proposed. Finally, it is required to map the abstract concepts into an implementation-ready architecture which allows for addressing the real-life SOA applications.

This chapter discusses the extent to which the proposed goal-driven adaptive monitoring approach resolves the mentioned research issues. Moreover, the chapter identifies the aspects which were not entirely covered by the dissertation and therefore, are important in the context of the future improvements.
8.1 Thesis Verification

The formulated thesis statement expects a proposition of the adaptable monitoring mechanisms which can be used for enriching the dynamic service oriented systems. The mechanisms are supposed to enable the goal-oriented management of the monitoring process that handles the SOA dynamics by adjusting the monitoring selectivity and identifying the root cause of encountered system anomalies.

This dissertation proposes a novel concept of the adaptive monitoring which resolves the research issue of gathering and representing knowledge about the monitored system. To grasp the knowledge about the system structure and related dynamic relations, the concept introduces several models and defines the following three layers: the SOA system, the measurement and the steering. The structure is represented by means of the topology model, while the dynamic relations are represented with the use of the causality model and the measurement model.

The management of the monitoring process is aided by a novel concept of the multi-dimensional monitoring goal, which instance is supplied to the steering layer. The layer follows the goal by adjusting the monitoring scope in accordance with the goal strategy. The monitoring goal strategy is a document which specifies what, where and how should be monitored. The realization of the monitoring process is divided into three phases. Firstly, the typical system behavior is discovered and remembered. Then, the monitoring is restricted to a minimal number of elements which ensure the following: (i) minimizing the monitoring overhead; (ii) verifying whether the system characteristic is similar to its typical behavior. The monitoring process sustains the monitoring scope restriction until a system anomaly is discovered. The detection of an anomaly triggers the third process phase, which covers the execution of the so-called adaptive drill-down, i.e., gradual increasing of the monitoring scope performed until the identification of the anomaly sources.

The proposed abstract concept can have many possible realizations. This dissertation leverages the BN theory for modeling the monitored system and for selecting the monitoring scope which is the most appropriate in the given monitoring process phase. The proposed Bayesian-based process steering imposes a specific expectations in the context of monitoring mechanisms. The dissertation proposes the novel Dynamic Adaptive Monitoring Framework (DAMON) covering detailed design of all the expected monitoring mechanisms. The framework involves also a complete architecture which integrates the mechanisms with the components of the steering layer.

The thorough evaluation performed on the OSGi-based DAMON prototype leads to several important conclusions. First of all, DAMON is able to correctly identify introduced models and allows for successful realization of all phases of the adaptive monitoring process. The performed experiments reveal that the proposed goal-orientation is capable of handling various aspects of the SOA dynamics related to the topology, the causality and
the measurements. It is possible thanks to the dynamic on-demand instrumentation performed upon activation of given monitoring goal strategy. Moreover, the DAMON framework allows for restricting the monitoring overhead in a significant manner. The overall overhead of DAMON is relatively low in comparison to the offered gain i.e., the difference between the normal non-adaptive monitoring and the approach proposed in this dissertation. Finally, the properties of the DAMON framework related to the aspect of the scalability ensure that the solution can be used in the SOA systems of a significant scale. The combination of all formulated conclusions ensure that the contribution of this dissertation was not only sufficient for the successful verification of the thesis statement, but it also introduces a significant improvement to the current state of the art in the field of the SOA systems adaptability and monitoring.

8.2 Future Work

It was proved that the proposed solution of the goal-driven adaptive monitoring can be successfully used in the dynamic service oriented systems. However, there are several aspects which could be improved in the scope of the future work. These are as follows.

- The steering model could be extended towards BN with continuous variables. The proposed solution uses BN with discrete states. This requires representing the service measurement with a set of metric ranges, which results in loosing the part of information about service behavior. Such an issue would be solved by introducing continuous BN variables.

- As noticed in experiment V, it would be valuable to automate the goal decomposition in case of large-scale systems. This is necessary to limit the complexity of the BN inferencing. For now the goal decomposition has to be performed manually by the user.

- Conflicting situations which can occur in case of overlapping BN, could be solved in a more sophisticated way. The current simplified approach assumes assigning priorities to overlapping BN and solving the conflict by choosing the highest priority. It would be valuable, to propose a novel approach which solves the conflict by merging different BN and combining their information.

- Reasoning upon the steering model could be implemented with the use of an approximate BN inferencing. Currently, the DAMON prototype uses the exact inferencing implemented by means of the Junction Tree algorithm, which can result in a high computational complexity. An approximate method which decreases the complexity but does not influence the inferencing quality, would be beneficial.

- Algorithms 2 and 3 could be redesigned with a use of more effective heuristics. For now, the algorithms are using a modification of greedy approach, which iterates
towards the most promising service combinations. A more sophisticated approach taking into consideration the measurement and the causality models, as well as the historical monitoring data, would be more appropriate.

Each enlisted aspect of the possible future improvements is focused on the particular solution element. A more general improvement could be achieved by implementing the DAMON framework for some other technologies than OSGi. For example, implementation for Web Services which are very common in the existing SOA systems, would allow for even more extensive framework evaluation.
Appendix A

EXTENDED EVALUATION RESULTS: EXPERIMENT II

This appendix contains extended results of experiment II described in Section 7.3.2

Fig. A.1: Active measurements of CCP application measurement model in experiment II after execution of action 6 but before deployment of credit-tax-new bundle. Measurements, which are not active are faded out.
### Appendix A. Extended Evaluation Results: Experiment II

#### Table A.1: Measurements received by $M_{HQ}$ in experiment II after execution of action 6 but before deployment of credit-tax-new bundle

<table>
<thead>
<tr>
<th>measurement</th>
<th>$LLB$</th>
<th>$LB$</th>
<th>$UB$</th>
<th>$UUB$</th>
<th>mcum formula</th>
</tr>
</thead>
</table>
| $MC^R(CCI)$ | 719   | 1439 | 1632 | 7632  | $MC^R(CCI) =$  
|             |       |      |      |       | $MD(CCI) + MD(OP3) + MC^R(P1)$ |
| $MC^G(CCI)$ | 103   | 206  | 233  | 2233  | $MC^G(CCI) =$  
|             |       |      |      |       | $MD(CCI) + MD(OP2)$ |
| $MC^B(CCI)$ | 1250  | 2500 | 2827 | 11827 | $MC^B(CCI) =$  
|             |       |      |      |       | $MD(CCI) + MC^B(OP4) + \max(MC^B(P1), MC^B(P3))$ |
| $MC^R(P1)$  | 612   | 1224 | 1389 | 5389  | $MC^R(P1) =$  
|             |       |      |      |       | $MD(P1) + MC^R(OVR)$ |
| $MC^B(P1)$  | 603   | 1205 | 1367 | 4367  | $MC^B(P1) =$  
|             |       |      |      |       | $MD(P1) + MC^B(OVR)$ |
| $MC^B(P3)$  | 584   | 1168 | 1325 | 4325  | $MC^B(P3) =$  
|             |       |      |      |       | $MD(P3) + MC^B(OVR)$ |
| $MC^B(OP4)$ | 601   | 1201 | 1353 | 6353  | $MC^B(OP4) =$  
|             |       |      |      |       | $MD(OP4) + MC^B(TH2) + MC^B(TH3)$ |
| $MC^R(OVR)$ | 374   | 747  | 848  | 3848  | $MC^R(OVR) =$  
|             |       |      |      |       | $MD(OVR) + MC^R(OC1)$ |
| $MC^G(OVR)$ | 364   | 729  | 827  | 2827  | $MC^G(OVR) =$  
|             |       |      |      |       | $MD(OVR) + MD(OC2)$ |
| $MC^B(OVR)$ | 364   | 729  | 827  | 2827  | $MC^B(OVR) =$  
|             |       |      |      |       | $MD(OVR) + MD(OC2)$ |
| $MC^B(TH2)$ | 271   | 542  | 615  | 2615  | $MC^B(TH2) =$  
|             |       |      |      |       | $MD(TH2) + MD(CH2)$ |
| $MC^B(TH3)$ | 260   | 519  | 580  | 2580  | $MC^B(TH3) =$  
|             |       |      |      |       | $MD(TH3) + MD(CH3)$ |
| $MC^R(OC1)$ | 224   | 448  | 509  | 2509  | $MC^R(OC1) =$  
|             |       |      |      |       | $MD(OC1) + MD(OP1)$ |

| $MD(CCI)$ | 47    | 93    | 106  | 1106 |
| $MD(P1)$  | 238   | 476   | 541  | 1541 |
| $MD(P3)$  | 220   | 439   | 498  | 1498 |
| $MD(OP2)$ | 56    | 112   | 127  | 1127 |
| $MD(OP3)$ | 61    | 121   | 138  | 1138 |
| $MD(OP4)$ | 70    | 140   | 159  | 1159 |
| $MD(OVR)$ | 149   | 299   | 339  | 1339 |
| $MD(TH2)$ | 154   | 308   | 350  | 1350 |
| $MD(TH3)$ | 162   | 323   | 357  | 1357 |
| $MD(OC1)$ | 187   | 374   | 424  | 1424 |
| $MD(OC2)$ | 215   | 430   | 488  | 1488 |
| $MD(CH2)$ | 117   | 234   | 265  | 1265 |
| $MD(CH3)$ | 98    | 196   | 223  | 1223 |
| $MD(OP1)$ | 37    | 75    | 85   | 1085 |

Table A.1: Measurements received by $M_{HQ}$ in experiment II after execution of action 6 but before deployment of credit-tax-new bundle
Appendix A. Extended Evaluation Results: Experiment II

Fig. A.2: Changes in measurements of CCP application measurement model in experiment II after execution of action 6 and after deployment of credit-tax-new bundle. Measurements, which are not active are faded out.

<table>
<thead>
<tr>
<th>Measurement</th>
<th>LLB</th>
<th>LB</th>
<th>UB</th>
<th>UUB</th>
<th>Mcum Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>(MC_B(CCI))</td>
<td>1450</td>
<td>2900</td>
<td>3248</td>
<td>14248</td>
<td>(MC_B(CCI) = MD(CCI) + MC_B(OP4) + \max(MC_B(P1), MC_B(P3)))</td>
</tr>
<tr>
<td>(MC_B(OP4))</td>
<td>799</td>
<td>1597</td>
<td>1787</td>
<td>8787</td>
<td>(MC_B(OP4) = MD(OP4) + MC_B(CH3) + MC_B(CH4))</td>
</tr>
<tr>
<td>(MC_B(CH3))</td>
<td>251</td>
<td>501</td>
<td>557</td>
<td>2557</td>
<td>(MC_B(CH3) = MD(CH3) + MD(CH4))</td>
</tr>
<tr>
<td>(MC_B(CH4))</td>
<td>206</td>
<td>412</td>
<td>463</td>
<td>2463</td>
<td>(MC_B(CH4) = MD(CH4))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Measurement</th>
<th>LLB</th>
<th>LB</th>
<th>UB</th>
<th>UUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>(MD(CH3))</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>(MD(CH4))</td>
<td>89</td>
<td>178</td>
<td>200</td>
<td>1200</td>
</tr>
<tr>
<td>(MD(CH4))</td>
<td>117</td>
<td>234</td>
<td>263</td>
<td>1263</td>
</tr>
</tbody>
</table>

Table A.2: Changes in measurements received by \(M_{HQ}\) in experiment II after execution of action 6 and after deployment of credit-tax-new bundle.
### Appendix A. Extended Evaluation Results: Experiment II

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP^R(CCI)$</td>
<td>12,37%</td>
<td>4,47%</td>
<td>83,16%</td>
<td>0,55</td>
</tr>
<tr>
<td>$MCP^B(CCI)$</td>
<td>5,89%</td>
<td>2,61%</td>
<td>91,51%</td>
<td>0,34</td>
</tr>
<tr>
<td>$MCP^R(P1)$</td>
<td>19,98%</td>
<td>6,41%</td>
<td>73,61%</td>
<td>0,72</td>
</tr>
<tr>
<td>$MCP^B(P1)$</td>
<td>26,93%</td>
<td>9,57%</td>
<td>63,51%</td>
<td>0,87</td>
</tr>
<tr>
<td>$MCP^B(P3)$</td>
<td>26,97%</td>
<td>9,47%</td>
<td>63,57%</td>
<td>0,86</td>
</tr>
<tr>
<td>$MCP^B(PP4)$</td>
<td>16,14%</td>
<td>5,25%</td>
<td>78,60%</td>
<td>0,64</td>
</tr>
<tr>
<td>$MCP^R(OR)$</td>
<td>24,71%</td>
<td>9,53%</td>
<td>65,76%</td>
<td>0,85</td>
</tr>
<tr>
<td>$MCP^G(CCI)$</td>
<td>31,32%</td>
<td>13,89%</td>
<td>54,79%</td>
<td>0,97</td>
</tr>
<tr>
<td>$MCP^G(OR)$</td>
<td>31,99%</td>
<td>15,48%</td>
<td>52,53%</td>
<td>0,99</td>
</tr>
<tr>
<td>$MCP^B(TH2)$</td>
<td>31,47%</td>
<td>16,43%</td>
<td>52,10%</td>
<td>1,00</td>
</tr>
<tr>
<td>$MCP^B(TH3)$</td>
<td>31,13%</td>
<td>16,68%</td>
<td>52,19%</td>
<td>1,00</td>
</tr>
<tr>
<td>$MCP^R(OC1)$</td>
<td>29,08%</td>
<td>18,18%</td>
<td>52,74%</td>
<td>1,01</td>
</tr>
</tbody>
</table>

$MDP(*)$      | 33,33%| 33,33%| 33,33%| 1,10    |

**mdirect cumulative entropy:** 15,38

Table A.3: Initial state of BN in experiment II. Range probabilities in all mdprobs are uniformly distributed, what is represented by single row containing $MDP$ with asterisk.
### Appendix A. Extended Evaluation Results: Experiment II

#### Table A.4: State of BN in experiment II after selection of sentinels. Rows containing mprobs of sentinel services are highlighted

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP^R(CCI)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^B(CCI)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^R(P1)$</td>
<td>5.07%</td>
<td>94.88%</td>
<td>0.05%</td>
<td>0.21</td>
</tr>
<tr>
<td>$MCP^B(P1)$</td>
<td>11.09%</td>
<td>88.61%</td>
<td>0.30%</td>
<td>0.37</td>
</tr>
<tr>
<td>$MCP^B(P3)$</td>
<td>36.00%</td>
<td>60.87%</td>
<td>3.13%</td>
<td>0.78</td>
</tr>
<tr>
<td>$MCP^B(OP4)$</td>
<td>3.53%</td>
<td>94.73%</td>
<td>1.74%</td>
<td>0.24</td>
</tr>
<tr>
<td>$MCP^R(OVR)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^G(CCI)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^G(OVR)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^B(OVR)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^B(TH2)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^R(TH3)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^R(OC1)$</td>
<td>9.17%</td>
<td>88.37%</td>
<td>2.46%</td>
<td>0.42</td>
</tr>
<tr>
<td>$MCP^B(OC1)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^B(P1)$</td>
<td>14.58%</td>
<td>85.02%</td>
<td>0.40%</td>
<td>0.44</td>
</tr>
<tr>
<td>$MCP^B(P3)$</td>
<td>45.20%</td>
<td>49.69%</td>
<td>5.11%</td>
<td>0.86</td>
</tr>
<tr>
<td>$MCP^B(OP2)$</td>
<td>7.35%</td>
<td>91.91%</td>
<td>0.74%</td>
<td>0.31</td>
</tr>
<tr>
<td>$MCP^B(OP3)$</td>
<td>40.77%</td>
<td>51.39%</td>
<td>7.84%</td>
<td>0.91</td>
</tr>
<tr>
<td>$MCP^B(OP4)$</td>
<td>41.39%</td>
<td>54.63%</td>
<td>3.98%</td>
<td>0.82</td>
</tr>
<tr>
<td>$MCP^B(OVR)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^B(TH2)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^B(TH3)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^R(OC1)$</td>
<td>12.44%</td>
<td>82.95%</td>
<td>4.61%</td>
<td>0.56</td>
</tr>
<tr>
<td>$MCP^R(OC2)$</td>
<td>0.53%</td>
<td>99.45%</td>
<td>0.03%</td>
<td>0.04</td>
</tr>
<tr>
<td>$MCP^R(CH2)$</td>
<td>17.89%</td>
<td>79.49%</td>
<td>2.62%</td>
<td>0.59</td>
</tr>
<tr>
<td>$MCP^R(CH3)$</td>
<td>20.43%</td>
<td>76.80%</td>
<td>2.76%</td>
<td>0.63</td>
</tr>
<tr>
<td>$MDP^R(OC1)$</td>
<td>35.01%</td>
<td>58.67%</td>
<td>6.33%</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Indirect cumulative entropy: 5.99
Appendix B

**EXTENDED EVALUATION RESULTS:**

**EXPERIMENT III**

This appendix contains extended results of experiment III described in Section 7.3.2.

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>(MCP^R(CCI))</td>
<td>3.95%</td>
<td>1.87%</td>
<td>94.18%</td>
<td>0.26</td>
</tr>
<tr>
<td>(MCP^G(CCI))</td>
<td>10.84%</td>
<td>3.91%</td>
<td>85.25%</td>
<td>0.50</td>
</tr>
<tr>
<td>(MCP^B(CCI))</td>
<td>6.90%</td>
<td>2.92%</td>
<td>90.17%</td>
<td>0.38</td>
</tr>
<tr>
<td>(MCP^R(P1))</td>
<td>19.73%</td>
<td>6.33%</td>
<td>73.93%</td>
<td>0.72</td>
</tr>
<tr>
<td>(MCP^B(P1))</td>
<td>27.36%</td>
<td>9.50%</td>
<td>63.15%</td>
<td>0.87</td>
</tr>
<tr>
<td>(MCP^G(P1))</td>
<td>26.77%</td>
<td>9.67%</td>
<td>63.56%</td>
<td>0.87</td>
</tr>
<tr>
<td>(MCP^B(P2))</td>
<td>27.05%</td>
<td>9.55%</td>
<td>63.40%</td>
<td>0.87</td>
</tr>
<tr>
<td>(MCP^G(P2))</td>
<td>24.66%</td>
<td>8.73%</td>
<td>66.61%</td>
<td>0.83</td>
</tr>
<tr>
<td>(MCP^R(OP2))</td>
<td>9.99%</td>
<td>6.45%</td>
<td>83.57%</td>
<td>0.56</td>
</tr>
<tr>
<td>(MCP^B(OP2))</td>
<td>19.74%</td>
<td>5.87%</td>
<td>74.40%</td>
<td>0.71</td>
</tr>
<tr>
<td>(MCP^R(OP3))</td>
<td>24.57%</td>
<td>9.50%</td>
<td>65.92%</td>
<td>0.84</td>
</tr>
<tr>
<td>(MCP^G(OP3))</td>
<td>31.97%</td>
<td>15.59%</td>
<td>52.44%</td>
<td>0.99</td>
</tr>
<tr>
<td>(MCP^B(OP3))</td>
<td>32.23%</td>
<td>15.93%</td>
<td>51.83%</td>
<td>1.00</td>
</tr>
<tr>
<td>(MCP^R(TH1))</td>
<td>31.22%</td>
<td>16.46%</td>
<td>52.32%</td>
<td>1.00</td>
</tr>
<tr>
<td>(MCP^G(TH1))</td>
<td>31.72%</td>
<td>15.03%</td>
<td>53.24%</td>
<td>0.98</td>
</tr>
<tr>
<td>(MCP^R(TH2))</td>
<td>31.12%</td>
<td>16.69%</td>
<td>52.19%</td>
<td>1.00</td>
</tr>
<tr>
<td>(MCP^B(TH2))</td>
<td>31.76%</td>
<td>16.69%</td>
<td>51.56%</td>
<td>1.00</td>
</tr>
<tr>
<td>(MCP^B(TH3))</td>
<td>31.53%</td>
<td>15.40%</td>
<td>53.07%</td>
<td>0.99</td>
</tr>
<tr>
<td>(MCP^B(TH4))</td>
<td>31.80%</td>
<td>14.63%</td>
<td>53.57%</td>
<td>0.98</td>
</tr>
<tr>
<td>(MCP^R(OC1))</td>
<td>28.80%</td>
<td>18.47%</td>
<td>52.73%</td>
<td>1.01</td>
</tr>
<tr>
<td>(MDP(*))</td>
<td>33.33%</td>
<td>33.33%</td>
<td>33.33%</td>
<td>1.10</td>
</tr>
</tbody>
</table>

**mdirect cumulative entropy:** 19.78

Table B.1: Initial state of BN in experiment III. Range probabilities in all mdprobs are uniformly distributed, what is represented by single row containing MDP with asterisk.
### Appendix B. Extended Evaluation Results: Experiment III

<table>
<thead>
<tr>
<th>Measurement</th>
<th>LLB</th>
<th>LB</th>
<th>UB</th>
<th>UUB</th>
<th>MCuM formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MC^R(CC1)$</td>
<td>970</td>
<td>1940</td>
<td>2197</td>
<td>10197</td>
<td>$MC^R(CC1) =$</td>
</tr>
<tr>
<td>$MC^G(CC1)$</td>
<td>975</td>
<td>1949</td>
<td>2207</td>
<td>9207</td>
<td>$MC^G(CC1) =$</td>
</tr>
<tr>
<td>$MC^B(CC1)$</td>
<td>1189</td>
<td>2378</td>
<td>2693</td>
<td>11693</td>
<td>$MC^B(CC1) =$</td>
</tr>
<tr>
<td>$MC^R(P1)$</td>
<td>611</td>
<td>1222</td>
<td>1383</td>
<td>5383</td>
<td>$MC^R(P1) =$</td>
</tr>
<tr>
<td>$MC^B(P1)$</td>
<td>602</td>
<td>1203</td>
<td>1362</td>
<td>4362</td>
<td>$MC^B(P1) =$</td>
</tr>
<tr>
<td>$MC^G(P2)$</td>
<td>616</td>
<td>1231</td>
<td>1394</td>
<td>4394</td>
<td>$MC^G(P2) =$</td>
</tr>
<tr>
<td>$MC^B(P3)$</td>
<td>583</td>
<td>1166</td>
<td>1320</td>
<td>4320</td>
<td>$MC^B(P3) =$</td>
</tr>
<tr>
<td>$MC^G(OP2)$</td>
<td>312</td>
<td>625</td>
<td>708</td>
<td>3708</td>
<td>$MC^G(OP2) =$</td>
</tr>
<tr>
<td>$MC^R(OP3)$</td>
<td>312</td>
<td>625</td>
<td>708</td>
<td>3708</td>
<td>$MC^R(OP3) =$</td>
</tr>
<tr>
<td>$MC^B(OP4)$</td>
<td>541</td>
<td>1082</td>
<td>1225</td>
<td>6225</td>
<td>$MC^B(OP4) =$</td>
</tr>
<tr>
<td>$MC^R(OVR)$</td>
<td>373</td>
<td>746</td>
<td>845</td>
<td>3845</td>
<td>$MC^R(OVR) =$</td>
</tr>
<tr>
<td>$MC^G(OVR)$</td>
<td>364</td>
<td>727</td>
<td>824</td>
<td>2824</td>
<td>$MC^G(OVR) =$</td>
</tr>
<tr>
<td>$MC^B(OVR)$</td>
<td>364</td>
<td>727</td>
<td>824</td>
<td>2824</td>
<td>$MC^B(OVR) =$</td>
</tr>
<tr>
<td>$MC^R(TH1)$</td>
<td>242</td>
<td>485</td>
<td>549</td>
<td>2549</td>
<td>$MC^R(TH1) =$</td>
</tr>
<tr>
<td>$MC^G(TH1)$</td>
<td>256</td>
<td>513</td>
<td>581</td>
<td>2581</td>
<td>$MC^G(TH1) =$</td>
</tr>
<tr>
<td>$MC^R(TH2)$</td>
<td>252</td>
<td>504</td>
<td>570</td>
<td>2570</td>
<td>$MC^R(TH2) =$</td>
</tr>
<tr>
<td>$MC^B(TH2)$</td>
<td>270</td>
<td>541</td>
<td>613</td>
<td>2613</td>
<td>$MC^B(TH2) =$</td>
</tr>
<tr>
<td>$MC^R(TH3)$</td>
<td>256</td>
<td>513</td>
<td>581</td>
<td>2581</td>
<td>$MC^R(TH3) =$</td>
</tr>
<tr>
<td>$MC^B(TH4)$</td>
<td>215</td>
<td>429</td>
<td>486</td>
<td>2486</td>
<td>$MC^B(TH4) =$</td>
</tr>
<tr>
<td>$MC^R(OC1)$</td>
<td>224</td>
<td>448</td>
<td>507</td>
<td>2507</td>
<td>$MC^R(OC1) =$</td>
</tr>
<tr>
<td>$MD(CC1)$</td>
<td>47</td>
<td>93</td>
<td>106</td>
<td>1106</td>
<td>$MD(CC1) =$</td>
</tr>
<tr>
<td>$MD(P1)$</td>
<td>238</td>
<td>476</td>
<td>539</td>
<td>1539</td>
<td>$MD(P1) =$</td>
</tr>
<tr>
<td>$MD(P2)$</td>
<td>252</td>
<td>504</td>
<td>570</td>
<td>1570</td>
<td>$MD(P2) =$</td>
</tr>
<tr>
<td>$MD(P3)$</td>
<td>219</td>
<td>438</td>
<td>496</td>
<td>1496</td>
<td>$MD(P3) =$</td>
</tr>
<tr>
<td>$MD(OP2)$</td>
<td>56</td>
<td>112</td>
<td>127</td>
<td>1127</td>
<td>$MD(OP2) =$</td>
</tr>
<tr>
<td>$MD(OP3)$</td>
<td>61</td>
<td>121</td>
<td>137</td>
<td>1137</td>
<td>$MD(OP3) =$</td>
</tr>
<tr>
<td>$MD(OP4)$</td>
<td>70</td>
<td>140</td>
<td>158</td>
<td>1158</td>
<td>$MD(OP4) =$</td>
</tr>
<tr>
<td>$MD(OVR)$</td>
<td>149</td>
<td>298</td>
<td>338</td>
<td>1338</td>
<td>$MD(OVR) =$</td>
</tr>
<tr>
<td>$MD(TH1)$</td>
<td>140</td>
<td>280</td>
<td>317</td>
<td>1317</td>
<td>$MD(TH1) =$</td>
</tr>
<tr>
<td>$MD(TH2)$</td>
<td>154</td>
<td>308</td>
<td>349</td>
<td>1349</td>
<td>$MD(TH2) =$</td>
</tr>
<tr>
<td>$MD(TH3)$</td>
<td>159</td>
<td>317</td>
<td>359</td>
<td>1359</td>
<td>$MD(TH3) =$</td>
</tr>
<tr>
<td>$MD(TH4)$</td>
<td>117</td>
<td>233</td>
<td>264</td>
<td>1264</td>
<td>$MD(TH4) =$</td>
</tr>
<tr>
<td>$MD(OC1)$</td>
<td>187</td>
<td>373</td>
<td>422</td>
<td>1422</td>
<td>$MD(OC1) =$</td>
</tr>
<tr>
<td>$MD(OC2)$</td>
<td>215</td>
<td>429</td>
<td>486</td>
<td>1486</td>
<td>$MD(OC2) =$</td>
</tr>
<tr>
<td>$MD(CH1)$</td>
<td>103</td>
<td>205</td>
<td>232</td>
<td>1232</td>
<td>$MD(CH1) =$</td>
</tr>
<tr>
<td>$MD(CH2)$</td>
<td>117</td>
<td>233</td>
<td>264</td>
<td>1264</td>
<td>$MD(CH2) =$</td>
</tr>
<tr>
<td>$MD(CH3)$</td>
<td>98</td>
<td>196</td>
<td>222</td>
<td>1222</td>
<td>$MD(CH3) =$</td>
</tr>
<tr>
<td>$MD(OP1)$</td>
<td>37</td>
<td>75</td>
<td>84</td>
<td>1084</td>
<td>$MD(OP1) =$</td>
</tr>
</tbody>
</table>

Table B.2: Nominal ranges established in experiment III after Phase I of the control loop
## Appendix B. Extended Evaluation Results: Experiment III

### Table B.3

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP^R(CCI)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^G(CCI)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^R(CCI)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^R(P1)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^B(P1)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^G(P2)$</td>
<td>7.24%</td>
<td>91.51%</td>
<td>1.25%</td>
<td>0.33</td>
</tr>
<tr>
<td>$MCP^B(P3)$</td>
<td>39.98%</td>
<td>54.80%</td>
<td>5.22%</td>
<td>0.85</td>
</tr>
<tr>
<td>$MCP^G(OP2)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^R(OP3)$</td>
<td>6.89%</td>
<td>90.62%</td>
<td>2.50%</td>
<td>0.37</td>
</tr>
<tr>
<td>$MCP^R(OP4)$</td>
<td>17.65%</td>
<td>78.63%</td>
<td>3.72%</td>
<td>0.62</td>
</tr>
<tr>
<td>$MCP^G(OPR)$</td>
<td>9.99%</td>
<td>85.89%</td>
<td>4.12%</td>
<td>0.49</td>
</tr>
<tr>
<td>$MCP^R(OPR)$</td>
<td>6.13%</td>
<td>93.42%</td>
<td>0.45%</td>
<td>0.26</td>
</tr>
<tr>
<td>$MCP^R(OPR)$</td>
<td>3.87%</td>
<td>96.08%</td>
<td>0.04%</td>
<td>0.17</td>
</tr>
<tr>
<td>$MCP^R(TH1)$</td>
<td>42.20%</td>
<td>55.13%</td>
<td>2.67%</td>
<td>0.79</td>
</tr>
<tr>
<td>$MCP^G(TH1)$</td>
<td>4.13%</td>
<td>95.76%</td>
<td>0.10%</td>
<td>0.18</td>
</tr>
<tr>
<td>$MCP^R(TH2)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^B(TH2)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MCP^B(TH3)$</td>
<td>31.36%</td>
<td>63.20%</td>
<td>5.44%</td>
<td>0.81</td>
</tr>
<tr>
<td>$MCP^R(OC1)$</td>
<td>19.11%</td>
<td>72.48%</td>
<td>8.41%</td>
<td>0.76</td>
</tr>
<tr>
<td>$MDP(CCI)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MDP(P1)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MDP(P2)$</td>
<td>17.96%</td>
<td>75.78%</td>
<td>6.26%</td>
<td>0.69</td>
</tr>
<tr>
<td>$MDP(P3)$</td>
<td>46.51%</td>
<td>45.83%</td>
<td>7.66%</td>
<td>0.91</td>
</tr>
<tr>
<td>$MDP(OP2)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MDP(OP3)$</td>
<td>36.06%</td>
<td>60.74%</td>
<td>3.19%</td>
<td>0.78</td>
</tr>
<tr>
<td>$MDP(OP4)$</td>
<td>39.55%</td>
<td>51.30%</td>
<td>9.15%</td>
<td>0.93</td>
</tr>
<tr>
<td>$MDP(OPR)$</td>
<td>4.17%</td>
<td>95.60%</td>
<td>0.24%</td>
<td>0.19</td>
</tr>
<tr>
<td>$MDP(TH1)$</td>
<td>11.70%</td>
<td>88.00%</td>
<td>0.30%</td>
<td>0.38</td>
</tr>
<tr>
<td>$MDP(TH2)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00</td>
</tr>
<tr>
<td>$MDP(TH3)$</td>
<td>31.95%</td>
<td>61.62%</td>
<td>6.43%</td>
<td>0.84</td>
</tr>
<tr>
<td>$MDP(OC1)$</td>
<td>20.42%</td>
<td>73.08%</td>
<td>6.50%</td>
<td>0.73</td>
</tr>
<tr>
<td>$MDP(OC2)$</td>
<td>5.50%</td>
<td>93.94%</td>
<td>0.56%</td>
<td>0.25</td>
</tr>
<tr>
<td>$MDP(CH1)$</td>
<td>47.52%</td>
<td>48.07%</td>
<td>4.41%</td>
<td>0.84</td>
</tr>
<tr>
<td>$MDP(CH2)$</td>
<td>3.79%</td>
<td>95.96%</td>
<td>0.25%</td>
<td>0.18</td>
</tr>
<tr>
<td>$MDP(CH3)$</td>
<td>11.11%</td>
<td>88.53%</td>
<td>0.36%</td>
<td>0.37</td>
</tr>
<tr>
<td>$MDP(OP1)$</td>
<td>37.02%</td>
<td>54.13%</td>
<td>8.85%</td>
<td>0.91</td>
</tr>
</tbody>
</table>

*mdirect cumulative entropy: 8.01*

Table B.3: State of BN in experiment III after selection of sentinels. Rows containing mprobs of sentinel services are highlighted.
Appendix B. Extended Evaluation Results: Experiment III

<table>
<thead>
<tr>
<th>iteration 1</th>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP^{OP}(P2)$</td>
<td>10.02%</td>
<td>84.99%</td>
<td>4.99%</td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(P3)$</td>
<td>26.14%</td>
<td>42.79%</td>
<td>29.08%</td>
<td>1.08</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OP3)$</td>
<td>5.16%</td>
<td>91.85%</td>
<td>2.99%</td>
<td>0.34</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OP4)$</td>
<td>0.48%</td>
<td>17.51%</td>
<td>82.01%</td>
<td>0.49</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OV$R)</td>
<td>11.28%</td>
<td>86.07%</td>
<td>2.66%</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OV$R)</td>
<td>8.43%</td>
<td>87.09%</td>
<td>4.48%</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(T$H1)</td>
<td>4.11%</td>
<td>91.15%</td>
<td>4.75%</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(T$H1)</td>
<td>40.67%</td>
<td>51.34%</td>
<td>7.99%</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(T$H2)</td>
<td>2.14%</td>
<td>97.80%</td>
<td>0.07%</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(T$H3)</td>
<td>2.43%</td>
<td>8.83%</td>
<td>88.71%</td>
<td>0.41</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OC$1)</td>
<td>25.78%</td>
<td>70.38%</td>
<td>5.84%</td>
<td>0.75</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iteration 2</th>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP^{OP}(P2)$</td>
<td>9.62%</td>
<td>88.98%</td>
<td>1.41%</td>
<td>0.39</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(P3)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OP3)$</td>
<td>4.53%</td>
<td>91.06%</td>
<td>4.41%</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OP4)$</td>
<td>0.00%</td>
<td>11.81%</td>
<td>88.19%</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OV$R)</td>
<td>8.44%</td>
<td>89.10%</td>
<td>2.46%</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OV$R)</td>
<td>4.71%</td>
<td>94.83%</td>
<td>0.46%</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(T$H1)</td>
<td>0.24%</td>
<td>99.75%</td>
<td>0.01%</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(T$H1)</td>
<td>40.18%</td>
<td>51.04%</td>
<td>8.78%</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(T$H2)</td>
<td>1.81%</td>
<td>98.19%</td>
<td>0.17%</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(T$H3)</td>
<td>1.75%</td>
<td>7.28%</td>
<td>90.94%</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OC$1)</td>
<td>15.86%</td>
<td>79.46%</td>
<td>4.66%</td>
<td>0.62</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iteration 3</th>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP^{OP}(P2)$</td>
<td>20.43%</td>
<td>79.84%</td>
<td>8.74%</td>
<td>0.78</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(P3)$</td>
<td>32.59%</td>
<td>35.14%</td>
<td>34.27%</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OP3)$</td>
<td>33.27%</td>
<td>62.94%</td>
<td>3.79%</td>
<td>0.78</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OP4)$</td>
<td>28.18%</td>
<td>32.24%</td>
<td>39.58%</td>
<td>1.09</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OV$R)</td>
<td>8.91%</td>
<td>80.31%</td>
<td>10.78%</td>
<td>0.63</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(TH1)$</td>
<td>10.43%</td>
<td>89.13%</td>
<td>0.44%</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(TH2)$</td>
<td>25.78%</td>
<td>70.38%</td>
<td>3.82%</td>
<td>1.68</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(TH3)$</td>
<td>23.56%</td>
<td>71.91%</td>
<td>5.25%</td>
<td>0.74</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OC$2)</td>
<td>12.47%</td>
<td>78.25%</td>
<td>9.28%</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(CH1)$</td>
<td>45.63%</td>
<td>44.83%</td>
<td>9.53%</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(CH2)$</td>
<td>2.22%</td>
<td>97.64%</td>
<td>0.14%</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(CH3)$</td>
<td>0.00%</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OC$1)</td>
<td>37.74%</td>
<td>54.60%</td>
<td>7.65%</td>
<td>0.89</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iteration 4</th>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MCP^{OP}(P2)$</td>
<td>20.06%</td>
<td>74.49%</td>
<td>5.45%</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(P3)$</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OP3)$</td>
<td>32.97%</td>
<td>62.58%</td>
<td>4.45%</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OP4)$</td>
<td>26.77%</td>
<td>31.60%</td>
<td>41.63%</td>
<td>1.08</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OV$R)</td>
<td>2.96%</td>
<td>96.74%</td>
<td>0.27%</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(TH1)$</td>
<td>10.55%</td>
<td>88.59%</td>
<td>0.85%</td>
<td>0.39</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(TH2)$</td>
<td>23.56%</td>
<td>71.91%</td>
<td>5.25%</td>
<td>0.74</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(TH3)$</td>
<td>16.78%</td>
<td>77.78%</td>
<td>5.44%</td>
<td>0.65</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OC$2)</td>
<td>3.90%</td>
<td>95.47%</td>
<td>0.54%</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(CH1)$</td>
<td>44.97%</td>
<td>44.96%</td>
<td>10.07%</td>
<td>0.95</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(CH2)$</td>
<td>2.33%</td>
<td>97.56%</td>
<td>0.11%</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(CH3)$</td>
<td>0.00%</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$MCP^{OP}(OC$1)</td>
<td>36.31%</td>
<td>56.92%</td>
<td>6.76%</td>
<td>0.87</td>
<td></td>
</tr>
</tbody>
</table>

Fig. B.1: States of monitored services in each iteration of adaptive drill-down execution triggered by single anomalies in service CH3 of CCP application. Sentinel services are excluded from the tables to preserve clarity. Suspicious services are highlighted in orange, while monitored services are highlighted in grey.

midirect cumulative entropy: 9.19
midirect cumulative entropy: 6.99
midirect cumulative entropy: 5.60
midirect cumulative entropy: 5.57
### Appendix B. Extended Evaluation Results: Experiment III

<table>
<thead>
<tr>
<th>mprob</th>
<th>under</th>
<th>exact</th>
<th>over</th>
<th>entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCP(^R)(CCI)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MCP(^G)(CCI)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MCP(^B)(CCI)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MCP(^R)(P1)</td>
<td>6,37%</td>
<td>92,91%</td>
<td>0,73%</td>
<td>0,28</td>
</tr>
<tr>
<td>MCP(^B)(P1)</td>
<td>11,59%</td>
<td>87,46%</td>
<td>0,95%</td>
<td>0,41</td>
</tr>
<tr>
<td>MCP(^G)(P2)</td>
<td>6,73%</td>
<td>90,71%</td>
<td>2,56%</td>
<td>0,36</td>
</tr>
<tr>
<td>MCP(^B)(P3)</td>
<td>36,38%</td>
<td>59,05%</td>
<td>4,56%</td>
<td>0,82</td>
</tr>
<tr>
<td>MCP(^G)(OP2)</td>
<td>10,51%</td>
<td>85,54%</td>
<td>3,96%</td>
<td>0,50</td>
</tr>
<tr>
<td>MCP(^R)(OP3)</td>
<td>12,88%</td>
<td>81,60%</td>
<td>5,52%</td>
<td>0,59</td>
</tr>
<tr>
<td>MCP(^B)(OP4)</td>
<td>10,05%</td>
<td>87,37%</td>
<td>2,57%</td>
<td>0,44</td>
</tr>
<tr>
<td>MCP(^R)(OVR)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MCP(^G)(OVR)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MCP(^B)(OVR)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MCP(^R)(TH1)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MCP(^G)(TH1)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MCP(^R)(TH2)</td>
<td>13,15%</td>
<td>85,34%</td>
<td>1,51%</td>
<td>0,47</td>
</tr>
<tr>
<td>MCP(^B)(TH2)</td>
<td>18,23%</td>
<td>80,16%</td>
<td>1,61%</td>
<td>0,55</td>
</tr>
<tr>
<td>MCP(^B)(TH3)</td>
<td>10,05%</td>
<td>87,37%</td>
<td>2,57%</td>
<td>0,44</td>
</tr>
<tr>
<td>MCP(^R)(OC1)</td>
<td>9,54%</td>
<td>88,43%</td>
<td>2,03%</td>
<td>0,41</td>
</tr>
<tr>
<td>MDP((CCI)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MDP((P1)</td>
<td>15,22%</td>
<td>83,56%</td>
<td>1,22%</td>
<td>0,49</td>
</tr>
<tr>
<td>MDP((P2)</td>
<td>17,71%</td>
<td>76,81%</td>
<td>5,48%</td>
<td>0,67</td>
</tr>
<tr>
<td>MDP((P3)</td>
<td>44,39%</td>
<td>49,09%</td>
<td>6,53%</td>
<td>0,89</td>
</tr>
<tr>
<td>MDP((OP2)</td>
<td>37,40%</td>
<td>57,10%</td>
<td>5,50%</td>
<td>0,85</td>
</tr>
<tr>
<td>MDP((OP3)</td>
<td>36,87%</td>
<td>53,91%</td>
<td>9,23%</td>
<td>0,92</td>
</tr>
<tr>
<td>MDP((OP4)</td>
<td>40,46%</td>
<td>51,78%</td>
<td>7,76%</td>
<td>0,91</td>
</tr>
<tr>
<td>MDP((OVR)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MDP((TH1)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MDP((TH2)</td>
<td>19,69%</td>
<td>78,49%</td>
<td>1,82%</td>
<td>0,58</td>
</tr>
<tr>
<td>MDP((TH3)</td>
<td>0,00%</td>
<td>100,00%</td>
<td>0,00%</td>
<td>0,00</td>
</tr>
<tr>
<td>MDP((OC1)</td>
<td>11,27%</td>
<td>84,87%</td>
<td>3,85%</td>
<td>0,51</td>
</tr>
<tr>
<td>MDP((OC2)</td>
<td>0,45%</td>
<td>99,52%</td>
<td>0,03%</td>
<td>0,03</td>
</tr>
<tr>
<td>MDP((CH1)</td>
<td>17,01%</td>
<td>80,26%</td>
<td>2,73%</td>
<td>0,58</td>
</tr>
<tr>
<td>MDP((CH2)</td>
<td>7,03%</td>
<td>92,70%</td>
<td>0,28%</td>
<td>0,27</td>
</tr>
<tr>
<td>MDP((CH3)</td>
<td>4,00%</td>
<td>95,70%</td>
<td>0,31%</td>
<td>0,19</td>
</tr>
<tr>
<td>MDP((OP1)</td>
<td>34,18%</td>
<td>60,05%</td>
<td>5,77%</td>
<td>0,84</td>
</tr>
</tbody>
</table>

**mdirect cumulative entropy:** 7,72

Table B.4: State of BN in experiment III after selection of sentinels performed upon finishing drill-down triggered by anomaly in service CH3 – Sentinels (highlighted in grey) have changed from CCI, OP2, TH2, P1 to CCI, TH1, TH3, OVR.
### Fig. B.2: States of monitored services in each iteration of adaptive drill-down execution triggered by anomalies in services OP1, CH1 and TH3 of CCP application. Sentinel services are excluded from the tables to preserve clarity. Suspicious services are highlighted in orange, while monitored services are highlighted in grey.

<table>
<thead>
<tr>
<th></th>
<th>iteration 1</th>
<th>iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mprob</td>
<td>under</td>
</tr>
<tr>
<td>$MCP^G(P2)$</td>
<td>11.91%</td>
<td>82.74%</td>
</tr>
<tr>
<td>$MCP^B(P3)$</td>
<td>25.91%</td>
<td>38.53%</td>
</tr>
<tr>
<td>$MCP^B(OP3)$</td>
<td>0.00%</td>
<td>7.08%</td>
</tr>
<tr>
<td>$MCP^B(OP4)$</td>
<td>0.79%</td>
<td>36.84%</td>
</tr>
<tr>
<td>$MCP^B(OVR)$</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>$MCP^G(OVR)$</td>
<td>11.32%</td>
<td>80.71%</td>
</tr>
<tr>
<td>$MCP^B(TH1)$</td>
<td>4.25%</td>
<td>84.74%</td>
</tr>
<tr>
<td>$MCP^B(TH3)$</td>
<td>18.87%</td>
<td>24.06%</td>
</tr>
<tr>
<td>$MCP^G(TH1)$</td>
<td>1.65%</td>
<td>98.19%</td>
</tr>
<tr>
<td>$MCP^G(TH3)$</td>
<td>10.36%</td>
<td>35.42%</td>
</tr>
<tr>
<td>$MCP^B(OC1)$</td>
<td>3.50%</td>
<td>6.53%</td>
</tr>
<tr>
<td>$MCP^B(OC2)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$MCP^B(OC3)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$MCP^B(OC4)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>iteration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mprob</td>
</tr>
<tr>
<td>$MCP^G(P2)$</td>
<td>3.54%</td>
</tr>
<tr>
<td>$MCP^B(P3)$</td>
<td>38.74%</td>
</tr>
<tr>
<td>$MCP^B(OP3)$</td>
<td>7.00%</td>
</tr>
<tr>
<td>$MCP^B(OP4)$</td>
<td>3.75%</td>
</tr>
<tr>
<td>$MCP^B(OVR)$</td>
<td>0.00%</td>
</tr>
<tr>
<td>$MCP^B(TH1)$</td>
<td>0.00%</td>
</tr>
<tr>
<td>$MCP^B(TH3)$</td>
<td>0.00%</td>
</tr>
<tr>
<td>$MCP^G(TH1)$</td>
<td>0.00%</td>
</tr>
<tr>
<td>$MCP^G(TH3)$</td>
<td>0.00%</td>
</tr>
<tr>
<td>$MCP^B(OC1)$</td>
<td>0.00%</td>
</tr>
<tr>
<td>$MCP^B(OC2)$</td>
<td></td>
</tr>
<tr>
<td>$MCP^B(OC3)$</td>
<td></td>
</tr>
<tr>
<td>$MCP^B(OC4)$</td>
<td></td>
</tr>
</tbody>
</table>

**Cumulative Entropy:**
- iteration 1: 11.45
- iteration 2: 5.28
- iteration 3: 3.41
ACRONYMS

CCP  Credit Card Process
CCI  Credit Card Issuing
OP   Offer Preparation
TH   Tax History Retrieval
CH   Credit History Retrieval
OC   Offer Calculation
OVR  Offer Verification
P    Persistence
AS3  Adaptive SOA Solution Stack
API  Application Programming Interface
AOP  Aspect Oriented Programming
BC   Binding Component
BPEL Business Process Execution Language
BN   Bayesian network
CBE  Common Base Event
CEP  Complex Event Processing
CPT  Conditional Probability Table
CPU  Central Processing Unit
DAG  Directed acyclic graph
DAMON Dynamic Adaptive Monitoring Framework
DAS  Dynamic Adaptive System
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTW</td>
<td>Dynamic-Time Warping</td>
</tr>
<tr>
<td>ECA</td>
<td>event-condition-action</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Beans</td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>EIS</td>
<td>Enterprise Information Systems</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>JAR</td>
<td>Java Archive</td>
</tr>
<tr>
<td>JBI</td>
<td>Java Business Integration</td>
</tr>
<tr>
<td>JCP</td>
<td>Java Community Process</td>
</tr>
<tr>
<td>JEE</td>
<td>Java Enterprise Edition</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management Extensions</td>
</tr>
<tr>
<td>JSR 208</td>
<td>Java Specification Request 208</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>LTW</td>
<td>Load-Time Weaving</td>
</tr>
<tr>
<td>NMR</td>
<td>Normalized Message Router</td>
</tr>
<tr>
<td>MAPE-K</td>
<td>Monitor Analyze Plan Execute - Knowledge</td>
</tr>
<tr>
<td>MOM</td>
<td>Message Oriented Middleware</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Interface Controller</td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraint Language</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OS</td>
<td>operating system</td>
</tr>
<tr>
<td>OSIMM</td>
<td>Open Group Service Integration Maturity Model</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>PIM</td>
<td>Platform-Independent Model</td>
</tr>
<tr>
<td>PSM</td>
<td>Platform-Specific Model</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RTBN</td>
<td>Response Time Bayesian network</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>SCA</td>
<td>Service Component Architecture</td>
</tr>
<tr>
<td>SE</td>
<td>Service Engine</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOA-RA</td>
<td>SOA Reference Architecture</td>
</tr>
<tr>
<td>SoaML</td>
<td>SOA Modeling Language</td>
</tr>
<tr>
<td>S3</td>
<td>Service Oriented Solution Stack</td>
</tr>
<tr>
<td>UCS</td>
<td>Cisco Unified Computing System</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>VLAN</td>
<td>Virtual LAN</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WS</td>
<td>Web Service</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RSA</td>
<td>Remote Service Admin</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Bibliography


[47] ——, “OSGi Core Release 5,” March 2012.


[125] “Java API of Delay Queue in JDK 1.5,” http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/DelayQueue.html, accessed on July 2013.


Bibliography


