PhD Dissertation

International Doctorate School in Information and Communication Technologies

DIT - University of Trento

CROSS-LAYER ADAPTATION OF SERVICE-BASED SYSTEMS

Asli Zengin

Advisor:

Prof. Marco Pistore

FBK-irst, Trento

November 2012

"Our greatest glory is not in never falling,
but in rising every time we fall."

Confucius

To my family.

Acknowledgements

This dissertation would not have been possible without the support of many people. First of all, I wish to express my gratitude to my advisor Marco Pistore who offered me his valuable support with a lot of patience. His guidance helped me understand how to conduct research and focus on getting solid results.

For having accepted to act as members of my doctoral examination committee, as well as for their constructive comments, I would like thank Maurizio Marchese, Luciano Baresi, Vittorio Cortellessa, Claudio Bartolini and Pierluigi Plebani.

I thank all my colleagues and collaborators from SOA team. In particular, I am grateful to Raman Kazhamiakin and Annapaola Marconi for their enthusiasm and availability to discuss my research problems and to assist me with fruitful ideas. Special thanks also go to my colleagues Heorhi Raik, Adina Sirbu and Piergiorgio Bertoli. In these years they have become my real friends beyond being colleagues and I have always felt their encouraging support. Not only them, I sincerely thank all my friends that I got to know throughout my PHD adventure in Trento and the older ones from Turkey.

My very speacial thanks go to Claudio, the beautiful surprise of my life. The last year of my PHD work passed with a lot of joy and hope, thanks to his love and care for me.

With my deepest gratitude I dedicate this work to my family: my beloved parents and my dear sisters who always believed in me and gave me their affectionate support and understanding.

Abstract

One of the key features of service-based systems (SBS) is the capability to adapt in order to react to various changes in the business requirements and the application context. Given the complex layered structure, and the heterogeneous and dynamic execution context of such systems, adaptation is not at all a trivial task.

The importance of the problem of adaptation has been widely recognized in the community of software services and systems. There exist several adaptation approaches which aim at identifying and solving problems that occur in one of the SBS layers. A fundamental problem with most of these works is their fragmentation and isolation. While these solutions are quite effective when the specific problem they try to solve is considered, they may be incompatible or even harmful when the whole system is taken into account. Enacting an adaptation in the system might result in triggering new problems.

When building adaptive SBSs precautions must be taken to consider the impacts of the adaptations on the entire system. This can be achieved by properly coordinating adaptation actions provided by the different analysis and decision mechanisms through holistic and multi-layer adaptation strategies. In this dissertation, we address this problem. We present a novel framework for Cross-layer Adaptation Management (CLAM) that enables a comprehensive impact analysis by coordinating the adaptation and analysis tools available in the SBS.

We define a new system modeling methodology for adaptation coordination. The SBS model and the accompanying adaptation model that we propose in *this thesis overcome the limitations of the existing cross-layer adaptation approaches: (i) genericness for accommodating diverse SBS domains with different system elements and layers (ii) flexibility for allowing new system artifacts and adaptation tools (iii) capability for dealing with the complexity of the SBS considering the possibility of a huge number of problems and adaptations that might take place in the system.*

Based on this model we present a tree-based coordination algorithm. On the one hand it exploits the local adaptation and analysis facilities provided by the system, and on the other hand it harmonizes the different layers and system elements by properly coordinating the local solutions. The outcome of the algorithm is a set of alternative cross-layer adaptation strategies which are consistent with the overall system.

Moreover, we propose novel selection criteria to rank the alternative strategies and select the best one. Differently from the traditional approaches, we consider as selection criteria not only the overall quality of the SBS, but also the total efforts required to enact an adaptation strategy. Based on these criteria we present two possible ranking methods, one relying on simple additive weighting - multiple criteria decision making, the other relying on fuzzy logic.

The framework is implemented and integrated in a toolkit that allows for constructing and selecting the cross-layer adaptation strategies, and is evaluated on a set of case studies.

Keywords

[Service-based Systems, Cross-layer Adaptation, System Modeling, Adaptation Impact]

Contents

1 Introduction 1

1.1 Contributions 6

1.1.1 Cross-layer Adaptation Approach 6

1.1.2 Addressing Specific Problems 9

1.1.3 Prototype Tool and Empirical Evaluation 13

1.2 Thesis Outline 15

2 State of the Art 17

2.1 Service-Based Systems 17

2.1.1 Business Process Management 18

2.1.2 Services and Service Compositions 19

2.1.3 Platforms and Infrastructures 20

2.2 Monitoring, Analysis and Adaptation of Service-Based Systems 21

2.2.1 Monitoring and Analysis at Business Processes Level . . 24

2.2.2 Monitoring and Analysis at Services and Service Compositions Level 25

2.2.3 Monitoring and Analysis at Platforms and Infrastructures Level 26

2.2.4 Adaptation at Business Processes Level 27

2.2.5 Adaptation at Services and Service Compositions Level 27

2.2.6 Adaptation at Platforms and Infrastructures Level 29

2.3 Modeling Service-Based Systems 30

2.3.1 Modeling for Application Deployment 31
2.3.2 Modeling for Monitoring and Analysis 32
2.3.3 Modeling for Adaptation 32
2.4 Cross-layer Approaches in Software Systems 33
2.5 Related Work on Cross-layer Adaptation 35
2.5.1 Cross-layer Adaptation Approaches in Service-Based Systems 36
2.5.2 Comparison of Cross-layer Adaptation Approaches . . . 38
2.6 Discussion 40

3 Cross-layer Adaptation Framework 43
3.1 Motivation 43
3.2 Reference Scenario 46
3.3 Formal Model of Service-Based System 48
3.3.1 System Model 48
3.3.2 System Configuration 50
3.3.3 System Layers 52
3.4 Formal Model of Adaptation 54
3.4.1 Extended System Configuration 56
3.4.2 Tools 58
3.5 The Framework 61
3.5.1 The initial trigger 61
3.5.2 Coordination and decision of the adaptation strategy . . 62
3.5.3 Deployment of the adaptation strategy 62
3.6 Discussion 63

4 Cross-layer Adaptation Manager 65
4.1 Problem Statement 65
4.1.1 Stable System Configuration 65
4.1.2 Cross-layer Adaptation Problem 66

4.2 Cross-layer Adaptation Approach 66
4.2.1 CLAM Algorithm 69
4.2.2 Correctness of the Approach 77
4.3 Discussion 87

5 Selection and Deployment of a Cross-layer Adaptation 89
5.1 Ranking and Selection 90
5.1.1 Selection Criteria 91
5.1.2 Multi Criteria Decision Analysis 95
5.1.3 An Alternative Fuzzy Logic-based Method 97
5.2 Adaptation Deployment 103
5.3 Discussion 104

6 Implementation and Design 107
6.1 CLAM Platform 107
6.2 Methodology for Cross-layer System Modeling 113
6.3 CLAM User Guide 115
6.3.1 Initial Setup of CLAM 115
6.3.2 Updating CLAM with the New Tools 120
6.3.3 Changing the System Model 121
6.4 Discussion 122

7 Evaluation 125
7.1 Experimental Set-up and a Sample Run 126
7.2 Heuristic Methods for Search Optimization and Termination . . 131
7.2.1 Optimization Heuristics 132
7.2.2 Termination Heuristics 136
7.3 Contribution of the Modeling Methodology 145
7.4 Discussion 151

8 Conclusions **155**

8.1 Achieved Results 155

8.2 Future Directions 158

Bibliography **161**

List of Tables

5.1 Mapping Changes in $\mathcal{SC}$ Nodes to Deployment Tasks 93
5.2 Required Efforts for Adaptation Deployment in terms of Time and Cost 93
5.3 Preferable Solution Layers for a Given Problem Layer 95

6.1 The Tools Available in “Call & Pay Taxi” SBS 117

7.1 Adaptation Spaces for the Solvers used in the “Call & Pay Taxi” Scenario 128
7.2 Cross-layer Adaptation Paths for a Sample Run 130
7.3 Qualitative Analysis of Effects of Solvers on the System 133
7.4 Calculation of Optimistic and Pessimistic Bounds of a CLAM Queue based on Solver Behaviors 133
7.5 Pruned solutions of Scenario 1 when $10\%|\mathcal{SC}_0|$ threshold is applied 142
7.6 Pruned solutions of Scenario 2 when $10\%|\mathcal{SC}_0|$ threshold is applied 143
7.7 Modeling Efforts: Scenario 1 - Case 0 149
7.8 Modeling Efforts: Scenario 1 - Case 1 149
7.9 Modeling Efforts: Scenario 2 - Case 0 150
7.10 Modeling Efforts: Scenario 2 - Case 1 150
7.11 Comparison of Analyzer Invocations at Different Modeling Abstractions 151

List of Figures

1.1 Prototype Cross-layer Adaptation Tool 13

2.1 Conceptual A&M framework 23

3.1 Feedback Control Systems 45
3.2 Adaptive Service-Based Systems - the Traditional Approach . . 45
3.3 Adaptive Service-Based Systems - the CLAM Approach 45
3.4 Adaptive Service-Based Systems - the CLAM Approach 47
3.5 Example of System Model 51
3.6 Example of System Configuration 51
3.7 Overall Approach to Modeling Adaptation 56
3.8 Conceptual Model of CLAM's Reasoner 58
3.9 Adaptation Tools of the Formal Framework 60
3.10 The Overall Framework 61

4.1 A Sample Solution Tree 69
4.2 CLAM algorithm - main functions. 70
4.3 CLAM algorithm - functions to create node and edge pairs. . . . 71
4.4 A Sample Cross-layer Adaptation Tree Produced by CLAM . . 73
4.5 Non Termination: Analysis of Recursive `expandTree` Function 85

5.1 The Overall Ranking Approach 97
5.2 Hierarchy of Fuzzy Inference Engines 99
a Process Execution Time 101

b Adaptation Deployment Time 101
c Adaptation Deployment Cost 101
5.3 Membership functions for quality and adaptation deployment parameters 101
5.4 Fuzzy rules to evaluate the aggregated QoS from quality criteria 102
5.5 Fuzzy rules to evaluate adaptation strategies in each path 102

6.1 The Functional Architecture of CLAM Platform 108

7.1 The System and Adaptation Models for "Call & Pay Taxi" Scenario 127
7.2 Greedy Search Algorithm. 134
a Scenario 1 137
b Scenario 2 137
7.3 Comparison of greedy search with BFS and DFS algorithms. . . 137
a Scenario 1 140
b Scenario 2 140
7.4 Effect of variation of $\Delta\mathcal{SC}$ weight on the number of solutions found at varying termination thresholds. 140
a Scenario 1 142
b Scenario 2 142
7.5 Observation of pruned tree nodes and solutions at varying termination thresholds when only $\Delta\mathcal{SC}$ weight is applied. 142
a Scenario 1 144
b Scenario 2 144
7.6 Variation of $\Delta\mathcal{SC}$ throughout the tree expansion when $\Delta\mathcal{SC}$ driven search heuristic is applied. 144
7.7 Two Different Levels of Abstraction for Scenario 1 and Scenario 2 147

Chapter 1

Introduction

The rapid evolution of software technology in the virtual world has brought paramount changes to every market sector and has created enormous opportunities for innovation. One such opportunity is constructing service-based applications (SBA) which allow for cross-application integration, interoperability, and the flexibility of implementation and reconfiguration when taking into account both the technological solution and the architectural style of the system. Those services of the SBA, collaborating together with an aim of achieving a set of business goals and providing certain business capabilities, have the power to provide utility to users in a much more dynamic and flexible way than the traditional software technology.

Every SBA has an overall execution context that is called a service-based system (SBS). In order to run the application and in the meanwhile to guarantee the conformance to the business metrics, SBSs comprise all the essential constituents such as functional and quality requirements of the application, the run-time application environment, underlying services, software and hardware platforms, and the corresponding support mechanisms to enable reconfigurations and modifications in case of changes in the application requirements as well as in the execution context.

SBSs have a complex structure given the horizontal components corresponding to the domain layers of the system and the vertical components, which correspond to the cross-cutting issues such as engineering and design, adaptation and monitoring, and quality assurance [102]. For what concerns domain layers, obviously, they vary based on the application of interest. For instance, for a typical SBS, one can think of the business process, service and infrastructure layers. Moreover, depending on the system capabilities, layers can be more fine-grained. E.g. instead of infrastructure layer, one can have the platform and resource layers, similarly, instead of business process layer, one can have the business model and service composition layers.

The complexity of service-based systems, the heterogeneity of the constituent layers together with an intrinsic distributed structure, and on top of all this, the dynamic run-time environment make the ability to adapt a key necessity that goes far beyond being a desired capability.

Existing works in SBS adaptation mostly solve the problem in a narrow scope, consider only a specific aspect of the system and do not take into account the impact of an adaptation on the overall SBS. Business process adaptation [77, 42], dynamic service binding [59, 29], self adaptation and self healing [70, 26], QoS-awareness [119, 12], mediator design for service interactions [67] and context-awareness [20] are research directions that have been and still remain in the central stage of SBS adaptation. These research directions delimit their specific areas of study; nevertheless, they are so interrelated that studying them separately is an endeavor that seems little promising.

Consequently, there should be holistic approaches that make explicit the knowledge of the horizontal layers that is relevant for the cross-cutting adaptation concern, and that currently is mostly hidden in languages, standards, mechanisms, and so on that are defined and investigated in isolation at the different layers. More precisely, the approach should be such that the domain layers offer the capabilities that are relevant for the adaptation, and a higher level cross-layer adaptation mechanism exploits these capabilities by properly coordinating the adaptation actions and eventually achieving the overall SBS consistency. In this way, one can ensure that the necessary precautions are taken to consider the interactions and possible conflicts between the different adaptation functionalities.

Few approaches consider cross-layer aspects of SBSs for the problem of adaptation. However, most of them oversimplify the important characteristics of the SBS that are fundamental to deal with such a non-trivial research problem. The research work presented in this dissertation was driven by the necessity to provide a cross-layer adaptation approach overcoming the limitations of the existing work. In particular, such approach should handle the following important features that are peculiar to service-based systems: (i) heterogeneity of system elements and a wide variety of domain layers depending on a specific SBS, (ii) dynamic changes in business goals, application constraints and the underlying execution environment, (iii) independent design and development of layer-specific monitoring and adaptation capabilities, (iv) possible contradictory adaptations that might be triggered by different adaptation mechanisms, and finally (v) the variety of adaptation actions one can take as solution to address a specific problem.

Heterogeneity of SBSs. Depending on the domain, there might be various types of SBS with different layers and building blocks. For instance, for one application it might be reasonable to consider solely the software platform as a layer, then for another application, resource consumption can be crucial, which in turn necessitates the consideration of a resource layer. Furthermore, within the same layer one can think of diverse elements. E.g. while for a security-critical application, quality attributes such as reliability and safety gain importance, for short-running processes response time can be much more of importance. Eventually, depending on the type of application, we might end up with different system aspects of concern. As a consequence, the adaptation solution cannot be grounded on a specific class of SBS which has a certain system representation with fixed system elements. In this regard, being generic enough to be able to accommodate various domains becomes an essential requirement for cross-layer adaptation.

Dynamicity of SBSs. There might be many and varied motives behind adaptation. The SBS owner might decide to update the business goals, for instance, relaxing some key performance indicators or introducing some new metrics. What's more, the application itself might change and a re-design of the overall process might be considered. On the other hand, various system layers that the application relies on might trigger a need for adaptation. To illustrate, we might have to cope with the unreliable network on which the application operate, and also deal with the changes in the execution environment, i.e., the underlying infrastructure, in the partner services with which the process interacts, and in the users preferences and context. This means that the system needs to be able to detect such potential problems and adapt its behavior respectively. On top of all this, such motives for adaptation might evolve by the time passes in the lifecycle of the SBS. Consequently, the cross-layer adaptation solution should be flexible enough to accommodate the introduction of new system elements and/or new capabilities for system maintenance.

Independence of SBS tools. Adaptation and monitoring tools are defined and developed independent from each-other and specialized only on a restricted part of the system being unaware of the rest of the SBS. Moreover, the complexity and the multi-layer nature of SBS make it impossible that there exists a centralized tool which knows all the cross-layer adaptation patterns, all the possible problems, solutions and their consequences. Thus, the layer-specific, fragmented knowledge of SBS should be exploited and properly coordinated through corresponding system capabilities but cannot be totally centralized such that there exists a single adaptation expert that takes control of all the possible cross-layer adaptations. This is because a cross-layer adaptation pattern, which can be represented as a sequence of adaptation actions, highly depends on the current state of the application as well as the current possible actions that a layer-specific adaptation mechanism can offer. With these regards, the solution approach should take into consideration this nature of SBS, which in turn implies that the cross-layer adaptation framework should utilize the existing adaptation and monitoring mechanisms as building blocks on the fly.

Contradictory adaptations. As a consequence of its layered structure, SBS has different constraints such as application-specific business goals and key performance indicators, functional and quality-related constraints on services, configuration and performance-related requirements for underlying platforms and infrastructures and so on. These constraints are monitored and analyzed continuously by system capabilities and in case of deviations, appropriate adaptation capabilities should be utilized to address the problems. However, the heterogeneity of such constraints and the reverse entailments that one expose to the other might lead to contradictory adaptations. Therefore, the cross-layer adaptation approach should ensure the analysis and identification of the impact of an adaptation on the entire SBS, and subsequently, in case of newly triggered problems, new actions should be taken to come up with a comprehensive solution where all the conflicts among all the constraints are resolved.

Variety of solutions. Given the complexity of SBSs and the diversity of adaptation capabilities available for such systems, there might exist several cross-layer adaptation solutions which address the same initial problem. Then each alternative solution should be assessed based on some evaluation metrics. These metrics, in other words, the selection criteria, should be carefully decided with respect to two main principles: the overall quality of the final system configuration proposed by the solution, and the efforts required to implement the necessary changes in the running system. Both of these aspects should be essentially taken into account to distinguish one solution from the other. Once some selection criteria are determined on the basis of these two principles, then the alternative cross-layer adaptation solutions can be scored against the criteria and in this way a final ranking can be produced, which in turn enables the selection of the best solution for deployment in the system.

1.1 Contributions

In this dissertation we develop a cross-layer adaptation approach for service-based systems. The aim of this approach is, overcoming the limitations of existing works, to propose a novel method that coordinates the layer-specific adaptation and analysis capabilities.

Our approach furnishes the SBS with the following capabilities: (i) It allows a novel way of system modeling that captures the cross-layer relations in the SBS. (ii) It defines a methodology to create this model for a given SBS and its adaptation capabilities. (iii) On top of the system model it proposes a holistic, iterative algorithm that enables the coordination of the adaptation capabilities. (iv) It gives the possibility to rank the cross-layer adaptation solutions and select the best one with respect to the diverse criteria.

1.1.1 Cross-layer Adaptation Approach

We propose a cross-layer adaptation framework that relies on a system model for the representation of service-based system, and an adaptation model for the representation of external analysis and adaptation tools. Eventually, the coordination algorithm, which exploits the system and adaptation models, realizes this framework and derives the cross-layer adaptation solutions.

By our cross-layer adaptation framework we target to help the owner of service-based application maintain the application such that its whole execution context, i.e., the service-based system, is taken into account. The proposed cross-layer adaptation is performed off-line in the sense that the adaptation does not modify an ongoing execution, instead it modifies the system model that will be applicable to the future executions. On the other hand, the approach is automated in the sense that once the necessary SBS and adaptation models are created by the system expert and fed to the algorithm, we identify the adaptations without requiring a human in the loop. Finally, the approach allows for derivation of cross-layer adaptations on the fly, which means that cross-layer adaptation patterns are not known a priori, but they are rather generated as a result of the coordination depending on the adaptation problem as well as the available adaptation solutions to coordinate at that moment.

Formal framework. We model the service-based system as a graph of nodes and edges. Nodes represent the different types of elements that are present in the system, and edges represent the relations among these elements. By identifying specific types of relations among elements such as “has”, “constrains” and “consumes”, we formally capture the layer concept and define layer as a set of nodes connected through “has” and “constrains” relations: these relations, indeed, serve the purpose to connect elements in the same layer, while “consumes” relations link elements belonging to different layers. The system model corresponds to a class of service-based systems, whereas a system configuration, corresponding to a specific service-based system, is an instantiation of the system model which is deployed and running or ready to be deployed. After having the definitions of system model and system configuration, we can extend them with the adaptation capabilities. For this extension, we take advantage of the formal model of adaptation where we introduce the concepts of adaptation need and adaptation action and subsequently related these concepts to the system capabilities that we call “tools”. We distinguish three different kinds of tools:

- *analyzer* is a system monitoring or analysis mechanism, which works on a subset of nodes in the system configuration to validate a system property and produces a set of alternative adaptation needs in case of a problem identification.

- *solver* is a system adaptation capability, which is specialized on a need and, similar to analyzers, works on a subset of nodes in the system configuration to produce a set of alternative adaptation actions to address the need.
- *enactor* is a tool type internal to our solution approach, which is responsible for applying in the system configuration the required changes imposed by an adaptation action.

Given the extended SBS representation which includes both system and adaptation models and the associated tools with these models, we define the cross-layer adaptation problem for a given initial system configuration as the problem of identifying a sequence of tools that is able to transform the initial system configuration into a stable system configuration where the stability implies that none of the analyzers of the system do not produce any adaptation needs.

CLAM algorithm. In order to solve the cross-layer adaptation problem, we propose an iterative coordination algorithm, namely the CLAM algorithm. It relies on the formal framework, and given the system tools, investigates the possible stable system configurations that can be reached to solve the adaptation problem of an initial system configuration.

The algorithm exploits a tree data structure such that at the tree nodes we keep a state of the system configuration and a queue of tools, and at the tree edges we keep the outputs of the tool invocations that we call "report". Here, the queue serves as a means for keeping and continuously updating an ordered set of analyzers and solvers that we identify to be invoked, and states of the system configuration correspond to the gradual transformation of the system configuration to reach a stable point. Indeed, the path of a tree leaf that has an empty queue implies a cross-layer adaptation solution.

CLAM algorithm performs three key steps in order to construct the tree, which in the end can disclose the cross-layer adaptation solutions:

- *Receiving the initial trigger.* The algorithm takes as input the running system configuration and the initial trigger, that is, a problem signaled by a monitor. Then, it creates a new queue instance, adds the analyzer relevant for the monitored data to the queue and instantiate a new tree with the root node.
- *Performing tool invocations.* Starting from the root node, each time a new node is created, we get the first tool of the queue in this node and invoke the tool.The output of a tool invocation, which we call a report, is of one of the following 4 types: (i) an analyzer validates a system configuration and does not produce any needs, (ii) an analyzer identifies a problem and produces a set of alternative needs, (iii) a solver produces a set of alternative actions for a given need, (iv) a solver cannot address an adaptation need.
- *Constructing the tree recursively.* Based on the output type of the report from a tool, we create new tree nodes and append them to the tree. We repeat iterations for the new nodes until all the nodes are visited and no new ones are created.

While the algorithm is proved to be correct and complete, and the proofs are presented in this dissertation; given the infinite input space of possible system configurations and adaptation actions, we remark that at theoretical level the algorithm does not terminate.

1.1.2 Addressing Specific Problems

Related to our cross-layer adaptation approach, we identified and addressed three specific problems: First one is the organization of the alternative cross-layer adaptation solutions produced by the CLAM algorithm so that the user can understand what a cross-layer adaptation solution implies and what are the consequences to deploy it in the SBS. Second one is the identification of the clear steps that one should follow to be able to use our approach. Since our approach works based on the SBS model proposed in this thesis, in particular, a modeling methodology to guide the users of our approach is essential. Third one is the identification of heuristic methods to optimize the tree construction, and to enforce the termination of the coordination algorithm so that in practical cases, we can overcome the non-termination issue of the algorithm and at the same time guarantee to find a reasonable set of cross-layer adaptation solutions.

Adaptation ranking and selection. Since cross-layer adaptation solutions are complex paths that involve diverse SBS elements, it is necessary to organize the produced results and present them based on some criteria so that the best one can be selected for deployment in the running system. While traditional selection approaches focus on only the quality dimension; conversely, understanding the consequences and convenience of an adaptation deployment is as crucial as the achieved SBS quality. In this regard, one can think of a short-running process where the time dedicated to deploy an adaptation must be small, or another situation can be where the SBS owner has to pay considerably big penalties to deploy the new, adapted system configuration. Thus, in order to address the selection problem in a holistic way, we propose novel criteria, which take into account not only the quality aspects but also the required efforts to deploy a cross-layer adaptation in terms of costs and time, and the adaptation locations in the SBS in terms of the domain layers involved in the solution.

For what concerns the ranking method, which should ground on the selection criteria, we propose two different approaches. The first one is the well known simple additive weighting – multi criteria decision making [55], which we we have implemented in the framework and present the achieved results in this dissertation. The second is a selection approach based on fuzzy logic where the selection is carried out by inferring the ranking criteria through a fuzzy inference system [130, 56]. While the first approach offers a mathematical model which every criterion has a certain weight and the normalized evaluations of the criteria are aggregated based on these weights, the second approach applies fuzzy logic to understand the complex relations between the diverse criteria in a more convenient way through using linguistic parameters, and performs if-then fuzzy rules to express the criteria relations and to figure out the aggregation.

An opportunist methodology for modeling. We identified a methodology to create system and adaptation models in a systematic way and with the right level of abstraction before feeding them to the cross-layer adaptation framework. The approach is opportunist in the sense that it benefits from a given set of tools to derive the system elements to be considered in the model. We gain two main advantages from this methodology. First, we identify the system elements more easily by taking into account simply the inputs and outputs of available tools. Second, we avoid creating unnecessary elements in the model, which do not have a corresponding tool, i.e., which will never be used by CLAM. The main steps of the methodology are as follows: (i) identify the analyzers and solvers available in the system, (ii) identify the inputs and outputs of analyzers and solvers, (iii) identify the system elements based on inputs and outputs of tools clarified in the previous step, here, we are particularly interested in analyzer inputs that are corresponding to the system parts which are subjected to be analyzed after adaptations, and in solver outputs that are corresponding to the system parts which are open to adaptation, (iv) identify the relations among system elements by the assistance of the SBS domain expert, (v) identify adaptation need types by looking at the solver inputs, (vi) identify adaptation action types by looking at the solver outputs, (vii) associate the analyzer outputs with the identified needs.

Following these steps, we guarantee to construct in an efficient way the system and adaptation models as well as integrating the tools with these models. The proposed methodology considerably facilitates the usage of our approach.

Heuristic methods for optimization and algorithm termination. We developed two heuristic methods to be used in practical settings, which the first one allows for guaranteeing the termination of our algorithm while finding a reasonable set of cross-layer adaptation solutions, and the second one allows for optimizing the tree construction.

Regarding termination, in a practical setting, we would like to avoid an explosion in the number of generated tree nodes due to the wide variety of tools, adaptations and the consequent system configurations. The principle of our termination method leans on the fact that we do not want to bring the system to a final configuration which is very different from the initial system configuration. The main idea behind is that we want to adapt the system, not to convert it into a totally new system as a result of a chain of adaptation actions because such cases imply a huge amount of efforts for the adaptation deployment, which obviously turns out to be comparable to the amount of efforts for a re-design. Thus, we propose a heuristic method to measure and control the number and distribution of changes imposed on the system due to a cross-layer adaptation. In this way, when we visit a tree node, if we identify that we overpass the threshold, which is defined on the basis of system configuration size and its maximum percentage that we permit for adaptation, we stop the tree expansion for that node.

Second, we utilize a greedy search algorithm [125] to efficiently traverse the CLAM tree under the termination conditions stated above. It is an informed search algorithm which first investigates the routes that appear to be most likely to lead towards the goal, in our case a cross-layer adaptation solution. In this algorithm, the node selection for the expansion of the tree is based on the goal-distance function that is an admissible "heuristic estimate" of the distance from the current node to the goal, i.e., for the given tree node, how close we are to the solution. It is not trivial at all to determine a promising function for the heuristic estimate of the distance to reach the goal. In this work, we present a novel optimization approach to search and find solutions faster compared to

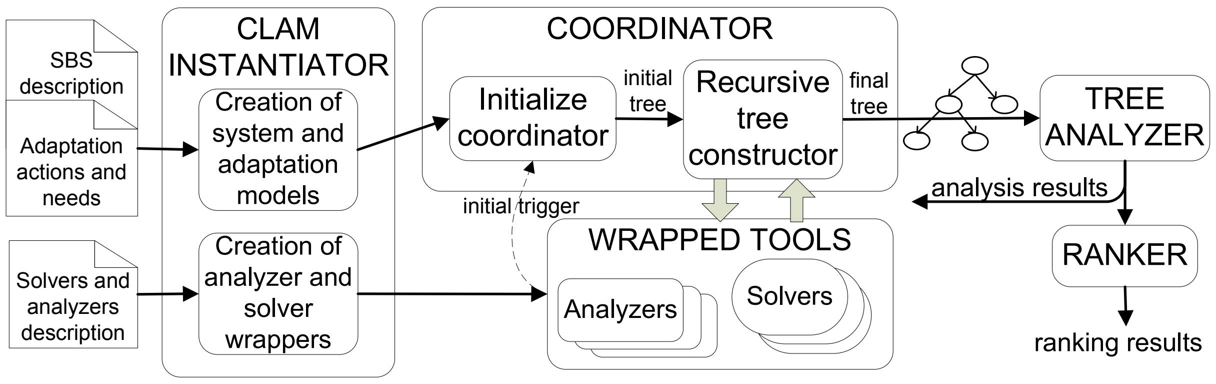


Figure 1.1: Prototype Cross-layer Adaptation Tool

the traditional approaches such as DFS and BFS. The main idea behind is the utilization of the system and adaptation models to observe solver behaviors at qualitative level, and thus to understand the best case and worst case consequent effects of adaptation actions on the system.

1.1.3 Prototype Tool and Empirical Evaluation

The cross-layer adaptation framework investigated in the dissertation has been implemented as a prototype toolkit, namely CLAM platform, and the implementation is evaluated on diverse case studies from an application scenario for the smart management of taxi reservations.

CLAM platform. The techniques and approaches presented here are implemented and incorporated into the CLAM Platform. The architecture of the platform is represented in Figure 1.1. Initially, the service-based system description, the identified adaptation actions and needs are translated into the system and adaptation models as described in the formal framework. Subsequently the tools are associated with these models and for each tool a wrapper is created in order to integrate the tool implementation with the platform and to be able to invoke it whenever it is required. Once tool wrappers are in place and models are created; coordinator, the core part of the platform where the CLAM algorithm is implemented, is ready to get an initial trigger from one of the wrapped tools, initial trigger can be an adaptation need as well as an adaptation action depending on whether it is an analyzer or a solver. Next, the recursive tree constructor iteratively expands the tree and continues to build it based on the heuristic termination method presented in the thesis. After the tree is constructed, it is passed to the tree analyzer for the paths analysis. First the solution paths are extracted from the tree, then each path is evaluated with respect to the selection criteria proposed in our approach. At the final step, the evaluation results are both published as output and fed to the tree ranker to get a final aggregate value, i.e., a rank for each path. For ranking, from the two proposed approaches, simple additive weighting – multi criteria decision making approach is applied.

Experimental evaluation. In order to provide a definitive account of the presented approach and incorporated methods, we evaluated our implementation with respect to various perspectives from the validation of the introduced techniques to the contribution of the proposed formal model.

The purpose of the experimental evaluations is threefold. First, we used the prototype tool to instantiate the cross-layer adaptation approach and the implemented ranking method. The results of the experiments demonstrated the viability of our approach, and its contribution with respect to the existing works when we consider the variety of adaptation capabilities that can be available in the SBS. Second, we validated the proposed heuristic methods, which enforces the termination of the CLAM algorithm in practical context as well as a more informed solution searching. The experiments revealed that the methods bring forward a considerable optimization with regard to the tree construction and a reasonable assurance of the termination. Third, we evaluated the contribution of the proposed modeling approach by changing the abstraction level of the model for the same SBS. The outcome of this empirical work showed that the opportunist modeling methodology minimizes the number of necessary analyzer invocations, and indeed, the efforts to create a very high level system model with a single node is comparable to the efforts to create the model adhering to the methodology which we propose.

1.2 Thesis Outline

The rest of the dissertation is organized as follows. In Chapter 2 we give an overview of the state of the art in the area of service-based systems and their monitoring and adaptation capabilities. This overview starts from the background information that provides a thorough insight into service-based systems and their layered structure. The chapter proceeds with the discussion of the current trends and technologies being applied for the adaptation and monitoring of these systems in general, and concludes with the review of the related works and research lines in the specific area of cross-layer adaptation. Chapter 3 provides a guided description of the formal framework for cross-layer adaptation, upon which the the techniques and approaches presented here are based. The chapter essentially motivates the need for such a framework and continues with the formal model of SBS and adaptation which are required by the framework. In Chapter 4 we present the cross-layer adaptation manager (CLAM) which realizes the proposed framework. We start with the formal definition of the cross-layer adaptation problem. In pursuit of this definition, we introduce the tree-based coordination algorithm which solves the problem, and finally, we provide the theoretical proofs demonstrating the correctness of the presented approach. Then, follows Chapter 5 which is dedicated to the selection and ranking problem regarding the alternative cross-layer adaptation solutions that CLAM produces. The chapter handles the problem in two phases; first we discuss the selection criteria and how to define them, then we propose two ranking methods which base on their operation on these criteria. Next, Chapter 6 presents the implementation of the CLAM platform in details and proceeds with the description of the proposed modeling methodology and an overall user guide in order to utilize the platform. Immediately afterwards, Chapter 7 is dedicated to present and discuss the empirical evaluation of the techniques and approaches proposed in the dissertation as well as the explanation of the heuristic methods that we use for optimization and termination. Eventually, concluding remarks and the future research directions are discussed in Chapter 8.

Chapter 2

State of the Art

2.1 Service-Based Systems

An SBA is an application that cannot be implemented by a singular service, but requires the aggregation of multiple services in a network to guarantee the application goals [95]. Services are independent entities that can achieve a specific task or a group of tasks and be re-utilized for different applications in different contexts. They can be provided by either the SBA owner itself or mostly by a third party.

An SBS is the overall environment of the SBA, which includes functional and quality requirements of the application, the execution context, underlying services, platforms and infrastructures, and the corresponding support mechanisms available for the monitoring and adaptation of the application.

SBSs are multi-layered in nature. Particularly the recent advancements in cloud computing introduce the service concept at different levels of abstraction and enables us to build software as a service (SaaS) on top of various platforms (PaaS) and infrastructures (IaaS) that are also provided as a service [115].

An SBS can have different layers depending on the application domain. For instance, in a geographically distributed complex application, agile service networks and cross-organizational relations can be important and introduced as an organizational layer. Whereas for an enterprise application this might not be the case since all the partner services might be provided by the same company.

An illustrative SBA can be presented by its three functional layers: application layer, constituent services layer and the underlying infrastructure layer. Application layer is the highest level functional layer where the business process is described in terms of application activities, constraints and requirements. At this layer, the entire business process can be constructed as a composition of services and key performance indicators (KPI) can be introduced as metrics that show quantitatively if the business performance meets the pre-defined business goals of the SBA. Secondly, at services layer we can have software or human-based partner services, which are composed for the business process. Composition can be achieved by service orchestration and services which are part of an orchestration can be atomic services or again service orchestrations. Atomic services are self-contained and do not use other services for implementing their business logic. Finally at infrastructure layer we have the underlying resources to build and run the SBAs and their constituent services. These resources include software platforms, operating systems as well as low level storage, computing and memory facilities.

2.1.1 Business Process Management

A process is an ordered set of activities with a start point and an end; it has inputs in terms of resources and the required information and an output, i.e., the results it produces [94]. Therefore, we may define a process as any ordering of steps that is initiated by an event and produces some output [32]. A business process is a process that aims at achieving a well-defined business outcome and is completed considering a set of procedures. The key points for business processes are that they may span organizations and may typically involve both people and systems.

Business process may reveal the entire SBA or in some cases part of it where SBA is actually a combination of business processes in a large-scale, cross-organizational domain. While business activities constitute the process by performing well-defined tasks, business rules together with business policies might have an implicit or explicit effect on the specification of business processes and activities.

Performance of business process is assessed by pre-defined business goals that are measured by KPIs. KPIs are formed by assigning target values to the business metrics that are relevant for the application. Business metrics can be financial such as revenues, customer-related such as customer satisfaction index, process-related such as order fulfillment cycle time or “learning and growth” related such as innovation rate [124].

In some cases there might be cross-organizational interactions among companies, which collaborate to construct geographically distributed complex SBAs. To illustrate such interactions, agile service network (ASN) model is used. ASN depicts the highest and most abstract business level where partners (constitutive companies) are nodes and their offerings and revenues are edges on the model. While ASN pictures overall relations of service providers without any details, the refinement of this model for a specific SBA signifies forming the business processes of the application [60].

2.1.2 Services and Service Compositions

There exist several definitions to define what a service is. One distinct definition is as follows [78]:

A Service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by an entity, namely the service provider, for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by *the provider.*

Services have functional and non-functional aspects. While the capabilities that a service offers to its consumers correspond to the functional aspects, service quality metrics (QoS) expose non-functional properties of a service building a base for the overall performance metrics of the SBA. QoS metrics talk about several aspects of a service such as response time, cost, availability, reliability or scalability.

We can compose services to create new services. Service composition is the combination of a set of services for achieving a certain purpose [65]. When talking about service composition, the concepts of orchestration and choreography come to one's mind. Description of these terms differ especially considering the interactions among services to be composed [98]. The main difference is that for orchestration the interaction protocol is described from the point of view of a specific service, on the other hand a choreography specification gives a global view of the interactions among all the services.

A service orchestration composes a new service by orchestrating several services in a process flow. Services which are part of an orchestration can be atomic services or again service orchestrations. Atomic services are self-contained and do not use other services for implementing their business logic.

There have been several efforts related to the creation of complex services out of simpler ones using various languages, e.g. "WSFL" [72], "XLANG" [112], "BPML" [8], "BPEL" [5].

2.1.3 Platforms and Infrastructures

Service platforms and infrastructures provision the necessary execution environment for SBAs. This run-time environment includes the hardware resources such as the cpu, memory and storage as well as operating systems, middleware, application containers and other software. Moreover, services and SBAs can be deployed on top of grids and clouds where resources include protocols and network infrastructure as well [84, 96].

In some cases infrastructures can be configurable and scalable based on the changing SBA requirements. Especially, clouds provide a new level of flexibility for SBAs in this sense. While pre-configured machine images decrease the deployment cost, several load-balancing mechanisms and easy resource allocation significantly decrease the run-time cost [115].

Like services, cloud providers as well offer a certain level of QoS for its consumers. QoS parameters may refer to several aspects of the clouds such as response time, cost, availability or reliability [39].

2.2 Monitoring, Analysis and Adaptation of Service-Based Systems

Monitoring and Analysis. The term "monitoring" has been extensively used in many disciplines and particularly in service-oriented applications. Depending on a particular purpose of the system and the kind of information being collected, the definition of the monitoring has different interpretations. In a broad sense, monitoring may be defined as a process of collecting and reporting relevant information about the status, execution and evolution of service-based applications. This general definition becomes more concrete when the monitoring goals are taken into account. Monitoring may be used to discover problems in the application configuration. In this case monitoring may be defined as an "analysis" process, i.e., a problem of observing the behavior of a system and determining if it is consistent with a given specification [33].

Especially for SOA, monitoring aspects may be both functional and non-functional [44]. Functional properties characterize the function (or behavior) that a given system is expected to provide. Typical examples of the functional properties are failures, assertions or behavioral properties, invariants. Non-functional properties define quality characteristics that often can be measured in a quantitative way. Typical non-functional properties refer to availability, latency, reliability.

Monitoring capabilities are the mechanisms and technologies defined at different parts of the SBS for collecting information about the behavior, functional and non-functional properties of the application and the changes in the environment. Indeed, monitoring capabilities at all layers are necessary to understand whether the SBA is executed and evolves in a normal mode and whether there are some deviations or violations of the desired functionality and performance.

At the Business Process Management level, monitoring capabilities include real-time monitoring of business activities, the measurement of KPIs, as well as mechanisms for a proactive identification and notification of deviations. Considering service compositions monitoring capabilities mainly address the problem of checking whether certain predefined properties are satisfied when the service composition is executed. Finally, we have monitoring mechanisms that report on quality aspects of services and the underlying infrastructures [60].

In [97] adaptation of services and processes is remarked as a research challenge in SBSs. Systems should be furnished with adaptation capabilities so that they can continually update themselves to respond to the problems reported by monitoring and analysis mechanisms.

Adaptation. We can define adaptation as a process of modifying SBS in order to satisfy new requirements and to fit new situations dictated by the environment. An adaptable SBS is a system augmented with the corresponding control loop that monitors and modifies itself on the basis of adaptation strategies. Notice that adaptations can be performed either because monitoring has caught a problem or because the application identifies possible optimizations or because its execution context has changed. The context here may have a broad meaning, that is the set of services available to compose SBAs, the computational resources available, the parameters and protocols being in place, user

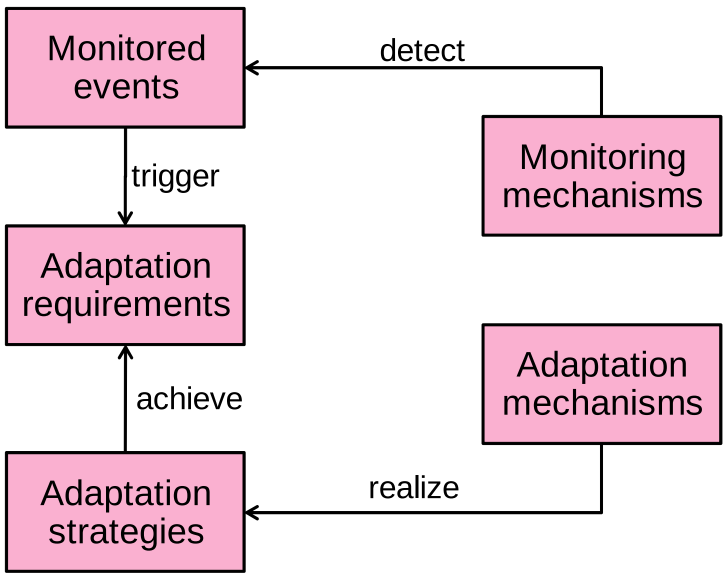


Figure 2.1: Conceptual A&M framework

preferences, environment characteristics [51].

SBSs have diverse adaptation capabilities regarding different layers of the system. Adaptation capabilities for the Business Process Management may range from temporary modifications to specific components of the business processes to permanent reconfigurations of the whole business model and business network the latter case often requires human intervention and re-design activities. Adaptation capabilities for service compositions and services usually correspond to the activation of pre-codified adaptation strategies. These strategies can be defined in various ways, ranging from procedural approaches (concrete actions to be performed are specified), over declarative approaches (the goals and requirements to be achieve are specified), to hybrid approaches. Adaptation capabilities for service infrastructures provide mechanisms related to resource brokering, load balancing and renegotiation of Quality of Service (QoS) parameters [60].

A general framework for SBA monitoring and adaptation defined in [51] is represented in Figure 3.5. In this framework *monitoring mechanism* refers to any analysis mechanism that can be used to check whether the actual situation corresponds to the expected one. Thus, the meaning is very broad; it goes beyond traditional run-time monitoring and may include run-time verification and testing, post-mortem analysis, data mining, etc. With these mechanisms one may detect *monitored events*, i.e., the events that deliver the relevant information about the application status and context. In turn, monitored events trigger *adaptation requirements* (or adaptation needs), which represent the necessity to update the SBS in order to remove the difference between the actual (or predicted) situation and the expected one. In order to satisfy those requirements it is necessary to define *adaptation strategies*, which in turn are realized with the appropriate *adaptation mechanisms* – the techniques and facilities provided by the application or by the operation and management platform in different SBS layers.

2.2.1 Monitoring and Analysis at Business Processes Level

Monitoring and analysis approaches at business process level constitute "Business Activity Monitoring (BAM)" and KPI monitoring [36]. BAM enables near real-time monitoring of business activities, measurement of KPIs, and automatic and proactive notification in case of deviations and violations [17, 107, 57]. However, there are approaches other than BAM as well to detect KPI deviations and violations in business processes [23, 122].

Among approaches that focus on BAM [17] presents the BP-Mon language and its underlying formal model. BP-Mon is a high-level intuitive graphical query language that allows for easy description of the execution patterns to be monitored. An important feature of the language's implementation is that BP-Mon queries are translated to BPEL processes that run on the same execution engine as the monitored processes. Differently, [107] enables automatical extension of WS-BPEL processes with probe points in order to report interesting business activities to an auditing web service. The resulting auditable process definition does not use proprietary elements but remains compliant with the WS-BPEL standard. Another approach is [57] which proposes an agent-based architecture with the aim of providing continuous analysis for business activities. The authors introduce a container-based method for the event processing that enables a near realtime event data integration.

Both [23] and [122] address monitoring and analysis of KPIs, and prediction of violations based on data mining. In [23], proposed by HP, abstract process monitor is the basic component responsible for collecting data and interpreting them at the level of abstract processes. Once process data is available, KPI values are computed by the metric computation engine and analyzed through decision tree-based prediction engine. Differently, in [122] also the QoS parameters of lower system layers are considered as factors of influence. Moreover, the authors provide a drill-down functionality to gain deeper knowledge about the structure of dependencies.

2.2.2 Monitoring and Analysis at Services and Service Compositions Level

At the services and service compositions level the monitoring engines and frameworks provide means to observe the execution of composed services (specified, e.g., in BPEL), including functional and non-functional properties and metrics of the compositions [13, 79, 137], or the constituent services [61, 75].

[13] focuses on the run-time checking of the assumptions under which the component services are supposed to participate to the composition, and the conditions that the composition is expected to satisfy. The authors propose the specification framework that relies on Run-Time Monitoring Language (RTML), and runtime monitoring environment that extends the standard BPEL engine with the monitoring and auditing capabilities. [79] allows expressing complex properties over events, fluents, and timestamps in order to monitor the SLA properties in the WS-Agreement notation. The monitoring process relies on automated extraction of the special templates that represent the formulas in event

calculus. Differently, [137] proposes an analysis method to estimate the QoS of a service composition, in which the probability distributions of the QoS of component services can be in any shape.

Regarding service monitoring, [61] presents the WSLA framework that defines a language for the specification of contract information that allows for describing the parties involved in the agreement, the relevant QoS characteristics, as well as the ways to observe and measure them. Similarly, [75] proposes the Cremona architecture for creation, management and monitoring service-level agreements represented as WS-Agreement documents.

2.2.3 Monitoring and Analysis at Platforms and Infrastructures Level

Monitoring and analysis at platforms and infrastructures level may be realized on top of Grid and cloud monitoring solutions [4, 63, 92], platform governance [69, 120].

The authors in [4] propose the GridICE tool to monitor the status of grid infrastructures in a centralized way. It is implemented on top of Globus MDS-2 grid information services [31]. As more recent work, [63] introduces a failure propagation approach for the cloud-based applications in order to avoid costly SLA violations. Differently, [92] proposes to monitor directly the cloud through Q-Clouds, a QoS-aware control framework that tunes resource allocations to mitigate performance interference effects. Q-Clouds uses online feedback to build a multi-input multi-output (MIMO) model that captures performance interference interactions.

Regarding the platforms, [69] presents an ecosystem-oriented model for developing and deploying enterprise applications in the cloud in order to support the governance of cloud application platforms. Differently, [120] performs platform monitoring from the service management point of view.

2.2.4 Adaptation at Business Processes Level

Several approaches exist to adapt business processes [18, 77, 42, 59, 46]. [18] presents a methodology for the automated generation of adapters for data flow modifications in a workflow, but its applicability is confined to resolution of mismatches, which occur as a result of a workflow modification. Differently, [77] considers the semantic verification of business goals upon an adaptation in the process, but the authors do not mention how an adaptation is triggered and applied. [42] presents a technique for automatic error correction during the design of process models. The approach is based on simulated annealing and it identifies a number of process model alternatives that resolve one or more errors in the original model. In [59] the authors propose to improve the flexibility of processes where modification of the control flow is possible at run time by adding or replacing services. The approach benefits from the analogy between publish/subscribe systems and the AOP paradigm. Finally, the authors of [46] bring a different perspective to the adaptation by introducing a variability framework for BPM which utilizes temporal logic formalisms to represent the essence of a process, leaving other choices open for later customization or adaption. The goal is to solve two main issues of BPM: improving reusability and flexibility.

2.2.5 Adaptation at Services and Service Compositions Level

Naturally, most of the adaptation approaches in SBSs focus on services and their compositions. Dynamic service binding [117, 29], QoS awareness [93, 7, 28, 119, 12], data mediator design [68, 90, 67, 24, 50] and context awareness [7, 81, 20, 19].

When unavailability or poor efficiency of some services are monitored during the application run; in this case one possibility to adapt is dynamic service binding and re-binding. This kind of adaptation is studied by [117, 29] where run-time execution and re-configuration of service composition are enabled. In [117], the METEOR-S framework is proposed to give flexibility to the service binding operation making use of semantic web languages. In [29], SCENE platform is introduced where proxies for dynamic binding and re-binding cases are generated and BPEL code is modified to include calls to these proxies. The suitable services are selected among the candidates based on the specified rules.

When QoS violation of a service is detected by the monitoring system, replacement of the service or re-negotiation with the service provider are among alternatives for adaptation. Usually Service Level Agreements (SLA) are utilized to keep QoS awareness in SBSs [93]. An SLA is a contract between the service provider and the service user and used to set the QoS values to be ensured by the service provider. [93] proposes an approach called NSLA for negotiation of SLAs. Both direct negotiation of the interested stakeholders and automatic negotiation are enabled in NSLA framework. Similarly, [7] presents a framework for service adaptation where QoS negotiation is enabled at design-time. Instead, in [28] an execution policy that comprises of restrictions for QoS attributes is specified and most suitable services are selected dynamically based on this policy. Similarly, [119] proposes a QoS-based composition. The authors present an approach that arranges functionally similar services in clusters and computes the QoS of each cluster, and finally the planning tool composes workflows consisting of these clusters. The authors of [12] modify processes for QoS conformance as well, they present ProAdapt, a framework for proactive adaptation of service composition due to changes in service operation response times, or unavailability of operations.

In some cases, the monitoring system detects mismatches in service interfaces, protocols or the data formats. To tackle this problem, mediators are designed to re-enable proper service interactions. In [68, 90, 67] adapter patterns are defined for both interface and protocol level mismatches. In [67] it is possible to separate the adaptation logic from the business logic by aspect orientation.

[24] proposes automatically generated adaptation scripts for protocol-level mismatches. Mismatch is defined as the difference in the order of operations offered by the expected service and the real concrete service. In [50] mismatches are categorized with respect to the necessity and possibility of an adaptation and then an adaptation procedure is described through operators for the solvable mismatches in the categorization.

The context refers to the dynamic environment in which an SBS is embedded and executed. The environment can encompass various aspects such as a user, a computing system and/or a physical environment. A context-aware monitor can report the changes in the context, and depending on the affected part of the application, various adaptation strategies can be used to adapt the application to the dynamicity of the environment. Approaches in [7, 81, 20] are in relevance to this kind of adaptation case. The approach proposed by [7], "PAWS", enables context-awareness as well as QoS awareness. In PAWS framework, the run-time environment exploits the design-time mechanisms to preserve the execution when a context change occurs. [81] introduces some basic constructs and principles to enable built-in adaptation for pervasive flows. The authors of [20, 19] provide an adaptation approach that can automatically adapt business processes to run-time context changes which prevent achievement of a business goal. They define a formal framework that adopts planning techniques to automatically derive necessary adaptation activities on demand.

2.2.6 Adaptation at Platforms and Infrastructures Level

Various self-* techniques are proposed to adapt SBSs at infrastructure level. While some adaptations may be service deployment/platform related [70, 99, 26, 86, 63], some other focuses on computing resources [2, 110, 74].

The work on self-adaptation and self-healing mostly addresses the monitored failures and problems in services infrastructures and platforms. Various repair mechanisms are applied to implement self-healing systems. In [70]

WS-DIAMOND framework is presented to allow easy diagnosis and recovery of services at run-time. The approach in [26] presents a plug-in architecture where adaptation actions can be integrated as aspects by using dynamic aspect-oriented workflow language (AO4BPEL). [99] proposes a methodology and a tool to incrementally learn the repair strategies based on keeping the history of previously performed repair actions for web services. Another self-healing approach is presented in [86]. The authors define a QoS-aware infrastructure for WS-based applications through auto-detection, diagnosis, and recovering of hardware and software problems. Finally, [63] introduces an SLA-aware service virtualization architecture that provides non-functional guarantees in the form of Service Level Agreements and enables on demand application deployment in the cloud.

Regarding resource management, [2] proposes an optimization model that identifies the optimal resource allocation in service oriented applications by maximizing a provider's revenues while satisfying customers QoS constraints and minimizing resource usage cost. [110] uses OpenNebula and Haizea together to provide a resource management solution that supports a variety of lease types in the clouds. Eventually, the authors of [74] formalize a model of a service-oriented computing environment and a workflow graph representation for the environment. Then, they propose SCPOR, a scientific workflow scheduling algorithm that is able to schedule workflows in need of elastically changing compute resources.

2.3 Modeling Service-Based Systems

Modeling can serve service-based systems in various ways. Considering the complexity of such systems, one clear advantage that modeling brings could be at the deployment stage of such systems to conceptualize and construct them better. Moreover, it helps throughout the whole SBS life-cycle, i.e., it facilitates maintaining the deployed systems as well in terms of both monitoring/analysis and adaptation.

2.3.1 Modeling for Application Deployment

The works in [10, 9, 11, 71, 83] propose models for customizable development and easy deployment of service-based applications.

The work in [10] presents SOAD, an approach proposed by IBM, which aims to give SOA practitioners concrete, tangible service modeling and realization advice by creating a reusable SOA decision model structured according to Model-Driven Architecture (MDA) principles. The methodology relies on the codification of existing knowledge in the form of architectural patterns and decision models to identify the required decisions, give domain-specific pattern selection recommendations, and provide a link from platform-independent patterns to platform-specific decisions. SOMA is another method developed by IBM [11] to guide in the modeling (design) of SOAs. It enables creation of SBAs by creating continuity between the business intent and the specific technical implementation. Differently, the authors of [9] focus on the infrastructural point of view and address the challenge of configuring the hosting infrastructure for SOA service deployment by formally capturing service deployment best-practices as model-based patterns. Yet, [71] presents another formal framework to provide expert users with the ability of rapidly building service-based applications according to new requirements and needs. The underlying technique integrates two visual and executable formalisms: live sequence charts, to describe control flow, and graph transformation systems, to describe data flow and processing. Finally, [83] presents a generic approach for the modeling, packaging, customization and on-demand provisioning of SBAs. They introduce a variability metamodel to model customizable aspects of the application.

2.3.2 Modeling for Monitoring and Analysis

There have been modeling approaches to facilitate monitoring and analysis of SBSs. While some of these approaches concentrate on directly modeling the monitoring properties [85, 14, 111], some other focus on modeling the SBS to support system analysis [127, 58].

The authors of [85] present an MDA approach which formalizes the monitoring requirements by focusing on supporting the specification and transformation of KPIs to an executable implementation. [14] utilizes MDA approach as well for Model-Driven Management of Services (MDMS). The approach supports the explicit modeling of quality dimensions, management objectives, and key performance indicators, and the transformations required to exploit these concepts at runtime for monitoring. [111] proposes modeling for monitoring lower levels for SOA compliance. Emitted events contain unique identifiers of models that can be retrieved dynamically during runtime from a model-aware repository and service environment. In [127], the authors model collaborative services to enforce strong accountability, analyze incompliances and enhance the trustworthiness of business processes. Finally, [58] presents an architectural approach to model, enact and manage business processes and their changes based on explicitly represented service relationships of a service composition.

2.3.3 Modeling for Adaptation

Although SBS modeling approaches in literature mostly target system deployment and system analysis, yet there are modeling approaches proposed to support adaptation of SBSs [113, 21, 43, 46].

[113] presents a methodology and system for changing SOA-based business process implementation. The authors distinguish two layers for change management: At the design layer processes are modeled in the ontology-based semantic markup language for web services OWL-S. At the execution layer the processes

are translated into BPEL. In [21] the authors present a model-based approach that utilizes goal and variability models to define design-time and runtime elements of service-oriented systems. Variability models are built from the analysis of goal models that capture stakeholders needs. They define the allowed system changes at the levels of stakeholder needs, architecture, and execution environment. Similarly, [46] introduces a variability framework for BPM. The framework utilizes temporal logic formalisms to represent the essence of a process, leaving other choices open for later customization or adaptation where the goal is to solve two major issues of BPM: enhancing reusability and flexibility. Differently, [43] focuses on the influence of the environment on the application and proposes a stochastic Petri net based method on modeling an environment-aware self-adaptive strategy for SBAs. The method proposed builds separate models for both service and environment.

2.4 Cross-layer Approaches in Software Systems

When we consider software systems, we always encounter interacting, interdependent parts in them. This is due to the fact that we integrate various small functionalities to create a bigger, more complex functionality that we call "the system". Usually, these parts group among each other with respect to their diverse levels of abstraction, which we call "layers" [16]. Various cross-layer approaches emerged in literature due to this characteristic of software systems.

There have been approaches that focus on the development and deployment of systems taking into account the cross-layer aspects of the system. In [9], the authors propose a pattern-based method to address the challenge of configuring the hosting infrastructure for SOA service deployment. Similarly, [83] considers the relation between the application and the underlying infrastructure and proposes an approach to configure and automatically deploy applications in the cloud. Instead, [25] deals with adaptable service development through identification of a set of variability types, which consider also the service execution platform.

There have been some cross-layer works that aims at addressing the monitoring problem. The approach in [88] proposes a cross-layer monitoring approach that considers service and infrastructure level events, which are produced by services communicating via a distributed enterprise service bus. Similarly, [40] proposes an extensible framework for monitoring business, software, and infrastructure services, yet the approach does not support the correlation of terms monitored at different layers.

Diagnosis and analysis of systems is another notable research attempt considering state-of-the-art cross-layer approaches. [126] presents a framework, proposed by SLA@SOI Project, which enables a process-centric business continuity analysis. In this approach, business impact of failed IT services are estimated and SLAs are validated through a top-down dependency analysis based on simulation. [69] is a similar approach that performs cross-layer diagnosis to report policy violations. Differently, [121] proposes an approach for business it alignment where there exist a mapping between the BPMN and BPEL so that they are synchronized in case of a change in the process at BPMN or BPEL level.

Finally there have been attempts for coordinated adaptation taking into account the cross-layer nature of systems. [128] supports supports application QoS under CPU and energy constraints via coordinated adaptation in the hardware, OS, and application layers in multimedia systems. Similarly, [34] focuses on operating systems and uses a common multi-level adaptation framework to adapt both the OS and the application layers in a coordinated way. Differently, [38] deals with the problem of management of resources shared by multiple applications interacting with each other in the same environment. While [128, 34] cover cross-layer adaptation issues in operating systems, the authors of [38] target mobile applications domain. We introduce and discuss the cross-layer adaptation approaches for the area of SBSs separately in next section and in more detail since they are the most relevant works regarding the results we present in this thesis.

2.5 Related Work on Cross-layer Adaptation

A well-known method of managing SBSs is MAPE (monitoring, analysis, planning, execution) [62, 91]. Thus, the traditional way to adapt a service-based application follows MAPE's rules: *(i)* The running application is continuously being monitored. *(ii)* Monitoring data are collected and analyzed to identify anomalies in the system. *(iii)* For each anomaly there exists an adaptation plan to get rid of the problem. *(iv)* Once the adaptation is decided, it is executed in the system, so the running SBS is updated accordingly.

There is a significant problem with this habitual practice of MAPE. For complex systems like SBSs, and given their multi-layer structure, it is probable that an adaptation creates a new problem while it solves another problem, which means that the traditional way of applying MAPE fails. That is to say, without understanding the consequences of an adaptation action for the whole system, it might be too costly, even harmful to update the system immediately after an adaptation decision. Therefore, we should have coordination mechanisms which take into account the cross-layer aspects of the system [80, 87] and derive holistic adaptation strategies accordingly. Only once the adaptation strategies are validated by these coordination mechanisms, then they should be applied to the system.

In this section we overview related works on cross-layer adaptation, presented in the literature. We discuss the approaches and the results obtained with respect to the goals and problems we presented in the introduction to this thesis, namely the ability *(i)* to accommodate different SBS domains, different SBS layers (heterogeneity of SBSs), *(ii)* to support new application goals, new adaptations (dynamicity of SBSs), *(iii)* to take into account the fragmented nature of SBS layers and layer-specific adaptation capabilities, *(iv)* to consider possible contradictory adaptations, i.e., to analyze the impact of adaptations on the SBS, *(v)* to propose alternative cross-layer solutions and a selection mechanism.

2.5.1 Cross-layer Adaptation Approaches in Service-Based Systems

Cross-layer adaptation of service-based systems has recently become a focus of a number of research works. Among them, the most notable ones are [45, 41, 53, 132, 103, 123, 108] that we would like to discuss in detail in this section.

In [45] the authors propose a framework to support cross-layer self-adaption in SBSs. They present a technologically agnostic middleware to facilitate coordinated cross-layer adaptations by integrating interface and application layer adaptation mechanisms. Service interface layer is the layer which constitute loosely coupled services, it hides service implementation details and the technology platform. The application layer is the layer in which service logic is developed and deployed on different technology platforms.

The framework developed in [41] addresses a different problem: How to model cross-layer SLA contracts for monitoring and adaptation. The proposed SLA contract model includes parameters of KPI, key goal indicators (KGI) and IT infrastructure metrics. The authors present a methodology for creating, monitoring, and adapting an SLA contract, in particular, leveraging aspects of Quality of Service (QoS) violations. “User”, “business” and “IT infrastructure” are the layers referred in this work.

[53] provides an intelligent traceability-based framework for analyzing the impact that changes in SOA-based systems can have on key performance indicators. The impact analysis is accomplished through evaluating KPIs based on the aggregated performance of lower level metrics. The framework targets a specific type of SBS where the layers are service, business and infrastructure, and exploits a dependency model among a set of fixed SOA artifacts.

The authors of [132] propose a framework, able to monitor and adapt SBAs across business process, service composition and service infrastructure layers. This is achieved by using techniques, such as event monitoring and logging, event-pattern detection, and mapping between event patterns and appropriate adaptation strategies. In addition, a taxonomy of adaptation-related events and a meta-model describing the dependencies among the SBA layers are introduced in order to ensure the cross-layer effects.

[103] presents a pattern-based approach for multilayer application adaptation, with layer-specific adaptation solution templates bound to application mismatches that are organized into hierarchical taxonomies. A mismatch is any kind of event that should be addressed by an adaptation template in the system, and a template is any adaptation capability available in the system that is exposed as a service with a WSDL interface.

In [123] the authors analyze the dependencies of KPIs on process quality factors from different functional levels of an SBS such as QoS parameters, and then an adaptation strategy is decided to improve all the negatively affected quality metrics in the system. To analyze KPI violations, machine learning techniques are used. Instead, adaptation strategies are predefined for every negatively affected metric.

The work presented in [108] proposes a solution to avoid SLA violations by applying cross-layer adaptation techniques. The approach exploits SBS layers, which are business, service composition and infrastructure, for the prevention of Service Level Agreement (SLA) violations. The identification of adaptation needs is based on QoS prediction, which uses assumptions on the characteristics of the running execution context. Multiple adaptation mechanisms are available to react on the adaptation need, acting on different layers of the SBS, the right one is selected by the adaptation strategy engine, which is a multi-agent platform.

2.5.2 Comparison of Cross-layer Adaptation Approaches

In this section we compare the cross-layer adaptation approaches presented in previous section as they are particularly relevant to the one we introduce in this dissertation.

The comparison we present here, without any claim to be thorough, focuses on the important aspects of SBSs that the cross-layer adaptation should take into consideration as discussed in the introduction of this thesis, namely: *(i) Heterogeneity of SBSs:* There is a wide variety of SBSs with different domains and layers. *(ii) Dynamicity of SBSs:* Business goals, application constraints, maintenance tools, even the application itself can change during the lifetime of an SBS. *(iii) Independent development and maintenance of SBS layers:* The SBS layers are designed, developed and maintained by different experts. Similarly, analysis, monitoring and adaptation tools, available for the SBS, work on only a restricted part of the system and are unaware of the rest. *(iv) Incompatible adaptations:* An adaptation that is performed in one part of the system might affect negatively another part of the system. *(v) Variety of solutions:* There might be various solutions to address an adaptation problem.

Heterogeneity of SBSs. The system elements considered in a cross-layer adaptation solution cannot be kept fixed. The solution should be generic enough to accommodate various application domains. Most of the approaches presented in previous section ([45, 41, 53, 132, 123, 108]) propose a specific SBS model with fixed layers and system aspects. While [45] proposes a two-layer system which consists of service interface and service application layers, [41] takes into account the user aspect and introduces user, business and IT infrastructure layers. The approaches [53, 132, 123, 108] follow the architecture proposed by S-Cube project [1] with the layers business process, service composition and service infrastructure. Differently, [103] has a general definition of multilayer applications and support the diversity of SBS domains and their system layers.

Dynamicity of SBSs. The solution should be flexible enough to accommodate introduction of new system elements and/or analysis, monitoring and adaptation tools during the lifetime of the SBS. Among the cross-layer adaptation approaches, while [45, 41, 132, 103, 123] supports introducing new adaptation actions in the solution, adding new monitoring and analysis tools to the overall analysis is supported only by [103, 123]. Moreover, only [103] has the flexibility to introduce new system elements, new system layers to the existing SBS.

Independent development and maintenance of SBS layers. While the existing fragmented adaptation capabilities should be utilized properly by the cross-layer coordination, the solution should keep in mind the fact that fragmented view of SBS should be coordinated but cannot be totally centralized in a way that there is a single adaptation expert who takes control of the whole system knowing all the probable problems and solutions that might occur in the SBS. Except for [53], all the current approaches support re-use of existing adaptation mechanisms used for the maintenance of the SBS. However, all of these approaches are centralized in the sense that they expect that the adaptation designer knows about the whole system to build the cross-layer adaptation solutions.

Incompatible adaptations. There might be several conflicting adaptation goals in the system, which means an adaptation can influence a system aspect negatively while it is reinforcing another system aspect. Therefore the solution should ensure that the proposed cross-layer adaptation strategy is compatible with the overall system. Among the works presented in the previous section, only [53] enables the impact analysis of the proposed adaptations on the whole system. However, if an adaptation has a problem, i.e., negatively influences some part of the system, no solution is provided to tackle this inconsistency.

Variety of solutions. There might be several cross-layer adaptation paths to address a problem in the system, the solution should have a selection and ranking mechanism to organize the alternatives. This is the case only in the works [103, 123]. In [123] the list of alternative adaptation strategies is filtered and ranked based on a constraints and preferences model. Constraints allow for defining conditions which should never be violated, while preferences are specified as weights on different quality metrics of the SBS. In [103] an adaptation strategy may involve specific adaptation solutions or the general ones. The authors propose a user-configurable ranking mechanism that employs the following criteria for an adaptation strategy: number of specific adaptation actions, number of general adaptation actions, number of total adaptation actions, and number of raised events for the strategy.

2.6 Discussion

One of the key features of service-based systems (SBS) is the capability to adapt in order to react to various changes in the business requirements and the application context. Holistic adaptation approaches are of primary importance to efficiently cope with the complex layered structure, and the heterogeneous and dynamic execution context of SBSs.

Several approaches have been proposed to tackle the adaptation problem. However, a fundamental issue with most of these works is their fragmentation and isolation. While these solutions are quite effective when the specific system problem they try to solve is regarded, they may be incompatible when the whole system is taken into account. This challenge should be handled by properly coordinating adaptation actions provided by the different analysis and decision mechanisms through cross-layer adaptation strategies.

Among the limited number of cross-layer adaptation approaches, most of them suffer from the fact that they ignore or oversimplify the important features of service-based systems. Some of them restrict the solution for a concrete, layered architecture of the SBS getting far from being generic, most of them ignore the effects of the adaptation on system layers, and finally, none of them take into account that SBS adaptation coordination cannot be totally centralized in a way that adaptation designer knows everything and predefines all the means for cross-layer adaptation.

In our research work we were primarily motivated by the necessity to propose a cross-layer adaptation approach that overcomes the limitations of existing works and is capable of tackling the issues specific to the heterogeneity and complexity of service-based systems. Among the most important characteristics provided by our approach are (i) the genericness for accommodating diverse SBS domains that can have different layers and system aspects, (ii) the flexibility for allowing new system artifacts and adaptation tools, (iii) the capability for dealing with the complexity of the SBS considering the possibility of a huge number of problems and adaptations that might interfere with each other in the same system.

Chapter 3

Cross-layer Adaptation Framework

In this chapter, we present a formal framework of our approach that allows for the representation of the SBS and its adaptation capabilities. In the end, the overall formal framework will enable the specification of cross-layer adaptation problem and form a basis to solve it.

To begin with, we would like to discuss the main purpose of our approach and motivate the need for the formal framework. Afterwards we will introduce the formal models required by this framework.

3.1 Motivation

A system can be defined as a set of interacting or interdependent components forming an integrated whole. Open dynamic systems extend the general definition of systems since they evolve by time and are subject to the environmental changes [76]. A well known method to support such systems is to apply the feedback control theory [35]. In control theory, the external input of a system is called the reference which corresponds to the desired behavior or characteristic of the system. The controller is responsible for ensuring the desired effect by checking the system status and manipulating it if needed (Figure 3.1).

Classical adaptive systems follow this control paradigm to react to the changes in the system context [82]. Most of the adaptive service-based systems, being an open and dynamic system, base their adaptation logic on the control theory. Figure 3.2 depicts such SBSs: (i) the monitors collect the data about the SBS status, (ii) the monitored data are passed to the system analysis facilities, i.e., analyzers, (iii) analyzers check the collected data with respect to the system constraints and signal problems in case they do not match the constraints, (iv) solvers, being the adaptation capabilities of the SBS, address the problem by proposing concrete actions, (v) and finally, executors update the SBS by deploying the proposed actions in the system.

A fundamental limitation in such SBSs is that they work on adaptation loops, demonstrated in Figure 3.2, which each loop operates totally in isolation from the other loops. Thus, what those SBSs propose is a set of (problem, solution) pairs and the effect of one pair on the others is ignored, and therefore unknown. While such approaches can achieve an overall system adaptiveness under the assumption that all the pairs are totally independent, i.e., do not imply any consequences on other pairs, we know that this assumption contradicts with the principal characteristic of systems: system components interact and are interdependent.

In this thesis we propose a novel framework to tackle this limitation. We break those closed loops of adaptation and propose a reasoner to coordinate the main constituents of adaptation, namely, analyzers and solvers (Figure 3.3). Note that in this case, the system constraints, on which the analyzers need to operate, are internal to the reasoner.

The main idea behind the overall approach is that, our understanding of adaptive SBS takes into account the dependencies of system elements, and the reasoner bases its adaptation coordination on those dependencies before enacting any adaptation decision in the system.

The remainder of this chapter focuses on the presentation of the main ingredients of the reasoner through formal definitions. We describe the *system dependency model* that the reasoner requires to operate on. Further, we introduce

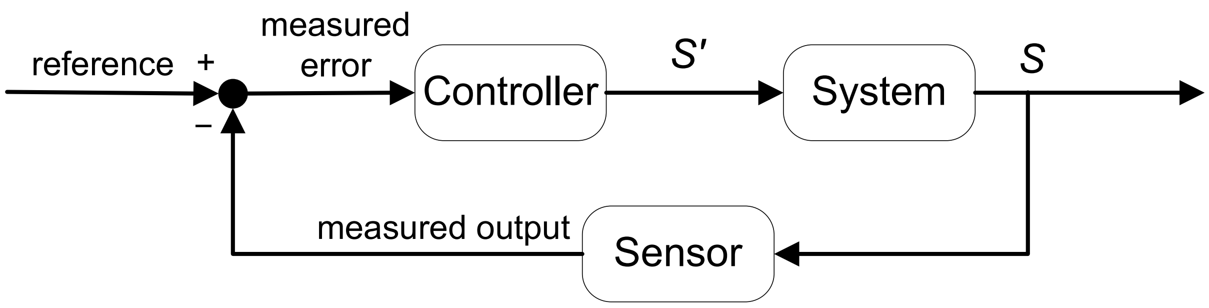


Figure 3.1: Feedback Control Systems

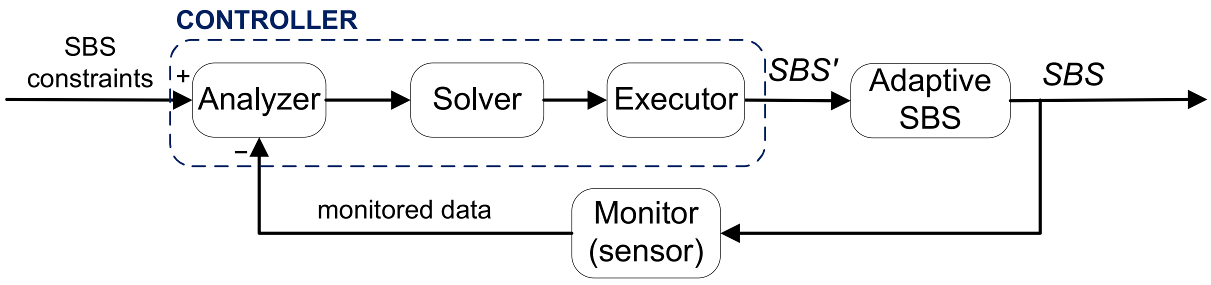


Figure 3.2: Adaptive Service-Based Systems - the Traditional Approach

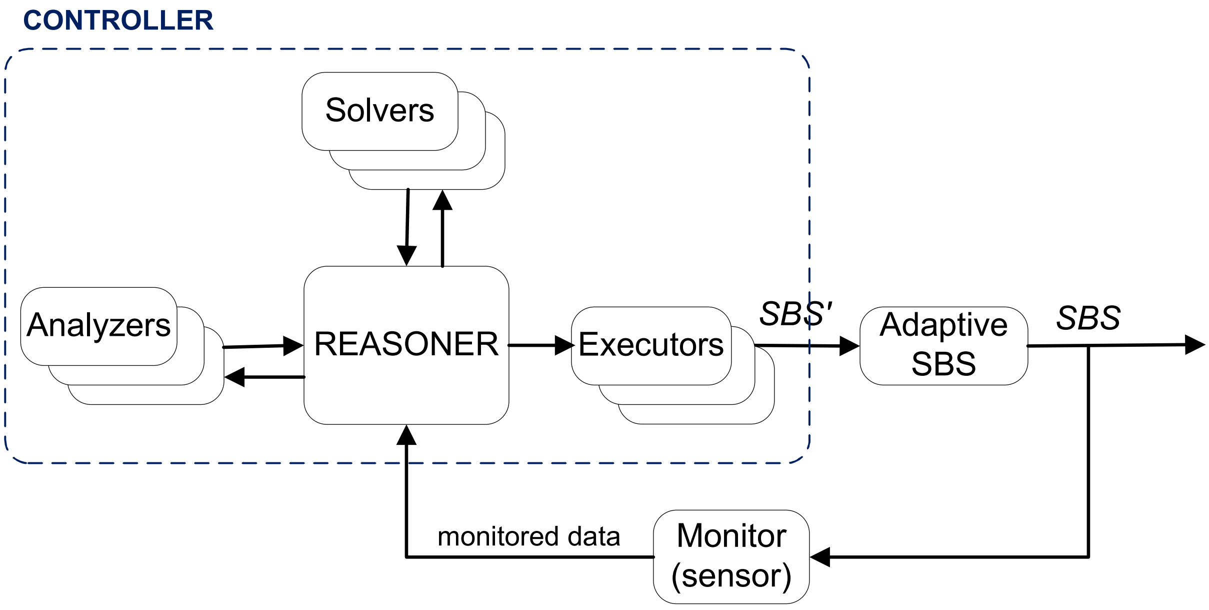


Figure 3.3: Adaptive Service-Based Systems - the CLAM Approach

the concepts concerning adaptiveness, namely, *adaptation need* and *adaptation action*, and relate these concepts to the system capabilities, i.e., *analyzers* and *solvers*.

3.2 Reference Scenario

Before moving forward with formal definitions, here, we present a reference scenario, a service-based system, which will be used in this chapter to exemplify the concepts and definitions. Moreover, we will benefit from the same scenario throughout the whole thesis to demonstrate our solution approach as well as the produced results.

Our scenario, "Call & Pay Taxi", is an SBS composed of the following layers: *application layer* where the application is implemented as a business process, run and monitored with respect to the key performance indicators (KPI) of interest, *service layer*, which corresponds to the partner services of the process catered by different service providers, and finally the underlying *infrastructure layer* for the composite service (application) and the partner services Figure 3.4).

The layers of our scenario comprise the following elements:

- *Application layer:* (i) "Call & Pay Taxi" composite service (CPTS), implemented as a BPEL process, (ii) application KPIs "process execution time" and "application cost".

- *Service layer:* (i) a short messaging service (SMS), a location service (LS) and a payment service (PS) provided by the telecom company, and the taxi service (TS) provided by the taxi company, (ii) service quality attributes (QoS) "execution time" and "cost".

- *Infrastructure layer:* (i) The underlying platforms on top of which CPTS, SMS, LS, PS and TS run. E.g., workflow engines, the application servers, hardware resources, (ii) infrastructure quality attributes “response time” and “cost”.

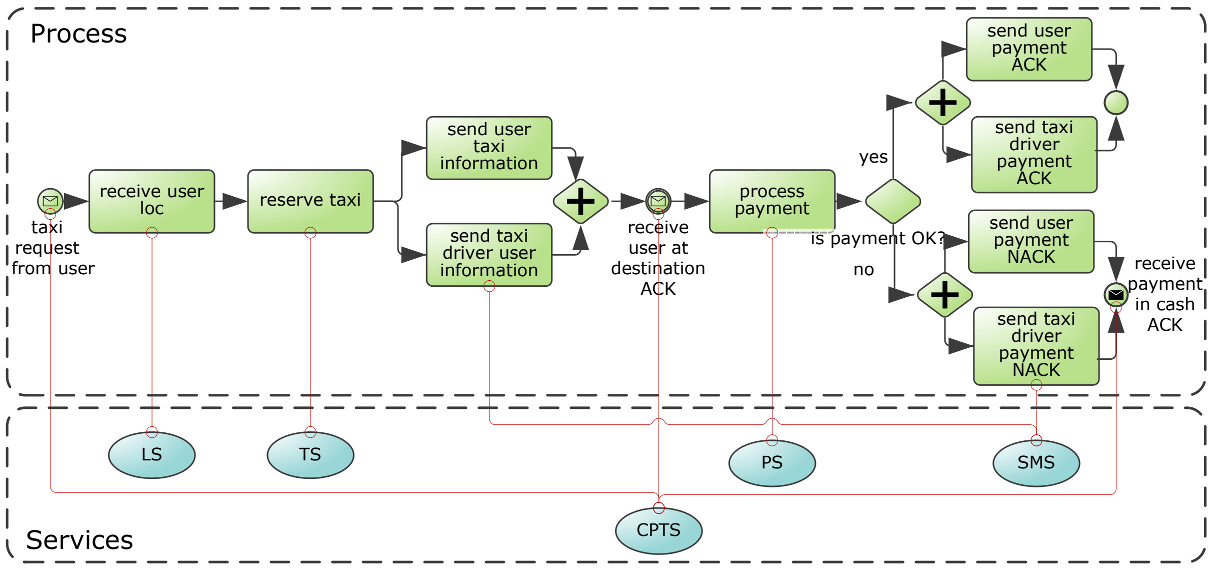

Process
taxi request from user
receive user loc
reserve taxi
send user taxi information
send taxi driver user information
receive user at destination ACK
process payment
is payment OK?
yes
no
send user payment ACK
send taxi driver payment ACK
send user payment NACK
send taxi driver payment NACK
receive payment in cash ACK
LS
TS
PS
SMS
CPTS
Services

Figure 3.4: Adaptive Service-Based Systems - the CLAM Approach

In CPTS, the client requests a taxi by sending a text message (SMS) to the application. Then, her location is identified and the taxi company is contacted to organize the pick-up service. After transportation to the destination, the process terminates upon a successful payment.

The application is maintained by various system capabilities to identify problems in the system and to provide the solutions, adaptations to those identified problems: To identify KPI violations we have the *time and cost analyzers*. To check the data flow constraints of the process we have the *data flow analyzer*. All these analyzers utilize some system data to check the relevant constraints and report violations. Second, we have a set of adaptation capabilities, i.e., solvers, to tackle the problems produced by analyzers. At application level we have the *KPI relaxer* enabling the modification of the target values of process execution time and application cost KPIs, *process optimizer* to perform possible parallelizations in the process, and the *data mismatch solver* to address the data incompatibilities of new partner services. At service level we have the *QoS* *negotiator* to negotiate the execution time and cost of services and the QoS-based *service replacer*. At infrastructure level we have the *resource allocator* to optimize the infrastructure provision.

We remark that we will explain analyzers and solvers in more detail later in Section 3.4.2.

3.3 Formal Model of Service-Based System

The formal model we use as a basis for defining the service-based system consists of three concepts, namely the *system model*, the *system configuration* and the *system layers*. While the system model provides a formalization of the main system elements and the interdependencies of those elements for a family of service-based systems sharing similar aspects, the system configuration represents a specific realization of an SBS from the family. On the other hand, system layers help perceive the SBS constitution at a high level of abstraction.

3.3.1 System Model

There is a large variety of service-based systems, which differ for the covered layers and for the way these layers are structured, as well as for the capabilities that can be exploited for adaptation. For this reason, our first step is to define a *system model* as a definition of a family of service-based systems that share the same structure. More precisely, a system model is defined by a graph, which nodes define the different types of elements that are present in the service-based system, and which edges define the relations among these elements.

Definition 3.1 *(System Model) A* system model *is a graph $\mathcal{M} = \langle N_{\mathcal{M}}, R_{\mathcal{M}} \rangle$, where:*

- *$N_{\mathcal{M}}$ is a finite set of types of admissible elements of the system;*

- $R_{\mathcal{M}}$, *where* $r \subseteq N_{\mathcal{M}} \times N_{\mathcal{M}}$ *for each* $r \in R_{\mathcal{M}}$, *is the set of possible relations among the element types.*

Notice that each edge r can connect different pairs of nodes, i.e., relations are polymorphic for what concerns the elements they connect. Whenever $\langle n_1, n_2 \rangle \in r$, we write $r(n_1, n_2)$.

Example 3.1 *Figure 3.5 represents the system model for our "Call and Pay Taxi" application. We can see each element type as a node of the graph. We have various element types from different parts of the system. For example at the application level we have the process, which implements the application, its constituent process activities, and the corresponding KPIs of the application. Then, we have the partner services, which our application relies on, together with their quality attributes. Finally, we have the elements that belong to the underlying infrastructures of the SBS. On the other hand, for our scenario, we have three types of relations represented by the graph edges: "has", "constrains" and "consumes". To illustrate:*

- *A process is composed of process activities, so, we relate the element type "process" to the element type "process activity" through the relation type "has".*
- *KPIs impose constraints on the application to enable the fulfilment of the business goals, thus, we have the "constrains" relation type between the "time KPI", "cost KPI" element types and the "process" element type.*
- *A process activity is realized by the invocation of a service operation. A service is implemented on top of an infrastructure, so, whenever we have this type of dependency, we relate two element types with the "consumes" relation type.*

We remark that $N_{\mathcal{M}}$ is a finite set of element types since the designer determines them when he creates the model.

3.3.2 System Configuration

While a system model defines a whole family of service-based systems, a *system configuration* defines a specific service-based system, which is deployed and running or which is ready to deploy and run. The system configuration is defined in terms of its elements and the existing relations among them. Formally, a system configuration is modeled as a graph which nodes –representing the elements of the SBS– are typed on the nodes of the system model, and which edges –representing the relations among these elements– are typed on the edges of the system model.

Definition 3.2 *(System Configuration) A* system configuration *of a system model $\mathcal{M} = \langle N_\mathcal{M}, R_\mathcal{M}\rangle$ is a typed graph $\mathcal{SC} = \langle N, T_N, R, T_R\rangle$, where:*

- *N is the set of elements of the system configuration;*
- *$T_N : N \to N_\mathcal{M}$ associates to each element $n \in N$ an element type $T_N(n) \in N_\mathcal{M}$;*
- *$R \subseteq N \times N$ is the set of relations among the element of the system configuration;*
- *$T_R : R \to R_\mathcal{M}$ associates to each relation $r = \langle n_1, n_2\rangle \in R$ a relation type $T_R(r) \in R_\mathcal{M}$ such that $\langle T_N(n_1), T_N(n_1)\rangle \in T_R(r)$.*

Example 3.2 *Figure 3.6 represents a system configuration for our "Call and Pay Taxi" application. We use the notation $n : T_N(n)$ for the nodes of the graph. As it can be seen from the graph, in some cases we have a single node for a given element type, but in some other cases we have multiple nodes having the same element type. For example, for the "process" element type, we have the "taxiBPEL" node. On the other side,"taxiBPEL" "has" multiple nodes having the "process activity"element type: "getTaxiReq", ..., "getPayment".*

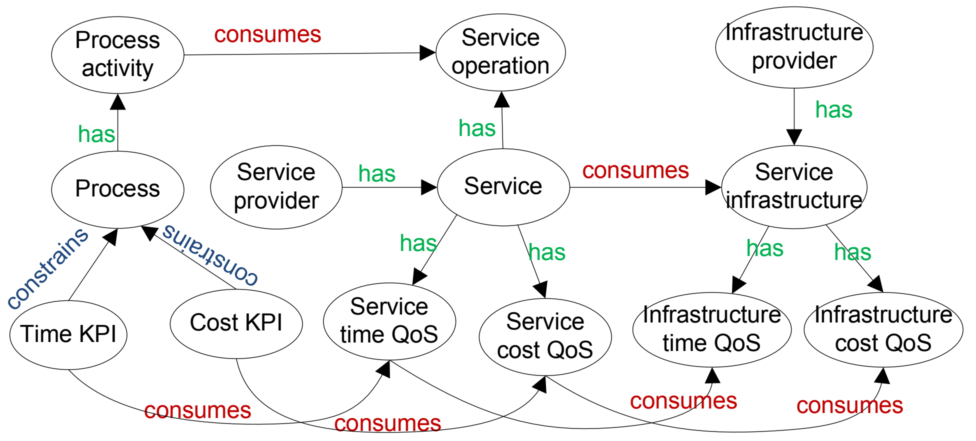


Figure 3.5: Example of System Model

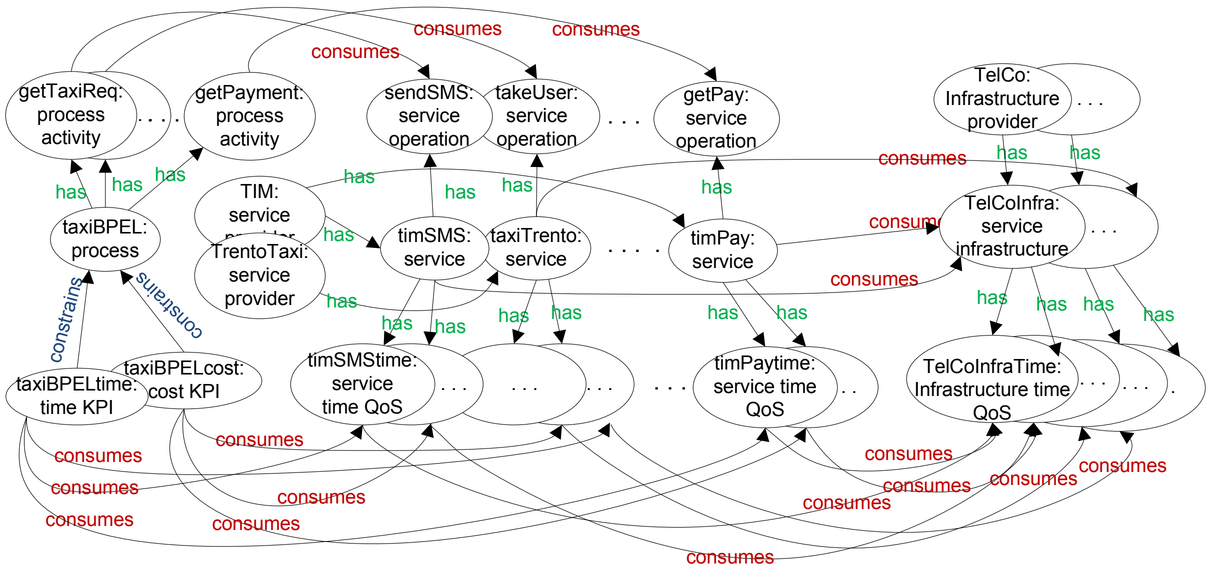

consumes
consumes
consumes
getTaxiReq: process activity
getPayment: process activity
sendSMS: service operation
takeUser: service operation
getPay: service operation
TelCo: Infrastructure provider
has
has
has
TIM: service provider
TrentoTaxi: service provider
taxiBPEL: process
timSMS: service
taxiTrento: service
timPay: service
TelCoInfra: service infrastructure
consumes
consumes
consumes
constrains
constrains
taxiBPELtime: time KPI
taxiBPELcost: cost KPI
timSMStime: service time QoS
timPaytime: service time QoS
TelCoInfraTime: Infrastructure time QoS
consumes
consumes
consumes
consumes
consumes
consumes
consumes
consumes
consumes
consumes
consumes

Figure 3.6: Example of System Configuration

We remark that the nodes of the system configuration keep references to the real data relevant for them. For instance, while a process node keeps a pointer to the process file such as a BPEL file, the service node keeps a pointer to its functional description such as a WSDL file. Similarly, a service quality node may keep a reference to its SLA agreement or to its online monitor and so on.

3.3.3 System Layers

Multi-layer systems are systems in which components are grouped, i.e., layered, usually in a hierarchical arrangement such that lower layers provide functions that support the functions of higher layers. SBSs are multi-layer systems by nature when their application workflows, constituent services, underlying platforms and infrastructures are considered.

One of the main characteristics of such layers is that, they are designed and developed in isolation by different experts, having different levels of abstraction with different technical aspects. While they are independently created layers, they become dependent on each other when they form the multi-layer system and need to work altogether. Let us consider an SBS with process, service, platform, operating system and hardware layers. We can easily see the reliant relations among them. E.g., the process consumes services and runs on top of a platform, and similarly platforms are a kind of middleware built on top of operating systems while operating systems are installed on computer hardware.

Note that the fundamental contribution of having a layered structure is that it helps understand the maintenance facilities of the system since usually each layer offers its own monitoring and adaptation capabilities. At this point, it gets clear that there is no concrete, unique set of layers, which describes all possible service-based systems. Depending on the application domain and what kind of adaptation capabilities the system has, different layers can be of consideration. For instance, if the operating system and the hardware do not contribute to the adaptiveness, i.e., they are not monitored and not adaptable either, then we do not need to have them as a separate layer.

After discussing the layer concept, now we are ready to give its formal definition:

We remark that, by exploiting the system model defined in Figure 3.5, we can formally capture the concept of *layer* as a set of nodes connected through "has" and "constrains" relations: these relations, indeed, serve the purpose to connect elements in the same layer, while "consumes" relations link elements belonging to different layers.

Definition 3.3 *(Layer) A* layer *in a system model $\mathcal{M} = \langle N_{\mathcal{M}}, R_{\mathcal{M}} \rangle$ is a set of nodes $\mathcal{L}_{\mathcal{M}} \subseteq N_{\mathcal{M}}$ that satisfies the following two conditions:*

- ***closed:*** *assume $n \in \mathcal{L}_{\mathcal{M}}$ and one of the following relations holds* has(n, n'), has(n', n), constrains(n, n'), constrains(n', n)*; then $n' \in \mathcal{L}_{\mathcal{M}}$;*
- ***connected:*** *if $\mathcal{L}'_{\mathcal{M}} \subseteq \mathcal{L}_{\mathcal{M}}$ and $\mathcal{L}'_{\mathcal{M}}$ also satisfies the previous condition "close", then either $\mathcal{L}'_{\mathcal{M}} = \emptyset$ or $\mathcal{L}'_{\mathcal{M}} = \mathcal{L}_{\mathcal{M}}$.*

Example 3.3 *E.g. The layers of "Call and Pay Taxi" application are as follows:*

- *application layer = {process, process activity, cost KPI, time KPI}*
- *service layer = {service, service operation, service provider, cost SQoS, time SQoS}*
- *infrastructure layer = {infrastructure, infrastructure provider, cost IQoS, time IQoS}*

Note that the given definition of layer can be applied also to system configurations. We remark that, in a configuration, more "layers" can correspond to the same system model "layer", in case there are different unconnected sets of elements corresponding to a layer in the system model (e.g., two unconnected infrastructures).

We also point out that while one can specify various types of relations among system elements, “has” and “constrains” are special type of relations which enable to formally define the layers. It is the domain expert’s responsibility to identify the principal element of a layer and subsequently to comprehend which main aspects this element “has” and which other elements impose requirements on this element, i.e. “constrains” the element. Let us consider a “service layer”. Obviously the main concept at this layer is the service. At this point, the expert should identify what service has as the main aspects and whether there are constraints imposed on the element. The relations identified should be semantically meaningful. For instance, a service can have quality attributes, but it cannot have a key performance indicator because key performance indicator is a concept related to the processes, not services.

3.4 Formal Model of Adaptation

The system model presented in the previous section represents an SBS in stable conditions, i.e., given the system does not need to be adapted, the definitions we have made so far fulfill the requirements of SBS representation. However, as we discussed in the beginning of this chapter, SBSs are exposed to many changes considering their surrounding context as well as their dynamic system parts. Hence, we have to extend the presented system model to capture the adaptiveness of service-based systems in the model.

First, let us clarify the “adaptation need” and the “adaptation action” concepts: when a system constraint is checked by an analysis mechanism and a problem is identified, we can exploit the relevant adaptation mechanisms available in the system to address this problem. Every possible way to solve the problem corresponds to an adaptation need. Hence, adaptation mechanisms, tackling specific problems, work on adaptation needs. What those mechanisms produces as concrete solutions to the problem correspond to adaptation actions.

We remind that in our approach we call every analysis mechanism of the SBS an “analyzer”, and its every adaptation capability a “solver”. At the same time, when we define the existing analysis mechanisms as “analyzers”, as an additional step we extend them by associating the adaptation needs with their outputs. Moreover, we introduce a third new type of tool to utilize in our approach: the “enactor”.

Now, let us explain how we model adaptation with these tools. The approach we follow, illustrated in Figure 3.7, is based on the system configuration graph and extends it with additional nodes representing identified adaptation needs and adaptation actions. More precisely, whenever an analyzer identifies an adaptation need, this is added to a system configuration as a new node, which is connected to the system elements that are associated to the need. Similarly, whenever a solver identifies an adaptation action that solves a given need, it updates the graph by replacing the need node with an action node, which is connected to the elements of the graph that are affected by the action. Finally, when an adaptation action is executed by an enactor, the corresponding node is removed, and the part of the graph connected to it is updated according to the effects of the adaptation actions.

Example 3.4 *Let us consider the adaptation example in Figure 3.7. At first, we have the initial subgraph of the system configuration, which consists of cost QoSs of the partner services of the application. Then, the “cost analyzer” triggers the adaptation need “negotiate service cost”, associated to this subgraph. Next, the need is passed to the solver “QoS negotiator”. Negotiator solves the need by the adaptation action “new service costs”, which proposes newly negotiated cost values for “timSMS”, “timLoc” and “timPay” services, and the graph is updated accordingly, i.e., the need is removed, the action is inserted. At final step, the corresponding enactor gets the action and applies it to the graph.*

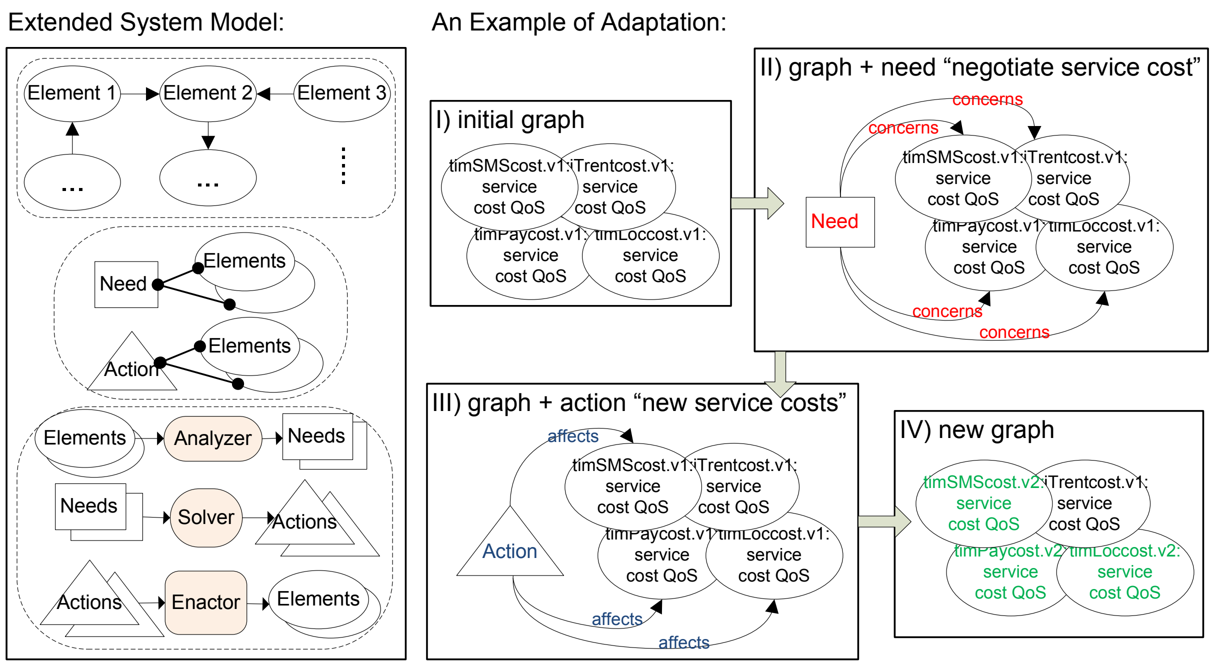

Extended System Model:
An Example of Adaptation:
Element 1
Element 2
Element 3
...
Need
Elements
Action
Analyzer
Needs
Solver
Actions
Enactor
I) initial graph
II) graph + need "negotiate service cost"
III) graph + action "new service costs"
IV) new graph
timSMScost.v1: service cost QoS
iTrentcost.v1: service cost QoS
timPaycost.v1: service cost QoS
timLoccost.v1: service cost QoS
timSMScost.v2: service cost QoS
timPaycost.v2: service cost QoS
timLoccost.v2: service cost QoS
concerns
affects

Figure 3.7: Overall Approach to Modeling Adaptation

3.4.1 Extended System Configuration

Definition 3.4 *(Extended System Model and Configuration)*
An extended system model *for system model $\langle N_{\mathcal{M}}, R_{\mathcal{M}} \rangle$ is a graph $\mathcal{EM} = \langle N_{\mathcal{M}} \cup D_{\mathcal{M}} \cup A_{\mathcal{M}}, R_{\mathcal{M}} \cup \{\text{concerns}, \text{affects}\} \rangle$, where:*

- *$D_{\mathcal{M}}$ is the finite set of need types and $A_{\mathcal{M}}$ is the finite set of action types; $N_{\mathcal{M}}$, $D_{\mathcal{M}}$ and $A_{\mathcal{M}}$ are disjoint;*

- concerns $\subseteq D_{\mathcal{M}} \times N_{\mathcal{M}}$ *and* affects $\subseteq A_{\mathcal{M}} \times N_{\mathcal{M}}$ *relate each need/action type to a set of associated nodes in the system model; for each $d \in D_{\mathcal{M}}$ (resp. $a \in A_{\mathcal{M}}$) there is at least one $n \in N_{\mathcal{M}}$ such that* concerns(d, n) *(resp.* affects(a, n)*).*

An extended system configuration *for extended system model $\mathcal{EM} = \langle N_{\mathcal{M}} \cup D_{\mathcal{M}} \cup A_{\mathcal{M}}, R_{\mathcal{M}} \cup \{\text{concerns}, \text{affects}\} \rangle$ is a system configuration $\mathcal{ESC} = \langle N \cup D \cup A, T_N, R \cup \{\text{concerns}, \text{affects}\}, T_R \rangle$ for $\mathcal{EM}$, according to Def. 3.2.*

Notice that having the system with finite adaptation and analysis capabilities, $D_{\mathcal{M}}$ and $A_{\mathcal{M}}$ are finite sets designed by the modeler.

Example 3.5 *In the adaptation example in Figure 3.7, while the initial graph and the final graph correspond to a subgraph of a "non extended $\mathcal{SC}$", the others, the second and the third, are examples illustrating (partially) an $\mathcal{ESC}$.*

Here, we would like to recall our reasoner that we have introduced in Figure 3.3. Now, having defined the extended system configuration, we can explain how our reasoner is supposed to work at a conceptual level, that is depicted in Figure 3.8. Suppose that we have a non-extended system configuration, i.e., $\mathcal{SC}$, which implies that it is a stable instance of the SBS and there is no problem in the system, neither an adaptation need d nor an adaptation action a triggered. Then whenever we observe a problem in the system, it necessitates a new need d, which in turn puts the system into an extended system configuration $\mathcal{ESC}$. To address the d in the new $\mathcal{ESC}$, we can apply an adaptation, which removes the d, and adds an a. This new action might cause a new problem, which introduces a new need to the $\mathcal{ESC}$. The system can stay in this loop for a while until the moment it arrives at a stable point again, i.e., no more problem is triggered and the system is back to a non-extended configuration. The responsibility of the reasoner is to coordinate those loops and to bring back the system into a non-extended configuration state whenever it falls into an extended one.

We remark that those action-need loops reflect the non-deployed $\mathcal{ESC}$'s. More precisely, once the system gets an initial problem in the running system and falls into an $\mathcal{ESC}$, we do not modify the running instance until the reasoner finds out a new non-extended, stable configuration, which is safe to deploy in the system.

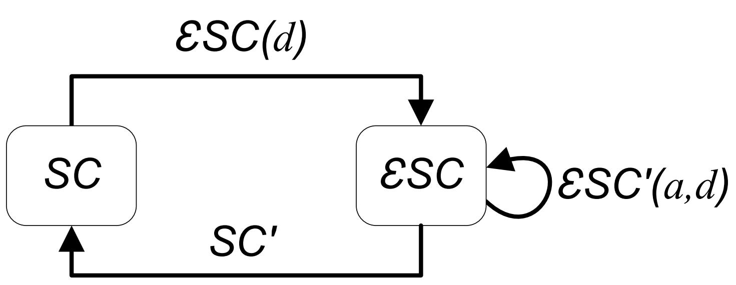


Figure 3.8: Conceptual Model of CLAM's Reasoner

3.4.2 Tools

We are now ready to understand how the $\mathcal{ESC}$ transformations in Figure 3.8 happen. The three kinds of tools which we have introduced in the beginning of this section, namely analyzers, solvers and enactors perform these transformations. Indeed, they are defined in terms of the transformations that they apply to an (extended) system configuration. More precisely:

- an *analyzer* is a tool that adds a need node to a system configuration – or leaves it unaffected, if no adaptation need is present;
- a *solver* is a tool that takes in input a system configuration containing a node d, removes it and adds an action node – or leaves it unaffected, if the solver is not able to solve the adaptation need;
- an *enactor* is a tool that takes in input a system configuration containing a node a, removes it, and possibly restructures the graph to reflect the changes implemented by the enactor.

Definition 3.5 *(Tools) An* adaptation tool $\mathcal{T}$ *is a nondeterministic function on extended system configurations such that, if* $\mathcal{ESC}' \in \mathcal{T}(\mathcal{ESC})$ *then* $\mathcal{ESC}'$ *satisfies one of the following conditions:*

- ***Analyzer:*** *either* $\mathcal{ESC}' = \mathcal{ESC}$ *or* $D = D' \setminus \{d\}$, $\mathrm{concerns}' \cap (D \times N) = \mathrm{concerns}$*, the other nodes and edges in* $\mathcal{ESC}'$ *are equal to those in* $\mathcal{ESC}$*;*

- ***Solver:*** *either* $\mathcal{ESC}' = \mathcal{ESC}$ *or* $D' = D \setminus \{d\}$*,* $A = A' \setminus \{a\}$*,* $\text{concerns}' = \text{concerns} \cap (D' \times N)$*,* $\text{affects}' \cap (A \times N) = \text{affects}$*, the other nodes and edges in* $\mathcal{ESC}'$ *are equal to those in* $\mathcal{ESC}$*;*

- ***Enactor:*** $A' = A \setminus \{a\}$*,* $\text{affects}' = \text{affects} \cap (A' \times N)$*,* N' *and* R' *are (arbitrary) variants of* N *and* R*, the other nodes and edges in* $\mathcal{ESC}'$ *are equal to those in* $\mathcal{ESC}$*.*

In the following, we assume to have three disjoint sets of tools $\mathcal{T}\text{ools}_A$ (analyzers), $\mathcal{T}\text{ools}_S$ (solvers) and $\mathcal{T}\text{ools}_E$ (enactors), and we will define $\mathcal{T}\text{ools}$ as $\mathcal{T}\text{ools}_A \cup \mathcal{T}\text{ools}_S \cup \mathcal{T}\text{ools}_E$.

Example 3.6 *Let us recall the adaptation example in Figure 3.7. As we mentioned previously, the "cost analyzer" triggers the adaptation need "negotiate service cost", and afterwards, the solver "QoS negotiator" triggers the action "new service costs". Here, the* $\mathcal{T}$ *definition applies as follows:*

- ***Analyzer:*** $D = \varnothing$*,* $D' = \{d_1\}$*,* $d_1 = negotiateServiceCost$*,* $\text{concerns} = \varnothing$*,* $\text{concerns}' = \{\text{concerns}(d_1, timSMScost.v1), \text{concerns}(d_1, timPaycost.v1), \text{concerns}(d_1, timLoccost.v1), \text{concerns}(d_1, taxiTrentocost.v1)\}$*,*

- ***Solver:*** $D = \{d_1\}$*,* $D' = \varnothing$*,* $A = \varnothing$*,* $A' = \{a_1\}$*,* $a_1 = newServiceCosts$*,* $\text{concerns} = \{\text{concerns}(d_1, timSMScost.v1), \text{concerns}(d_1, timPaycost.v1), \text{concerns}(d_1, timLoccost.v1), \text{concerns}(d_1, taxiTrentocost.v1)\}$*,* $\text{concerns}' = \varnothing$*,* $\text{affects} = \varnothing$*,* $\text{affects}' = \{\text{affects}(a_1, timSMScost.v1), \text{affects}(a_1, timPaycost.v1), \text{affects}(a_1, timLoccost.v1)\}$

- ***Enactor:*** $A = \{a_1\}$*,* $A' = \varnothing$*,* $\text{affects} = \{\text{affects}(a_1, timSMScost.v1), \text{affects}(a_1, timPaycost.v1), \text{affects}(a_1, timLoccost.v1)\}$*,* $\text{affects}' = \varnothing$*,*
$\{timSMScost.v1, timPaycost.v1, timLoccost.v1\} \in N$*,*
$\{timSMScost.v1, timPaycost.v1, timLoccost.v1\} \notin N'$*,*
$\{timSMScost.v2, timPaycost.v2, timLoccost.v2\} \notin N$*,*
$\{timSMScost.v2, timPaycost.v2, timLoccost.v2\} \in N'$

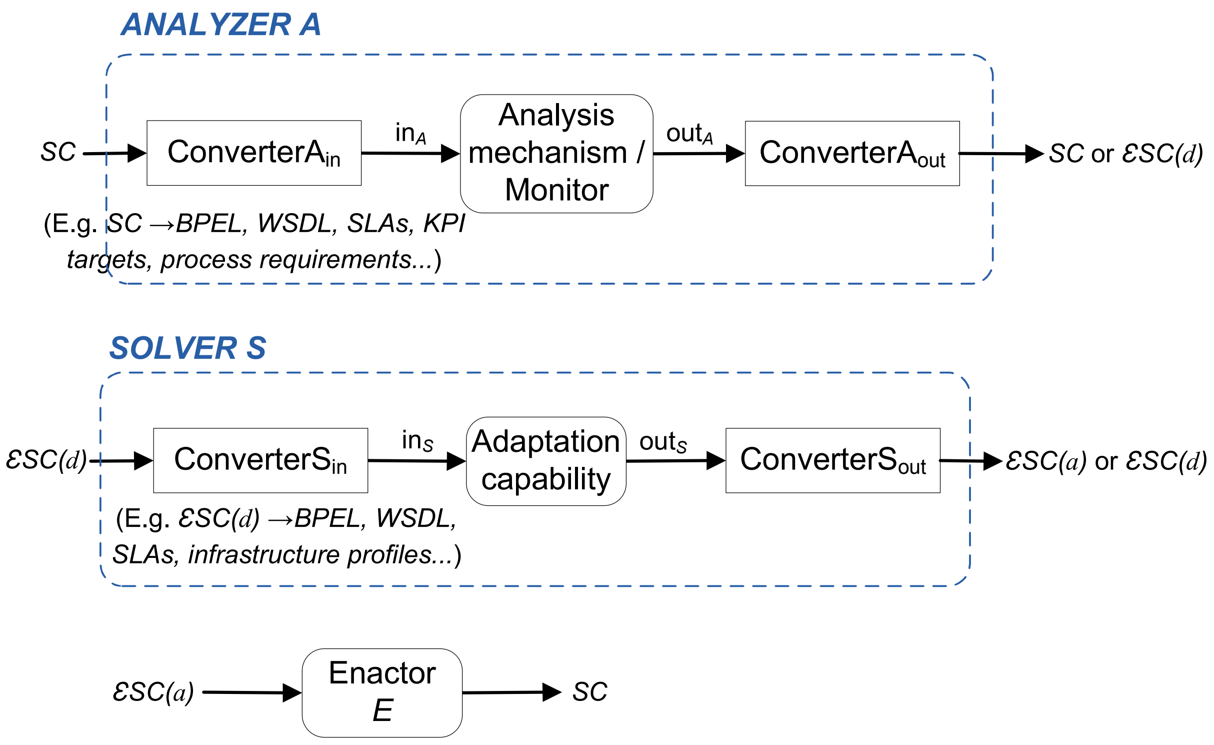


Figure 3.9: Adaptation Tools of the Formal Framework

We expect that tools work on specific elements of an extended system configuration. In particular, we require that solvers are specialized on needs d of a specific type and, similarly enactors are specialized on actions a of a specific type. Instead, analyzers work on a subset of system nodes, in other words, for each system node n in the $\mathcal{ESC}$, there exists a set of analyzers associated with it. We recall that analyzers and solvers correspond to the real adaptation and analysis mechanisms existing in current works, whereas enactors are internal to our framework (Figure 3.9).

The analyzers (resp. solvers) have nondeterministic behavior because in some cases they might be producing alternative needs d (resp. actions a). E.g. when a cost analyzer identifies that cost KPI is violated, it can report that we need to either relax the KPI target value or re-negotiate service costs or replace some existing services with cheaper ones. Or a service replacer solver can identify a set of alternative services from different providers having the

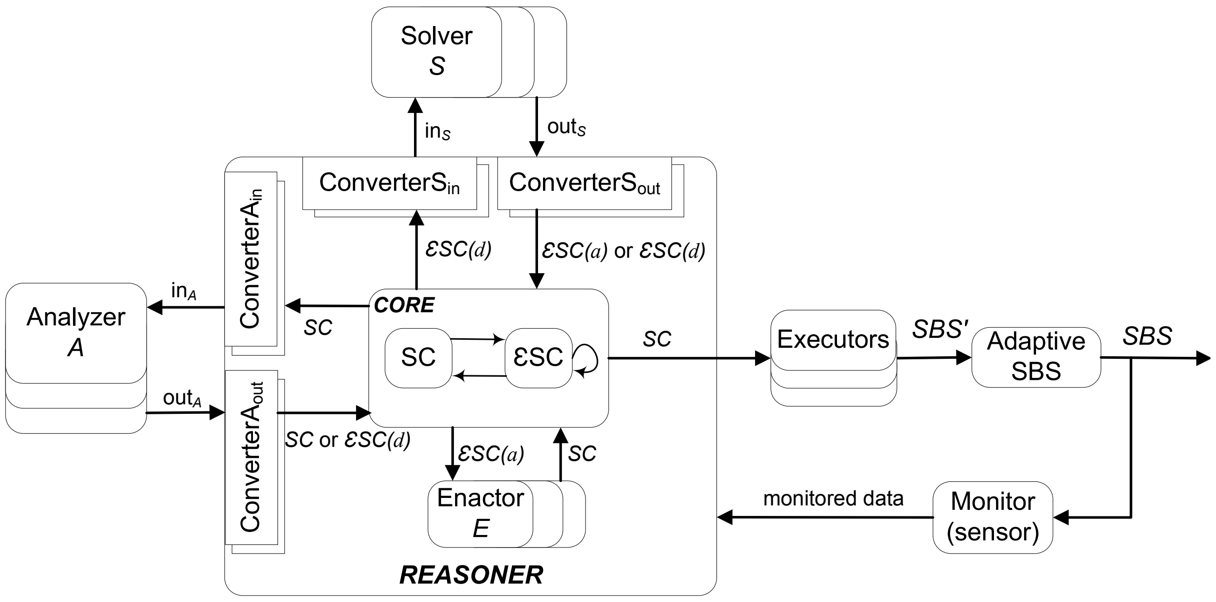


Figure 3.10: The Overall Framework

same functionality.

3.5 The Framework

Now, having defined all the required components, we can elaborate on our reasoner, which we have introduced in Figure 3.3, and present the overall framework. The architecture of the framework is depicted in Figure 3.10. Its main constituents are the reasoner, analyzers, solvers, executors and the monitor. The reasoner, one of the principal contributions of this dissertation, comprises the core, converters and enactors. Given these components, let us see below how the framework is expected to work.

3.5.1 The initial trigger

The execution of the service-based application and its context, i.e., the overall SBS, are continuously observed through the monitoring mechanisms available in the system. Whenever a *monitor* signals a problem to the reasoner, *the rea-*

soner should start its operation.

3.5.2 Coordination and decision of the adaptation strategy

Whenever the reasoner receives a signal from a monitor, it is considered as a new problem to be addressed by an adaptation strategy. *The core* is responsible for building up the strategy through an iterative coordination of the tools $\mathcal{T}$ools, and by keeping track of the modifications made to the (extended) system configuration by the tools. The first thing to do is to identify the needs that correspond to the problem signaled by the monitor. We remark that monitors are in the set of analyzers $\mathcal{T}\text{ools}_A$, which means by Def. 3.5 the core knows the corresponding needs d for this monitor. Once the needs are identified, the core can initiate its coordination starting from the solvers, which can tackle those needs. The actions a, proposed by the *solvers* and executed by *enactors*, should be validated by the proper *analyzers*. If the analyzers identify new needs, similarly they should be addressed by the proper solvers. A continuous invocation of tools should be performed until a non-extended system configuration is found without any need d and any action a. Notice that the core requires the *converters* in order to invoke analyzers and solvers as they are external tools, whereas this is not the case for enactors since they are internal to the reasoner.

3.5.3 Deployment of the adaptation strategy

Once the coordination is completed and an adaptation strategy is identified by the reasoner, the eventual step should be the invocation of *executors* to deploy the strategy in the running system and to bring the SBS to a new state. Notice that the executors are external components like analyzers and solvers: The framework should utilize the existing deployment mechanisms to apply the adaptation strategy to the running SBS.

3.6 Discussion

We presented a formal adaptation framework, which allows for the representation of the SBS and its adaptation and analysis mechanisms. In the next chapter, the proposed framework will aid us to define of cross-layer adaptation problem and to provide a basis for its solution.

A crucial contribution of our framework is that, differently from the numerous adaptation works, it proposes a coordination procedure before executing an adaptation in the running service-based system. This is a fundamental aspect that cannot be neglected when we consider the complexity of such systems and the interdependencies of their components. Moreover, the capability to accommodate diverse SBSs –with a diverse set of maintenance facilities– in the model brings a remarkable genericness and a distinct flexibility to the approach. The last but not least, the framework aims at aligning the existing monitors and analysis mechanisms, i.e. analyzers, and the adaptation capabilities, i.e., solvers, which run in isolation from each other. We remark that in real world we might have analyzers working on more than one problem, and similarly, solvers addressing more than one problem. However, in our approach we separate them functionally to enable their easy management by the reasoner. One example can be the work in [37], which proposes an aggregation mechanism to analyze the execution time, cost and reliability aspects of the process. This analysis tool corresponds to three different analyzers in our framework, i.e., the time, cost and reliability analyzers.

Modeling the service-based system for adaptiveness is proposed also by previous works [113, 21, 43, 46]. These approaches, in particular, targets a specific SBS and lacks the flexibility to allow SBSs with different system elements and layers. In [113], a method for change management is introduced for the business process model and the underlying service composition implemented in BPEL. However, the definition of change management is limited to the necessary changes to be done in BPEL upon the modification of the process. Like in our approach, [21] considers the system elements and their interdependencies in the SBS representation. The authors utilize goal and variability models to facilitate adaptiveness. Similarly, [46] proposes a variability framework to model the business process management for reusability and flexibility. However, the authors do not explain how one can decide and perform the adaptation benefiting from the proposed variability framework. Instead, [46] focuses on modeling environment-aware service compositions to enable self-adaptiveness, but similar to the approach in [46], it is not clear how the adaptation is achieved.

Chapter 4

Cross-layer Adaptation Manager

In this chapter, we formally describe cross-layer adaptation problem, present the algorithm that solves it and prove that the algorithm is complete and correct with respect to the definition of cross-layer adaptation problem. Moreover, we discuss the conditions under which the algorithm terminates.

4.1 Problem Statement

Given the framework presented in the previous chapter, we would like to coordinate the adaptation capabilities in such a way that the resultant strategy brings the system to a stable point in which none of the system analyzers produce any more problems.

We are now ready to give the formal definition of cross-layer adaptation problem.

4.1.1 Stable System Configuration

Before defining cross-layer adaptation problem, we should define the "stability" concept.

We define the *stable* system configurations as those configurations that do not imply any adaptation need.

Definition 4.1 *(Stable System Configuration) A (non-extended) system configuration $\mathcal{SC}$ is* stable *if, for each analyzer $\mathcal{T} \in \mathcal{T}ools_A$, $\mathcal{T}(\mathcal{SC}) = \mathcal{SC}$.*

4.1.2 Cross-layer Adaptation Problem

A cross-layer adaptation problem for a given initial system configuration is the problem of identifying a sequence of tools that is able to transform the initial system configuration into a stable system configuration.

Definition 4.2 *(Cross-layer Adaptation) A cross-layer adaptation for system configuration $\mathcal{SC}_0$ is a sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$, such that:*

- $\mathcal{SC}_i \in \mathcal{T}_i(\mathcal{SC}_{i-1})$, *for each* $i \in \{1, 2, \ldots, n\}$;
- $\mathcal{SC}_{i-1} \neq \mathcal{SC}_i$, *for each* $i \in \{1, 2, \ldots, n\}$ *(i.e., all tools contribute to the solution);*
- $\mathcal{SC}_n$ *is stable.*

4.2 Cross-layer Adaptation Approach

We propose a tree-based solution to the cross-layer adaptation problem in Definition 4.2, which the initial idea originated in the works [134, 136, 135, 133]. More precisely, there should be a cross-layer adaptation solution, which can be extracted from a tree produced by a cross-layer adaptation manager (CLAM).

A CLAM Tree is a connected graph without cycles. Nodes of the tree keep a system configuration $\mathcal{SC}$ and an ordered set of tools $\mathcal{T}$, which we call a queue. The queue elements correspond to the necessary tools that we need to invoke for the given system configuration in the same tree node. Subsequently, edges keep information about the invocation of tools, thus we label them with the first element of the queue in the parent node.

When a tool invocation happens, depending on the response of the tool, the system configuration may remain unchanged or may be updated by a need d or an action a. We distinguish one case from the other. If the node transition, i.e., a tool invocation, ends up in the same system configuration, we call the edge a *useless edge*, otherwise it is a *useful edge*.

Definition 4.3 *(CLAM Tree) A* CLAM Tree *is a tree* $\mathsf{T} = \langle V, E, T_E, rt, L\rangle$*, where:*

- V *is the set of nodes of the CLAM tree where node* $v \in V$ *is a tuple* $\langle \mathcal{SC}, Q\rangle$*, queue* Q *is an ordered set of tools* $\mathcal{T}$ *in which a solver has always a priority over an analyzer,* $\mathcal{SC}$ *is a system configuration;*
- $E \subseteq V \times V$ *is the set of edges among the tree nodes such that* $E = E_{useful} \cup E_{useless}$*, where:*
  - *an* $e_i \in E_{useful}$ *is a useful edge iff* $e_i(v_{i-1}, v_i)$ *such that* $\mathcal{SC}_{v_{i-1}} \neq \mathcal{SC}_{v_i}$*;*
  - *an* $e_j \in E_{useless}$ *is a useless edge iff* $e_j(v_{j-1}, v_j)$ *such that* $\mathcal{SC}_{v_{j-1}} = \mathcal{SC}_{v_j}$*;*
  - $E_{useful} \cap E_{useless} = \varnothing$*;*
- $T_E : E \rightarrow \mathcal{T}ools$ *associates to each edge* $e \in E$ *an edge label* $T_E(e) \in \mathcal{T}ools$*, where there exists an edge* $e_i(v_{i-1}, v_i)$ *iff:*
  - *there exists the tool* $\mathcal{T}_i = T_E(e_i)$ *such that* $\mathcal{T}_i(\mathcal{SC}_{v_{i-1}}) = \mathcal{SC}_{v_i}$ *and* $\mathcal{T}_i$ *is the first tool of the queue* Q_{v_i}*, and*
  - $\mathcal{SC}_{v_i} \neq \mathcal{SC}_{v_{i-1}}$ *if* $\mathcal{T}_i$ *is a solver such that* $\mathcal{T}_i \in \mathcal{T}ools_S$*;*
- rt *is the root node of the CLAM tree;*
- $L \subseteq V$ *is the set of leaves of the CLAM tree.*

Notice that every path from the root node to a leaf of the CLAM Tree T corresponds to a sequence of tools which are applied to an initial system configuration one after the other such that the sequence brings the system to a final configuration. Here, we can define a collapsed tree path in which we ignore the useless edges since they do not contribute to the transformation of the system configuration.

Definition 4.4 *(Tree Paths and Collapsed Tree Paths)* Tree paths *P is the set of paths in a CLAM Tree T where the path $p \in P$ is a sequence of tools $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ for a sequence of tree nodes $v_0, v_1, \ldots, v_k$ such that $v_0 = rt$, $v_k \in L$, (v_{i-1}, v_i) is an edge e_i for $i = 1, 2, \ldots, k$, $T_E(e_i) = \mathcal{T}_i$.*

Collapsed tree paths *P_c is the set of collapsed paths in a CLAM Tree T where a collapsed path $p_c \in P_c$ is a reduced form of a path $p \in P$ such that if for a $\mathcal{T}_i \in p$, $e_i = T_E^{-1}(\mathcal{T}_i)$ is a useless edge, then $\mathcal{T}_i \notin p_c$.*

Intuitively, we can notice that if a collapsed tree path p_c ends up in a leaf which has a non empty queue Q, it cannot be a solution to a cross-layer adaptation problem since some of the tools, which are identified to be necessary to invoke, remain in the queue.

Now we are ready to give the definition of a solution tree. A CLAM tree is a solution tree if and only if there exists at least one collapsed tree path that terminates in a leaf with an empty queue (Figure 4.1). Intuitively we expect that such path corresponds to a solution to a cross-layer adaptation problem since (i) all the necessary tools that we identify on the path are exhausted, and (ii) they all contribute to the solution, i.e., modify the system configuration.

Definition 4.5 *(Solution Tree) A CLAM Tree $\mathsf{T} = \langle V, E, T_E, rt, L \rangle$ is a* Solution Tree T_s *iff T has at least one leaf $l \in L$ such that $Q_l = \varnothing$.*

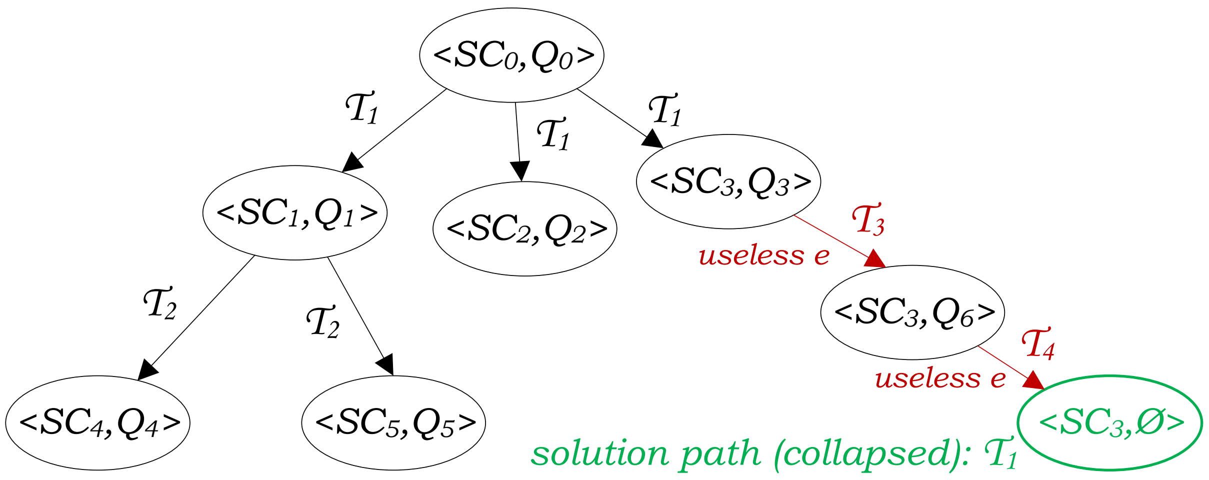


Figure 4.1: A Sample Solution Tree

4.2.1 CLAM Algorithm

We now present the "CLAM algorithm" that grounds on the cross-layer adaptation framework presented in Chapter 3 and realizes the reasoner of the framework, which is responsible for coordinating the tools integrated (Figure 3.10. Given an initial $\mathcal{SC}_0$, a set of tools $\mathcal{T}$ools and an *initial trigger* from a monitor, the algorithm implements the tree-based approach that we propose for the solution of cross-layer adaptation problem.

The algorithm exploits a tree data structure such that at the tree nodes we keep the statuses of system configuration $\mathcal{SC}$ and a queue $\mathcal{Q}$ of tools $\mathcal{T}$, and at the tree edges we keep the outputs, called "report", of the tool invocations. We remark that the queue serves as a means for keeping and continuously updating an ordered set of analyzers and solvers that we identify to be invoked. Notice that in a queue, a solver is always inserted in the queue head, which in turns implies that solvers have priority over analyzers and invoked first as soon as they are identified.

Figures 4.2 & 4.3 present the algorithm. To find the cross-layer adaptations, CLAM algorithm performs three key steps:

1. **Receiving the initial trigger.** First, we get the running system configura-

```
function main(SC, monTrigger)
  analyzer a := monTrigger.getAnalyzer()
  treeType tree := new tree()
  nodeType root := new node()
  root.setSC(SC)
  root.addQueue(a)
  root.setLeaf(true)
  tree.addNode(root)
  return expandTree(tree)

function expandTree(tree)
  List<node> leaves := tree.getLeafNodes()
  if (leaves.isEmpty()) || stopCondition(tree)
    return tree
  leaf := pickOneLeaf(leaves)
  if !leaf.getQueue().isEmpty()
    report := invokeTool(leaf.getSC(), leaf.getQueueHead())
    nodes_and_edge_pairs := expandNode(report, leaf)
    forall <node, edge> in nodes_and_edge_pairs
      if !(report.fromSolver() && node.isInTree())
        tree.addNode(node)
        tree.addEdge(leaf, edge, node)
  leaf.setLeaf(false)
  return expandTree(tree)

function expandNode(report, node)
  if node.getQueue().isEmpty()
    return empty_list
  switch report.getType()
    case OK:
      return expandForNextQueuedTool(report, node)
    case adapt:
      return expandAdaptations(report, node)
    case need:
      return expandNeeds(report, node)
    case NOK:
      return empty_list
```

Figure 4.2: CLAM algorithm - main functions.

```
function expandForNextQueuedTool(report, node)
  List<nodeType, edgeType> result := empty_list
  if node.getQueue().isEmpty()
    return empty_list
  new_n := node.clone()
  new_n.removeQueueHead()
  new_e := new edge(report.getType(), report.getTool())
  result.add(<new_n, new_e>)
  return result
```

```
function expandAdaptations(report, node)
  List<nodeType, edgeType> result := empty_list
  forall adapt in report.getAdapts()
    tmp_n := node.clone()
    tmp_n.getSC().attachAdaptToSC(adapt)
    tmp_n.getSC().applyAdaptToSC(adapt, adapt.getEnactor())
    List<analyzer> analyzers := adapt.getSystemNodes.getAllUniqueAnalyzers()
    tmp_n.removeQueueHead()
    forall analyzer in tmp_n.getQueue()
      analyzers.addUnique(analyzer)
    List<List<analyzer>> allPermutations := permute(analyzers)
    tmp_n.clearQueue()
    forall analyzer_list in allPermutations
      new_n := tmp_n.clone()
      new_n.addAllToQueue(analyzer_list)
      new_e := new edge(adapt, report.getTool())
      result.add(<new_n, new_e>)
  return result
```

```
function expandNeeds(report, node)
  List <nodeType, edgeType> result := empty_list
  List<solver> solvers := empty_list
  forall need in report.getNeeds()
    solvers.add(need.getSolvers())
  forall solver in solvers
    new_n := node.clone()
    new_n.removeQueueHead()
    new_n.getSC().attachNeedToSC(solver.getNeed())
    new_n.addToQueueHead(solver)
    new_e := new edge(solver.getNeed(), report.getTool())
    result.add(<new_n, new_e>)
  return result
```

Figure 4.3: CLAM algorithm - functions to create node and edge pairs.

tion and the initial trigger, that is, a problem signaled by a monitor. Then, we create a new queue instance, add the analyzer relevant for the monitored data to the queue and instantiate a new tree with the root node, which keeps the queue and the system configuration (see `main` in Figure 4.2).

2. **Performing tool invocations.** Starting from the root node, each time a new node is created, we get the first tool of the queue in this node and invoke the tool.The output of a tool invocation, which we call a report, is of one of the following 4 types: (i) *OK*: an analyzer validates a system configuration and does not produce any needs, (ii) *need*: an analyzer identifies a problem and produces a set of alternative needs, (iii) *adapt*: a solver produces a set of alternative actions for a given need, (iv) *NOK*: a solver cannot address an adaptation need (see `expandNode` in Figure 4.2). Notice that the first tool invocation is always an analyzer invocation and the report is a *need*. This is because we initiate the algorithm with a problem triggered by a monitor (see step 1).
3. **Constructing the tree recursively.** Based on the output type of the report from a tool, we create new *(node, edge)* pairs (see `expandForNext-QueuedTool`, `expandAdaptations`, and `expandNeeds` in Figure 4.3) and append them to the tree (lines 11, 12 in `expandTree`).We repeat steps 2 & 3 for the new nodes until all the nodes are visited and no new ones are created (see the recursion, lines 3, 4, 14 in `expandTree`).

Let us see how the algorithm works on a sample tree produced by CLAM (Figure 4.4). When we have an initial trigger inside CLAM (see `main`), we identify a corresponding analyzer for the monitoring data we receive –in our case *time analyzer*–, create a queue Q_0, insert the analyzer in Q_0 and create the root node $< Q_0, \mathcal{SC}_0 >$ to instantiate the tree, which the initial system configuration $\mathcal{SC}_0$ is input to the algorithm (lines 1-8 in `main`). Then we call `expandTree` function (lines 9 in `main`). The core of the algorithm is realized by this function. First, it identifies the set of leaf nodes in the tree (line 3 in

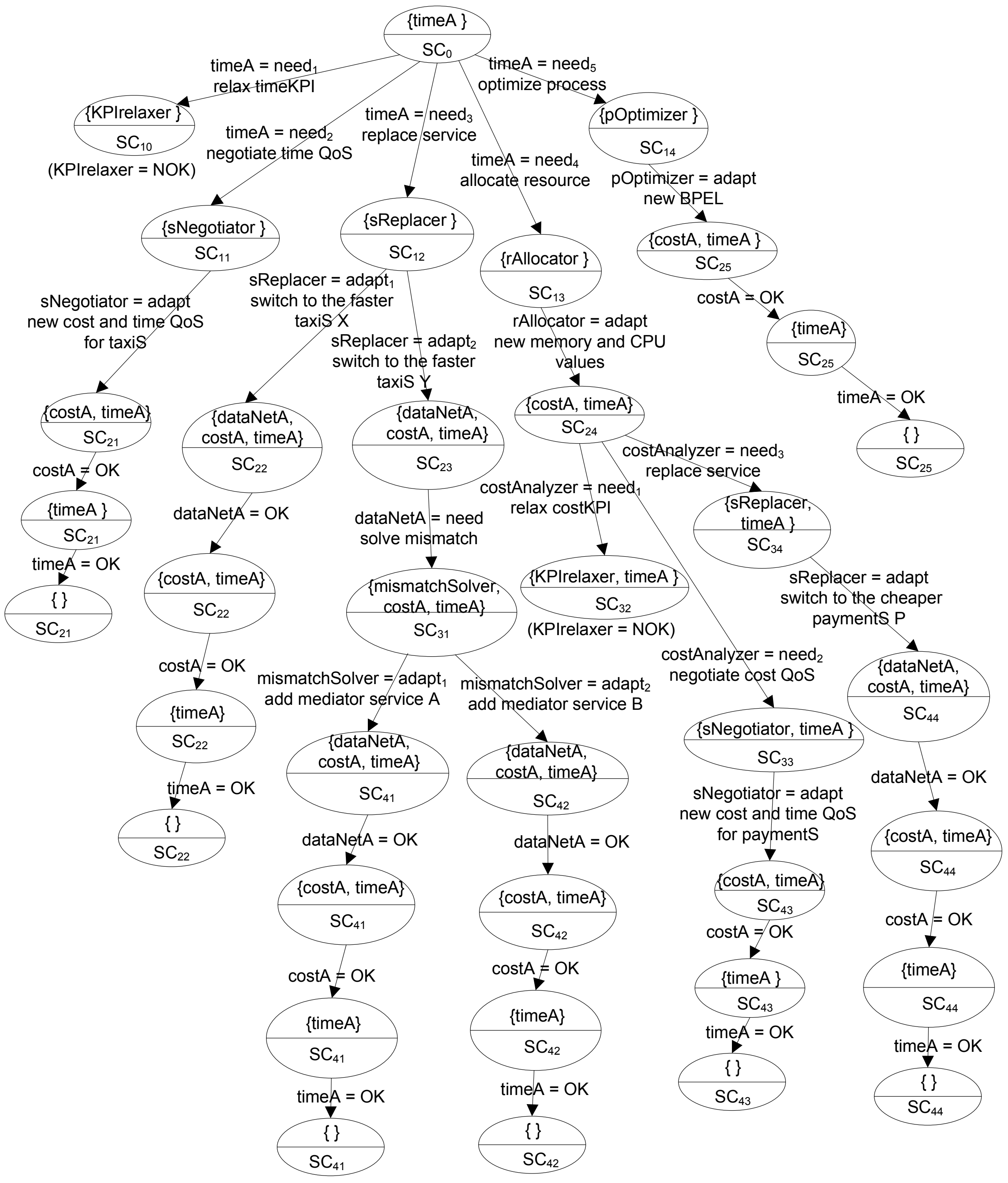


Figure 4.4: A Sample Cross-layer Adaptation Tree Produced by CLAM

expandTree), which is only the root node for the initial call. Then, it picks up one of the leaves (based on the setting it can be breadth-first or depth-first tree traversal) and invokes the first tool from the queue of this node (lines 5, 7 in expandTree). In this case, since we have only the root node, we invoke the time analyzer from the queue. Afterwards, we get back a report from the tool and we call the expandNode function (line 8 in expandTree). This function is responsible for figuring out the report type and calling the proper procedure which will return the new *(node, edge)* pairs to be appended to the tree.

Based on the report type, expandNode calls one of the following 3 functions (Figure 4.3): (i) in case of an *OK*: expandForNextQueuedTool (ii) in case of an *adapt*: expandAdaptations (iii) in case of a *need*: expandNeeds. Otherwise if it is a *NOK* from a solver, it returns back an empty list of *(node, edge)* pairs, which means there is nothing more to expand in that branch. In our example, we get a *need* report, i.e., a set of alternative needs from the time analyzer. So we call the expandNeeds. Using our adaptation model, for each need d it identifies a solver (lines 4, 5 in expandNeeds), updates the queue, and extends the system configuration (lines 6-10 in expandNeeds). Notice that extending the system configuration with need d implies attaching the need to the relevant elements in the system configuration (in this case $\mathcal{SC}_0$). Then, having identified an updated queue Q and system configuration $\mathcal{SC}$, for each need d it creates the *(node, edge)* pair (lines 11,12 in expandNeeds). For instance, in this case, pairs are:

Node:	Edge:
$Q_1 = \{KPIrelaxer\}, \mathcal{SC}_{10}$	$timeA = need, d_1 = relaxtimeKPI$
$Q_2 = \{sNegotiator\}, \mathcal{SC}_{11}$	$timeA = need, d_2 = negotiatetimeQoS$
$Q_3 = \{sReplacer\}, \mathcal{SC}_{12}$	$timeA = need, d_3 = replaceservice$
$Q_4 = \{rAllocator\}, \mathcal{SC}_{13}$	$timeA = need, d_4 = allocateresource$
$Q_5 = \{pOptimizer\}, \mathcal{SC}_{14}$	$timeA = need, d_6 = optimizeprocess$

After all the pairs are computed, `expandNode` returns them back to the main procedure `expandTree` so that they can be appended to the leaf under investigation (lines 11,12 in `expandTree`). As a next step, `expandTree` makes a recursive call in order to perform the same analysis steps for the updated tree (line 14 in `expandTree`). In the next iteration, all the leaves contain a queue with a solver. Then a leaf is picked, the solver in its queue is invoked and a set of adaptation actions are received as an invocation output, which implies that this time `expandAdaptations` function is called (line 8 in `expandNode`): For each action a, first we extend the system configuration. Notice that extending the system configuration with action a (`attachAdapt-ToSC`, line 5 in `expandAdaptations`) implies attaching the proposed action a to the relevant elements in the system configuration, and removing the corresponding need d. Subsequently, the action a is applied to the system configuration (`applyAdaptToSC`, line 6 in `expandAdaptations`) by calling the relevant *enactor* of the action a, which implies adding a subset of new elements and/or removing a subset of existing elements in the system configuration. Next, for each action a, using our adaptation model, we get the affected system elements and identify the analyzers associated to these elements (line 7 in `expandAdaptations`). Then, we combine the identified analyzers with the existing analyzers of the queue and create an overall list of unique analyzers (lines 8-10 in `expandAdaptations`). Afterwards, we create the permutations of this list to identify all the possible orders of analyzer invocations (line 11 in `expandAdaptations`). We add every permuted analyzer list to a different queue, which every queue results in a separate new tree node (lines 12-17 in `expandAdaptations`). Notice that, for the sake of space, our example in Figure 4.4 illustrates only one specific order of identified analyzers and does not show all the permutations of queues.

Subsequent to the execution of `expandAdaptations` function, we will end up with the new tree nodes which have the queues with the analyzers identified to check the changes in an adapted system configuration. After adding them to the tree, `expandTree` will initiate a new iteration inside the recursion. Suppose that this time we pick a leaf in which the queue head is an analyzer and after invoking it we receive an *OK* report. In this case, `expandNode` calls the `expandForNextQueuedTool` function (line 6 in `expandNode`): Since we get a positive response from the analyzer, there is nothing to modify in the system configuration. Only the queue is updated, i.e., the analyzer just invoked is removed from the queue (line 6 in `expandForNextQueuedTool`).

`expandTree` keeps on building the tree until all the nodes in the tree are visited. To prevent the infinite trigger of *needs* and *adapts*, at each iteration `expandTree` can exploit a stop condition, which is optional to use in the algorithm (line 3 in `expandTree`). A stop condition can be, for instance, a threshold on the tree size. Furthermore, during the iterations, the algorithm checks the repetition of a node creation, which can be the case after a solver triggers an adaptation action and the enactor applies it to the configuration: If a newly created tree node is already in the tree, we stop the iteration on that tree path (line 10 in expandTree).

The resultant tree paths correspond to the possible cases that one can have as a consequence of the initial trigger. The paths that have an empty queue in the leaf node represent the solutions (cross-layer adaptations) which ensure that there is no *need* left connected to the final $\mathcal{SC}$ and it is a stable one, that is safe to be deployed in the running system. For instance, in Figure 4.4 the paths having the $\mathcal{SC}_{21}$, $\mathcal{SC}_{22}$, $\mathcal{SC}_{41}$, $\mathcal{SC}_{42}$, $\mathcal{SC}_{43}$, $\mathcal{SC}_{44}$ and $\mathcal{SC}_{25}$ final system configurations constitute alternative cross-layer adaptations. Whereas, the paths with the final configurations $\mathcal{SC}_{10}$ and $\mathcal{SC}_{32}$ constitute the unsuccessful ones since the analysis was terminated due to a *NOK* report from a solver.

4.2.2 Correctness of the Approach

In the following we will prove the correctness of the proposed approach and discuss the termination conditions of the presented algorithm. In particular, we will show that the algorithm produces a CLAM tree T as defined in Definition 4.3, and if it is a solution tree T_s it solves the cross-layer adaptation problem with respect to the Definition 4.2. Moreover, we will demonstrate the completeness of the algorithm.

Lemma 4.1 *The CLAM algorithm produces a CLAM Tree* T *as defined in Definition 4.3.*

Proof.

1) The output of the algorithm is a graph which nodes keep a system configuration $\mathcal{SC}$ and a queue of tools $\mathcal{T}$, and edges keep a tool $\mathcal{T}$:

From the algorithm we see that we initialize the graph with the root node in lines 3-8 of `main`. Lines 5,7 of `expandForNextQueuedTool`, lines 14,16 of `expandAdaptations`, and lines 7,11 of `expandNeeds` show the creation of new nodes and edges. Then, `expandTree` recursive function shows that we continuously add them to the graph in lines 11, 12.

From line 5 of `main`, lines 5,6 of `expandAdaptations`, and line 9 of `expandNeeds` we see that the "node" data structure keeps a system configuration $\mathcal{SC}$. From line 6 of `main`, lines 8,9,12,15 of `expandAdaptations` and lines 8,10 of `expandNeeds` we see that the nodes keep also a queue of tools $\mathcal{T}$. Finally, line 7 of `expandForNextQueuedTool`, line 16 of `expandAdaptations` and line 11 of `expandNeeds` show that the edges comprise a tool $\mathcal{T}$.

2) The output graph of the algorithm is a tree, which implies that the graph is acyclic:

To conclude that the output of the algorithm is an acyclic graph, we should prove that every transition always produces a novel node. Let us see all the possible cases in which we add a node to the tree: We expand the tree in lines 11-12 of `expandTree` and the *(node, edge)* pairs to be appended to the tree are decided in the `expandNode` function (see line 8 of `expandTree`).

Let us consider an instantaneous leaf $l = \langle \mathcal{SC}, Q \rangle$ in the tree and invoke the first tool $\mathcal{T}$ from its queue Q. If we look inside `expandNode`, as a result of an invocation we may have three cases in which new *(node, edge)* pairs are created (lines 5-10). Let us analyze each case one by one:

Case I. A solver is invoked and a set of alternative adaptation actions is produced (lines 7-8 in `expandNode`).

In this case, `expandAdaptations` function is called. For each action a that the solver produced, the system configuration $\mathcal{SC}$ is updated to an $\mathcal{SC}'$ (lines 5,6) , and the queue Q is updated to a set of Q's (lines 12-15), each of which has the same updated set of tools, but with different ordering. Notice that *(i)* $\mathcal{SC}$ changes because of the execution of the action a by the enactor, and *(ii)* Q changes because the solver, which has been just invoked, is deleted from the queue, and new analyzers, associated with the changed $\mathcal{SC}$ nodes, are added to the queue. However, it is possible that a newly generated node $\langle \mathcal{SC}', Q' \rangle$ already exists in the tree. In this case, we do not append it to the tree (line 10 in `expandTree`).

Case II. An analyzer is invoked and a set of alternative adaptation needs is produced (lines 9-10 in `expandNode`).

In this case, `expandNeeds` function is called. For each need d that the analyzer produced, the system configuration $\mathcal{SC}$ is updated to an $\mathcal{SC}'$ (line 9) , and the queue Q is updated to a Q' (lines 8, 10): *(i)* $\mathcal{SC}$ changes because of the attachment of the need d to the configuration, and *(ii)* Q changes because the analyzer, which has been just invoked, is deleted from the queue, and the new alternative solver, associated with the need, is added to the queue. Notice that it is impossible that the new node $\langle \mathcal{SC}', Q' \rangle$ already exists in the tree because its parent node is a novel tree node which has been proposed by a solver (**Case I**).

Case III. An analyzer is invoked and an acknowledgement is received for the $\mathcal{SC}$.

In this case, `expandForNextQueuedTool` function is called. Since there is no new need d or new action a proposed, the system configuration $\mathcal{SC}$ remains unchanged, and the queue Q is updated to Q' (line 6). Q changes because the analyzer, which has been just invoked, is deleted from the queue. In the next iterations either Q' will be reduced by a continuous invocation of a series of analyzers and end up with an empty queue, or a new need will be identified by one of the forthcoming analyzers, which in turn will invoke a solver. Notice that, even if the $\mathcal{SC}$ remains unmodified, the fact that the queue is always reduced guarantees the creation of a novel tree node in this function.

From **1)** and **2)** we conclude that the output of the algorithm is a CLAM Tree T. $\square$

When the algorithm constructs the tree T, if in an iteration it receives an *adapt* report from a solver and accordingly an adaptation action a is applied to the system configuration, later it does not take into account all the analyzers to validate the new configuration $\mathcal{SC}_{new}$, instead it identifies a subset of analyzers from the set $\mathcal{T}\text{ools}_A$. This is because in the cross-layer adaptation framework, each analyzer works on a subset of system configuration nodes, which in turn implies that if an a does not change the input of an analyzer, then this analyzer remains *unaffected* by the a, and consequently, the new configuration $\mathcal{SC}_{new}$ does not require to be checked by this analyzer.

Definition 4.6 *(Unaffected Analyzers) In a CLAM Tree* T, *for every edge* $e_i(v_{i-1}, v_i)$ *with an edge lable* $T_E(e_i) = \mathcal{T}_i$ *such that* $\mathcal{T}_i \in \mathcal{T}ools_S$, *there exists a set of* Unaffected Analyzers $\mathcal{T}ools_{A_{unaf}}$ *such that* $Q_{v_i} = (Q_{v_{i-1}} \setminus \{\mathcal{T}_i\}) \cup \{\mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \ldots, \mathcal{T}_{a_n}\}$ *where* $\mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \ldots, \mathcal{T}_{a_n} \in \mathcal{T}ools_A$ *and* $\forall \mathcal{T}_a \in \mathcal{T}ools_A$ *if* $\mathcal{T}_a \notin \{\mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \ldots, \mathcal{T}_{a_n}\}$

then $\mathcal{T}_a \in \mathcal{T}ools_{A_{unaf}}$ *where* $\mathcal{T}_a(\mathcal{SC}_{v_i}) = \mathcal{SC}_{v_i}$.

Lemma 4.2 *(CLAM Path Structure)*
Let T *be a CLAM tree produced by the algorithm. For all path* $p \in P$ *of* T, *there exists a path structure* $p = p_{NA_0}p_{OK_0}p_{NA_1}p_{OK_1}\dots p_{NA_k}p_{OK_k}$ *such that:*

- *a* $p_{NA} = \{\mathcal{T}_a, \mathcal{T}_s \mid \mathcal{T}_a \in \mathcal{T}ools_A, \mathcal{T}_s \in \mathcal{T}ools_S\} \cup \{\mathcal{T}_a \mid \mathcal{T}_a \in \mathcal{T}ools_A, \mathcal{T}_a(\mathcal{ESC}) = \mathcal{ESC}'\}$;
- *a* $p_{OK} = \{\mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \dots \mathcal{T}_{a_n} \mid \mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \dots \mathcal{T}_{a_n} \in \mathcal{T}ools_A, 0 \le n \le Size(\mathcal{T}ools_A), \mathcal{T}_{a_0}(\mathcal{SC}) = \mathcal{T}_{a_1}(\mathcal{SC}) = \dots = \mathcal{T}_{a_n}(\mathcal{SC}) = \mathcal{SC}\} \cup \perp$.

Proof.

From Lemma 4.1 we know that the algorithm produces the CLAM tree, T. Let us consider an arbitrary p of T and analyze its construction from the root node rt:

Step 1. $rt = \langle \mathcal{SC}_0, Q_0 \rangle$ has always the queue $Q_0 = \{\mathcal{T}_a\}$ such that $\mathcal{T}_a \in \mathcal{T}\text{ools}_A$. Thus p will always start with an analyzer edge, which produces a need d (lines 2,6 in `main`).

Step 2. For the need d produced by $\mathcal{T}_a$ in Step 1, there are two cases: (i) There is a *solver* which can address d. In this case, we delete the analyzer of the last invocation from the queue and add the newly identified solver: $Q_1 = \{\mathcal{T}_s\}$, $\mathcal{T}_s \in \mathcal{T}\text{ools}_S$. We create a new (node,edge) pair (v_1, e_1) where $v_1 = \langle \mathcal{SC}_1, Q_1 \rangle$, $T_E(e_1) = \mathcal{T}_a$, add v_1 and e_1 to the tree and the iteration continues. (ii) there is no *solver* found to address d. In this case, the iteration stops, and since there is no edge created, p does not exist.

Step 3. We follow by the solver invocation from the queue Q_1, created in Step 2. There are two cases: (i) It proposes an adaptation action a. In this case, after the enactor executes a and updates $\mathcal{SC}_1$ to $\mathcal{SC}_2$, the algorithm identifies a set of analyzers, associated with the system configuration nodes which are updated by the enactor (line 7 in `expandAdaptations`). Thus, we delete the solver of the last invocation from the queue and add in the queue the identified analyzers: $Q_2 = \{\mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \ldots, \mathcal{T}_{a_k}\}$, $\mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \ldots, \mathcal{T}_{a_k} \in \mathcal{T}\text{ools}_A$. We create a new (node,edge) pair (v_2, e_2) where $v_2 = \langle \mathcal{SC}_2, Q_2 \rangle$, $T_E(e_2) = \mathcal{T}_s$, add v_2 and e_2 to the tree and the iteration continues. (ii) The solver could not find any adaptation actions. In this case, the iteration stops for p and we end up with $p = p_{NA_0} p_{OK_0}$ where $p_{NA_0} = T_E(e_1) = \mathcal{T}_a$ and $p_{OK_0} = \perp$.

Step 4. We follow by a tool invocation from Q_2, created in Step 3. The first element of Q_2 is $\mathcal{T}_{a_0}$, i.e., it will be an analyzer invocation. There are two cases:

Step 4a. The analyzer produces a new need d', in this case, it implies that we go back to Step 2 until we get at Step 4 again.

Step 4b. The analyzer does not produce any need, i.e., leaves the $\mathcal{SC}_2$ unchanged, which implies a reduced queue by the deletion of the lastly invoked analyzer. Afterwards, there are two possibilities: (i) $Q_3 = \varnothing$, which implies that the iteration stops for p and we end up with $p = p_{NA_0} p_{OK_0}$ where $p_{NA_0} = \mathcal{T}_a, \mathcal{T}_s$ and $p_{OK_0} = \mathcal{T}_{a_0}$. (ii) $Q_3 = \{\mathcal{T}_{a_1}, \ldots, \mathcal{T}_{a_k}\}$. We create a new (node,edge) pair (v_3, e_3) where $v_3 = \langle \mathcal{SC}_2, Q_3 \rangle$, $T_E(e_2) = \mathcal{T}_{a_0}$, add v_3 and e_3 to the tree and the iteration continues with a new application of Step 4.

From above steps, we can easily see that every path $p \in P$ has always a path structure $p = p_{NA_0} p_{OK_0} p_{NA_1} p_{OK_1} \ldots p_{NA_k} p_{OK_k}$ such that

- In case the iteration stops in Step 2: if there exists a path p, it finishes with $p_{NA_k} p_{OK_k}$ such that $p_{NA_k} = \mathcal{T}_{a_k}, \mathcal{T}_{s_k}$ and $p_{OK_k} = \perp$ if the previous step is Step 4a, or $p_{OK_k} \neq \perp$ if the previous step is Step 4b;

- In case the iteration stops in Step 3: it finishes with $p_{NA_k} p_{OK_k}$ such that $p_{NA_k} = \mathcal{T}_{a_k}$ and $p_{OK_k} = \perp$;

- In case the iteration stops in Step 4b: it finishes with $p_{NA_k} p_{OK_k}$ such that $p_{NA_k} = \mathcal{T}_{a_k}, \mathcal{T}_{s_k}$ and $p_{OK_k} \neq \perp$. $\square$

Lemma 4.3 *In a solution tree* T_s*, each leaf* $l \in L$ *with an empty queue* $Q_l = \varnothing$ *has a stable system configuration with respect to the Definition 4.1.*

Proof.

From Lemma 4.1 and by Definition 4.5, the algorithm produces a solution tree T_s if it has at least one tree path ending with a leaf that has an empty queue. Let us consider a path p of T_s –produced by the algorithm– such that its leaf $l = \langle \mathcal{SC}_l, Q_l \rangle$, $Q_l = \varnothing$. From Lemma 4.2 it has a structure $p = p_{NA_0} p_{OK_0} p_{NA_1} p_{OK_1} \dots p_{NA_k} p_{OK_k}$ such that $p_{OK_k} = \mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \dots \mathcal{T}_{a_n}, \mathcal{T}_{a_0}(\mathcal{SC}_l) = \mathcal{T}_{a_1}(\mathcal{SC}_l) = \dots = \mathcal{T}_{a_n}(\mathcal{SC}_l) = \mathcal{SC}_l$. $Q_l = \varnothing$ implies that all the identified analyzers on p are invoked and deleted from the queue. Then, Definition 4.6 guarantees that the execution of p_{NA_k} introduces all the analyzers affected by the last adaptation action, which in turn implies that $\mathcal{T}\text{ools}_{A_{unaf}} = \mathcal{T}\text{ools}_A \setminus \{\mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \dots \mathcal{T}_{a_n}\}$ and $\forall \mathcal{T}_a \in \mathcal{T}\text{ools}_{A_{unaf}} \mathcal{T}_a(\mathcal{SC}_l) = \mathcal{SC}_l$. Thus, given the fact that we consider an initial trigger as the only concern which impedes the *system stability* –the overall framework works with a single problem at a time, Lemma 4.2 and Definition 4.6 enforce that $\mathcal{SC}_l$ of the leaf l of the path p is stable, i.e., for each analyzer $\mathcal{T} \in \mathcal{T}\text{ools}_A$, $\mathcal{T}(\mathcal{SC}_l) = \mathcal{SC}_l$. $\square$

Theorem 4.1 (Correctness of the CLAM Algorithm)
If the algorithm finds a cross-layer adaptation solution, which implies a leaf node with an empty queue in the produced tree, it is correct with respect to the Definition 4.2.

Proof.

From Lemma 4.1, the algorithm produces a CLAM tree T. From Lemma 4.3, if the output of the algorithm is a solution tree T_s, then it has at least one path p which leads to a *stable* system configuration $\mathcal{SC}_f$ in the leaf l of p. From Definition 4.4, we can always obtain the collapsed path p_c of the path p such that all the edges, which are not contributing to the modification of an instantaneous system configuration, are removed, which in turn enforces a sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_f$, such that $\mathcal{SC}_{i-1} \neq \mathcal{SC}_i$, for each $i \in \{1, 2, \ldots, f\}$. Thus, from Lemma 4.1, Lemma 4.3 and Definition 4.4, the collapsed path p_c is a cross-layer adaptation solution, and is correct with respect to the Definition 4.2. □

Theorem 4.2 (Completeness of the CLAM Algorithm)
If there exists a solution for cross-layer adaptation problem defined in Definition 4.2, the algorithm guarantees to find it.

Proof.

Let the sequence $p_s = \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ be a cross-layer adaptation for system configuration $\mathcal{SC}_0$ with respect to the Definition 4.2. The definition enforces two concerns: (i) Every tool in the sequence must contribute to the solution, which in turn implies that for every analyzer $\mathcal{T}_{a_i} \in p_s$, $\mathcal{T}_{a_i}(\mathcal{SC}_i) = \mathcal{SC}_i \cup \{d_i\}$. (ii) The final system configuration $\mathcal{SC}_n$, produced from the execution of $\mathcal{T}_n$ is stable, which implies that $\mathcal{SC}_n$ must be non extended and $\mathcal{T}_n$ can only be an enactor. Given the fact that the enactor is a tool type internal to the cross-layer adaptation framework and executed always subsequent to an action a produced by a solver, we can consider the following equivalence: If $\mathcal{T}_s$ is a solver producing an action a and $\mathcal{T}_e$ is its enactor, i.e., working on a, then we can combine the functionalities of two tools by generalizing the definition of a solver: $\mathcal{T}_s(\mathcal{SC}) \equiv \mathcal{T}_e(\mathcal{T}_s(\mathcal{SC}))$. Given this generalization, the last tool of the sequence is a solver: $\mathcal{T}_n \in \mathcal{T}\text{ools}_S$.

Notice that (ii) also enforces that for every need d_i, produced by $\mathcal{T}_{a_i} \in p_s$, there must be a corresponding solver $\mathcal{T}_{s_i} \in p_s$, which works on d_i. Similarly, for every $\mathcal{T}_{s_i} \in p_s$, there must be its corresponding need d_i, already produced by a $\mathcal{T}_{s_i} \in p_s$. Thus, given that each solver works on a single need, the sequence p_s must have an equal number of analyzers and solvers.

Since an analyzer $\mathcal{T}_{a_i}$ modifies a $\mathcal{SC}_i$ only by extending it with a need d_i, it is a consequent solver $\mathcal{T}_{s_i}$ –responsible for addressing d_i– which is really modifying a $\mathcal{SC}_i$ by introducing variants of $N, R \in \mathcal{SC}$. Therefore, what really brings $\mathcal{SC}_0$ to $\mathcal{SC}_n$ is a specific order of solver invocations.

In the algorithm, as soon as a need d_i is triggered by an analyzer $\mathcal{T}_{a_i}$, all the possible corresponding solvers are identified and each alternative solver is inserted to an alternative queue as the queue head (lines 4,5,10 in `expandNeeds`). This ensures the identification of all the combinations of solvers. However, this is not sufficient, we must have all the possible sequences of solver combinations because if we invoke the same set of solvers in different sequences, each sequence will produce a different final $\mathcal{SC}_n$. The algorithm addresses this issue as follows:

From Lemma 4.2 every tree path has a structure $p = p_{NA_0} p_{OK_0} p_{NA_1} p_{OK_1} \cdots p_{NA_k} p_{OK_k}$ such that $p_{OK_k} = \mathcal{T}_{a_0}, \mathcal{T}_{a_1}, \ldots \mathcal{T}_{a_n}$, $\mathcal{T}_{a_0}(\mathcal{SC}_l) = \mathcal{T}_{a_1}(\mathcal{SC}_l) = \ldots = \mathcal{T}_{a_n}(\mathcal{SC}_l) = \mathcal{SC}_l$, which in turn implies that by Definition 4.4, every collapsed path p_c of p has the following structure: $p = p_{NA_0} p_{NA_1} \cdots p_{NA_k}$ where $p_{OK_0} = p_{OK_1} = \ldots = p_{OK_k} = \perp$. Thus, every sequence of tools that CLAM Tree T can find as a potential solution for cross-layer adaptation is in the form $\mathcal{T}_{a_1}, \mathcal{T}_{s_1}, \mathcal{T}_{a_2}, \mathcal{T}_{s_2}, \ldots, \mathcal{T}_{a_k}, \mathcal{T}_{s_k}$ where $\mathcal{T}_{a_1}, \mathcal{T}_{a_2}, \ldots, \mathcal{T}_{a_k} \in \mathcal{T}\text{ools}_A$, $\mathcal{T}_{s_1}, \mathcal{T}_{s_2}, \ldots, \mathcal{T}_{s_k} \in \mathcal{T}\text{ools}_S$. Therefore, for us, finding all the possible sequences of solver combinations is equivalent to finding all the possible sequences of analyzers, which produce the relevant needs, i.e., inputs of the solvers. In the algorithm, each time an adaptation action a is triggered by a solver $\mathcal{T}_{s_i}$, all the analyzers of affected system nodes are identified, by Definition 4.6 we know that we do not need to consider the remaining analyzers of the system (line 7 in `expandAdaptations`). Then, the identified analyzers are combined with the ones already existing in the queue and this combined set is permuted to create all the possible sequences and reset the queues with these sequences (lines 9-11 in `expandAdaptations`). In this way, we guarantee all the specific orders of solver invocations, which in turn implies that if there exists a solution sequence p_s with respect to Definition 4.2, the algorithm finds it. $\square$

Termination.

In order to apprehend if the CLAM algorithm terminates, let us analyze function `expandTree`, which is responsible for constructing the CLAM Tree T recursively. Figure 4.5, starting from the creation of the root node of T, elaborates all the possible transitions inside `expandTree` function. As the figure demonstrates, we might have three cases for the construction of a tree path:

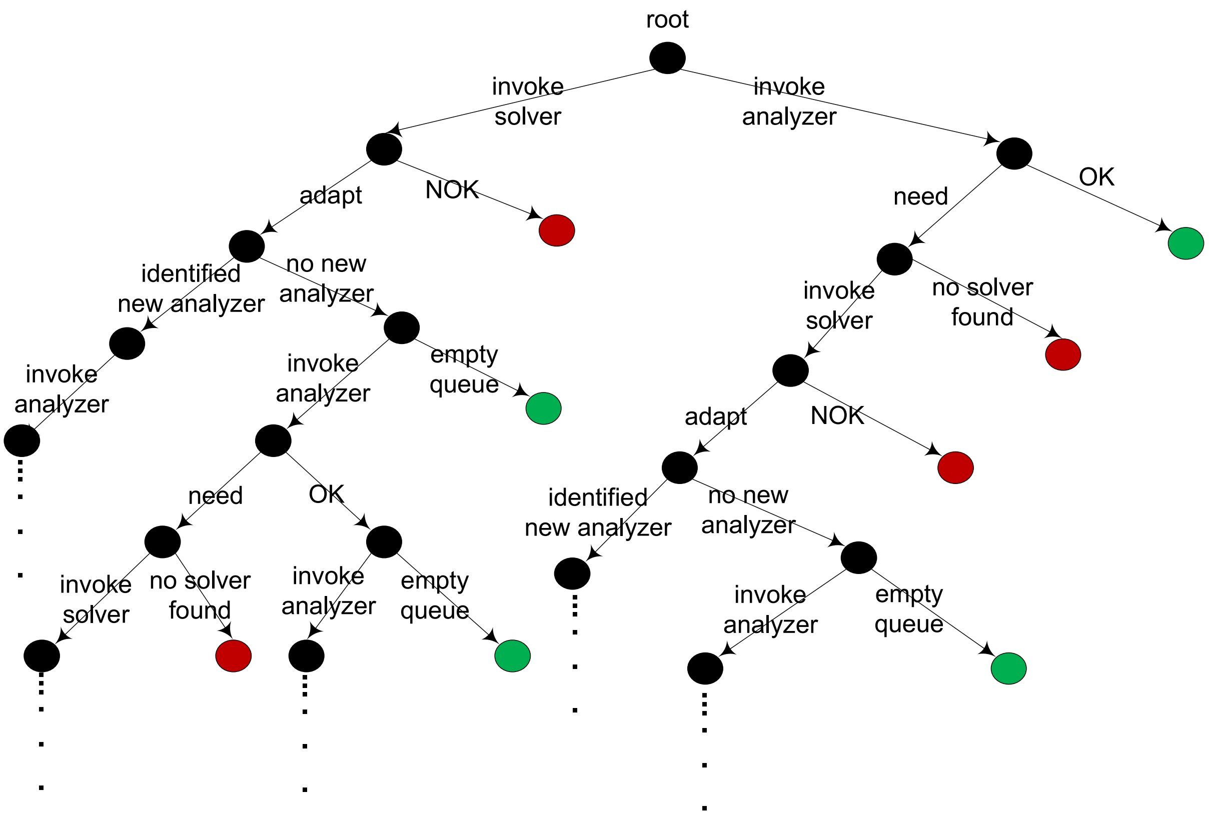


Figure 4.5: Non Termination: Analysis of Recursive `expandTree` Function

1) A finite path with a solution: This case is depicted with a green node in the figure. As soon as there is a state with an empty queue, it implies that the algorithm finds a cross-layer adaptation and the relevant tree path p can be finalized. Note that in case of the first invocation, if it is an analyzer and the response is an *OK*, it directly implies that there is no problem in the system, and thus, no need to search for a cross-layer adaptation.

2) A finite path without a solution: This case is depicted with a red node in the figure. It implies that there is no solution for the path p under investigation, and thus, we can finalize the path. It happens in two different ways: (i) a solver cannot propose an adaptation for the given need, (ii) there is no corresponding solver for the need.

3) An infinite path: This case occurs whenever analyzers keep on triggering new needs in some of the tree paths, which results in further invocations of solvers and subsequently they produce new adaptation actions again to be checked by analyzers. Such paths can go to infinity.

As we can observe from the analysis of the algorithm, in the third case it does not terminate. This is due to the fact that there might be cross-layer adaptation solutions with arbitrary path lengths, and moreover, the solution space might be infinite. Then, we might end up with combinations of infinitely many adaptation actions. However, from Theorem 4.2 and given the fact that the algorithm constructs the tree in a breadth-first manner, the algorithm eventually finds all the solutions, which in turn implies that if a path goes to infinity, there exists no cross-layer adaptation solution that could be found in that path.

In a practical setting, non-termination is a problem as infinitely long paths are never desired for a good design of an adaptation framework. Moreover, such paths do not serve us since they do not contain a solution, which can be proposed for system stability. Getting encouraged by these two phenomena, in Chapter 7 we present practical cases that we can work on a finite tree depth and still find a reasonable set of solutions by applying a heuristic method.

4.3 Discussion

In this chapter, we defined formally the cross-layer adaptation problem, which we grounded on the framework introduced in Chapter 3, and presented the CLAM algorithm in order to tackle it. We proved that the algorithm is correct and complete with respect to the formal definition of the cross-layer adaptation problem.

As we have discussed in related work in Chapter 2, there exists different approaches for addressing the cross-layer adaptation problem. One intention of this chapter was to clarify our position within existing cross-layer adaptation approaches by presenting in depth our understanding of a cross-layer adaptation problem and its possible solution.

The problem we want to tackle can be summarized as follows: Given (i) a set of adaptation capabilities, called solvers, and monitoring and analysis mechanisms, called analyzers, i.e., a set of tools, and (ii) an initial monitoring trigger, i.e., an adaptation need, we aim at coordinating the system tools to find a cross-layer adaptation strategy, which will bring the system back to an overall stable configuration where all the system analysis tools are satisfied with the resultant configuration.

Given this description, we can clearly define the characteristics of our proposed approach: (i) it identifies cross-layer adaptation solutions on the fly since it is an iterative coordination approach, (ii) it is a static approach since the tools to be used are decided at design time, (iii) it can identify complex or unexpected cross-layer adaptation cases since the solutions are discovered at run-time during the coordination process.

In addition to summarizing the overall idea of our approach, we would also like to comment on more specific aspects: We remark that even if the presented algorithm starts with an analyzer (initial trigger is always an adaptation need), it can be easily seen that we can apply the same algorithm with an initial adaptation action, in this case the root node will comprise the system configuration in which the triggered adaptation is not applied yet, and the queue in which there is the solver, which triggered the adaptation.

One limitation of our approach is that the framework, which the algorithm grounds on, receives one problem at a time, which means that we do not allow the correlation of problems that might come from the monitors simultaneously. In future work, this limitation can be overcome by integrating a cross-layer monitoring tool to the CLAM, which will first diagnose the various monitoring events triggered from different system parts, then will produce a resultant trigger to the CLAM algorithm.

Another issue regarding our current solution is that, even if it allows for alternative solvers and their alternative adaptation solutions, it does not allow for alternative analyzers. All the analyzers that we identify after an adaptation action are to be invoked for system stability, thus, put in the same queue. However, assuming that tools behave correctly, we do not need alternative analyzers to investigate the validity of the same system constraint.

Coming back to the discussion about our position within the existing cross-layer adaptation approaches, in particular, related work lacks severely for the capability to identify a wide range of possible cross-layer adaptation solutions. E.g. the approaches presented in [132, 103] are based on a centralized knowledge of cross-layer adaptation patterns, which in turn implies that if a pattern for a possible cross-layer adaptation solution is not predefined, this solution will never be found.

Furthermore, our approach differs significantly in being holistic. In every iteration of the coordination, we take into account the possible consequences of each adaptation action, and do not propose any final solution unless all the identified problems are handled. Most of the existing works oversimplify the cross-layer adaptation problem by not considering the impact of the proposed adaptations on the overall system [45, 41, 132, 103, 123, 108].

Chapter 5

Selection and Deployment of a Cross-layer Adaptation

In this chapter we propose two approaches to rank and select the produced cross-layer adaptation solutions (strategies), and furthermore, discuss the deployment issues once a strategy is selected to be enacted in the running system.

Given the variety & complexity of adaptations, it is important to understand what each solution means, which ranking criteria should be taken into account and how a cross-layer adaptation strategy could be selected based on these criteria.

To address the selection problem, we propose novel criteria, which take into account the required efforts to enact an adaptation strategy as well as the system quality. For selection method, we propose two different approaches. The first one is the well known multi criteria decision making, which we we have implemented in the framework and present the achieved results in Chapter 7. However, the cross-layer adaptation framework is open to exploit different techniques for the selection problem. Regarding this, we have collaborated with Politecnico Milano and investigated a second selection approach based on fuzzy logic.

5.1 Ranking and Selection

Selecting an appropriate adaptation strategy in service-based systems (SBS) is not an easy task. Basically a problem in the system can be addressed using various sets of adaptation actions, each having different consequences once deployed in the system. The problem gets even more complex when we consider the multi-layer nature of such systems. As we have observed in Chapter 4.1, given the variety of layer-specific adaptation capabilities, and the interdependencies among system elements, CLAM can identify various cross-layer adaptation strategies to tackle the same problem. Regarding this fact, apparently, traditional QoS-based selection approaches, e.g. [123, 28, 3, 27] are not sufficient to address the complexity of adaptation selection problem in multi-layer systems. Let us elaborate the problem on a motivating example from our reference scenario “Call & Pay Taxi” which we introduced in Section 3.2:

Example 5.1 *Let us recall the case study that we utilized in Chapter 4 to describe the CLAM algorithm: The monitor detects a violation for the KPI process execution time of “Call & Pay Taxi” composite service (CPTS). Given the adaptation capabilities, i.e., the available solvers, one can take various decisions to address this problem: We can make changes at the application level such as trying to optimize the process o relaxing the KPI target a bit. Moreover, we can move to the service level by negotiating the execution time QoS of partner services, i.e., messaging, location, payment and taxi services, or replacing some of them with some faster ones. Finally, we can improve the response time of the underlying infrastructure by allocating more resources. Note that some of these actions can have also alternatives: e.g., for service replacements, we might have various alternative services with the same functionality. In the end, each option might have various consequences. For instance, negotiating for a faster execution of services might increase the overall cost and cause a new KPI violation, the newly replaced service might have data incompatibility, which in* *turn requires a solution to the data mismatch problem. In such cases, CLAM needs to identify new, additional actions to remove the negative effects of these adaptations.*

As the example above reveals, CLAM might produce several solution paths for the cross-layer adaptation problem. In the end, path selection in a CLAM tree is a complex task not only because of the large number of solutions, but also because we need to understand what each solution path means when we consider its deployment in the system. This is not straightforward to figure out: First of all, adaptation actions on a path may be originated from different system elements with their own characteristics. Usually it is not trivial to compare two adaptations from two different system layers. Moreover, resultant QoS of the entire SBS should be taken into account for ranking. As a result, it is an important issue to carefully decide on the right selection criteria.

Finally, on top of our selection criteria, we need a technique to evaluate each path and eventually to select the best one for deployment in the SBS.

5.1.1 Selection Criteria

The main difficulty behind determining the selection criteria originates from the heterogeneity of the adaptation paths produced by CLAM. The adaptation actions on a path might have different deployment aspects, e.g., the time required to deploy the adaptation, the cost of adaptation. Moreover, each cross-layer adaptation solution might constitute different layers. It is reasonable that, if possible, we prefer to solve the problem by staying in the same layer without modifying further layers when a problem originates in a specific layer. Moreover, each path brings the system to a different level of quality in terms of various dimensions such as process execution time, process cost or the underlying infrastructure cost. We must take into account all these concerns for ranking and selection.

Adaptation Deployment Criteria. Let us consider a cross-layer adaptation solution which brings an initial system configuration $\mathcal{SC}_0$ to a final system configuration $\mathcal{SC}_f$. In order to determine the efforts required to deploy this solution in the system, initially we have to understand what are the changes in $\mathcal{SC}_f$ with respect to $\mathcal{SC}_0$.

We associate a number of deployment tasks with each updated node, which are necessary to realize the change in the running system, and each deployment task has two main aspects: (i) required time for deployment, (ii) deployment cost. In this way, once we identify the updated nodes of the system, we can reason about the overall efforts required to deploy a cross-layer adaptation solution. Note that the identification of deployment costs and times for each task and its association with the system nodes are carried out at design time and we use normalized values in a range of integers (0-5) for each deployment time and cost.

Definition 5.1 *(Deployment Tasks)* Deployment Tasks $\mathcal{DT}$ *is a set of deployment tasks dt available for a system model $\mathcal{M} = \langle N_\mathcal{M}, R_\mathcal{M} \rangle$ such that*

- *$dt \in \mathcal{DT}$ is a tuple $\langle dt_{time}, dt_{cost} \rangle$ where dt_{time} is the required time, dt_{cost} is the cost for the deployment task dt;*
- *$\forall n \in N_\mathcal{M}\ \exists \mathcal{DT}(n) = \mathcal{DT}'$ such that $\mathcal{DT}' \subseteq \mathcal{DT}$.*

Example 5.2 *Table 5.1 illustrates the association of the $\mathcal{SC}$ nodes of our "Call & Pay Taxi" scenario with the deployment tasks available for the SBS. Subsequently, Table 5.2 depicts the normalized values for the required time and cost to deploy each task.*

We remark that the cost and time values of deployment tasks depend on the system mechanisms, which we call *executors* in our cross-layer adaptation framework (Figure 3.10), since they are responsible for implementing the deployment tasks in the system. For instance, if we have an executor that modifies

Changed System Node:	Deployment Tasks:
process	migrate process; create machine image
service	prepare new SLA; set up new monitor
service QoS (time, cost)	modify SLA; modify monitor
KPI (time, cost)	modify monitor
infrastructure	migrate machine image
infrastructure QoS (time, cost)	reconfigure resources

Table 5.1: Mapping Changes in $\mathcal{SC}$ Nodes to Deployment Tasks

Deployment Tasks:	Efforts for Deployment	
	Required Time:	Deployment Cost:
migrate process	2	3
create machine image	0	1
prepare new SLA	1	2
set up new monitor	1	2
modify SLA	0	1
modify monitor	0	1
migrate machine image	2	4
reconfigure resources	0	1

Table 5.2: Required Efforts for Adaptation Deployment in terms of Time and Cost

the monitors automatically once an updated SLA is in place, then the required time to enact this deployment task will be short. Instead, for a semi-automated mechanism where there needs to be also human involvement, both the time and the cost of the deployment will be high. Notice that the cost of the deployment task involves also the penalties that the SBS owner might get due to the adaptations.

Adaptation Location Criteria. Let us consider a problem signaled to CLAM by a monitor. We can see from which layer the problem arises since we know which $\mathcal{SC}$ elements a monitor works on. Once the problematic layer is identified, through the execution of CLAM, various ways to solve the problem can come out. Each cross-layer adaptation solution involves a subset of system layers. Then we can distinguish among these solutions considering their locations in terms of system layers. In general we prefer to solve the problem by modifying a minimum number of layers. If we are able to find a solution by staying in the same layer of the problem, this is the most preferable case. For other cases, we define a preference list of layers for a given problem layer. So when two solutions involve the same number of layers, we investigate how preferable the constituent layers. Preferable solution layers of a problem layer is an ordered list such that each list element corresponds to the constituent layers of a cross-layer adaptation solution. The elements in the list are ordered by considering (i) the total number of layers involved and (ii) the preference list of layers for the given problem layer. The first list element is the most preferable for a cross-layer adaptation solution and the last is the worst one. Notice that the preference list for every system layer is decided at design time.

Definition 5.2 *(Preferable Solution Layers) Let $\mathcal{L}ayers$ be the set of layers $\mathcal{L}$ in a system model $\mathcal{M} = \langle N_{\mathcal{M}}, R_{\mathcal{M}} \rangle$. Let $\mathcal{L}_p \in \mathcal{L}ayers$ be a problem layer. Let $\mathcal{L}_0, \mathcal{L}_1 \ldots \mathcal{L}_n$ be an ordered list of $\mathcal{L}ayers$ denoting the preferences for the solution layer of the problem layer $\mathcal{L}_p$ where the most preferable, $\mathcal{L}_0$, is always equal to $\mathcal{L}_p$.* Preferable Solution Layers *$\mathcal{L}ayers_S$ of the problem layer $\mathcal{L}_p$ is a set of possible combinations of solution layers such that:*

- $\mathcal{L}ayers_S = \{\mathcal{L}_0\mathcal{L}_1 \ldots \mathcal{L}_n \mid \mathcal{L}_0 = \{\mathcal{L}_p \cup \varnothing\}, \mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_n \in \{\mathcal{L}ayers \cup \varnothing\}\}$
- *There exists an order relation $\leq$: $\mathcal{L}ayers_S \times \mathcal{L}ayers_S$ such that $\forall \mathcal{L}_{i_0}\mathcal{L}_{i_1} \ldots \mathcal{L}_{i_k}, \mathcal{L}_{j_0}\mathcal{L}_{j_1} \ldots \mathcal{L}_{j_m} \in \mathcal{L}ayers_S$:*
  - $\mathcal{L}_{i_0}\mathcal{L}_{i_1} \ldots \mathcal{L}_{i_k} < \mathcal{L}_{j_0}\mathcal{L}_{j_1} \ldots \mathcal{L}_{j_m}$ *if* $k < m$
  - $\mathcal{L}_{i_0}\mathcal{L}_{i_1} \ldots \mathcal{L}_{i_k} \leq \mathcal{L}_{j_0}\mathcal{L}_{j_1} \ldots \mathcal{L}_{j_m}$ *if* $k = m$ *and* $i_0 + i_1 + \ldots + i_k \leq j_0 + j_1 + \ldots + j_m$

Example 5.3 *Let us consider the layers of our "Call & Pay Taxi" scenario, namely, application, service and infrastructure layers. Table 5.3 illustrates the preference list for each layer in case a problem occurs there. The ordered list of preferable solution layers is then generated based on these preference lists by Definition 5.2.*

Problem Layer:	Layer Preference for Solution:	Ordered List of Preferable Solution Layers:
Application (A)	A, S, I	**OneOf** (A, S, I, AS, AI, SI, ASI)
Service (S)	S, A, I	**OneOf** (S, A, I, AS, SI, AI, ASI)
Infrastructure (I)	I, A, S	**OneOf** (I, A, S, AI, SI, AS, ASI)

Table 5.3: Preferable Solution Layers for a Given Problem Layer

SBS Quality Criteria. Each cross-layer adaptation solution brings an initial system configuration $\mathcal{SC}_0$ to a different final system configuration $\mathcal{SC}_f$. Thus, before deployment in the system, we should also take into consideration the quality aspects of the alternative $\mathcal{SC}_f$s. We consider two quality dimensions: process execution time and overall application cost, which includes the costs of external service invocations and the cost of the underlying infrastructure. We can easily obtain the values for these selection criteria by taking advantage of quality degrees that are committed in the SLAs signed with the service and infrastructure providers.

5.1.2 Multi Criteria Decision Analysis

As discussed above, to summarize, we have the following ranking and selection criteria for alternative cross-layer adaptation solutions: (i) overall adaptation deployment time, (ii) overall adaptation deployment cost, (iii) system layers involved in the solution, (iv) system quality in terms of process execution time, and (v) system quality in terms of overall application cost, i.e., external services and infrastructures. Based on those criteria, alternative cross-layer adaptation solutions are evaluated and ranked as follows:

Aggregated Score for a Cross-layer Adaptation Solution. We calculate the overall value for a cross-layer adaptation solution by aggregating the selection criteria presented. In our technique we apply Simple Additive Weighting - Multiple Criteria Decision Making Approach (SAW – MCDA) for criteria aggregation [55].

Let y_{p_x} be the value of a selection criterion y for a particular p_x which denotes an alternative cross-layer adaptation path in the CLAM tree. Then, for each cross-layer adaptation solution p_x; the normalized score of a criterion y, namely $n(y_{p_x})$, and the aggregated score of all the criteria, namely $score(p_x)$, are:

$$n(y_{p_x}) = \begin{cases} \frac{max(y)-y_{px}}{max(y)-min(y)} & \text{if lower values of y are preferred} \\ \\ \frac{y_{px}-min(y)}{max(y)-min(y)} & \text{if higher values of y are preferred} \end{cases} \tag{5.1}$$

$$score(p_x) = \frac{1}{k}\sum_{y=1}^{k} w_y n(y_{p_x}) \tag{5.2}$$

Above, $max(y)$ is the maximum value of the selection criterion y available in a solution tree created by CLAM, similarly, $min(y)$ is the minimum value of the selection criterion y available in the solutions. We remark that for all our selection criteria, lower values are preferred. More precisely, we would like to minimize the overall adaptation deployment time and cost, and similarly, process execution time and application cost. Considering the criterion for the layers involved in the solution, we apply Definition 5.2 to create the set of *Preferable Solution Layers*. In this way, we order all the possibilities and give each element in the set a value where the first element in the order must take the minimum value since it is the most preferable one.

The weight w_y is assigned to each selection criterion y depending on its importance for us, and the value k represents the number of used criteria. The

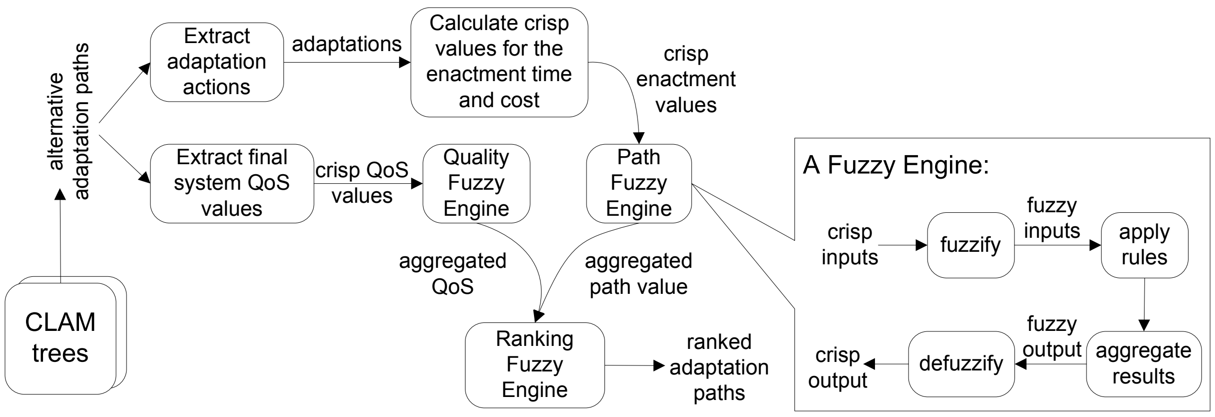


Figure 5.1: The Overall Ranking Approach

sum of all weights must be equal to 1. The result of this procedure, $score(p_x)$, is a score by which the overall path value of a cross-layer adaptation solution can be compared with the overall path value of other solutions. Consequently, the path having the highest score can be selected to be deployed in the running system.

5.1.3 An Alternative Fuzzy Logic-based Method

In this section we present an alternative ranking and selection technique, which is the outcome of our collaboration with Polimi [109]. The approach makes use of fuzzy logic [130] to evaluate the alternative cross-layer adaptation solutions and the selection is carried out by inferring the ranking criteria through a fuzzy inference system (FIS).

Fuzzy logic has been applied to many fields for decision making where the relation of involving parameters is too complex to be modeled by conventional mathematical techniques. Fuzzy logic, applying fuzzy set theory [129], is suitable to understand the relation between input and output parameters which uses linguistic parameters and applies if-then fuzzy rules to express input-output relations.

The overall ranking approach that shows the exploitation of fuzzy logic through fuzzy engines is illustrated in Figure 5.1. We apply a fuzzy inference system (FIS) for each fuzzy engine in Figure 5.1, which is a system to interpret if-then fuzzy rules. It consists of five major steps [56] as follows: The first step is fuzzification of the input parameters in which crisp parameters are converted to linguistic parameters using membership functions. The second and third steps are defining and evaluating fuzzy rules by applying fuzzy operators to the if-part, later applying the result to the then-part. This will result an output fuzzy set for each rule. Since there are several rules, the next step is applying an aggregation method so that the result of each rule can be combined and aggregated to a single fuzzy set. The final step is called defuzzification to convert the fuzzy set (from the aggregation step) to a single crisp value.

More precisely, the approach comprises the following phases:

i) Ranking parameters and the hierarchy of the fuzzy inference engines. Selection criteria are inputs to the fuzzy inference engines, which have a hierarchical structure. Let us start with the highest level of hierarchy: We consider two main criteria to rank cross-layer adaptation solutions: (i) the overall quality of the resultant service-based system, (ii) aggregated path value for the adaptation deployments in a solution. Those criteria depend on lower level aspects in the hierarchy. For the quality, we have time and cost dimensions. Thereby, the overall system quality relies on the process execution time, process execution cost and underlying infrastructure cost. Similarly, for the path value we consider the required time and cost for the deployment of a solution. Each of the dependence relation among parameters corresponds to a fuzzy inference engine. The hierarchy of the parameters can be seen in Figure 5.2. At the lower level of the hierarchy, we have two different fuzzy engines. While one of them serves for the inference of the aggregated system QoS, the other is used for inferring the path value for an alternative cross-layer adaptation solution. After deciding on these two parameters, at the higher level of the hierarchy a third engine is used to rank all the alternatives and come up with a final result.

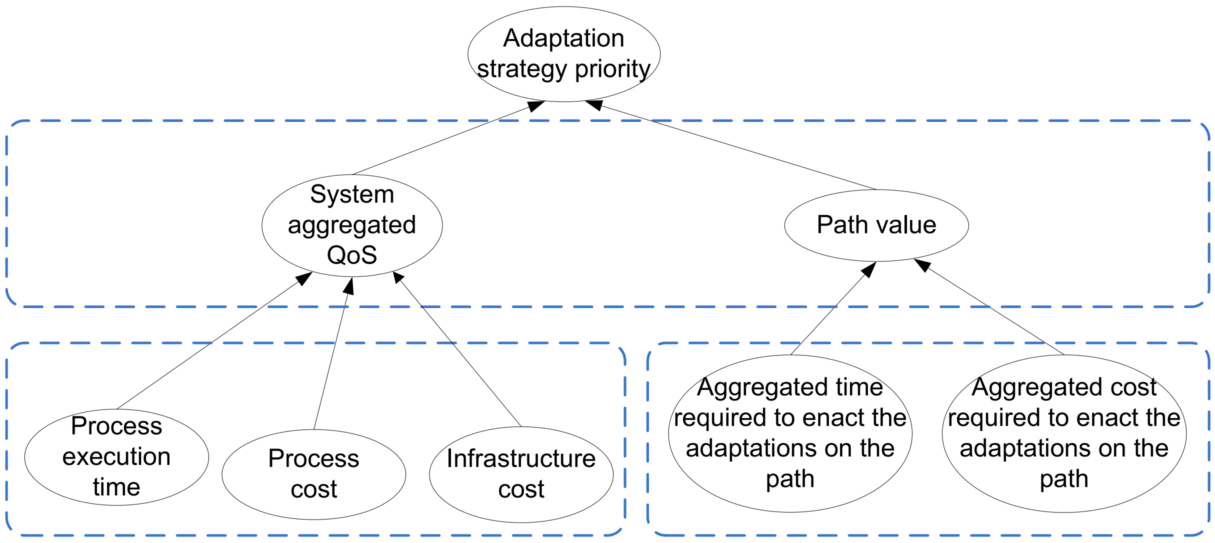

Adaptation strategy priority
System aggregated QoS
Path value
Process execution time
Process cost
Infrastructure cost
Aggregated time required to enact the adaptations on the path
Aggregated cost required to enact the adaptations on the path

Figure 5.2: Hierarchy of Fuzzy Inference Engines

ii) Assigning crisp values to the criteria. Fuzzy engines need crisp values of parameters as input to perform the fuzzification process. As discussed in the previous step, the ranking parameters are (i) overall adaptation deployment time, (ii) overall adaptation deployment cost, (iii) process execution time, (iv) process execution cost, and (v) underlying infrastructure cost. Their crisp values are obtained as described in Section 5.1.1.

iii) Creation of fuzzy parameters via membership functions. In order to operate our fuzzy ranker, all the input parameters need to be converted to fuzzy parameters. Therefore, we introduce linguistic parameters defined by fuzzy sets as follows: All the parameters are defined by three linguistic variables, however, they have different impression according to their membership function that we define later. For instance, the process execution time, process execution cost, infrastructure cost and aggregated QoS are defined by fuzzy variables as in the set $\{bad, normal, good\}$. In this set, obviously "good" is more preferable than the other fuzzy values. Similarly, the adaptation deployment time, adaptation deployment cost and the overall path value for deployment are defined by three linguistic variables as in the set $\{low, medium, high\}$. In this case, since we would like less cost and less time for an adaptation deployment, “low” is more preferable than the other fuzzy values. While we use the same set $\{low, medium, high\}$ for cross-layer adaptation value (final ranking), here differently “high” is more preferable than the others, therefore, at the end of the overall process we rank the highest cross-layer adaptation value as first.

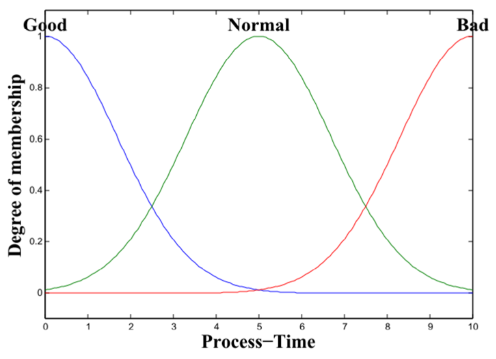
We define membership functions in order to understand the mathematical meaning of the linguistic variables. A fuzzy set represents the degree to which an element belongs to a set and it is characterized by membership function $\mu_{\tilde{A}}(x) : X \mapsto [0, 1]$. A fuzzy set $\tilde{A}$ in X is defined as a set of ordered pairs

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in X, \mu_{\tilde{A}}(x) \in [0, 1]\} \tag{5.3}$$

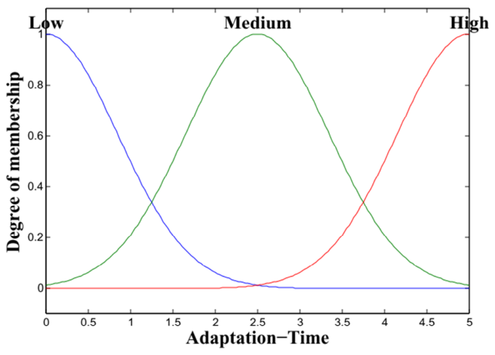
where $\mu_{\tilde{A}}(x)$ is the membership function of x in $\tilde{A}$. Therefore, a membership function shows the degree of affiliation of each parameter by mapping its values to a membership value between 0 and 1. We associate membership functions to a given fuzzy set to define the appropriate membership value of linguistic variables. We apply the Gaussian function to define membership functions for all parameters. The initial ranges of parameters are taken from a predefined contract (refer to [100] and [101] for further detail) and the shape of the functions are defined by experts of the system through experiments.

The membership functions are illustrated in Figure 5.3 for process execution time (5.3a), adaptation deployment time (5.3b) and adaptation deployment cost (5.3c) respectively. For instance, the adaptation deployment cost of each cross-layer adaptation path in our scenario is derived as described in Section 5.1.1 and then its membership value can be calculated using the membership function in Figure 5.3c.

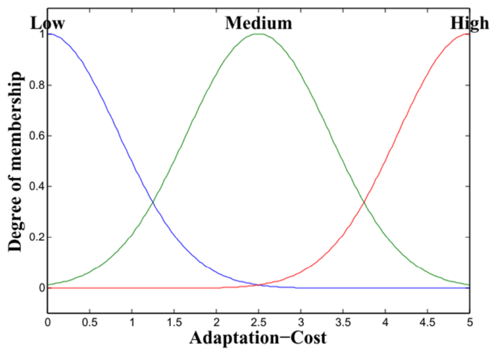
Note that the ranges of parameters are fixed in our approach and we do not perform optimization to tune the shape and range of membership functions. A well known optimization technique, which can also be used in our case, is to apply learning as proposed in [56].



Process Execution Time



Adaptation Deployment Time



Adaptation Deployment Cost

Figure 5.3: Membership functions for quality and adaptation deployment parameters

iv) Implication of fuzzy rules. Having defined the fuzzy parameters and their membership functions, we introduce fuzzy rules in order to find relations between parameters.

In our approach we use a hierarchical fuzzy system and apply three fuzzy inference engines according to Figure 5.2. Each inference engine should use a specific set of fuzzy rules. Therefore, we use three sets of fuzzy rules: (i) quality rules to assess the aggregated QoS, (ii) path rules to evaluate the value of different paths regarding their deployment efforts and finally (iii) ranking rules related to the final evaluation of the cross-layer adaptation alternatives, which we call adaptation strategies. In this way, the relation between aggregated QoS and the quality parameters can be defined using fuzzy rules and aggregated QoS value can be calculated using an inference mechanism. Similarly the overall path value is derived from the adaptation deployment time and cost of each path.

At the end, an overall ranking degree, i.e., adaptation value, can be derived from taking into account both adaptation deployment value and aggregated quality of the SBS.

In Figure 5.4 and 5.5 we present the quality and ranking rules defined by the expert of the system. Figure 5.4 shows a subset of the fuzzy quality rules that we use to measure the aggregated QoS from the involving quality criteria. For example the first rule, highlighted in the Figure 5.4, shows that when the process execution time is good, process execution cost is good and infrastructure cost is good, then the aggregated QoS is good. Note that path rules to evaluate deployment efforts are defined in a similar way.

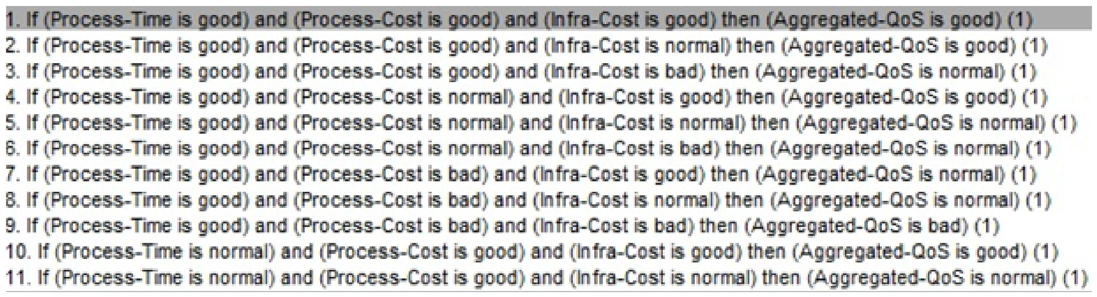

1. If (Process-Time is good) and (Process-Cost is good) and (Infra-Cost is good) then (Aggregated-QoS is good) (1)
2. If (Process-Time is good) and (Process-Cost is good) and (Infra-Cost is normal) then (Aggregated-QoS is good) (1)
3. If (Process-Time is good) and (Process-Cost is good) and (Infra-Cost is bad) then (Aggregated-QoS is normal) (1)
4. If (Process-Time is good) and (Process-Cost is normal) and (Infra-Cost is good) then (Aggregated-QoS is good) (1)
5. If (Process-Time is good) and (Process-Cost is normal) and (Infra-Cost is normal) then (Aggregated-QoS is normal) (1)
6. If (Process-Time is good) and (Process-Cost is normal) and (Infra-Cost is bad) then (Aggregated-QoS is normal) (1)
7. If (Process-Time is good) and (Process-Cost is bad) and (Infra-Cost is good) then (Aggregated-QoS is normal) (1)
8. If (Process-Time is good) and (Process-Cost is bad) and (Infra-Cost is normal) then (Aggregated-QoS is normal) (1)
9. If (Process-Time is good) and (Process-Cost is bad) and (Infra-Cost is bad) then (Aggregated-QoS is bad) (1)
10. If (Process-Time is normal) and (Process-Cost is good) and (Infra-Cost is good) then (Aggregated-QoS is good) (1)
11. If (Process-Time is normal) and (Process-Cost is good) and (Infra-Cost is normal) then (Aggregated-QoS is normal) (1)

Figure 5.4: Fuzzy rules to evaluate the aggregated QoS from quality criteria

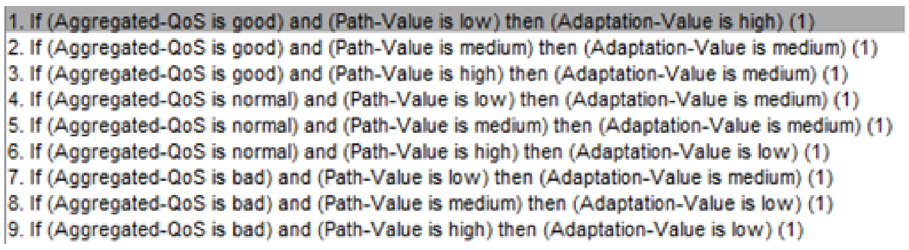

1. If (Aggregated-QoS is good) and (Path-Value is low) then (Adaptation-Value is high) (1)
2. If (Aggregated-QoS is good) and (Path-Value is medium) then (Adaptation-Value is medium) (1)
3. If (Aggregated-QoS is good) and (Path-Value is high) then (Adaptation-Value is medium) (1)
4. If (Aggregated-QoS is normal) and (Path-Value is low) then (Adaptation-Value is medium) (1)
5. If (Aggregated-QoS is normal) and (Path-Value is medium) then (Adaptation-Value is medium) (1)
6. If (Aggregated-QoS is normal) and (Path-Value is high) then (Adaptation-Value is low) (1)
7. If (Aggregated-QoS is bad) and (Path-Value is low) then (Adaptation-Value is medium) (1)
8. If (Aggregated-QoS is bad) and (Path-Value is medium) then (Adaptation-Value is low) (1)
9. If (Aggregated-QoS is bad) and (Path-Value is high) then (Adaptation-Value is low) (1)

Figure 5.5: Fuzzy rules to evaluate adaptation strategies in each path

Figure 5.5 shows the ranking rules that we use to measure the overall value of each cross-layer adaptation alternative. For example the rule highlighted in the Figure 5.5 expresses that when the aggregated QoS is good and the path-value is low, then the adaptation value is high, which means it is a preferable strategy to deploy among the alternatives. Note that the number at end of each rule in Figure 5.4 and 5.5 represents the rule's weight. Since we do not provide weight in this illustration, they are all indicated by number 1.

v) Aggregation of implication results and defuzzification Since there are several rules, the output fuzzy set of each rule is required to be aggregated to a single fuzzy set, which is called aggregation. The last step is defuzzification that converts the fuzzy set resulted from the aggregation to a single crisp value. Eventually, having a resultant crisp value for each cross-layer adaptation solution, we can rank them and deploy the highest ranked in the system.

5.2 Adaptation Deployment

As proposed by Definition 5.1, for each updated node in the system configuration, we associate a number of deployment tasks from the overall set of deployment tasks available for the SBS. We assume that we have a mechanism, which we call an "executor", to realize each deployment task in the system. Thus, given the selected cross-layer adaptation solution and its resultant deployment tasks, we can eventually enact this solution in the running system through the executors.

Definition 5.3 *(Executors)* Executors *$\mathcal{E}xec$ is a set of deployment mechanisms where each mechanism is called an executor $\mathcal{E}$ such that $\forall dt \in \mathcal{DT}$, there exists an $\mathcal{E}(dt)$.*

We remark that although deployment is not part of the cross-layer adaptation problem, as discussed above, our framework proposes a method to map a selected cross-layer adaptation solution to a set of deployment mechanisms, i.e., executors.

There have been several state-of-the-art approaches which can be used as executors in the cross-layer adaptation framework. [131, 15] propose techniques to migrate BPEL process instances. The authors of [131] present a migration data meta-model for enhancing existing processes with the ability for runtime migration. The approach permits the inclusion of intensions and privacy requirements of both process modelers and initiators and supports execution strategies for sequential and parallel execution of processes. Instead, [15] proposes a monitoring and recovery framework for the self-supervision of BPEL processes where the supervision-aware runtime environment enables the migration of updated processes. Among the approaches [22, 118], while [22] focuses on the automated generation of service-level agreements to address the changes in quality provisions due to negotiations, [118] tackles the problem of dynamic generation of monitors in case the SLAs are modified. Differently, [64, 116] present deployment mechanisms at the infrastructure level. [64] proposes an SLA-aware self-configuration approach in the cloud through autonomic service virtualization, and [116] introduces a phase-driven step-by-step strategy for migrating applications to the Amazon Web Services (AWS) Cloud.

5.3 Discussion

In this chapter, we dealt with the problem of selection and deployment of a cross-layer adaptation solution. Selection is not trivial considering the variety of solution alternatives in a CLAM tree. Initially, we presented our selection criteria, and subsequently we proposed two different ranking approaches, which both ground on the identified selection criteria, but apply different techniques for evaluating the ranks. Finally, we discussed the deployment issue.

The first approach presented, simple additive weighting - multiple criteria decision making, is a widely used technique in the field of service and service composition selection [6, 47, 54, 123]. However, these approaches consider only the QoS parameters as selection criteria.

Regarding the second approach, fuzzy logic has been applied to many fields (e.g. control and communication) for solving problems and decision making where the relation of existing parameters are complicated to be modeled by conventional mathematical models and techniques [101, 104, 66, 73]. However, similar to the previous case, the existing fuzzy logic approaches mostly focus on quality aspects of the system.

Notice that we can easily introduce new criteria to both techniques presented. The selection criteria can be decided based on the application domain. E.g. usually process KPIs are driving forces for the quality criteria. Moreover, the SBS owner should decide what kind of criteria are more important for ranking and selection. Such decisions affect the weights of each criterion in the overall evaluation. For instance, for a long running process it is an expected thing that the adaptation deployment time is not a very significant selection criterion.

Among the existing cross-layer adaptation approaches, only [123] and [103] propose a method to select an adaptation strategy. [123] considers the quality dimensions of the application and applies simple additive weighting - multiple criteria decision making like we do. Instead, [103] totally leaves out the system quality and considers only the metrics related to the length of a cross-layer adaptation pattern such as number of specific adaptation actions, number of general adaptation actions, number of total adaptation actions, and number of raised events in the pattern. Evidently, our proposed ranking methods surpass those approaches since we take into account various selection criteria, which in turn results in a more comprehensive ranking.

Chapter 6

Implementation and Design

The cross-layer adaptation approach and the SAW – MCDA adaptation selection technique presented in previous chapters were implemented as a prototype toolkit, namely CLAM Platform. The tool provides means for integrating in a common platform system analysis and adaptation capabilities, coordinating them for cross-layer adaptation, and selecting the best solution once all the alternative cross-layer adaptation solutions are identified by the coordination.

In this chapter we describe the CLAM platform and its implementation in detail. Moreover, we present a methodology to decide on the system elements, and subsequently create the overall model, which is required for CLAM operation. Finally, starting from the creation of the model, we guide the SBS owners in the concrete steps that they should follow in order to use our platform.

6.1 CLAM Platform

The CLAM platform presented here is implemented as a Java API and provides facilities for integrating and coordinating adaptation and analysis mechanisms in order to tackle an initial adaptation problem, and furthermore, allows for selecting a solution among alternatives.

The functional architecture of the CLAM Platform is presented in Figure 6.1. The operations in the platform pass several steps, starting from the CLAM in-

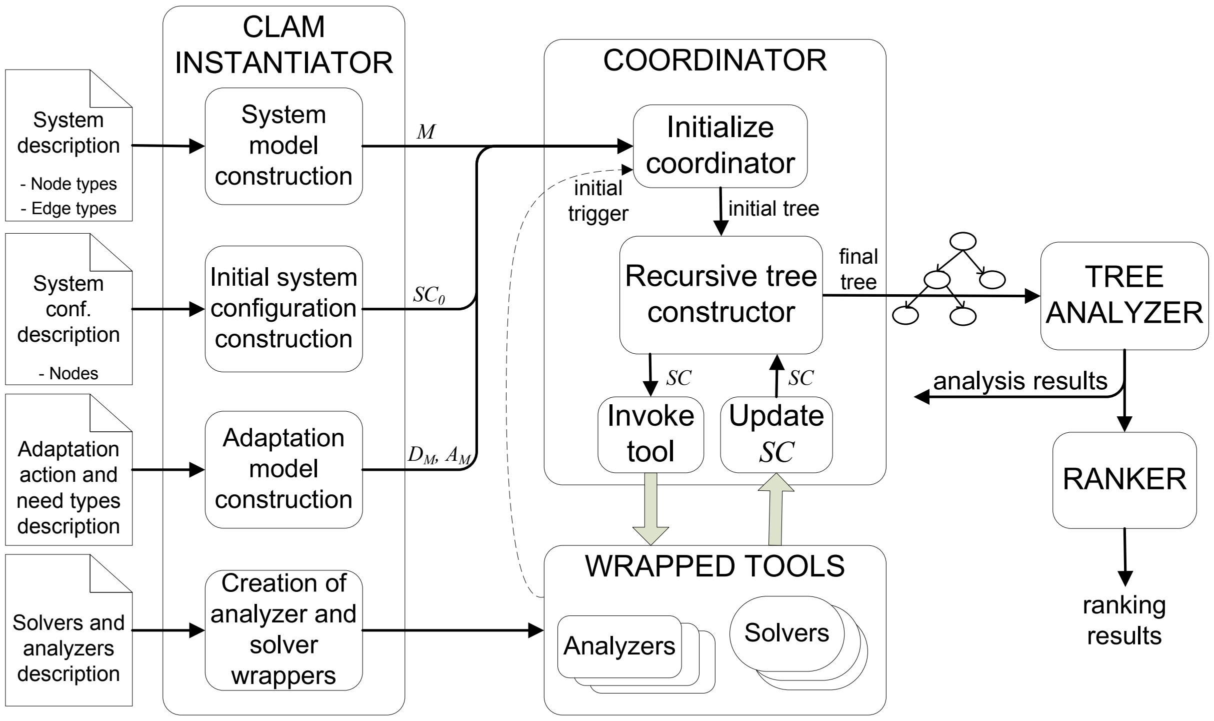


Figure 6.1: The Functional Architecture of CLAM Platform

stantiation to the coordination phase where there takes place an iterative analysis approach upon receiving an initial problem, and eventually to the selection phase where the produced results –cross-layer adaptation solutions– from the previous step are analyzed and ranked to select a final solution.

Input specification

The CLAM operation accepts the following input parameters:

- *System description.* We need to identify the elements of the system model, i.e., admissible node types of the service-based system and the relation types among them. Notice that this is a very high-level description of the system without referring to any concrete system parts, i.e., without including any implementation details.
- *System configuration description.* Once we have the system model, we can have its various instantiations, each corresponding to a specific deployment

which is running or ready to run. As discussed in previous chapters, we call each instantiation a system configuration and we need to specify the initial system configuration in order to enable the initialization of the CLAM operation. We remark that it is sufficient to identify the nodes of the system configuration. Next, relations among them can be derived from the system model. Notice that for each node, we must specify the links pointing to the locations of its implementation and/or configuration files.

- *Adaptation action and need types description.* The CLAM platform requires an extended system model which includes need and adaptation action types in the system. For each need type, we must identify a set of system nodes which it concerns, and similarly for each action type, we must identify a set of nodes which it affects. Furthermore, for each action type, we must also identify a set of deployment tasks together with the cost and the time values required for each deployment task.

- *Solvers and analyzers description.* Finally, we must identify the system tools, more precisely; the analyzers, i.e., monitoring and analysis mechanisms, and the solvers, i.e., adaptation capabilities that we want to coordinate in the CLAM platform. For each tool, we must specify the input and the output. For an analyzer, the input is a set of node types from the system model which the tool works on, and the output is a set of alternative need types that the tool can trigger in case of a negative analysis result. For a solver, the input is a need type, and the output is a set of alternative adaptation action types that the tool can produce to address the given need. Once tool inputs and outputs are described, they are integrated to the extended system model. However, this is not adequate. We must also define a wrapper function for each tool, which is responsible for extracting the required data from the system configuration, invoking the real tool with this data, and finally converting the tool response to the appropriate output format, i.e., need or adaptation action instantiations in the system configuration. Notice that wrapper functions enable the integration of tools with the CLAM execution environment, which in turn reflects the overall degree of adaptability of the service-based system.

CLAM execution: coordination phase

CLAM coordination phase is performed in several steps. Depending on the specified parameters, certain steps may be omitted or executed in different modes.

1. Inside the *CLAM instantiator module*, the set of input parameters –descriptions for the system, system configuration nodes, adaptation action and need types, and solvers and analyzers– are translated into the extended system model and the initial system configuration of this model. In this way adaptation model and tools are integrated in the overall system representation according to the definitions presented in Chapter 3. Moreover, the wrappers for analyzers and solvers are created based on their descriptions and the constructed models.

2. Once CLAM is instantiated with the required models, it is ready to get the initial trigger. Notice that even if the initial trigger is described as a monitored event coming from an *analyzer*, the implementation easily handles any trigger from an integrated tool. Thus, the initial trigger can be an adaptation action from a solver or an adaptation need from an analyzer. When we get the initial trigger, the *coordinator module* starts its execution by creating the initial CLAM tree, which is a root node including the initial system configuration and the queue with the tool that sends the initial trigger (see *initialize coordinator* in Figure 6.1).

3. After the initial tree is created, we are ready to recursively construct the CLAM tree inside the *recursive tree constructor module*, which is the core

part of the coordinator and implements the CLAM algorithm presented in Chapter 4. The tree construction corresponds to an iterative impact analysis of the initial trigger by checking in each step the required analyzers or solvers to invoke and each time we put them in the queue or delete them from the queue, we create a new tree node. Notice that tools are invoked through executing *invoke tool* method and their responses are reflected to the system configuration through *update SC* method. The iteration on a path terminates in the following cases: (i) the queue is emptied, there are no more tools to invoke, (ii) a solver could not find an adaptation action. However, we remark that there are configuration parameters for CLAM and some of them enforce the path termination even if neither of those mentioned cases hold. The configuration parameters, which must be set before the coordination starts, are as follows:

(a) *Tree traversal.* To construct the CLAM tree, the implementation supports both DFS and BFS traversal algorithms as well as a greedy search heuristic, presented in Chapter 7.

(b) *Stop condition.* Since in theory the CLAM algorithm does not terminate, we must put a stop condition to ensure that the tree construction is finalized. We have the following options: (i) stopping the iteration on a path when it reaches a maximum number of changes allowed in the system configuration with respect to a threshold defined in terms of a percentage of the total system size, (ii) stopping after the coordinator finds the first cross-layer adaptation solution. Note that the second case allows for finding a fast solution under BFS traversal, but impedes completeness, i.e., we are not producing all the possible solutions. Whereas, for the first case, if not all, we can still identify a good set of solutions by utilizing a heuristic method. We present in Chapter 7 the method which we apply.

(c) *Analyzer invocation.* Recalling the CLAM algorithm, to ensure completeness, we try all the possible permutations of analyzer invocations when we want to validate an adaptation action proposed by a solver. However, this leads to the enlargement of the tree size exponentially. Thereby, we introduce an optional parameter in which we can define a fixed order of analyzers and invoke them only with this order. This practice, similarly, impedes finding all the possible solutions.

CLAM execution: tree analysis phase

After the tree construction, the next phase is the tree analysis with the following steps:

1. The *tree analyzer module* receives the constructed CLAM tree and extracts the cross-layer adaptation solutions, which correspond to the tree paths ending with an empty queue.
2. After the extraction of solutions, we should understand what each solution signifies. This is achieved by evaluating each criterion introduced in Section 5.1.1:
   (a) *Adaptation deployment criteria.* The tree analyzer identifies the corresponding deployment tasks for each solution and based on them calculates the time and cost required to deploy the solution.
   (b) *Adaptation location criteria.* Once a solution path is extracted by determining the node differences between the initial and final system configuration, we can easily figure out which layers are included in the solution. Then the value of each path with respect to the adaptation location is calculated according to the Definition 5.2.
   (c) *System quality criteria.* Process execution time and the overall application cost are the system quality criteria which we consider in the

current version of the implementation. It is sufficient to take into account the final system configuration in order to calculate the values of these criteria.

3. Once all the criteria are evaluated for all the cross-layer adaptation solutions in the CLAM tree, the tree analyzer outputs the analysis results which comprise the entire set of criteria values for each solution. This output is fed also to the ranker module.

CLAM execution: adaptation selection phase

In the final phase of the CLAM execution, the *ranker module* takes the tree analysis results and produces an overall ranking of the solutions according to the SAW – MCDA technique presented in Section 5.1.2. Furthermore, it also publishes a separate ranked list of solutions with respect to the each criterion. The default selection of the cross-layer adaptation solution implies the first ranked solution in the overall ranking.

Notice that in current version of the CLAM platform, we have a definite set of selection criteria, each assigned with a default weight to calculate the overall value of a solution. However, this set of inputs may be extended in order to accept further selection criteria. Moreover, the weight of each criterion can be specified by the SBS owner. Similarly, at the final step, the SBS owner can ignore the default selection and takes his own decision by examining the tree analysis and ranking results.

6.2 Methodology for Cross-layer System Modeling

The operation of the coordinator in the CLAM platform relies on the continuous updates to the system configuration through tool invocations. Thus, the overall coordination tightly depends on the system model. Yet, it is not trivial how designers can decide on the elements of the system and what is the right level of abstraction to represent the service-based system comprehensive enough, but at the same time, not too intricate. Consequently, we want to guide designers in creating the system model so that they can reason out the right level of abstraction when they determine the elements of the system model.

We follow an opportunist methodology to decide on system elements and create the overall model: Given a service-based system and a set of tools available to analyze(monitor) and adapt this system, we create the model based on the tools available. There are two main benefits of this methodology. First, we identify the system elements more easily by looking at the inputs and outputs of the available tools. Second, we avoid putting unnecessary elements in the model, which do not have a corresponding tool, i.e., which would never be used by CLAM.

The main steps of the modeling methodology is as follows:

1) Identify the analyzers and solvers available in the system. As the primary step, we need to identify the existing system facilities, i.e., the monitoring, analysis and adaptation capabilities of the SBS.

2) Identify the inputs and outputs of analyzers and solvers. To derive the system element types, need and action types to be used at CLAM operation, we need to clarify what kind of information each system capability needs to operate on and what kind of output it produces.

3) Identify the system element types. Analyzer inputs reveal which parts of the system are under observation, and solver outputs reveal which parts of the system are modifiable. Hence, we can derive the system element types to be used by CLAM by looking at the analyzer inputs and the solver outputs.

4) Identify the relation types among system element types. Once the system element types to be used in the model get clear, the SBS domain expert should identify the relations among those element types.

5) Identify the need and action types. Each solver targets an adaptation problem, i.e., a need in the system. Therefore, we can identify a set of need types by looking at the problems targeted by solvers, and similarly, we can look at the outputs of solvers to identify a set of adaptation action types that can be coordinated in CLAM.

6) Associate the analyzer outputs with needs. A system analyzer reports whether a system constraint is violated or not. In case of a violation, this problem should be transferred to a set of solvers which can address this type of problem. Thus, we should map the analyzer outputs to the solver inputs, i.e., the needs.

6.3 CLAM User Guide

In this section we illustrate on our reference scenario "Call & Pay Taxi" – introduced in Section 3.2– the main steps to do in order to get CLAM working. We demonstrate how we can benefit from the modeling methodology we present in the previous section. Moreover, we describe the concrete actions to be performed in case we need to introduce new, additional tools to the CLAM platform.

6.3.1 Initial Setup of CLAM

Setting up CLAM requires that the system model is created and instantiated, i.e., the initial system configuration is specified, and the tools are integrated to the model and to the java execution environment. Let us see in more detail what are the main steps to get CLAM ready to work:

i. Create the model. Here we apply our opportunist modeling methodology to determine the system element types and eventually to create the system model.

1) Identify the analyzers and solvers available in the system. We start with the analysis and adaptation capabilities available for the "Call and Pay Taxi" application. As also introduced in Section 3.2, the application is equipped with the following state-of-the-art tools:

System analyzers: We use the *time and cost analyzers* [37] for the process KPIs. They take as input the BPEL file of the application and the execution times/costs of each process activity in the BPEL, then produce an aggregate value. Then the aggregate values can easily be compared with KPI target values to detect violations. We used the *data flow analyzer* [67] to check the data compatibility of the services newly introduced to the process.

System solvers: In our scenario we use as solvers a number of adaptation capabilities from different SBS layers. At the application layer, we use the *process optimizer* [105] to reduce the execution time of the process through possible task parallelizations, and the *data mismatch solver* [67] to resolve data incompatibility problems through mediators. We assume that from the beginning business analyst identifies the possible KPI margins for the application. At the service layer, we utilize the *service quality negotiator* [30] to negotiate the time and cost quality metrics of the partner services, and the *service replacer* [89] to replace the services in the process, which have poor performance or high cost. Finally, at the infrastructure layer we use the cloud for flexible resource allocation. We designed a custom *resource allocator* based on pre-selected Amazon EC2 Cloud, IBM Smart Cloud instances. We created profiles for time performance of these instances by relying on the benchmarking results presented in [49].

2) Identify the inputs and outputs of analyzers and solvers. After the identification of the tools, we examine what kind of information each tool needs to operate on and what kind of output it produces. The description of the tool interfaces in our scenario are presented in Table 6.1.

3) Identify the system element types. If we observe the analyzer inputs and the solver outputs in Table 6.1, we can easily figure out the system element types, which are analyzable and adaptable, i.e., which must be taken into considera-

Tool	Inputs	Outputs
Analyzer–Time	BPEL Process $\wedge$ process execution time KPI target $\wedge$ service execution times	OK $\vee$ NOK
Analyzer–Cost	BPEL Process $\wedge$ application cost KPI target $\wedge$ service costs	OK $\vee$ NOK
Analyzer–Data Flow	BPEL Process $\wedge$ service WSDL	OK $\vee$ NOK
Solver–Process Optimizer	old BPEL process	optimized new BPEL process
Solver–Data Mismatch	BPEL Process $\wedge$ incompatible service WSDL	a set of alternative mediator services
Solver–QoS Negotiator	old service execution times / old service execution costs	new service execution times / new service execution costs
Solver–Service Replacer	old services to be replaced by the faster / old services to be replaced by the cheaper	a set of new alternative faster services / a set of new alternative cheaper services
Solver–Resource Allocator	old infrastructure to be reconfigured for cost / old infrastructure to be reconfigured for time	a set of new alternative infrastructure configurations with less cost / a set of new alternative infrastructure configurations with faster response time

Table 6.1: The Tools Available in “Call & Pay Taxi” SBS

tion for the adaptation coordination in CLAM platform. From the presented tool data we infer the following types: process, time KPI, cost KPI, service, service execution time, service cost, infrastructure, infrastructure response time, infrastructure cost. Notice that this set of element types is smaller than the illustrated system model for “Call & Pay Taxi” scenario in Chapter 3. In that example, the system model has further element types such as process activity, service operation, service provider and infrastructure provider. This observation reconfirms that the methodology, which we follow, avoids putting unused element types in the model.

4) Identify the relation types among system element types. In this step the SBS domain expert must specify the relation types among the element types just identified in the previous step. At a first glance we can quickly see that time KPI and cost KPI are process-related element types, whereas, service execution time and service cost are service-related, and infrastructure response time and infrastructure cost are infrastructure-related. This observation, in turn, clarifies the system layers, namely process, service and infrastructure layers. From this starting point, we can specify more concretely the possible relations among element types, namely, “has” and “constrains” relations among element types of the same layer and "consumes" relations to address the inter-layer dependencies.

5) Identify the need and action types. The need and action types reflect the adaptation capabilities of solvers. So we can easily identify them by examining the solvers. Let us pick a solver from Table 6.1, e.g. Service Replacer. From its inputs we identify two needs: (i) replace service for cost and (ii) replace service for time, and actually the outputs, i.e., the adaptation actions just follow accordingly: (i) new alternative services to replace for cost (ii) new alternative services to replace for time. Notice that since the service replacer can work on two different problems, i.e., replacement for time or cost, in our platform, we treat it as two separate solvers being consistent with the solver description in Chapter 3 that each solver in CLAM works on a single need. We remark that once an action type is identified, we must also specify its deployment tasks. For instance, for service replacement, since we modify the process with the new partner service, we must migrate the process instances to the new version. Moreover we must create the SLA and the service monitor.

6) Associate the analyzer outputs with needs. Every analyzer checks a specific system constraint given a system configuration. In our example we have three analyzers, thus, three constraints: the process execution time constraint which is imposed by the time KPI, the SBS cost constraint which is similarly imposed by the cost KPI, and finally the data flow constraint which is imposed by the composition requirements of the process. To map the needs to the analyzer outputs, we go through the solvers and try to find out which aspect they target to improve. Let us consider the process optimizer. Since it performs parallelizations inside the workflow, it is obvious that it tries to improve time, so we can associate its need with the output of time analyzer. We perform the same exercise for each solver and map its need to the appropriate analyzer.

ii. Create initial system configuration. After we create the model –by following the steps above, next, we can create the initial system configuration: Being the instantiation of the system model, it keeps the references to the implementations of system element types. E.g. in case of process system element type, we will have the “taxiBPEL” node and we need to know where the running BPEL file is, similarly; for partner services the WSDL files, for quality attributes the signed SLAs. Note that we are not interested in how system configuration nodes are implemented in the running environment, what platform or technology is used. We only need to know and have access to the locations of the data required for analyzers and solvers. Notice that once nodes are identified, the relations among nodes can be easily inferred from the relation types identified in the system model.

iii. Create analyzer and solver wrappers. Finally, for each tool that we would like to integrate in the CLAM platform, we must create a wrapper. A wrapper function is responsible for the data conversion between the actual tool to be invoked and the CLAM platform. Let us pick an analyzer from Table 6.1, e.g. Time Analyzer. As mentioned in the table, it requires the BPEL file, execution times of partner services and the target value of the time KPI. This implies that when the wrapper function gets the relevant input nodes from the system configuration, i.e., process, time KPI and service execution times, it must pass to the tool real implementation data which are referenced inside the nodes. Then when the tool response is received, in case no violation is reported, the wrapper must send back just an “OK” to the coordinator. Otherwise, it must send back the set of alternative needs associated with the time analyzer.

Let us see also the case of a solver, e.g. Service Replacer. As we mentioned before, for CLAM platform it corresponds to two separate solvers. So we have one wrapper of Service Replacer for time and another wrapper for cost. Let us consider the one for time. Table 6.1 shows that it requires the current services involved in the BPEL. Similar to the analyzer case, this implies that the wrapper must pass to the tool the relevant data referenced inside service nodes of system configuration. Then when the tool response is received, in case no adaptation is found, i.e., no new service is identified for substitution, the wrapper must send back just a “NOK” to the coordinator. Otherwise, it must send back the set of alternative adaptation actions in the following format: The old system configuration nodes to be removed, the new system configuration nodes to be added. This implies that the service quality profiles, WSDLs and any other relevant data of these services must be referenced in the newly created nodes.

6.3.2 Updating CLAM with the New Tools

Adaptive systems are by nature dynamic systems, which in turn implies that throughout the SBS life cycle, we might need to furnish the system with new monitoring, analysis and adaptation capabilities. Let us see what we precisely need to do in case we would like to introduce a new tool to the CLAM platform:

i. Integrate the tool with the system model. We should follow the same order of steps proposed above to create the system model. Therefore we start with the identification of tool inputs and outputs. Based on that, we check if there are new system elements to be added in the system model to ensure the utilization of the tool in CLAM platform. If this is the case, we create the new system element and connect it to the relevant elements in the model. Next, if it is a solver, we must create the new need type which corresponds to its input, and the new action type which corresponds to its output. Moreover, we must go through the analyzers to associate the new need type with the relevant ones. Instead, if it is an analyzer, we must simply associate its output with a set of existing needs.

ii. Integrate the tool with the CLAM run-time environment. This step requires the creation of the tool wrapper. Once the wrapper is prepared, then the tool is ready to be invoked by the coordinator, as a result, it is ready to be included in the overall solution search.

6.3.3 Changing the System Model

Above we discussed extension of the system capabilities by introducing new tools to the CLAM platform. However, this is not the only case that we might need to update our platform. One other case, which is a more considerable alteration in the SBS life cycle, is the need to update the system itself. This might happen under circumstances that require more fundamental changes such as re-design of the system. Let us see what we precisely need to do in case we would like to change the system model by adding new system element type:

i. Update the system model. First of all, we insert the new system element type to the system model by determining properly its relations with the existing element types. Next, we identify the new analyzers and solvers associated with the element type. Then, similarly to the steps in Section 6.3.2 we integrate the new tools with the system model by defining the new need and adaptation action types for them.

ii. Update the system configuration. After we update the model, next, we can update the existing system configuration. Being an instantiation of the system model, the updated system configuration should contain the corresponding node instance for the new system element type, including the references to the implementation of the element type.

iii. Create wrappers of the new tools. Again, similarly to Section 6.3.2, this step necessitates the creation of wrappers, which is required to incorporate the new tools in the overall CLAM coordination.

Rather than adding a new system element type in case we would like to remove an existing element type from the system model, the procedure is more straightforward: we should remove the corresponding tools associated with this element type in the model as well as removing the relevant element instances from the system configuration.

6.4 Discussion

We presented the prototype implementation of the cross-layer adaptation framework, the CLAM platform. In CLAM platform we exploit separation of concerns, which is one of the main advantages of our approach making it flexible and extensible. The implemented coordination algorithm, namely the CLAM algorithm, does not depend on the application domain nor the specific implementation of tools.

Second, we introduced a methodology that proposes an easy way to determine the system elements and subsequently the overall system and adaptation models required for the operation of CLAM. We illustrated on our reference scenario the concrete steps that the designers must follow in order to utilize our platform. In addition to the initial set up, the presented user guide includes the must-do's for updating the CLAM platform with new tool integrations, and for changing the system model by adding new element types and/or removing some existing ones.

As previous sections showed, our opportunist modeling methodology, which proposes to start from the available tools, brings us the following benefits: (i) we identify the system elements more easily by looking at the required inputs and produced outputs of the available tools. (ii) we keep the model at the right level of abstraction, which means that we avoid putting unnecessary details in the model which would never be used by CLAM since there wouldn't be the corresponding tool.

We remark that one can create the models also starting from the system elements, and afterwards, determining the tools to be included. We will later analyze this case and compare it with our methodology in Chapter 7.

Eventually, it requires some efforts to set up CLAM, however, once we start using it, we have substantial advantages: (i) we get the overall adaptation impact analysis and the coordination in an automated way, (ii) we get an extensible run-time set-up that can easily accommodate new tools without interrupting an ongoing CLAM operation.

Chapter 7

Evaluation

In this chapter we present the evaluation of our approach. In order to evaluate the cross-layer adaptation framework, as well as the presented techniques and algorithms, we conducted a range of empirical research methods on the case study described in the thesis, namely "Call & Pay Taxi" application. In the evaluation we exploited the implemented CLAM platform together with a set of state-of-the-art analysis and adaptation tools integrated with the platform, which overall enabled to analyze an initial adaptation problem through the coordination of tools, identify the possible cross-layer adaptation solutions, and eventually rank them for selection.

The chapter discusses the results of the empirical work in three directions: *(i) Viability of the approach.* We used the prototype toolkit, i.e., the CLAM platform, to instantiate the cross-layer adaptation approach and the presented "simple additive weighting – multi criteria decision making" ranking method. The results of the experiments demonstrated the viability of our approach, and its contribution with respect to the existing works when we consider the variety of adaptation capabilities that can be available in the SBS. *(ii) Heuristics for optimization and algorithm termination.* We identified and implemented two novel heuristic methods to optimize the tree construction and to enforce the termination of the CLAM algorithm in practical context. *(iii) Contribution of the* *modeling methodology.* We evaluated the contribution of the proposed modeling methodology by changing the level of abstraction of the model for the same SBS, and calculating the design-time efforts required for each abstraction.

7.1 Experimental Set-up and a Sample Run

Here, we present the experimental set-up of the implemented CLAM platform on our reference scenario, "Call & Pay Taxi" application and demonstrate the output of a sample run of the CLAM platform. We use the system model, which we create based on the adaptation and analysis tools and the methodology presented in Section 6.3. Figure 7.1 depicts the consequent system and adaptation models for the scenario after following the methodological steps one by one.

As it can be seen from the displayed figure, we have in total 8 adaptation needs defined in the system, and correspondingly 8 solvers to tackle them. Each solver can propose diverse adaptation actions depending on the adaptation space it utilizes. For instance, let us consider the service replacer for time. The number of alternative adaptations which it can propose depends on not only the service repository it exploits, but also the quality profiles of the available alternatives from the same service category. Thus, the service repository that a service replacer contacts underlies the adaptation space for this solver. Table 7.1 presents an illustrative list of adaptation spaces for the solvers of "Call & Pay Taxi" presented in Figure 7.1.

Given the adaptation space in Table 7.1 we created a case study where the process execution time is higher than the expected value, i.e., time KPI is violated. For this problem, CLAM coordinated the tools introduced in Figure 7.1 and, using BFS for tree construction, produced a tree of 1615 nodes with 8 successful paths, i.e., alternative cross-layer adaptation strategies. The experiment took around 2 minutes on a 4GB 2.53GHz Dual Core machine running Windows. 85% of the time was elapsed inside the tools.

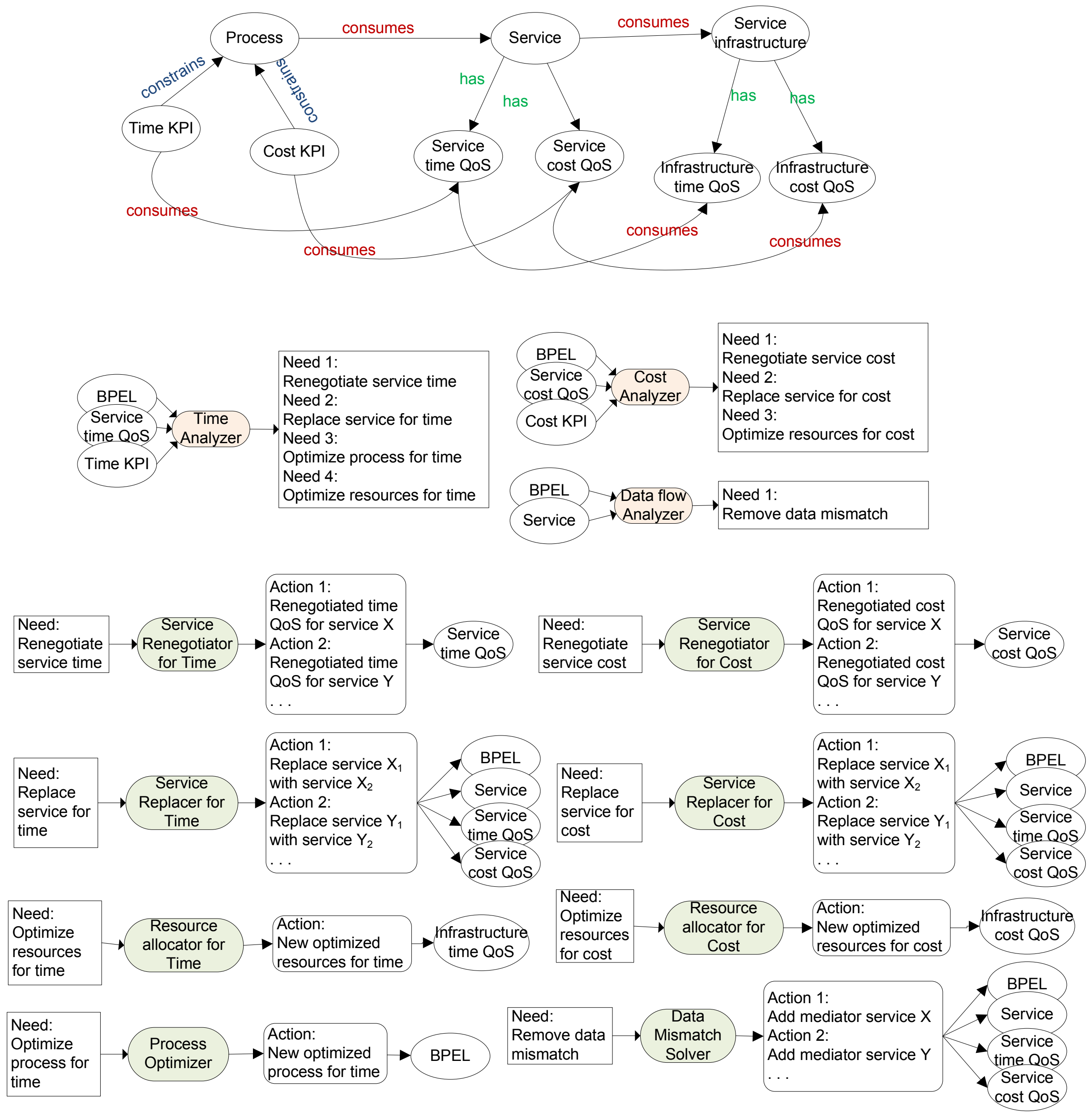

Process
consumes
Service
consumes
Service infrastructure
constrains
constrains
has
has
has
has
Time KPI
Cost KPI
Service time QoS
Service cost QoS
Infrastructure time QoS
Infrastructure cost QoS
consumes
consumes
consumes
consumes
BPEL
Service time QoS
Time KPI
Time Analyzer
Need 1: Renegotiate service time
Need 2: Replace service for time
Need 3: Optimize process for time
Need 4: Optimize resources for time
BPEL
Service cost QoS
Cost KPI
Cost Analyzer
Need 1: Renegotiate service cost
Need 2: Replace service for cost
Need 3: Optimize resources for cost
BPEL
Service
Data flow Analyzer
Need 1: Remove data mismatch
Need: Renegotiate service time
Service Renegotiator for Time
Action 1: Renegotiated time QoS for service X
Action 2: Renegotiated time QoS for service Y
. . .
Service time QoS
Need: Renegotiate service cost
Service Renegotiator for Cost
Action 1: Renegotiated cost QoS for service X
Action 2: Renegotiated cost QoS for service Y
. . .
Service cost QoS
Need: Replace service for time
Service Replacer for Time
Action 1: Replace service X_1 with service X_2
Action 2: Replace service Y_1 with service Y_2
. . .
BPEL
Service
Service time QoS
Service cost QoS
Need: Replace service for cost
Service Replacer for Cost
Action 1: Replace service X_1 with service X_2
Action 2: Replace service Y_1 with service Y_2
. . .
BPEL
Service
Service time QoS
Service cost QoS
Need: Optimize resources for time
Resource allocator for Time
Action: New optimized resources for time
Infrastructure time QoS
Need: Optimize resources for cost
Resource allocator for Cost
Action: New optimized resources for cost
Infrastructure cost QoS
Need: Optimize process for time
Process Optimizer
Action: New optimized process for time
BPEL
Need: Remove data mismatch
Data Mismatch Solver
Action 1: Add mediator service X
Action 2: Add mediator service Y
. . .
BPEL
Service
Service time QoS
Service cost QoS

Figure 7.1: The System and Adaptation Models for “Call & Pay Taxi” Scenario

Solver:	**Adaptation Space:**
Service QoS negotiator for time	service-taxiCoop.0 → service-taxiCoop.1 service-timSMS.0 → service-timSMS.1 service-vodafonePAY.0 → service-vodafonePAY.1
Service QoS negotiator for cost	service-TaxiTrento.0 → service-taxiTrento.2 service-timLBS.0 → service-timLBS.2 service-timPAY.0 → service-timPAY.2 service-vodafoneLBS.0 → service-vodafoneLBS.2 service-vodafoneSMS.0 → service-vodafoneSMS.2
Service replacer for time	service-taxiTrento → service-taxiCoop service-vodafoneLBS → service-timLBS service-vodafonePAY → service-timPAY service-vodafoneSMS → service-timSMS
Service replacer for cost	service-taxiCoop → service-taxiTrento service-timLBS → service-vodafoneLBS service-timPAY → service-vodafonePAY service-timSMS → service-vodafoneSMS
Resource allocator for time	infra-amazonEC2.0 → infra-amazonEC2.1
Resource allocator for cost	infra-amazonEC2.0 → infra-amazonEC2.2
Process optimizer	process-callAndPayTaxi.bpel.0 → process-callAndPayTaxi.bpel.1
Data mismatch solver	service-mediatorFull2Geo service-mediatorGeo2Full

Table 7.1: Adaptation Spaces for the Solvers used in the “Call & Pay Taxi” Scenario

Performance improvement. We remark that such a large number of tree nodes despite a relatively small adaptation search space is due to the fact that we are using permutations of analyzers to ensure completeness as already discussed in Section 4.2.2. However, in reality, for the given example the tree size should be much larger than 1615 since permutations introduce an exponential growth. We tackled this performance issue as follows: (i) We updated the CLAM algorithm such that as soon as a set of analyzers are identified, they are immediately invoked and only in case of problems they are added into the CLAM Queue and reflected as tree nodes in the CLAM tree. (ii) In fact, it is sufficient to generate the tool sequences of the queue where only queue head is important. This is because each time a solver proposes an adaptation, we are checking the affected analyzers and updating the queue. All in all this implies that for a queue of size n, we reduce the number of permutations from $n!$ to n.

Solutions for a sample run. Table 7.2 presents the paths, i.e., cross-layer adaptations, which CLAM identified for our sample run. Note that CLAM searches for the whole solution space through numerous iterations and organizes the paths in the end. It implies that it discards the adaptations that are rolled back during the iterations and eventually keeps the ones who make direct effects on the final system configuration with respect to the initial system configuration. This leads to obtain interesting results such as paths 5, 6, 7 and 8 in which there exist adaptations for improving cost although the initial problem was time and could be solved by actions performed only for this problem (see paths 1, 2, 3 and 4). In general unexpected solutions like paths 5, 6, 7 and 8 optimize overall system quality considering both time and cost at the same time, but, entailing many adaptation actions they might require too much effort to be deployed in the running SBS. In the end, it is the SBS owner's decision what kind of path he prefers. If the SBS owner is not very concerned about optimizing the system

Path id	Final System Configuration	Triggered Adaptations	Required Enactment Efforts	Adaptation Location	Adaptation Deployment Cost	Adaptation Deployment Time	Total System Cost	Process Cycle Time	Aggregated Normalized Path Value
path 1	CallAndPayTaxi-opt.bpel, taxiTrento.0, timLBS.0, timSMS.1, timPAY.0, amazonEC2.0	negotiated service time for timSMS; new optimized process	modify service monitor; modify service SLA; migrate process; create new machine image	application and service layers	6.0	2.0	31.7	19.7	0.146 (BEST RANKED)
path 2	CallAndPayTaxi-opt.bpel, taxiTrento.0, timLBS.0, timSMS.0, timPAY.0, amazonEC2.0	new optimized process	migrate process; create new machine image	application layer	4.0 (BEST DEPLOYMENT COST)	2.0 (BEST DEPLOYMENT TIME)	31.7	20.3	0.137
path 3	CallAndPayTaxi-opt.bpel, taxiTrento.0, timLBS.0, timSMS.1, timPAY.0, amazonEC2.1	new resources for time; negotiated service time; new optimized process	modify service monitor; modify service SLA; migrate process; create new machine image; migrate machine image	application, service and infrastructure layers	8.0	4.0	31.7	19.3 (BEST SC TIME)	0.129
path 4	CallAndPayTaxi-opt.bpel, taxiTrento.0, timLBS.0, timSMS.0, timPAY.0, amazonEC2.1	new resources for time; new optimized process	migrate process; create new machine image; migrate machine image	application and infrastructure layers	6.0	4.0	31.7	19.9	0.122
path 5	CallAndPayTaxi-opt.bpel, taxiTrento.0, timLBS.2, vodafoneSMS.0, timPAY.2, amazonEC2.1	new resources for time; replaced service for cost; negotiated service cost; new optimized process	modify service monitor; modify service SLA; create new service monitor; sign new service SLA; migrate process; create new machine image; migrate machine image	application, service and infrastructure layers	14.0	6.0	30.7	20.09	0.101
path 6	CallAndPayTaxi-opt.bpel, taxiTrento.0, timLBS.2, vodafoneSMS.0, vodafonePAY.1, amazonEC2.0	replaced service for cost; replaced service for time; negotiated service cost; negotiated service time; new optimized process	modify service monitor; modify service SLA; create new service monitor; sign new service SLA; migrate process; create new machine image	application and service layers	14.0	6.0	30.8	20.2	0.091
path 7	CallAndPayTaxi-opt.bpel, taxiTrento.0, timLBS.2, vodafoneSMS.0, vodafonePAY.1, amazonEC2.2	replaced service for cost; new resources for cost; negotiated service cost; negotiated service time; new optimized process	modify service monitor; modify service SLA; create new service monitor; sign new service SLA; migrate process; create new machine image; migrate machine image	application, service and infrastructure layers	16.0	8.0	30.5 (BEST SC COST)	20.2	0.079
path 8	CallAndPayTaxi-opt.bpel, taxiTrento.0, timLBS.2, vodafoneSMS.0, vodafonePAY.1, amazonEC2.1	replaced service for cost; new resources for time; negotiated service cost; negotiated service time; new optimized process	modify service monitor; modify service SLA; create new service monitor; sign new service SLA; migrate process; create new machine image; migrate machine image	application, service and infrastructure layers	16.0	8.0	30.8	19.8	0.075

Table 7.2: Cross-layer Adaptation Paths for a Sample Run

quality, he could go for the simplest path which is not costly to deploy in the system, i.e., path 2.

Moreover, the presented paths demonstrate two important characteristics of our approach: First, it is able to identify the combinations of alternative adaptations to solve a single problem. Paths 1, 3 and 4 illustrate such cases that they propose more than one action to improve the time performance of the process. Second, CLAM is capable of tackling new problems triggered on the path due to the proposed adaptations. Paths 5, 6, 7 and 8 are the examples in which new cost problem is arisen and addressed during the iterations. We remark that, as a consequence of our flexible, on-the-fly cross-layer adaptation detection, these are important aspects distinguishing our approach from the existing works. E.g. the pattern-based approach proposed in [103] requires the description of cross-layer adaptation patterns at design time, which in turn implies that the approach cannot catch an unexpected problem, which might originate from a proposed cross-layer adaptation. Moreover, in the case that such problem patterns are envisaged at design time, (i) it will be very costly to go through all the possible interactions among tools, and (ii) it will not be realistic for an open adaptive system since the adaptations proposed by solvers may vary by time as well as the set of solvers and analyzers used in the system.

7.2 Heuristic Methods for Search Optimization and Termination

Regarding the implementation of our framework, we addressed two practical issues through heuristic methods and evaluated the proposed methods through experiments. First, we optimized the tree construction by designing a greedy search algorithm [125], which investigates the tree nodes that seem to be most likely to reach a cross-layer adaptation solution. Second, we devised a principled way for termination in practical settings, which relies on our primary intuition that we do not want to deploy a cross-layer adaptation, which brings the system to a very different configuration with respect to the initial configuration.

7.2.1 Optimization Heuristics

In our optimized search algorithm, as being greedy, the node selection for expanding the tree is based on the goal-distance function that is an admissible "heuristic estimate" of the distance from the current node to the goal, i.e., for the given tree node, how close we are to the solution. In fact, It is not straightforward to determine a promising function for the heuristic estimate of the distance to reach the goal. In this dissertation, we present a novel optimization approach to search and find solutions faster compared to the well known tree traversal algorithms such as DFS and BFS.

The main idea behind is to be able to utilize as much information as possible so that our algorithm performs an informed search rather than picking up an arbitrary node to continue. At this point it is obvious that we should turn our head towards the system and adaptation models because they are the only ones that keep information about tool behaviors. Especially we concentrate on solver behaviors since they are the actual builders of a cross-layer adaptation solution. At a qualitative level we can extract the following information from the models: We know that each analyzer works on a single system constraint and produces a set of alternative needs to address a constraint violation. Thus, we know which constraint problem is solved when a solver addresses a need, which in turn implies at best case a solver can solve the given problem. However, as discussed throughout the thesis, solvers might have negative effects on other system constraints. Again, from the model, we know the associated system parts where a solver work on, and similarly the parts on which a constraint is imposed, i.e., the analyzer inputs. Thus, from this bunch of information, we can easily extract the best case and worst case consequent effects of solvers on the system.

Table 7.3 depicts the solver behaviors for our reference scenario. It displays for each solver the set of affected analyzers and the analysis of effects if they are positive or negative. For all the unaffected analyzers, the effect is neutral. Using Table 7.3 we can calculate optimistic and pessimistic bounds on a CLAM queue size at a given tree node. At optimistic case, the solver addresses the problem and does not touch the other constraints, so the queue is reduced by 1. At pessimistic case, even if the solver proposes an adaptation, it is not enough to solve the problem, moreover it negatively affects all its associated analyzers, so the queue is increased by the number of negatively affected analyzers. The resultant bound calculations are given in Table 7.4.

SOLVER:	**List of triggered analyzers:**	**Impact on time:**	**Impact on cost:**	**Impact on data flow:**
Process optimizer	time analyzer	positive	neutral	neutral
Resource optimizer for time	time analyzer	positive	neutral	neutral
Resource releaser for cost	cost analyzer	neutral	positive	neutral
Service time QoS negotiator	time analyzer	positive	neutral	neutral
Service cost QoS negotiator	cost analyzer	neutral	positive	neutral
Data Mismatch solver	time, cost, data flow analyzers	negative	negative	positive
Service Replacer for time	time, cost, data flow analyzers	positive	negative	negative
Service Replacer for cost	time, cost, data flow analyzers	negative	positive	negative

Table 7.3: Qualitative Analysis of Effects of Solvers on the System

SOLVER:	**Optimistic bound on CLAM Queue size:**	**Pessimistic bound on CLAM Queue size:**
Process optimizer	Q - 1	Q
Resource optimizer for time	Q - 1	Q
Resource releaser for cost	Q - 1	Q
Service time QoS negotiator	Q - 1	Q
Service cost QoS negotiator	Q - 1	Q
Data Mismatch solver	Q - 1	Q +2
Service Replacer for time	Q - 1	Q +2
Service Replacer for cost	Q - 1	Q +2

Table 7.4: Calculation of Optimistic and Pessimistic Bounds of a CLAM Queue based on Solver Behaviors

Optimized Search Algorithm. Figure 7.2 presents the greedy search algorithm that we have implemented based on the optimistic and pessimistic bounds in-

```
function pickBestLeaf(treeType t)
  nodeType best := null
  List<node> leaves := t.getLeafNodes()
  if leaves == null || leaves.isEmpty()
    return null
  forall leaf in leaves
    if !leaf.getQueue().hasSolver()
      return leaf
    if best == null
      best = leaf
    elseif leaf.getOptimisticBound() <= best.getOptimisticBound() &&
           leaf.getPessimisticBound() < best.getPessimisticBound()
      best = leaf
    elseif leaf.getOptimisticBound() < best.getOptimisticBound() &&
           leaf.getPessimisticBound() <= best.getPessimisticBound()
      best = leaf
    elseif leaf.getOptimisticBound() == best.getOptimisticBound() &&
           leaf.getPessimisticBound() == best.getPessimisticBound() &&
           leaf.getCurrProblemList() < best.getCurrProblemList()
      best = leaf
    elseif leaf.getOptimisticBound() == best.getOptimisticBound() &&
           leaf.getPessimisticBound() == best.getPessimisticBound() &&
           leaf.getDistanceFromCriteria() < best.getDistanceFromCriteria()
      best = leaf
  return best
```

Figure 7.2: Greedy Search Algorithm.

troduced above. It first expands all the tree nodes which do not have any solvers in the queue, i.e. the nodes with analyzer queues (lines 7-8). Once we have only the tree nodes in which we have a solver to invoke, we start utilizing the information we have in hand to select the best leaf. We go through all the leaves (line 6), and first, we compare the optimistic and pessimistic bounds and we try to find the leaf minimizing both values (lines 11-14). If we find a leaf having the same bounds with the current "best", in order to break the ties we check the current number of analyzers in their queues, which correspond to current number of problems to be solved (lines 15-16). Finally, if also the current number of problems are equal, we apply tie breaking for a second time, this time using the overall quantitative distance from the target constraint values such as KPI target for process cycle time and KPI target for overall application cost (lines 17-18). Notice that we have the third system constraint in our reference scenario, which is the data flow compatibility. Naturally, this type of constraints take boolean values instead of numeric values. We assume that the distance is 1 when the boolean value is $false$. Furthermore, when we calculate the overall distance from the target values, we tackle this kind of heterogeneities by the normalization of values.

Comparison with BFS and DFS. We conducted two experiments to see the contribution of our greedy search algorithm compared to the traditional tree traversal algorithms such as BFS and DFS. For both experiments, we used the same SBS presented in Figure 7.1 and the same adaptation search space introduced in Table 7.1. In the first scenario we used the same adaptation case study introduced in the previous section where the process cycle time KPI is violated and all the other system constraints are OK. In the second scenario we generated a different problem where the overall application cost KPi is violated and all the other system constraints are OK:

Scenario 1. There is a time problem in the system. The initial values are as follows:

Current application cost:	31.7
Current process cycle time:	26.1
Current data compatibility status:	$true$
Max application cost allowed:	32
Max process cycle time allowed:	20.3
Data compatibility must be:	$true$

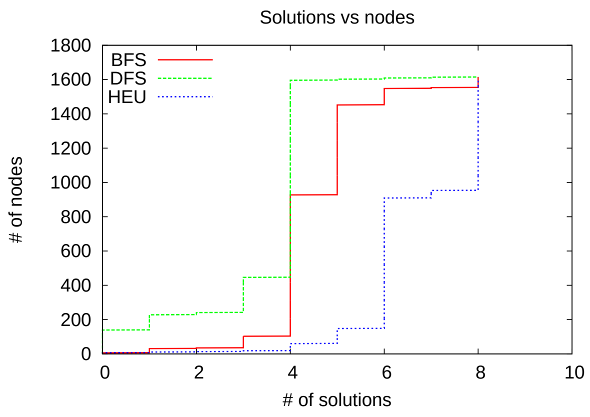
Scenario 2. There is a cost problem in the system. The initial values are as follows:

Current application cost:	32.5
Current process cycle time:	19.5
Current data compatibility status:	$true$
Max application cost allowed:	32
Max process cycle time allowed:	20.3
Data compatibility must be:	$true$

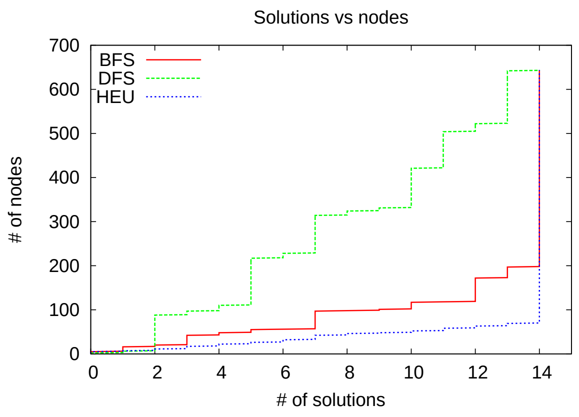
Figure 7.3 presents the results of experiments. As it can clearly be seen from both experiments, our proposed greedy algorithm, taking advantage of performing an informed search, finds the solutions much earlier compared to both BFS and DFS algorithms. This observation validates that the information that we feed to the search algorithm is indeed meaningful and helps optimize the search.

7.2.2 Termination Heuristics

Previously, in Section 4.2.2 we discussed the termination of CLAM algorithm and concluded that the algorithm does not terminate from the theoretical point



Scenario 1



Scenario 2

Figure 7.3: Comparison of greedy search with BFS and DFS algorithms.

of view. This is due to the fact that we might have infinitely many adaptations proposed by solvers, and infinitely many ways to configure the system even with a finite set of solvers, even with a finite set of actions proposed by a solver. One illustrative solver would be a BPEL reconstructor which is adding an exception handler in front of an activity. So, each time a new BPEL activity is observed, the solver adds a new exception handler. Suppose that we have a second solver, which is parallelizing the process activities. When we continuously apply these two solvers one after the other, we can generate infinitely many versions of BPEL by adding an exception handler, then putting it in parallel, then adding a new exception handler in front of the new parallel activity and so on.

In a practical setting, it is not feasible continuously modifying a BPEL process to address an adaptation problem. Moreover, it is not feasible to replace a big part of the system configuration either: We do not want to introduce new better services, new faster infrastructures to address a local adaptation problem. We would like find the most reasonable solution to an adaptation problem, which will not imply a continuous modification of a system part nor a big revision of the overall system.

The principle of our termination method leans on the fact that we do not want to bring the system to a final configuration which is very different from the initial system configuration, because eventually we want to adapt the system, not to convert it into a totally new system as a result of a chain of adaptation actions. Such cases indeed imply a huge amount of efforts for the adaptation deployment, which obviously turns out to be comparable to the amount of efforts we would require for a re-design. Therefore, we propose a heuristic method to measure and control the number and distribution of changes imposed on the system due to a cross-layer adaptation. In this way, when we visit a tree node, if we identify that we exceed the threshold, which is defined on the basis of system configuration size and its maximum percentage that we permit for adaptation, we stop the tree expansion for that node.

Termination threshold. We consider the required efforts to design a system, at a very high level, is equivalent to the size of initial system configuration. We assume that if we have a system with $2n$ elements, it is more costly to design and deploy it compared to system of size n. Our main motivation to specify a termination threshold in CLAM iterations exactly bases upon this universal threshold, which definitely we do not want to do actions which are equivalent to re-design in terms of the amount of efforts. To be more realistic, we propose a parameter to specify the threshold, which is the percentage of the initial system configuration size, $|\mathcal{SC}_0|$. This parameter can be fixed by the domain expert to a reasonable value at design-time given the complexity and dynamicity of the system.

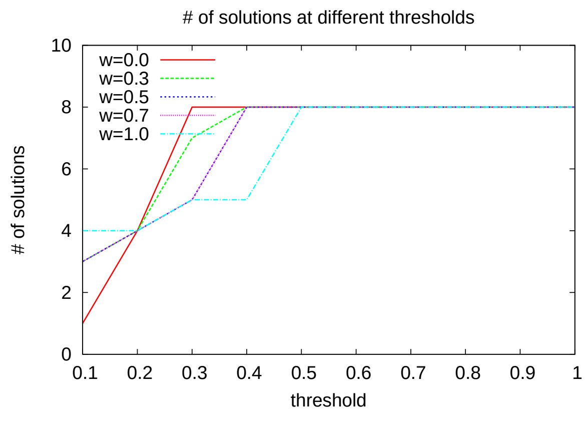
Node acceptability. Intuitively, a cross-layer adaptation which brings the system from size n to size $2n$ is not acceptable since each node addition counts for increasing the initial size of the system, i.e., $|\mathcal{SC}_0|$, by a factor of $1/|\mathcal{SC}_0|$ and in the end we would arrive at the same amount of efforts required for a system design. Similarly, replacing a node or removing a node are significant changes in a system configuration, so they have the same effects like adding a node. Whereas, modifying a node is in general a less significant change for a system configuration, so it should be handled in a different way. Thus, we propose to assign weights to each action type when we are summing up all the actions required to reach a final system configuration from an initial one.

Moreover there is another aspect that we should take into account. Not only the total number of changes matters, but also we should consider the node distribution of changes. Definitely, modifying the same node three times is not equal to modifying three distinct nodes, each of them just once. Therefore, we aggregate these two metrics into a single metric that we call "node acceptability". Then, each time we are visiting a node, we can calculate its acceptability and stop the expansion at that node if it has reached the termination threshold.

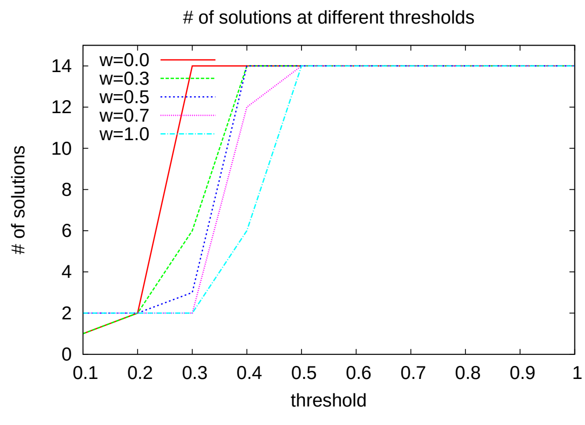
Definition 7.1 *(Node Acceptability) A node v in the CLAM tree T is* acceptable *if and only if its acceptability $accept_v$ is smaller than the termination threshold $Threshold$ where*

- $accept_v = W_1 \Delta\mathcal{SC}_v + W_2 numChange_v$ *such that* $W_1, W_2 \in \mathbb{R}$ *and* $W_1 + W_2 = 1.0$*;*
- $\Delta\mathcal{SC}_v$*, the* $\mathcal{SC}$ *distance factor, is the weighted sum of* $\{$*add, remove, replace, modify*$\}$ *actions to arrive from* $\mathcal{SC}_0$ *to* $\mathcal{SC}_v$ *and the weight for each action, represented by* w_a*, is a real number such that* $w_a \in [0, 1]$*;*
- $numChange_v$ *is the change distribution factor which denotes the total number of changed nodes in* $\mathcal{SC}_v$ *with respect to the* $\mathcal{SC}_0$*;*
- $Threshold = W_{thr}|\mathcal{SC}_0|$ *where* W_{thr} *is a real number such that* $W_{thr} \in [0, 1]$ *and it denotes the maximum percentage of the system that we allow for adaptation.*

In our experiments while we tried different value sets for the parameters W_1, W_2 and W_{thr}, we fixed the weights of actions to calculate $\Delta\mathcal{SC}$. For each *add,*



Scenario 1



Scenario 2

Figure 7.4: Effect of variation of $\Delta\mathcal{SC}$ weight on the number of solutions found at varying termination thresholds.

remove and *replace* action we assigned 1 as weight, instead for each *modify* action we assigned 0.5 since they impose less significant changes on the system compared to the other actions as also discussed above. We remark that one can have a more elaborate weight list for different adaptation actions, e.g., one can distinguish between adding a service node and adding an infrastructure node. Such refinements can easily be handled by our approach.

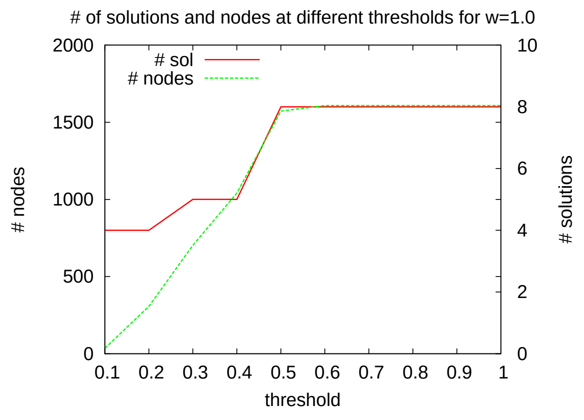
Let us consider the experiments that we conducted to evaluate our termination heuristics. We remark that for all the experiments, we used the two scenarios, **Scenario 1** and **Scenario 2**, which we already introduced in the previous section.

The first concern we would like to observe is the variation of results when we consider different weights for the $\mathcal{SC}$ distance factor $\Delta\mathcal{SC}$ and the change distribution factor $numChange$. Figure 7.4 depicts the experiment results where we tried different variations of W_1, W_2 and W_{thr}. As it can be seen from the experiment results, when we totally ignore $\Delta\mathcal{SC}$ factor, i.e., it has a 0 weight, then, only relying on the change distribution factor, we do not prune solutions at high thresholds since even if many changes are made to the system configuration, roughly, we only consider the number of changed nodes. So in that case, we start pruning solutions much later only around 30% of the system configuration size. However, on the contrary, when we consider lower percentages such as 10%, we see that "applying purely $numChange$ factor" prunes more solutions than "applying purely $\Delta\mathcal{SC}$ factor". This is because when we consider smaller thresholds $numChange$ might be overestimating the meaning of the changes. More precisely, it will treat a modification equally as it treats replacement, addition or removal of nodes. For instance, we have a system configuration of 20 nodes and we have the threshold 10%. If we perform 3 modifications in 3 nodes, "applying purely $\Delta\mathcal{SC}$ factor" would keep the solution since the acceptability would imply 3 times the weight of modification action, which is 0.5, so the node acceptability value would be 1.5. On the other hand, "applying purely $numChange$ factor" would discard the solution since the acceptability would imply the number of changed nodes, which is 3.

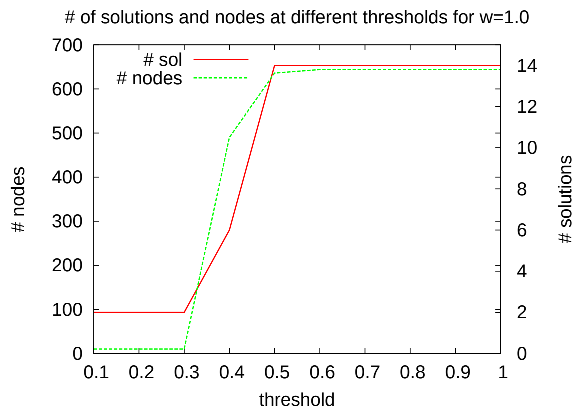
From this observation, we conclude that $\mathcal{SC}$ distance factor $\Delta\mathcal{SC}$ proposes a more refined way of understanding changes in the system. However, one can still give more weight to $numChange$ if the localization of adaptations is of highest importance.

Our second concern is to understand if we are really pruning the bad solutions and what remains is really a good set of solutions at low thresholds. We will use purely $\Delta\mathcal{SC}$ factor since it is more refined, and pruning the solution more gradually compared to $numChange$ factor.

Figure 7.5 presents the results when we observe the pruned tree nodes and solutions at varying termination thresholds. Let us consider the 10% threshold. In the first scenario we keep 4 solutions out of 8 possible solutions and in the second we keep 2 solutions out of 14. We should understand if the remaining solutions are actually the preferable ones that we have intended to keep. Table 7.5 presents the remaining solutions of Scenario 1 and Table 7.6 presents the remaining solutions of Scenario 2. Both tables demonstrate that we find the



Scenario 1



Scenario 2

Figure 7.5: Observation of pruned tree nodes and solutions at varying termination thresholds when only $\Delta\mathcal{SC}$ weight is applied.

Solutions:	Tree size at the solution node:	Distance from initial SC / threshold	Nodes affected from adaptation:	Final SC cost:	Final SC time:	Overall rank among the whole set of 8 solutions:
solution#1	5	1.0 (2 modify) / 1.8	process, service time QoS	31.7	19.7	1st
solution#2	9	1.5 (3 modify) / 1.8	process, service time QoS, infra time QoS	31.7	19.3	3rd
solution#3	17	1.0 (2 modify) / 1.8	process, infra time QoS	31.7	19.9	4th
solution#4	20	0.5 (1 modify) / 1.8	process	31.7	20.3	2nd

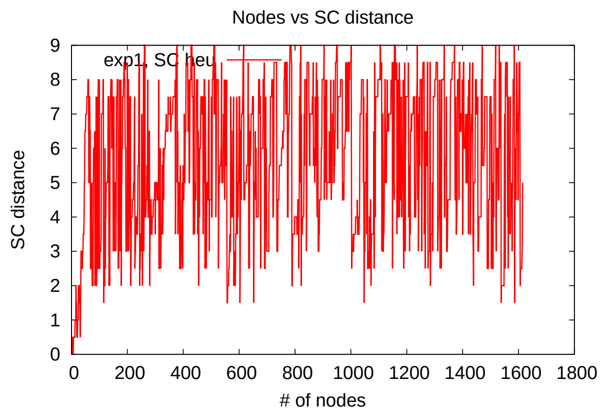
Table 7.5: Pruned solutions of Scenario 1 when $10\%|\mathcal{SC}_0|$ threshold is applied

Solutions:	Tree size at the solution node:	Distance from initial SC / threshold	Nodes affected from adaptation:	Final SC cost:	Final SC time:	Overall rank among the whole set of 8 solutions:
solution#1	5	1.5 (3 modify) / 1.8	2 service cost QoS + infra cost QoS	31.6	19.5	2nd
solution#2	8	1.0 (2 modify) / 1.8	2 service cost QoS	31.9	19.5	1st

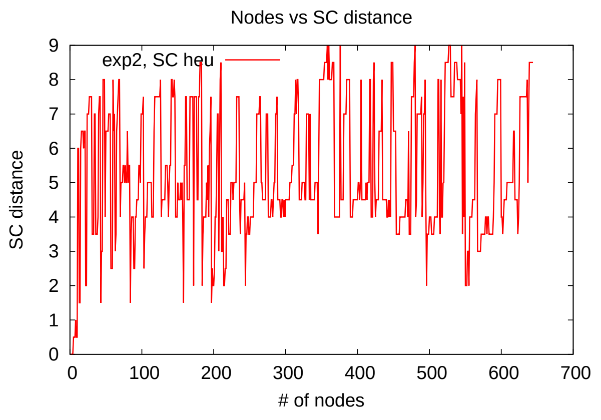
Table 7.6: Pruned solutions of Scenario 2 when $10\%|\mathcal{SC}_0|$ threshold is applied

minimum $\Delta\mathcal{SC}$ solutions which are based on modification actions and much simpler paths compared to the discarded solutions. If we again have a look at the full solution list for Scenario 1 (Table 7.2), we can easily see that the pruning keeps the first 4 highly ranked paths and discarded the worse ones with respect to our overall adaptation selection criteria. The discarded solutions are indeed bad since they include service replacements, which are definitely more costly than optimizing the process or reconfiguring the infrastructure. This is an expected result since we have a fine-grained evaluation of $\mathcal{SC}$ distance which is aligned with adaptation selection criteria especially considering the adaptation deployment efforts. Moreover, the presented results show once more that our optimized search algorithm works very efficiently since it finds the solutions very early. In addition, from the plots it can be seen that at 10% termination threshold, the final tree size is reduced from around 1600 nodes to less than 40 nodes in Scenario 1, and from around 600 nodes to less than 20 nodes in Scenario 2. This is a motivating result to consider the use of our approach online in run-time adaptation scenarios.

One other observation that we deduce from Figure 7.5 is the relation between the number of pruned tree nodes and the number of pruned solutions when we vary the termination threshold. Intuitively as we decrease the threshold, we expect that we start pruning the tree before we start pruning the solutions. So,



Scenario 1



Scenario 2

Figure 7.6: Variation of $\Delta\mathcal{SC}$ throughout the tree expansion when $\Delta\mathcal{SC}$ driven search heuristic is applied.

there should be a range for the threshold value in which we decrease the tree size but still we do not lose any solutions. However, in our plots we observe that when we are going down from 100% to ~50%, we do not prune neither the tree nor the solutions, then almost at the same threshold we start pruning them both. More precisely, experiment results report that there does not exist a range for the threshold value in which we guarantee to find all the solutions and at the same time we prune the tree.

We conducted an additional experiment to investigate this behavior more elaborately. We implemented an ad-hoc $\Delta\mathcal{SC}$ driven search algorithm to understand the variation of $\Delta\mathcal{SC}$ in case that we construct the complete tree without applying any termination condition. By "$\Delta\mathcal{SC}$ driven search algorithm", we mean that –instead of applying our solution driven greedy search algorithm– at every iteration we pick the leaf which has the minimum $\mathcal{SC}$ distance from the root node. By replacing the proposed greedy search with this algorithm, we can eliminate the possible effects of the greedy search on the variation of $\Delta\mathcal{SC}$, and instead, we can purely concentrate on what happens to the $\Delta\mathcal{SC}$ during the construction of the whole tree.

Figure 7.6 depicts the experiment results when we utilize the "$\Delta\mathcal{SC}$ driven search algorithm". We observe from the results that in the beginning there is quite a gradual increase in $\Delta\mathcal{SC}$ as expected since we use a search heuristic which always tries to minimize $\Delta\mathcal{SC}$. However, later, even if we construct the tree purely $\Delta\mathcal{SC}$ driven, the $\Delta\mathcal{SC}$ starts oscillating. This is due to the fact that when we apply as adaptation "replace" action to a $\mathcal{SC}$ element, we might be decreasing the current $\Delta\mathcal{SC}$ since we might be going back to the initial state of this element, i.e., its state at the tree root. We call this "roll back effect of adaptations". In our analyses we check the roll back to the same $\mathcal{SC}$ and we do not expand such tree nodes, however, as mentioned, also some of the $\mathcal{SC}$ elements can roll back to a previous version. This clarifies why we have oscillations in $\Delta\mathcal{SC}$ even if we use a tree search algorithm which is minimizing the $\Delta\mathcal{SC}$ at each node pick, and due to the same reason, we cannot prune the tree by ensuring at the same time that we do not prune any solutions.

Moreover, Figure 7.6 shows that the oscillations never exceed $\Delta\mathcal{SC} = 9$, which corresponds to 50% of threshold as the $\mathcal{SC}$ size of our reference scenario is 18. The reason why they do not exceed 9 is because this is the maximum level of adaptations we can perform on the system given the adaptation search space presented in Table 7.1. All in all, this explains why we do not prune any tree nodes down to 50% in Figure 7.5.

7.3 Contribution of the Modeling Methodology

In this section we discuss the contribution of the modeling methodology presented in Chapter 6. We evaluate the contribution in two extents: (i) the efforts required to create system and adaptation models (ii) the resultant outputs of different model abstractions on an adaptation case study.

For evaluation we considered two different scenarios. In the first experiment we worked on the system model introduced in Section 7.1. Instead, in the second experiment we worked on a more complicated system with more adaptation and system tools (Figure 7.7).

For each experiment, first we measured the design-time efforts required in the case that we follow our opportunist modeling methodology, that is derivation of system elements from the available system tools. Then we measured the efforts in the case that we follow another modeling methodology where we start from the identification of very high level system elements, and then associating the tools with these elements. In order to demonstrate that the system model in the first case, which is created via our methodology, has the right level of abstraction, intentionally we kept the model of the second case more simple with less number of system elements. Then as the second stage of the evaluation, we ran CLAM platform for the adaptation case study presented in Section 7.1 and observed the results in both cases.

Measuring Modeling Efforts. We used the techniques proposed by [52] in order to identify and calculate required design-time efforts for each case presented in Figure 7.7. The calculation comprises the efforts required to create system and adaptation models.

[52] defines the modeling effort as the number of actions required to complete a high-level modeling goal. The general equation for measuring modeling effort $M(T)$ is defined as:

$M(T) = \sum Z_t + time(t)$

For each action t in task T, modeling effort is measured as the summation of the think time for the action, that is Z_t, i.e., human portion of the task, and the time for the computer to complete its given portion of the task, i.e., $time(t)$. For our computations we ignore the computer portion since it is relatively too small compared to the human portion.

Following the approach in [52], we performed the measurement for modeling efforts in three steps: (i) identification of modeling tasks in chronological

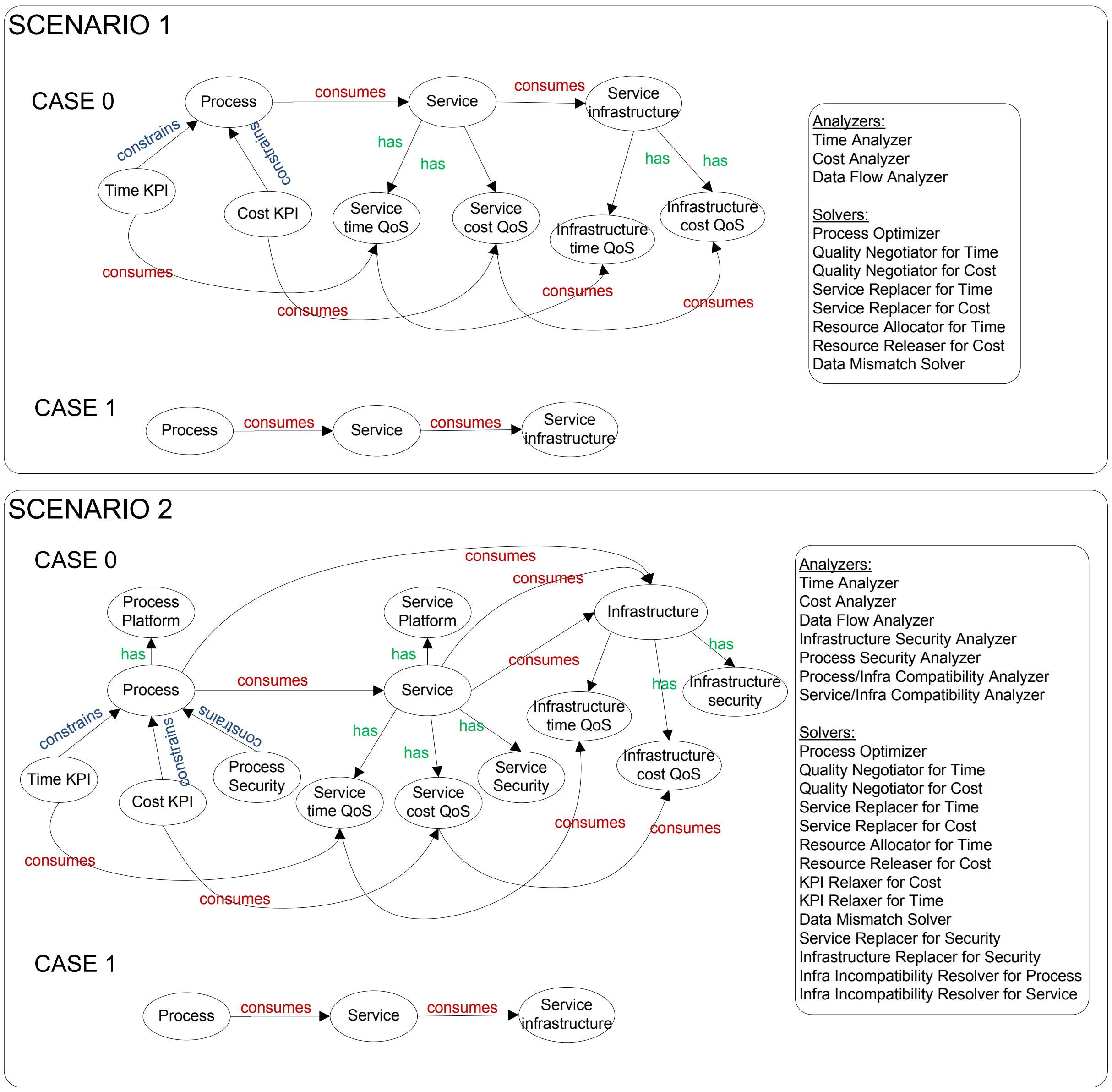


Figure 7.7: Two Different Levels of Abstraction for Scenario 1 and Scenario 2

order, (ii) identification of number of actions we need to consider for each modeling task, (iii) identification of effort required for each single action. Tables 7.7, 7.8, 7.9 and 7.10 illustrate the calculation of overall efforts for "Case 0 of Scenario 1", "Case 1 of Scenario 1", "Case 0 of Scenario 2" and "Case 1 of Scenario 2" respectively.

Notice that when we follow our approach, i.e., starting from the tools, we have the "identify relations among system elements" and "associate needs with analyzer outputs" modeling actions more costly than the other actions. This is because, while other actions can be easily derived from the tool inputs and outputs, these two actions should be reasoned based on interdependences of system elements and tools behavior. Also in the second modeling approach, i.e., when we start from the system elements to design, we have some actions more costly compared to some other. To elaborate, while we have the same issues for the identification of relations and the mapping of needs to analyzer outputs, additionally, we have a remarkable difficulty to determine the system elements since in this case they are not obtained by derivation. Similarly, we need to make more efforts to understand how to map the decided system elements to the analyzer inputs and solver outputs. We remark that the time units in the results are normalized values in order to compare different scenarios and cases. The actual values strongly depend on the domain expertise of the designer.

The results reveal the following facts: In case of not following our modeling methodology (Tables 7.8 and 7.10), even if we keep the model at a very high level by increasing the level of abstraction, we still need a considerable amount of efforts to create the system and adaptation models. This gets more obvious when we consider more complicated systems with more adaptation and analysis tools like in Scenario 2. In that scenario while we generalize the model from 14 system elements to 3 system elements, the modeling effort is only halved compared to the case where we follow our methodology, i.e., deciding the right level of abstraction by deriving the system elements from tool inputs and out-

Modeling tasks (in chronological order)	**Number of total actions in the task**	**Human effort required for a single action**	**Total effort required for the task**
Identify analyzers	3 analyzers	1 unit of time	3
Identify solvers	8 solvers	1 unit of time	8
Identify solver inputs (needs)	8 inputs	1 unit of time	8
Identify solver outputs (system elements involved in produced adaptations)	17 outputs	1 unit of time	17
Identify analyzer inputs (system elements involved in analyses)	10 inputs	1 unit of time	10
Identify system elements	9 elements	1 unit of time	9
Identify relations among system elements	9 x 9 relations	2 unit of time	162
Associate needs with analyzer outputs	3 analyzers x 8 needs	2 unit of time	48
OVERALL MODELING EFFORT			**265 units of time**

Table 7.7: Modeling Efforts: Scenario 1 - Case 0

Modeling tasks (in chronological order)	**Number of total actions in the task**	**Human effort required for a single action**	**Total effort required for the task**
Identify system elements	3 elements	5 unit of time	15
Identify relations among system elements	3 x 3 relations	2 unit of time	18
Identify analyzers	3 analyzers	1 unit of time	3
Identify solvers	8 solvers	1 unit of time	8
Identify solver inputs (needs)	8 inputs	1 unit of time	8
Identify solver outputs (system elements involved in produced adaptations)	11 outputs	2 unit of time	22
Identify analyzer inputs (system elements involved in analyses)	8 inputs	2 unit of time	16
Associate needs with analyzer outputs	3 analyzers x 8 needs	2 unit of time	48
OVERALL MODELING EFFORT			**138 units of time**

Table 7.8: Modeling Efforts: Scenario 1 - Case 1

Modeling tasks (in chronological order)	**Number of total actions in the task**	**Human effort required for a single action**	**Total effort required for the task**
Identify analyzers	7 analyzers	1 unit of time	7
Identify solvers	14 solvers	1 unit of time	14
Identify solver inputs (needs)	14 inputs	1 unit of time	14
Identify solver outputs (system elements involved in produced adaptations)	37 outputs	1 unit of time	37
Identify analyzer inputs (system elements involved in analyses)	17 inputs	1 unit of time	17
Identify system elements	14 elements	1 unit of time	14
Identify relations among system elements	14 x 14 relations	2 unit of time	392
Associate needs with analyzer outputs	7 analyzers x 14 needs	2 unit of time	196
OVERALL MODELING EFFORT			**691 units of time**

Table 7.9: Modeling Efforts: Scenario 2 - Case 0

Modeling tasks (in chronological order)	**Number of total actions in the task**	**Human effort required for a single action**	**Total effort required for the task**
Identify system elements	3 elements	5 unit of time	15
Identify relations among system elements	3 x 3 relations	2 unit of time	18
Identify analyzers	7 analyzers	1 unit of time	7
Identify solvers	14 solvers	1 unit of time	14
Identify solver inputs (needs)	14 inputs	1 unit of time	14
Identify solver outputs (system elements involved in produced adaptations)	17 outputs	2 unit of time	34
Identify analyzer inputs (system elements involved in analyses)	15 inputs	2 unit of time	30
Associate needs with analyzer outputs	7 analyzers x 14 needs	2 unit of time	196
OVERALL MODELING EFFORT			**328 units of time**

Table 7.10: Modeling Efforts: Scenario 2 - Case 1

puts. Thus, even if in the end it might be proposing a more refined system model, the required efforts are still reasonable.

Scenario:	Analyzer Invocations		
	Case 0	Case 1	Increase
Scenario 1	1417	1927	36%
Scenario 2	2179	3797	74%

Table 7.11: Comparison of Analyzer Invocations at Different Modeling Abstractions

Minimization of Analyzer Invocations. We would like to underline that our modeling approach finds the right level of abstraction. More precisely, it helps us identify a model, which is not overdetailed due to the unnecessary system elements that are never used by system tools, and at the same time, which is not too general that causes unnecessary analyzer invocations. When a model is too general with only few elements, each time we adapt even a small part of the system, we might have to call a series of irrelevant analyzers, which are in reality working on some other small parts of the system. The experiments verify our intuitive outlook. Table 7.11 depicts the total number of analyzer invocations required inside CLAM to identify the cross-layer adaptations. Consequently, even if we put some efforts to create a little bit more detailed model, we considerably minimize the number of tool invocations, which leads to the further optimization of the overall CLAM approach. The results get even more obvious in case of more complicated systems such as Scenario 2.

7.4 Discussion

In this chapter, first we presented the practical results obtained from the application of the presented cross-layer adaptation framework to our reference scenario. The results show that the cross-layer adaptation solutions identified by CLAM are more comprehensive compared to the state-of-the-art approaches, especially considering new problems that might arise due to adaptations. To the best of our knowledge except for [53], none of the cross-layer adaptation approaches consider the impact analysis of a local adaptation on the overall system. Differently, [103] proposes an exhaustive taxonomy of cross-layer adaptation patterns. As we discussed before, this approach brings forward drawbacks such as the design-time costs of patterns and inadequacy of handling unexpected consequences of adaptations. However, one advantage of [103] over our approach is that the authors propose Web Service wrappers to integrate the adaptation tools, which makes it more standard and reusable from the service oriented point of view.

After presenting the produced cross-layer adaptations of a sample CLAM run, we introduced our greedy search algorithm, which is solution-driven when picking a tree leaf for the coordination algorithm. Compared to using well known tree search algorithms such as DFS and BFS , the results reveal a big optimization in terms of finding the solutions faster.

We also introduced a novel practical approach to terminate the algorithm in a principled way while at the same time we can still work with infinitely many adaptations proposed by solvers. This is a significant contribution, because none of the existing cross-layer adaptation approaches consider the infiniteness of the possible solutions. The termination approach takes advantage of the fact that in practice we never want to change the overall SBS significantly for a problem which arises locally.

Finally, we evaluated our modeling methodology in terms of the required efforts for constructing the system and adaptation models. The results show that applying our methodology, i.e., starting from the system tools and deriving the models out of them, is a convincing approach and it entails reasonable modeling efforts compared to the case that we follow the other way around, i.e., we start from the system elements and then associate the tools with them. However, we remark that in any case, CLAM is not totally blind for the behavior of tools.

It is true that we do not need to know how an analyzer works internally or how a solver produces the adaptations, but in order to integrate them in our framework, we still need to know for what purposes we want to use them, what kind of inputs they need, and what kind of outputs they produce.

Chapter 8

Conclusions

In this dissertation, we presented a novel approach to the cross-layer adaptation problem in service-based systems. With cross-layer adaptation we mean generating an adaptation strategy, which may involve various adaptation actions from different SBS domain layers, such that its deployment in the running system ensures an overall system stability where the system is stable if and only if all the system parts are satisfied with regard to their individual constraints that they impose on the system.

8.1 Achieved Results

To address the cross-layer adaptation problem we have developed a holistic framework that supports the identification of the problems that might occur in different parts of the SBS layers due to an adaptation, and tackles these problems by proposing new adaptations, and finally brings the system to an overall consistency. This framework integrates the existing monitoring, analysis and adaptation capabilities of the system by defining a formal model for the representation of the system and adaptation concepts. Based on this model, it introduces a tree-based algorithm which is capable of deriving the cross-layer adaptation solutions through a proper coordination of the integrated system capabilities.

We build our framework on top of a formal model, where we have the system model for the representation of service-based system, and the adaptation model for the representation of external system capabilities, i.e., analysis and adaptation tools. The system model is represented as a graph of nodes and edges. Nodes correspond to the different types of elements that are present in the system, and edges correspond to the relations among these elements. While the system model represents a class of service-based systems, a system configuration of a service-based system represents a specific service-based system, which is deployed and running or ready for deployment. On the other hand the adaptation model extends a system model and its system configurations with the system capabilities, i.e., analyzer and solver tools. In order to enable a convenient creation of the proposed models we introduced an opportunist modeling methodology which benefits from a given set of tools to derive the system elements to be considered in the model.

The CLAM algorithm, incorporated in the presented framework and responsible for solving the cross-layer adaptation problem, is a recursive tree expansion algorithm, which coordinates the analyzer and solver tools, invokes them iteratively, and investigates the possible stable system configurations that can be reached to solve the adaptation problem of an initial system configuration.

While we showed that the presented algorithm is correct and complete with respect to the definition of cross-layer adaptation problem, given the infinite input space of possible system configurations and also adaptation actions, we detected that at theoretical level the algorithm does not terminate. To address this issue in practical settings, we presented a novel termination method that is based on the fact that we do not want to bring the system to a final configuration which is very different from the initial system configuration.

Following the presentation of the cross-layer adaptation framework, we presented selection criteria to evaluate and rank the alternative cross-layer adaptation solutions identified by the CLAM algorithm. We used as criteria the overall system quality in terms of time and cost aspects, the total amount of efforts to deploy a solution again in terms of time and cost, and finally the adaptation location in terms of the SBS domain layers involved in the solution path. On top of these selection criteria we proposed two ranking methods, one of them based on simple additive weighting – multi criteria decision making, the other based on fuzzy inference systems.

We implemented our cross-layer adaptation approach and the presented algorithms into a prototype tool that we call the CLAM platform where the coordinator is the core part which realizes the CLAM algorithm. The platform allows for creation of the required models, integration of the tools with the models and the coordinator, coordination of the tools to discover the cross-layer adaptation solutions, and ranking of the discovered solutions to select a best one for deployment.

Using the prototype implementation, we evaluated our framework on "Call & Pay Taxi" SBS scenario using the state-of-the-art analysis and adaptation mechanisms as analyzers and solvers. The experiments revealed substantial results: First, our approach for cross-layer adaptation problem overcomes the limitations of existing works especially with regard to the extension of SBS with new system elements and new tools, and the construction of cross-layer adaptation paths on the fly, which leads to the identification of non-trivial solutions. Second, the proposed heuristic methods not only achieve to guarantee termination, but also optimize the tree construction. Last, but not least, the proposed modeling methodology minimizes the number of necessary analyzer invocations, and indeed, the effort to create a very high level system model with a single node, which does not follow the presented methodology, is comparable to the effort to create the model adhering to the methodology which we propose.

To sum up, the main contributions of the presented research work are: (i) a new generic and flexible modeling approach which is capable of capturing the cross-aspects of the system in terms of both its elements and the supporting tools, (ii) a novel iterative approach for solving cross-layer adaptation problem on the fly, (iii) a heuristic method to cope with infinite-range adaptations and system configuration options, (iv) a convenient modeling methodology to facilitate the creation of the system and adaptation models, and finally (v) novel adaptation selection criteria, which flexibly allow for the usage of diverse ranking techniques.

8.2 Future Directions

There is a broad range of research directions that we can investigate for future work. These directions would potentially tackle important limitations of the current state of the presented work both from the methodological and from the technical points of view. We take them into account as future extensions of the presented cross-layer adaptation framework and its implementation, namely, the CLAM platform.

Currently in our approach we do not support the correlation of monitoring events. We assume that in the running SBS, a single problem occurs at a single time, which is quite a restrictive assumption considering the real world cases. In this sense, an appropriate cross-layer monitoring technique should be incorporated into our cross-layer adaptation framework. A starting point could be the investigation of the existing methods presented in [122, 88]. In this way, we can deal with multiple monitoring events at the same time and accept only the resultant aggregate event as the initial trigger for cross-layer adaptation management.

Furthermore, beyond the correlation of monitoring events, one can think of other type of correlations as an extension. It could be useful information to have the correlation of observations, predictions and events from different sources, and provided by different analysis, decision and adaptation mechanisms. This kind of research challenges for multi-layered applications are emphasized also in [80]. For instance, monitoring events could be cross-correlated with prediction mechanisms concerning future situations, and further with results of similar monitors to confirm the validity of a triggered event. Such correlations would imply introduction of new tool types to the cross-layer adaptation framework and extend the overall coordination approach accordingly.

On the contrary to what is discussed above, instead of correlating similar information of similar tools, another extension could be –like alternative solvers– the consideration of alternative analyzers during the iterative CLAM operation. Then, all the analysis results and their consequences would appear as alternative paths and at the final stage they would be analyzed to extract the solutions.

Another limitation of the presented approach is that it does not support receiving multiple adaptations at the same time. In order to be able to deal with real systems, it is necessary to take into account a more dynamic run-time system setting where it is possible to receive adaptation actions being triggered at different system parts at the same time. In such a setting, CLAM should be able to first reason on the possible contradictions among triggered adaptations, next combine them properly, and then analyze the impact of the aggregate adaptations on the entire system.

In the evaluation of the prototype tool, we did not consider in depth the online applicability of our approach, which implies to update the system configuration without affecting the already running process instances. For such purposes, the tree construction algorithm should be further optimized properly so that the overall computation time is decreased to a reasonable level. We remark that this also highly depends on the response times of the integrated tools. For the online adaptation of SBSs, the analyzers and solvers should be carefully selected so that they do not cause unaffordable delays. Moreover, it is necessary to investigate the deployment mechanisms in more detail to select the appropriate ones for this kind of setting.

For what concerns the overall impact analysis of an initial adaptation, The current approach is pragmatistic in the sense that it only identifies the short-term consequences of an adaptation on the system constraints through immediate invocations of analyzers. Whereas, it would be very useful to investigate additionally the long-term consequences as also mentioned in [82], which are based on the past execution of the system and the adaptation history. As a starting point, we could study a possible exploitation of the existing learning, process mining and system evolution approaches [106, 48, 114].

In addition to all the foregoing, we underline that the presented cross-layer adaptation approach is generic and flexible enough to be applied to any system, i.e., not necessarily service-based systems. We remark that this thesis reflects and validates its first application in the field of service-based systems.

Bibliography

[1] S-cube project - software services and systems network. http://www.s-cube-network.eu.

[2] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian. Resource management in the autonomic service-oriented architecture. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 84–92. IEEE, 2006.

[3] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *Proceedings of the 18th international conference on World wide web*, pages 881–890. ACM, 2009.

[4] Sergio Andreozzi, Natascia De Bortoli, Sergio Fantinel, Antonia Ghiselli, Gian Luca Rubini, Gennaro Tortone, and Maria Cristina Vistoli. GridICE: a monitoring service for grid systems. *Future Generation Computer Systems*, 21(4):559–571, April 2005.

[5] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, et al. Business process execution language for web services, 2003.

[6] D. Ardagna and B. Pernici. Global and local qos guarantee in web service selection. In *Business Process Management Workshops*, pages 32–46. Springer, 2006.

[7] Danilo Ardagna, Marco Comuzzi, Enrico Mussi, Barbara Pernici, and Pierluigi Plebani. Paws: A framework for executing adaptive web-service processes. *IEEE Softw.*, 24(6):39–46, 2007.

[8] A. Arkin et al. Business process modeling language. *BPMI.org*, 2002.

[9] W. Arnold, T. Eilam, M. Kalantar, A. Konstantinou, and A. Totok. Pattern based soa deployment. *Service-Oriented Computing–ICSOC 2007*, pages 1–12, 2007.

[10] A. Arsanjani. Service-oriented modeling and architecture. *IBM developer works*, 2004.

[11] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley. Soma: A method for developing service-oriented solutions. *IBM systems Journal*, 47(3):377–396, 2008.

[12] R. Aschoff and A. Zisman. Qos-driven proactive adaptation of service composition. *Service-Oriented Computing*, pages 421–435, 2011.

[13] Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. Run-Time Monitoring of Instances and Classes of Web Service Compositions. In *IEEE International Conference on Web Services (ICWS 2006)*, pages 63–71, 2006.

[14] L. Baresi, M. Caporuscio, C. Ghezzi, and S. Guinea. Model-Driven Management of Services. In *ECOWS'10*, pages 147–154, 2010.

[15] L. Baresi and S. Guinea. Self-supervising bpel processes. *Software Engineering, IEEE Transactions on*, 37(2):247–263, 2011.

[16] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[17] Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg. Monitoring Business Processes with Queries. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 603–614, 2007.

[18] Antonio Brogi and Razvan Popescu. Automated Generation of BPEL Adapters. In *ICSOC 2006:Service-Oriented Computing*, pages 27–39, 2006.

[19] A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik. Dynamic adaptation of fragment-based and context-aware business processes. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 33–41. IEEE, 2012.

[20] A. Bucchiarone, M. Pistore, H. Raik, and R. Kazhamiakin. Adaptation of service-based business processes by context-aware replanning. In *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, pages 1–8. IEEE, 2011.

[21] B. Burgstaller, D. Dhungana, X. Franch, P. Grunbacher, L. López, J. Marco, M. Oriol, R. Stockhammer, J.K. Universitat, and J.K. Universitat. Monitoring and Adaptation of Service-oriented Systems with Goal and Variability Models. Technical report, Universitat Politècnica de Catalunya, 2008.

[22] C. Cappiello, M. Comuzzi, and P. Plebani. On automated generation of web service level agreements. In *Advanced Information Systems Engineering*, pages 264–278. Springer, 2007.

[23] Malu Castellanos, Fabio Casati, Ming-Chien Shan, and Umesh Dayal. iBOM: A Platform for Intelligent Business Operation Management. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 1084–1095, 2005.

[24] Luca Cavallaro, Elisabetta Di Nitto, and Matteo Pradella. An automatic approach to enable replacement of conversational services. In Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors, *ICSOC/ServiceWave*, volume 5900 of *Lecture Notes in Computer Science*, pages 159–174, 2009.

[25] S.H. Chang and S.D. Kim. A variability modeling method for adaptable services in service-oriented computing. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 261–268. Ieee, 2007.

[26] Anis Charfi, Tom Dinkelaker, and Mira Mezini. A plug-in architecture for self-adaptive web service compositions. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 35–42, Washington, DC, USA, 2009. IEEE Computer Society.

[27] P. Châtel, J. Malenfant, and I. Truck. Qos-based late-binding of service invocations in adaptive business processes. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 227–234. IEEE, 2010.

[28] Kareliotis Christos, Costas Vassilakis, Efstathios Rouvas, and Panayiotis Georgiadis. Qos-driven adaptation of bpel scenario execution. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 271–278, Washington, DC, USA, 2009. IEEE Computer Society.

[29] Massimiliano Colombo, Elisabetta Di Nitto, and Marco Mauri. SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In *In ICSOC06*, pages 191–202, 2006.

[30] M. Comuzzi and B. Pernici. An architecture for flexible web service qos negotiation. In *EDOC Enterprise Computing Conference, 2005 Ninth IEEE International*, pages 70–79. IEEE, 2005.

[31] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 181–194. IEEE, 2001.

[32] HT Davenport. The e-process evolution. *Business Process Management Journal*, 10(1):12–15, 2004.

[33] N. Delgado, A.Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *Software Engineering, IEEE Transactions on*, 30(12):859–872, 2004.

[34] D. Dib, N. Parlavantzas, M. Christine, et al. Towards multi-level adaptation for distributed operating systems and applications. In *ICA3PP-12*, 2012.

[35] J.C. Doyle, B.A. Francis, and A. Tannenbaum. *Feedback control theory*, volume 134. Macmillan New York, 1992.

[36] H. Dresner. Business activity monitoring: Bam architecture. In *Gartner Symposium ITXPO*, 2003.

[37] M. Dumas, L. García-Bañuelos, A. Polyvyanyy, Y. Yang, and L. Zhang. Aggregate quality of service computation for composite services. *Service-Oriented Computing*, pages 213–227, 2010.

[38] C. Efstratiou, K. Cheverst, N. Davies, and A. Friday. An architecture for the effective support of adaptive context-aware applications. In *Proceedings of Mobile Data Management (MDM'01)*, pages 15–26, Berlin, January 2001. Springer.

[39] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini. Qos–aware clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 321–328. IEEE, 2010.

[40] H. Foster and G. Spanoudakis. Smart: a workbench for reporting the monitorability of services from slas. In *Proceedings of the 3rd International Workshop on Principles of Engineering Service-oriented Systems, PESOS*, pages 36–42, 2011.

[41] M. Fugini and H. Siadat. Sla contract for cross-layer monitoring and adaptation. In *Business Process Management Workshops*, pages 412–423. Springer, 2010.

[42] M. Gambini, M. La Rosa, S. Migliorini, and A. Ter Hofstede. Automated error correction of business process models. *Business Process Management*, pages 148–165, 2011.

[43] L. Ge and B. Zhang. A modeling approach on self-adaptive composite services. In *Multimedia Information Networking and Security (MINES), 2010 International Conference on*, pages 240–244. IEEE, 2010.

[44] C. Ghezzi and S. Guinea. Run-time monitoring in service-oriented architectures. *Test and analysis of web services*, pages 237–264, 2007.

[45] E. Gjørven, R. Rouvoy, and F. Eliassen. Cross-layer self-adaptation of service-oriented architectures. In *Proceedings of the 3rd workshop on Middleware for service oriented computing*, pages 37–42. ACM, 2008.

[46] H. Groefsema, P. Bulanov, and M. Aiello. Declarative enhancement framework for business processes. *Service-Oriented Computing*, pages 495–504, 2011.

[47] R. Grønmo and M.C. Jaeger. Model-Driven Methodology for Building QoS-Optimised Web Service Compositions. In *Proceedings of the 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS05*, pages 68–82. Citeseer, 2005.

[48] C.W. Gunther, S. Rinderle-Ma, M. Reichert, and W.M.P. Van Der Aalst. Using process mining to learn from process changes in evolutionary systems. *International Journal of Business Process Integration and Management*, 3(1):61–78, 2008.

[49] Cloud Harmony. Cloudharmony benchmarks. `http://cloudharmony.com/benchmarks`.

[50] Marcel Hiel and Hans Weigand. Interoperability changes in an adaptive service orchestration. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 351–358, Washington, DC, USA, 2009. IEEE Computer Society.

[51] Julia Hielscher, Andreas Metzger, and Raman Kazhamiakin, editors. *Taxonomy of Adaptation Principles and Mechanisms*. S-Cube project deliverable, March 2009. S-Cube project deliverable: CD-JRA-1.2.2. http://www.s-cube-network.eu/achievements-results/s-cube-deliverables.

[52] J.H. Hill. Measuring and reducing modeling effort in domain-specific modeling languages with examples. In *Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on*, pages 120–129. IEEE, 2011.

[53] MA Hirzalla, A. Zisman, and J. Cleland-Huang. Using traceability to support soa impact analysis. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 145–152. IEEE, 2011.

[54] A.F.M. Huang, C.W. Lan, and S.J.H. Yang. An optimal QoS-based Web service selection scheme. *Information Sciences*, 2009.

[55] C.L. Hwang, K. Yoon, et al. *Multiple attribute decision making: methods and applications: a state-of-the-art survey*, volume 13. Springer-Verlag New York, 1981.

[56] J.-S.R. Jang. Anfis: adaptive-network-based fuzzy inference system. *Systems, Man and Cybernetics, IEEE Transactions on*, 23(3):665 –685, 1993.

[57] J.J. Jeng, J. Schiefer, and H. Chang. An agent-based architecture for analyzing business processes of real-time enterprises. In *Enterprise Distributed Object Computing Conference, 2003. Proceedings. Seventh IEEE International*, pages 86–97. IEEE, 2003.

[58] M. Kapuruge, J. Han, and A. Colman. Controlled flexibility in business processes defined for service compositions. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 346–353. IEEE, 2011.

[59] D. Karastoyanova and F. Leymann. Bpel'n'aspects: Adapting service orchestration logic. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 222–229. IEEE, 2009.

[60] R. Kazhamiakin, M. Pistore, and A. Zengin. Cross-layer adaptation and monitoring of service-based applications. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pages 325–334. Springer, 2010.

[61] Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. Network Syst. Manage.*, 11(1), 2003.

[62] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[63] A. Kertész, G. Kecskeméti, and I. Brandic. Autonomic sla-aware service virtualization for distributed systems. In *Parallel, Distributed and*

Network-Based Processing (PDP), 2011 19th Euromicro International Conference on, pages 503–510. IEEE, 2011.

[64] A. Kertesz, G. Kecskemeti, and I. Brandic. Autonomic sla-aware service virtualization for distributed systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 503–510. IEEE, 2011.

[65] R. Khalaf and F. Leymann. On web services aggregation. *Technologies for E-Services*, pages 1–13, 2003.

[66] C. Koliver, K. Nahrstedt, J.M. Farines, J.D.S. Fraga, and S.A. Sandri. Specification, mapping and control for qos adaptation. *Real-Time Systems*, 23(1):143–174, 2002.

[67] W. Kongdenfha, H.R. Motahari-Nezhad, B. Benatallah, F. Casati, and R. Saint-Paul. Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters. *IEEE Transactions on Services Computing*, pages 94–107, 2009.

[68] Woralak Kongdenfha, Rgis Saint-paul, Boualem Benatallah, and Fabio Casati. An aspect-oriented framework for service adaptation. In *In ICSOC06*, pages 15–26. ACM Press, 2006.

[69] D. Kourtesis and I. Paraskakis. A registry and repository system supporting cloud application platform governance. In *Service-Oriented Computing-ICSOC 2011 Workshops*, pages 255–256. Springer, 2012.

[70] M. Fugini L. Console. Ws-diamond: An approach to web services - diagnosibility, monitoring and diagnosis, 2007.

[71] L. Lambers, L. Mariani, H. Ehrig, and M. Pezzè. A formal framework for developing adaptable service-based applications. *Fundamental Approaches to Software Engineering*, pages 392–406, 2008.

[72] F. Leymann. Web services flow language (wsfl 1.0), may 2001. *Web site available at http://www-3. ibm. com/software/solutions/webservices/pdf/WSFL.pdf.*

[73] B. Li and K. Nahrstedt. A control-based middleware framework for quality-of-service adaptations. *Selected Areas in Communications, IEEE Journal on*, 17(9):1632–1650, 1999.

[74] C. Lin and S. Lu. Scpor: An elastic workflow scheduling algorithm for services computing. In *Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications*, pages 1–8. IEEE Computer Society, 2011.

[75] Heiko Ludwig, Asit Dan, and Robert Kearney. Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. In *Service-Oriented Computing - ICSOC 2004, Second International Conference*, pages 65–74, 2004.

[76] D. Luenberger. Introduction to dynamic systems: theory, models, and applications. 1979.

[77] Linh Thao Ly, Stefanie Rinderle, and Peter Dadam. Integration and verification of semantic constraints in adaptive process management systems. *Data Knowl. Eng.*, 64(1):3–23, 2008.

[78] C.M. MacKenzie et al. Reference model for service oriented architecture. *Public Review Draft 2*, 2006.

[79] Khaled Mahbub and George Spanoudakis. Monitoring WS-Agreements: An Event Calculus-Based Approach. In Luciano Baresi and Elisabetta Di Nitto, editors, *Test and Analysis of Web Services*, pages 265–306. Springer, 2007.

[80] A. Marconi, A. Bucchiarone, K. Bratanis, A. Brogi, J. Cámara, D. Dranidis, H. Giese, R. Kazhamiakin, R. de Lemos, C.C. Marquezan, et al. Research challenges on multi-layer and mixed-initiative monitoring and adaptation for service-based systems. In *Proceedings of the ICSE 2012 Workshop on European Software Services and Systems Research–Results and Challenges (S-Cube)*, 2012.

[81] Annapaola Marconi, Marco Pistore, Adina Sirbu, Hanna Eberle, Frank Leymann, and Tobias Unger. Enabling adaptation of pervasive flows: Built-in contextual adaptation. In Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors, *ICSOC/ServiceWave*, volume 5900 of *Lecture Notes in Computer Science*, pages 445–454, 2009.

[82] J.A. Martín H, J. de Lope, and D. Maravall. Adaptation, anticipation and rationality in natural and artificial systems: computational paradigms mimicking nature. *Natural Computing*, 8(4):757–775, 2009.

[83] R. Mietzner and F. Leymann. A self-service portal for service-based applications. In *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on*, pages 1–8. IEEE, 2010.

[84] D. Minoli. *Enterprise architecture A to Z: frameworks, business process modeling, SOA, and infrastructure technology*. Auerbach Publications, 2008.

[85] C. Momm, M. Gebhart, and S. Abeck. A model-driven approach for monitoring business performance in web service compositions. In *Internet and Web Applications and Services, 2009. ICIW'09. Fourth International Conference on*, pages 343–350. IEEE, 2009.

[86] F. Moo-Mena, J. Garcilazo-Ortiz, L. Basto-Díaz, F. Curi-Quintal, and F. Alonzo-Canul. Defining a self-healing qos-based infrastructure for

web services applications. In *Computational Science and Engineering Workshops, 2008. CSEWORKSHOPS'08. 11th IEEE International Conference on*, pages 215–220. IEEE, 2008.

[87] A. Mos, A. Boulze, S. Quaireau, and C. Meynier. Multi-layer perspectives and spaces in soa. In *Proceedings of the 2nd international workshop on Systems development in SOA environments*, pages 69–74. ACM, 2008.

[88] A. Mos, C. Pedrinaci, G. Rey, J. Gomez, D. Liu, G. Vaudaux-Ruth, and S. Quaireau. Multi-level monitoring and analysis of web-scale service based applications. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pages 269–282. Springer, 2010.

[89] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for ws-bpel. In *Proceedings of the 17th international conference on World Wide Web*, pages 815–824. ACM, 2008.

[90] Hamid Reza Motahari Nezhad, Boualem Benatallah, Axel Martens, Francisco Curbera, and Fabio Casati. Semi-automated adaptation of service interactions. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 993–1002, New York, NY, USA, 2007. ACM.

[91] M. Nami and M. Sharifi. A survey of autonomic computing systems. *Intelligent Information Processing III*, pages 101–110, 2007.

[92] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, pages 237–250. ACM, 2010.

[93] Elisabetta Nitto, Massimiliano Penta, Alessio Gambi, Gianluca Ripa, and Maria Luisa Villani. Negotiation of service level agreements: An archi-

tecture and a search-based approach. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 295–306, Berlin, Heidelberg, 2007. Springer-Verlag.

[94] M. Papazoglou and P. Ribbers. *E-business: organizational and technical foundations*. Wiley, 2006.

[95] Mike P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: A research roadmap. *International Journal of Co-operative Information Systems*, 17(2):223–255, (2008).

[96] M.P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 3–12. IEEE, 2003.

[97] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[98] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[99] Barbara Pernici and Anna Maria Rosati. Automatic learning of repair strategies for web services. In *ECOWS '07: Proceedings of the Fifth European Conference on Web Services*, pages 119–128, Washington, DC, USA, 2007. IEEE Computer Society.

[100] Barbara Pernici and Seyed Hossein Siadat. A fuzzy service adaptation based on qos satisfaction. In *CAiSE*, pages 48–61, 2011.

[101] Barbara Pernici and Seyed Hossein Siadat. Selection of service adaptation strategies based on fuzzy logic. In *SERVICES*, pages 99–106, 2011.

[102] M. Pistore, R. Kazhamiakin, and A. Bucchiarone. Integration framework baseline. *S-Cube Deliverable CD-IA-3.1*, 1, 2009.

[103] R. Popescu, A. Staikopoulos, A. Brogi, P. Liu, and S. Clarke. A formalized, taxonomy-driven approach to cross-layer application adaptation. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 7(1):7, 2012.

[104] B. Qiu. The application of fuzzy prediction for the improvement of qos performance. In *Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on*, volume 3, pages 1769–1773. IEEE, 1998.

[105] N. Rasadka and A. Marconi. Optimizing bpel compositions via automatic process re-writing. *Technical Report*, 2011.

[106] S. Rinderle, B. Weber, M. Reichert, and W. Wild. Integrating process learning and process evolution–a semantics based approach. *Business Process Management*, pages 252–267, 2005.

[107] Heinz Roth, Josef Schiefer, and Alexander Schatten. Probing and Monitoring of WSBPEL Processes with Web Services. In *CEC-EEE '06: Proceedings of the The 8th IEEE International Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, page 30, 2006.

[108] E. Schmieders, A. Micsik, M. Oriol, K. Mahbub, and R. Kazhamiakin. Combining sla prediction and cross layer adaptation for preventing sla violations.

[109] Seyed Hossein Siadat, Asli Zengin, Annapaola Marconi, and Barbara Pernici. A fuzzy approach for ranking adaptation strategies in clam. In *SOCA*, 2012.

[110] B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. Capacity leasing in cloud systems using the opennebula engine. In *Workshop on Cloud Computing and its Applications*, volume 3, 2008.

[111] E.M. Taid Holmes, U. Zdun, and S. Dustdar. Model-aware monitoring of soas for compliance. *Service Engineering: European Research Results*, page 117, 2010.

[112] S. Thatte. Xlang: Web services for business process design. *Microsoft Corporation*, page 13, 2001.

[113] U.K. Tripathi, K. Hinkelmann, and D. Feldkamp. Life cycle for change management in business processes using semantic technologies. *Journal of Computers*, 3(1):24, 2008.

[114] W. van der Aalst, M. Pesic, and M. Song. Beyond process mining: from the past to present and future. In *Advanced Information Systems Engineering*, pages 38–52. Springer, 2010.

[115] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[116] J. Varia. Migrating your existing applications to the aws cloud. *Amazon Web Services Whitepaper. Available at http://www. media. amazonwebservices. com/[Last Accesed: December 2011]*, 2010.

[117] K. Verma, K. Gomadam, A.P. Sheth, J.A. Miller, and Z. Wu. The meteor-s approach for configuring and executing dynamic web processes. *LSDIS METEOR-S project. http://lsdis. cs. uga. edu/projects/meteor-s/techRep6-24-05. pdf. Technical report*, 2005.

[118] N.M. Villegas, HA Muller, and G. Tamura. Optimizing run-time soa governance through context-driven slas and dynamic monitoring. In *Main-*

tenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2011 International Workshop on the, pages 1–10. IEEE, 2011.

[119] F. Wagner, F. Ishikawa, and S. Honiden. Qos-aware automatic service composition by applying functional clustering. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 89–96. IEEE, 2011.

[120] H. Wei, J. Shao, B. Liu, H. Liu, Q. Wang, and H. Mei. A self-management approach for service developers of paas. In *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*, pages 85–92. IEEE, 2011.

[121] M. Weidmann, M. Alvi, F. Koetter, F. Leymann, T. Renner, and D. Schumm. Business process change management based on process model synchronization of multiple abstraction levels. In *Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications*, pages 1–4. IEEE Computer Society, 2011.

[122] B. Wetzstein, P. Leitner, F. Rosenberg, I. Brandic, S. Dustdar, and F. Leymann. Monitoring and analyzing influential factors of business process performance. In *Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International*, pages 141–150. IEEE, 2009.

[123] B. Wetzstein, A. Zengin, R. Kazhamiakin, A. Marconi, M. Pistore, D. Karastoyanova, and F. Leymann. Preventing kpi violations in business processes based on decision tree learning and proactive runtime adaptation. *Journal of Systems Integration*, 3(1):3–18, 2012.

[124] Branimir Wetzstein, editor. *Initial Models and Mechanisms for Quantitative Analysis of Correlations between KPIs, SLAs and Underlying Business Processes.* S-Cube project deliverable, March

2009. S-Cube project deliverable: CD-JRA-2.1.2. http://www.s-cube-network.eu/achievements-results/s-cube-deliverables.

[125] D.P. Williamson and D.B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[126] U. Winkler and W. Gilani. Erp b3: Business continuity service level agreement translation and optimisation. In *Service-Oriented Computing-ICSOC 2011 Workshops*, pages 239–240. Springer, 2012.

[127] J. Yao, S. Chen, C. Wang, D. Levy, and J. Zic. Modelling collaborative services for business and qos compliance. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 299–306. IEEE, 2011.

[128] W. Yuan, K. Nahrstedt, S. Adve, D.L. Jones, and R.H. Kravets. Design and evaluation of a cross-layer adaptation framework for mobile multimedia systems. In *Proceedings of SPIE*, volume 5019, page 1, 2003.

[129] Lotfali Askar Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[130] Lotfi A. Zadeh. Fuzzy logic, neural networks, and soft computing. *Commun. ACM*, 37(3):77–84, 1994.

[131] S. Zaplata, K. Hamann, K. Kottke, and W. Lamersdorf. Flexible execution of distributed business processes based on process instance migration. *Journal of Systems Integration*, 1(3):3–16, 2010.

[132] C. Zeginis, K. Konsolaki, K. Kritikos, and D. Plexousakis. Ecmaf: An event-based cross-layer service monitoring and adaptation framework.

[133] A. Zengin. Clam: cross-layer adaptation management in service-based systems. In *Service-Oriented Computing-ICSOC 2011 Workshops*, pages 213–219. Springer, 2012.

[134] A. Zengin, R. Kazhamiakin, and M. Pistore. Clam: Cross-layer management of adaptation decisions for service-based applications. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 698–699. IEEE, 2011.

[135] A. Zengin, A. Marconi, L. Baresi, and M. Pistore. Clam: Managing cross-layer adaptation in service-based systems. In *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, pages 1–8. IEEE, 2011.

[136] A. Zengin, A. Marconi, and M. Pistore. Clam: cross-layer adaptation manager for service-based applications. In *Proceedings of the International Workshop on Quality Assurance for Service-Based Applications*, pages 21–27. ACM, 2011.

[137] H. Zheng, J. Yang, W. Zhao, and A. Bouguettaya. Qos analysis for web service compositions based on probabilistic qos. *Service-Oriented Computing*, pages 47–61, 2011.