A Formal Approach for Designing Distributed Self-Adaptive Systems
A Formal Approach for Designing Distributed Self-Adaptive Systems
A FORMAL APPROACH FOR DESIGNING DISTRIBUTED SELF-ADAPTIVE SYSTEMS

DIDAC GIL DE LA IGLESDIA

LINNAEUS UNIVERSITY PRESS
A Formal Approach for Designing Distributed Self-Adaptive Systems
Doctoral dissertation, Department of Media Technology, Linnaeus University, Växjö, Sweden, 2014

Published by: Linnaeus University Press, 351 95 Växjö
Printed by: Elanders Sverige AB, 2014
Abstract


Engineering contemporary distributed software applications is a challenging task due to the dynamic operating conditions in which these systems have to function. Examples are dynamic availability of resources, errors that are difficult to predict, and changing user requirements. These dynamics can affect a number of quality concerns of a system, such as robustness, openness, and performance. The challenges of engineering software systems with such dynamics have motivated the need for self-adaptation. Self-adaptation is based on the principle of separation of concerns, distinguishing two well defined systems: a managed system that deals with domain specific concerns and a managing system that deals with particular quality concerns of the managed system through adaptation with a feedback loop. State of the art in self-adaptation advocates the use of formal methods to specify and verify the system’s behavior in order to provide evidence that the system’s goals are satisfied. However, little work has been done on the consolidation of design knowledge to model and verify self-adaptation behaviors.

To support designers, this thesis contributes with a set of formally specified templates for the specification and verification of self-adaptive behaviors of a family of distributed self-adaptive systems. The templates are based on the MAPE-K reference model (Monitor-Analyze-Plan-Execute plus Knowledge). The templates comprise: (1) behavior specification patterns for modeling the different MAPE components of a feedback loop, and (2) property specification patterns that support verification of the correctness of the adaptation behaviors. The target domain are distributed applications in which self-adaptation is used for managing resources for robustness and openness requirements. The templates are derived from expertise with developing several self-adaptive systems, including a collaborative mobile learning application in which we have applied self-adaptation to make the system robust to degrading GPS accuracy, and a robotic system in which we apply self-adaptation to support different types of openness requirements. We demonstrate the reusability of the templates in a number of case studies.

**Keywords**
Self-Adaptive Systems, Formal Methods, MAPE-K, Robustness, Openness, Mobile Learning
To my family and friends
To my family and friends
Acknowledgments

With this thesis, I accomplish another milestone in my life, but the final product is not ready yet. I think I started my “research career” in engineering when I was a kid wondering how could that small car drive front after pulling it backwards.

“Let’s open it. The tools are in the garage”.

There were no guarantees that the car would run as fast (or run at all) after the disassemble & assemble process, but it was worth it to learn. Toys became more complex, with LEDs, buzzers and electronics. My interest was growing even further. At one point, electronics were not sufficient to explain the features they had. I don’t believe in magic, so eventually I realized it was because of something called software. I should thank those toys that I “killed” in my childhood: they triggered me to do engineering research. However, I reserve my acknowledgments for all the people that kindly spent their time and efforts to help me in this research.

I started my PhD studies in November 2008, and since then, I have faced a lot of challenges in my life: new country, new job, being far from my family and childhood friends and, of course, the intrinsic challenges that come with research. However, together with these challenges, I have also lived lots of great experiences. I had the chance to do research without the limitations and pressure that research in industry may face. I really enjoyed this freedom, guided by great advisors (please, do not confuse guidance with constraints). I have been very fortunate to do research at CeLeKT. To give some examples, I have been working with people from more than 10 different countries and cultures, and I learnt lots from them; I had the chance to travel more than I would have ever imagined; and I could work with new technologies whenever I needed.

Have you ever had a web server on your mobile phone? I had the chance already in 2008. And I met so many supportive people, that I just want to apologize in advance if I didn’t mention someone. This acknowledgment is for you.

I want to thank the research team at CeLeKT and the people at the Faculty of Technology at Linnaeus University. We spent lots of hours together in projects design and development, discussing methods, analyzing data, writing and fixing articles, searching for the disguised and other typos in the writings, etc. Thanks Alisa, Andreas, Arianit, Aris, Bato, Daniel, Danny Weyns, David, Dennis, Ilir, Jesper Andersson, Joshy, Kalle, Katrin, Lars, Magnus, Marc, Marcelo Milrad, Martin, Mattias, Max, Miranda, Nico, Nuno, Oleg, Osama, Oskar, Sadaf, Tonya, Usman, Yeray. The experience becomes much nicer when you have somebody to share it with.

And thanks to the people with whom I had the pleasure to collaborate during my research. Thanks Juan Felipe, Miguel Nussbaum, and all the students that participated during the empirical studies that we carried out in Växjö, Stockholm and in Santiago de Chile.
Acknowledgments

With this thesis, I accomplish another milestone in my life, but the final product is not ready yet. I think I started my "research career" in engineering when I was a kid wondering how could that small car drive front after pulling it backwards. "Let's open it. The tools are in the garage". There were no guarantees that the car would run as fast (or run at all) after the disassemble\&assemble process, but it was worth it to learn. Toys became more complex, with LEDs, buzzers and electronics. My interest was growing even further. At one point, electronics were not sufficient to explain the features they had. I don’t believe in magic, so eventually I realized it was because of something called software. I should thank those toys that I "killed" in my childhood: they triggered me to do engineering research. However, I reserve my acknowledgments for all the people that kindly spent their time and efforts to help me in this research.

I started my PhD studies in November 2008, and since then, I have faced a lot of challenges in my life: new country, new job, being far from my family and childhood friends and, of course, the intrinsic challenges that come with research. However, together with these challenges, I have also lived lots of great experiences. I had the chance to do research without the limitations and pressure that research in industry may face. I really enjoyed this freedom, guided by great advisors (please, do not confuse guidance with constraints). I have been very fortunate to do research at CeLeKT. To give some examples, I have been working with people from more than 10 different countries and cultures, and I learnt lots from them; I had the chance to travel more than I would have ever imagined; and I could work with new technologies whenever I needed. Have you ever had a web server on your mobile phone? I had the chance already in 2008. And I met so many supportive people, that I just want to apologize in advance if I didn’t mention someone. This acknowledgment is for you.

I want to thank the research team at CeLeKT and the people at the Faculty of Technology at Linnaeus University. We spent lots of hours together in projects design and development, discussing methods, analyzing data, writing and fixing articles, searching for the disguised bababa and other typos in the writings, etc. Thanks Alisa, Andreas, Arianit, Aris, Bato, Daniel, Danny Weyns, David, Dennis, Ilir, Jesper Andersson, Joshy, Kalle, Katrin, Lars, Magnus, Marc, Marcelo Milrad, Martin, Mattias, Max, Miranda, Nico, Nuno, Oleg, Osama, Oskar, Sadaf, Tonya, Usman, Yeray. The experience becomes much nicer when you have somebody to share it with.

And thanks to the people with whom I had the pleasure to collaborate during my research. Thanks Juan Felipe, Miguel Nussbaum, and all the students that participated during the empirical studies that we carried out in Växjö, Stockholm and in Santiago de Chile.
També vull donar les gràcies a la meva família. El suport que m’heu donat en tots i cadascún dels durs moments que he passat durant el doctorat no es pot pagar amb diners. També us dec molt de temps que no he pogut disfrutar al vostre costat, encara sort de les noves tecnologies que ens han deixat parlar per Internet de tant en tant, però no tant com deuria. *Mea culpa.* M’ho apunto a la llista de tasques pendents.

Quiero agradecer también a los buenos amigos que desde la distancia se han preocupado por mí. Han sido ya más de 5 años desde que me fui, y en todo este tiempo nos hemos visto en esporádicas ocasiones. Sin embargo, a pesar del tiempo y la distancia, es hermoso ver que nuestra amistad no ha sufrido. Gracias Aitor, David, Gloria, Juan Pablo, Juanma, Maria, Oscar, Pasquale, Puchu, Xavi, y as resto de *la familia.* Seguís siendo todos como antes, aunque alguno se vaya quedando calvo.

Thanks Yuliya, for your unconditional support. You helped me in taking decisions, in analyzing situations, etc. You even reviewed my thesis, from the distance, while having a heavy load of work, and without a software engineering background. I cannot thank you enough. Two years ago, when I finished my Licentiate, I wrote "*I hope I can pay it back*." Now, the debt has even increased. If there is a board of directors inside your head, I hope they consider me a profitable long term investment. I will pay it back. Quoting Scissor Sisters, "*if it takes another life, I wait for you on the other side*". You deserve it.

As I said at the beginning, the final product is not finished. I am certain that more challenges are waiting for me, and I have learnt that you will be there once again to give me strength whenever I require it. Thank you all for being YOU.

Växjö, Sweden
April 14\textsuperscript{th}, 2014
També vull donar les gràcies a la meva família. El suport que m'heu donat en tots i cadascún dels durs moments que he passat durant el doctorat no es pot pagar amb diners. També us dec molt de temps que no he pogut disfrutar al vostre costat, encara sort de les noves tecnologies que ens han deixat parlar per Internet de tant en tant, però no tant com deuria.

Mea culpa. M'ho apunto a la llista de tasques pendents.

Quiero agradecer también a los buenos amigos que desde la distancia se han preocupado por mi. Han sido ya más de 5 años desde que me fui, y en todo este tiempo nos hemos visto en esporádicas ocasiones. Sin embargo, a pesar del tiempo y la distancia, es hermoso ver que nuestra amistad no ha sufrido. Gracias Aitor, David, Gloria, Juan Pablo, Juanma, Maria, Oscar, Pasquale, Puchu, Xavi, y as resto de la familia. Seguís siendo todos como antes, aunque alguno se vaya quedando calvo.

Thanks Yuliya, for your unconditional support. You helped me in taking decisions, in analyzing situations, etc. You even reviewed my thesis, from the distance, while having a heavy load of work, and without a software engineering background. I cannot thank you enough. Two years ago, when I finished my Licentiate, I wrote "I hope I can pay it back". Now, the debt has even increased. If there is a board of directors inside your head, I hope they consider me a profitable long term investment. I will pay it back. Quoting Scissor Sisters, "if it takes another life, I wait for you on the other side". You deserve it.

As I said at the beginning, the final product is not finished. I am certain that more challenges are waiting for me, and I have learnt that you will be there once again to give me strength whenever I require it. Thank you all for being YOU.

Växjö, Sweden
April 14th, 2014
## Contents

1 Introduction 1
   1.1 Problem Definition .............................. 2
   1.2 Research Goals ................................. 3
   1.3 Scope of the Research ............................ 4
   1.4 Scientific Approach .............................. 5
   1.5 Contributions .................................. 6
   1.6 Overview ...................................... 7

2 Research Approach 11
   2.1 Overview ...................................... 11
   2.2 Systematic Literature Review ....................... 12
   2.3 Formal Methods .................................. 14
   2.4 Consolidating Patterns ............................ 15
   2.5 Case Studies ................................... 17
   2.6 Research Methods Summary ......................... 18

3 Research Background 21
   3.1 Self-Adaptation ................................ 21
   3.2 Quality Concerns ................................ 26
   3.3 Assurances ..................................... 28
   3.4 Formal Methods .................................. 29

4 Systematic Literature Review on Formal Methods for Self-Adaptive Systems 33
   4.1 Introduction .................................... 35
   4.2 Research Method ................................ 36
   4.3 Planning Review ................................ 38
   4.4 Results ......................................... 42
   4.5 Limitations of the Study ......................... 50
   4.6 Future Research Directions ....................... 50
   4.7 Selected Studies ................................ 52

5 Guaranteeing Robustness using Formally Verified MAPE-K Loops 55
   5.1 Introduction .................................... 57
   5.2 Background and Related Work ...................... 59
   5.3 Towards a Robust M-Learning Application .......... 59
   5.4 Behavioral Design ................................ 63
   5.5 Verification of Self-Adaptation .................... 72
   5.6 From Design to Implementation .................... 75
   5.7 Conclusions and Future Work ...................... 75

6 MAPE-K Formal Templates 79
   6.1 Introduction .................................... 81
   6.2 Related Work .................................... 83
   6.3 Target Domain ................................... 86
   6.4 Behavior Specification Patterns for Modeling MAPE-K Feedback Loops ......................... 89
   6.5 Property Specification Patterns for Verifying MAPE-K Feedback Loops ......................... 101
6.6 Assessment of the MAPE-K Formal Templates .................. 104
6.7 Conclusions ........................................ 114
7 Conclusions ............................................ 117
  7.1 Contributions ....................................... 117
  7.2 Lessons Learned from Applying Formal Methods in Self-Adaptive Systems ........................................ 118
  7.3 Future Work ........................................ 119
  7.4 Closing Reflection .................................. 121
Appendix ..................................................... 123
A Mobile Learning Domain ..................................... 123
  A.1 Introduction ....................................... 125
  A.2 Related Efforts .................................... 127
  A.3 Learning Scenarios .................................. 129
  A.4 System Architecture ................................ 134
  A.5 Architecture Implementation and Assessment ............... 139
  A.6 Self-Adaptation ..................................... 144
  A.7 Conclusions and Future Efforts ......................... 149
  B.1 Background and Justification .......................... 154
  B.2 Overview of Research Method .......................... 155
  B.3 Research Questions .................................. 156
  B.4 Search Scope and Strategy ............................. 157
  B.5 Data Items .......................................... 158
  B.6 Data Analysis and Reporting ......................... 161
C Protocol for Case Studies: Reusability of MAPE-K Formal Templates 163
  C.1 Background .......................................... 164
  C.2 Terminology ........................................ 166
  C.3 Research Questions .................................. 167
  C.4 Design .............................................. 169
  C.5 Case Selection ....................................... 171
  C.6 Procedures and Roles ................................ 172
  C.7 Data Collection ...................................... 175
  C.8 Analysis ............................................ 178
  C.9 Study Limitations .................................... 178
  C.10 Reporting .......................................... 181
  C.11 Schedule .......................................... 181
  C.12 Interview Questions ................................ 185
  C.13 Propositions ........................................ 187
D MAPE-K Formal Templates Documentation 189
  D.1 Introduction ........................................ 190
  D.2 Preliminaries ........................................ 192
  D.3 Behavior Specification Patterns ......................... 199
  D.4 Property Specification Patterns ....................... 233
E List of Publications ....................................... 237
List of Figures

1.1 Disposition of the thesis ........................................... 9
2.1 Context in the evolution of the research contributions ........... 12
3.1 Envision of architecture based self-adaptive software from [144] . 22
3.2 MAPE-K reference model for self-adaptive systems from [102] . . 23
3.3 Feedback control loop adapted from [137] .......................... 24
3.4 Feedback control loop for Adaptive systems from [137] .......... 24
3.5 Rainbow framework for self-adaptive systems from [70] ......... 25
3.6 3-Layer Architecture Model adapted from [121] .................. 26
3.7 Event- (left) and time- (right) triggering approaches for component behaviors ................................................ 31
4.1 Overview of the systematic review process (adapted from [28]) . . 37
4.2 Overview of the studies on formal methods in self-adaptation over time ......................................................... 43
4.3 Overview of the studies on formal methods in self-adaptation per venue ......................................................... 44
4.4 Modeling languages used for formalization .......................... 45
4.5 Property specification languages used for formalization ........... 45
4.6 Adaptation concerns .................................................... 46
4.7 Types of software systems used for formalization .................. 47
4.8 Verified properties of formalization .................................. 48
4.9 Formal approach used in the study .................................... 48
4.10 Zones in the state-space that represent different behaviors of a self-adaptive system with properties of interest ................. 49
5.1 Use case scenario of a learning activity ............................... 60
5.2 Exemplification of GPS accuracy ..................................... 61
5.3 Structural view of the self-adaptive system .......................... 63
5.4 Master/Slave MVD self-healing pattern (for 3 phones) ......... 64
5.5 Process mapping for GPS service self-adaptation ................... 65
5.6 Activity Agent .......................................................... 66
5.7 Context ................................................................. 66
5.8 GPS Module ........................................................... 67
5.9 GPS Quality Probe ..................................................... 68
5.10 GPS SA Monitor ....................................................... 68
5.11 GPS SA Analyze ....................................................... 68
5.12 GPS SA Plan .......................................................... 69
5.13 GPS SA Execute ....................................................... 69
5.14 GPS Service Effector .................................................. 69
5.15 Process mapping for MVD self-healing ............................... 70
5.16 MVD Requirement Probe ............................................ 71
5.17 MVD Requirement Monitor .......................................... 71
5.18 MVD Membership Monitor .......................................... 71
5.19 MVD SH Analyze ..................................................... 72
5.20 MVD SH Plan ........................................................ 72
5.21 MVD SH Execute ...................................................... 73
5.22 UML component diagram of the GPS Service self-adaptation

6.1 Sky environment (top), GPS module (middle) and GPS-Probe behaviors for a mobile learning application

6.2 Automaton describing the map layout of a robot

6.3 Behavior specifying interfaces that managers use to update the map layout (left) and updating the robot’s current location and destination (right)

6.4 MAPE-K. Model for self-adaptive systems [113]

6.5 Mobile learning GPS-Monitor behavior

6.6 Robot transportation interface Monitor behaviors

6.7 Monitor behavior pattern with event-triggering

6.8 Monitor behavior pattern with time-triggering

6.9 Analyze behavior for the mobile learning application

6.10 Analyze behavior for a robotic scenario

6.11 Analyze behavior pattern with event-triggering

6.12 Plan behavior for the mobile learning application

6.13 Plan behaviors for the robot transportation application

6.14 Plan behavior pattern with time-triggering

6.15 Mobile learning Execute process

6.16 Robot transportation Execute process

6.17 Robot transportation Effector process

6.18 Execute behavior pattern with time-triggering

6.19 Groups of property specification patterns

6.20 Overview of 6-weeks course in which the study took place

6.21 Analyze behavior for a vehicular traffic system

6.22 Plan behavior for a vehicular traffic system system (left). Execute behavior for a smart house system (right)

6.23 Analyze behavior for a smart house system (right)

6.24 Use Cases for the described learning scenarios

A.1 Organization with three participants to order randomly generated numbers

A.2 (Left) Treasure map. (Right) Instance of a point search

A.3 Use Cases for the described learning scenarios

A.4 Architecture described through a layers diagram

A.5 Architecture described through a components diagram

A.6 Self-adaptation components on mobile nodes

B.1 Overview of the systematic review process (adapted from [28])

C.1 Example of Behaviors Specifications Pattern

C.2 Example of a behavior specification (SAAnalyze) refining a Behaviors Specifications Pattern for an Analyze behavior

D.1 MAPE-K. Model for self-adaptive systems [102]

D.2 Example of automaton with sender behavior. Sky environment

D.3 Example of automaton with receiver behavior. GPS module

D.4 Event- (left) and Time- (right) triggered interaction between components

D.5 Event- versus time-triggered interactions between MAPE behaviors
Chapter 1

Introduction

The use of systems that depend on software is growing at speeds that could not have been imagined before. One large class of such systems is characterized by being distributed with mobile units or components that can come and go at will. Some examples are emerging mobile learning systems where mobile devices are used for collaborative pedagogical activities; autonomous robots for transportation and warehouse management; smart houses equipped with sensors and actuators to provide daily home services; and e-health systems.

These examples have some characteristics in common. They are composed of multiple units or nodes; the nodes have an explicit position in the environment, which may be dynamic as the nodes may be mobile; the nodes in these systems have a continual (or periodic) communication access, which is necessary to realize the goals of the distributed system; and, due to the characteristics of these systems, they are located in dynamic environments.

The dynamic environment and changing operating conditions can make such systems behave in undesired ways. Some examples of these undesired behaviors may originate from the failure of certain resources or nodes in the system (such as power batteries and parts of embedded systems) and the degradation in the quality of services provided by resources (such as web services, accuracy of geolocation services and networking connection). Under these circumstances, it becomes necessary to implement mitigation mechanisms. Self-adaptation is a well-recognized and effective approach to deal with the increasing complexity and dynamism of modern software systems. Self-adaptation follows the principle of separation of concerns. Ideally, domain specific concerns are managed by a domain specific application, also known as the managed system. An adaptation engine, also known as the managing system, extends the features offered by the managed system in order to provide some quality concerns, such as performance, robustness and openness that are desired in the system. Self-adaptive systems require representations of the system and its environment in order to reason about the adaptation behaviors. One widely used technique applies architecture models that provide an abstraction of the managed system, in order to support the reasoning for self-adaptation performed in the managing system. This is known as architecture-based self-adaptation. One well-recognized approach to realize architecture-based self-adaptation is through MAPE-K feedback loops [113]. A MAPE-K feedback loop consists of four distinct components that separate the following functions: monitoring the managed system.
Chapter 1

Introduction

The use of systems that depend on software is growing at speeds that could not have been imagined before. One large class of such systems is characterized by being distributed with mobile units or components that can come and go at will. Some examples are emerging mobile learning systems where mobile devices are used for collaborative pedagogical activities; autonomous robots for transportation and warehouse management; smart houses equipped with sensors and actuators to provide daily home services; and e-health systems.

These examples have some characteristics in common. They are composed of multiple units or nodes; the nodes have an explicit position in the environment, which may be dynamic as the nodes may be mobile; the nodes in these systems have a continual (or periodic) communication access, which is necessary to realize the goals of the distributed system; and, due to the characteristics of these systems, they are located in dynamic environments.

The dynamic environment and changing operating conditions can make such systems behave in undesired ways. Some examples of these undesired behaviors may originate from the failure of certain resources or nodes in the system (such as power batteries and parts of embedded systems) and the degradation in the quality of services provided by resources (such as web services, accuracy of geolocation services and networking connection). Under these circumstances, it becomes necessary to implement mitigation mechanisms. Self-adaptation is a well-recognized and effective approach to deal with the increasing complexity and dynamism of modern software systems. Self-adaptation follows the principle of separation of concerns. Ideally, domain specific concerns are managed by a domain specific application, also known as the managed system. An adaptation engine, also known as the managing system, extends the features offered by the managed system in order to provide some quality concerns, such as performance, robustness and openness that are desired in the system. Self-adaptive systems require representations of the system and its environment in order to reason about the adaptation behaviors. One widely used technique applies architecture models that provide an abstraction of the managed system, in order to support the reasoning for self-adaptation performed in the managing system. This is known as architecture-based self-adaptation. One well-recognized approach to realize architecture-based self-adaptation is through MAPE-K feedback loops (Monitor-Analyze-Plan-Execute) [113]. A MAPE-K feedback loop consists of four distinct components that separate the following functions: monitoring the managed system...
1 Introduction

and its environment, analyzing the managed system’s behavior, planning adaptation actions to address detected undesired system’s behaviors and executing the planned architecture adaptations to adapt the managed system. Other prominent architecture-based approaches for the design of self-adaptive systems are 3-layer architecture model [71, 121], Rainbow [70] and feedback control loops [32]. 3-layer architecture model [71] separates goals, algorithms for changes and component control in three layers, as an approach to structure self-adaptation through the separation of concerns. Rainbow [70] offers a reusable architectural framework for building self-adaptive systems. The architectural layer that deals with self-adaptation resembles similarities with a MAPE loop. Rainbow supports monitoring and adaptation of software systems that are distributed in a network. However, the control of adaptation is centralized. Control feedback loops [32] are software approaches inspired by principles from biology. A control mechanism for self-adaptation is fed with feedback from the system, which may be positive or negative, whether it reinforces or counteracts changes in the system.

These approaches provide engineering guidelines that support the architectural design of self-adaptive systems and created a significant impact in the self-adaptive community [127]. However, it is still a challenge to provide rigorous evidence to prove that self-adaptation goals are achieved through particular self-adaptation approaches, as reflected in "Assurances for Self-Adaptive Systems" [126].

However, providing evidence to assure that the adaptation goals are achieved through particular self-adaptation approaches remains a significant challenge in the field. In a recent Dagstuhl Seminar, assurances for self-adaptive systems were the focus of discussion [126]:

"One key aspect of self-adaptive systems that poses important challenges yet to be tackled in depth is assurances: that is, providing evidence that systems satisfy their functional and non-functional requirements during operation." [128]

1.1 Problem Definition

In order to provide evidence of system goals’ satisfaction, the state of the art in architecture-based self-adaptation advocates the use of formal methods. Formal methods provide the means to rigorously specify and verify the behavior of a system [51].

In [188] we performed a systematic literature review to study the current state of the art in the use of formal methods for self-adaptive systems. The study showed that there is an increase in the use of formal methods in the field. Most of these studies used formal methods to reason about the design of the self-adaptive solutions. Only around 30% of the studies that use formal methods apply the formalisms to provide evidence that the adaptation mechanisms can accomplish the desired adaptation goals. We identified an interest in the community for the formalization of self-adaptive behaviors in order to guarantee desired adaptation properties, even though this is not extensively spread. Some representative studies are [13, 37, 73, 187], which cover aspects such as techniques for structural
1.2 Research Goals

The goal of this thesis is to consolidate knowledge for rigorous design of self-adaptive systems. This goal is achieved by consolidating knowledge and insights arising from experience and empirical evidence. This thesis focuses on the study of the adaptive behavior for a relevant family of systems that comprise resources deployed on multiple nodes that are connected via a network and that have an explicit, potentially dynamic position in the environment. We use mobile learning as one of the application domains to drive our research. Mobile learning applications are decentralized systems composed of mobile devices that are used by students for collaborative outdoor learning activities, such as distances calculations and triangulation. A second domain we use in this research are robot systems for transportation in warehouses. The thesis focusses on consolidating formal models and verification properties of MAPE-K feedback loops to deal with robustness and openness concerns. With robustness we refer to self-adaption
1 Introduction

to deal with failures of resources in the system. With openness, we refer to self-adapation to deal with resources that enter or leave the system at will.

We refine the main research question with a set of subquestions. The following questions allow us to center our efforts towards the consolidation of knowledge in rigorous design of self-adaptive systems:

**RQ1:** "Which formal methods have been used in self-adaptive systems?"

**RQ2:** "How to model the behaviors of MAPE-K components to deal with robustness and openness requirements?"

**RQ3:** "What properties must be verified to provide evidence that the required self-adaptation behaviors are assured?"

With RQ1, the thesis aims are providing an insight in the use of formal methods for self-adaptive systems in the research community. This research question is motivated by the need to get an insight in what are the formal languages and tools used for the rigorous specification and property verification of self-adaptive systems’ behaviors. We also aim at determining the parts in self-adaptive systems that were formalized and how these were applied by the research community.

RQ2 is centered on studying and elaborating the necessary formal models that specify behaviors for the targeted self-adaptive systems in order to provide robustness and openness concerns.

RQ3 focuses on the specification of adaptation properties for the verification of robustness and openness system properties. With this research question we aim at studying properties under different levels of granularity.

### 1.3 Scope of the Research

The research presented in this dissertation is scoped in four particular dimensions: type of systems, quality properties, formal languages, and time of application in the software life-cycle.

Regarding the type of systems, we target a family of self-adaptive systems that are characterized with the description listed below. Systems that fit under these descriptions become the *target domain* in this study.

- The systems comprise software deployed on distributed nodes;
- The nodes have an explicit position in the environment and may be mobile;
- The system has continual communication access;
- The dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software;
- Some resources in the system may come and go.
The target type of systems is a representative part of nowadays systems. Some examples of systems that fit under this definition are robotic systems that must collaborate and coordinate to manage goods in a warehouse; vehicle traffic systems to manage traffic flow via monitoring devices, such as cameras, and control devices such as traffic lights; web applications that maintain their services distributed among multiple server nodes; systems for e-health with multiple node units to monitor patient well-being and to provide health-care services; smart and security home systems with distributed sensors and actuators deployed in a building; and distributed mobile applications for collaborative mobile tasks, such as pedagogical outdoor activities performed with mobile devices to support traditional learning approaches.

Regarding quality properties, we have a primary focus on robustness concerns, and openness as a secondary quality concern. By robustness, we study self-adaptation to deal with parts of the system that fail. Regarding openness, we study self-adaptive systems that deal autonomously with parts that come and go dynamically. To that end, local feedback loops can observe local managed systems and their execution context and adapt the managed systems by adding and removing resources. Resources are abstractly defined and controllable parts of the managed system, such as components, devices, etc.

In this thesis, we introduce a set of reusable formal models to specify self-adaptive behaviors and a set of reusable properties to verify desired properties that the self-adaptive behaviors should hold. In a recent study in the use of formal methods for self-adaptive systems [188], automata were identified as one of the mostly used languages in the field. We use timed-automata (TA) as a formal language for the specification of self-adaptive behaviors. In order to specify desired self-adaptive properties, the thesis uses timed computational tree logic (TCTL). TCTL is a formal language based on computational tree logic extended with clock variables that allows specifying properties. Expressions in TCTL describe state and path formulae enabling the verification of properties of interest, such as reachability (a system should/can/cannot etc. reach a particular state or states), liveness (something eventually will hold), safety (given a particular condition in the system, all possible system executions should hold certain desired conditions or avoid specific undesired conditions), etc. TA and TCTL can successfully be combined to specify systems’ behaviors and to verify that the desired properties hold.

In this thesis, the templates are used offline, by architects, both during initial system development and redesign activities. The focus of this thesis is not on using the templates at runtime, but initial work has shown that the templates can be used also as a basis to provide evidence for assurances of self-adaptive systems at runtime [105].

1.4 Scientific Approach

To realize the research goals and provide scientific underpinning and validity of the research results, we combined three established methods: a systematic literature review, formal methods, and case studies.
1 Introduction

We applied Systematic methods in order to obtain a clear and complete understanding of current research efforts in the application of formal methods in self-adaptive systems. To answer RQ1, we applied the Systematic literature review (SLR) [115] method. Through the application of SLR, we obtained a complete and systematic view on efforts that applied formal methods in self-adaptive systems.

Formal methods offer the means to rigorously study the behavior of systems through the application of mathematical techniques [7]. We applied formal methods to specify the behavior of a set of distributed self-adaptive systems in order to study desired adaptation goals. The first domain where we apply formal methods includes the specification of part of the collaborative mobile learning application behavior that is relevant to analyze system’s robustness concerns. In a second domain, we apply formal methods to model self-adaptive behaviors for robustness and openness concerns in an autonomous robots system.

The Case study method offers the means to analyze the benefits of a particular tool or technique by studying one or more case instances in which the tool or technique is applied [110]. We consolidated our knowledge gained from the design of multiple formal specifications of behaviors in self-adaptive systems, generating a set of formal templates, which include behavior and property patterns for the targeted self-adaptation domain. We call this work, MAPE-K Formal Templates. We applied the Case study method to demonstrate the reusability of the MAPE-K Formal Templates in modeling behaviors of MAPE-K components for robustness and openness. Applying this method, we provide evidence for the templates reuse and answer RQ2. Through the case study, we also provide evidence regarding the benefits of the property patterns included in the MAPE-K Formal Templates, and answer RQ3.

The different methods are explained in detail in Chapter 2.

1.5 Contributions

Over the past 15 years, researchers in the field of architecture-based self-adaptive systems have developed a vast body of work. Key contributions include reference models for architecture-based self-adaptation [32, 70, 71, 113, 187, 198], formal approaches to design of architecture-based self-adaptive systems e.g., [13, 73, 74], and the analysis and evaluation of different implementation approaches [6, 48, 127]. These efforts provide a solid foundation for designers of self-adaptive systems. However, as indicated by leading researchers in the field, significant challenges remain on providing evidence that the requirements of self-adaptive systems are effectively assured [126, 127, 128]. Furthermore, there is a need for consolidating design knowledge in the form of reusable assets to support software designers [65]. This thesis contributes to these challenges as follows:

1. It contributes with a systematic survey of the research area that offers an exhaustive and synthesized view on the use of formal methods for designing self-adaptive systems [188]. The survey provides an overview of the state-of-the-art and trends in the use of formal methods for self-adaptive systems. This
contribution offers the possibility to identify commonalities, particularities and potential beneficial combinations from existing efforts in the field, and helps identifying open problems for future research.

2. It contributes with formal designs of a number of self-adaptive systems that include both models to specify self-adaptation behaviors following the MAPE-K architectural style and properties to verify the self-adaptation requirements. This thesis provides one complete formal design for a collaborative mobile learning application [81] and presents particular cases for fields such as mobile learning, robotics, vehicle traffic and smart homes [82].

3. It contributes with a set of formal templates, called MAPE-K Formal Templates [82], that consolidate knowledge gained from designing self-adaptation behaviors following the MAPE-K architectural model for a family of systems. The formal templates offers support to designers for the specification and verification of MAPE-K based self-adaptive systems for robustness and openness concerns. In particular, the templates support rigorous behavior design through a set behavior specification patterns and verification through a set of property specification patterns.

1.6 Overview

The remainder of this dissertation is structured in three parts spread over six chapters. The first part with Chapters 2 and 3 puts the research in context by zooming in on methodological aspects, the research questions, and background material. The second part with Chapters 4 to 6 provides the core contributions of this thesis: a systematic review of the research area, the application of MAPE-K to a concrete application, and the MAPE-K Formal Templates that consolidate the design expertise. Finally, the third part with Chapter 7 draws conclusions and outlines venues for future research.

Chapter 2 elaborates on the different methodological approaches that were applied during the research. After a brief overview of each method, results are summarized and linked to the concrete research questions they aim to answer. Finally, the validity of the results are discussed. Chapter 3 provides an overview of the theoretical foundations that underly the research efforts presented in this thesis. The particular focus is on design principles of architecture-based self-adaption, the quality properties of interest in this thesis: robustness and openness, and the formal languages that we use in the thesis for rigorous specification and verification of behaviors of self-adaptive systems.

Chapter 4 presents the results of the systematic literature review on the use of formal methods for self-adaptive systems. The review shows a steady increase in the use of formal methods for self-adaptation; it provides and overview of the formal languages that have been used for behavior and property specifications, the concerns that have been considered for adaptation, the types of properties that have been used for verification, and the purposes for which formalization efforts have been applied. In Chapter 5, we demonstrate how we have applied formal methods to rigorously specify and verify behaviors of a concrete self-adaptive system
following the MAPE-K architecture style. This effort demonstrates the benefits and tradeoffs of rigorously specified behaviors of a particular self-adaptive system when separating monitor, analyze, plan and execute behaviors. In Chapter 6, present the MAPE-K Formal Templates that consolidate our expertise from designing self-adaptation behaviors for a variety of systems. We illustrate the different patterns with excerpt from two applications from which they were derived. The formal templates comprise a set behavior specification patterns and a set of property specification patterns that designers can reuse when specifying and verifying self-adaptive systems. Chapter 6 concludes with the results of a set of case studies that we used for validating the reusability of the formal templates.

Finally, in Chapter 7, we summarize the contributions of this research, derive conclusions, and outline open challenges for future research. The thesis contains four additional appendices. The first appendix describes the design of a self-adaptive application in the domain of mobile learning. The second appendix provides the protocol that we developed for the execution of the systematic literature review presented in Chapter 4. The third appendix provides the protocol we developed for the case studies described in Chapter 6. Finally, the fourth appendix provides the detailed documentation of the MAPE-K Formal Templates.

Fig. 1.1 shows the structure of the thesis.
1.6 Overview

**Motivation:**
There is a need for reusable formal models for self-adaptive behaviors

**Chapter 2: Research Approach**
- Research Methods
- Results
- Validity of Results

**Chapter 3: Research Background**
- Self-Adaptation
- Self-Adaptive Approaches
- Robustness
- Openness
- Formal Methods

**Chapter 4: Survey on Formal Methods in Self-Adaptive Systems**
- Survey Research Questions
- Data Collection
- Data Analysis
- Conclusions

**Chapter 5: Guaranteeing Robustness using Formally Verified MAPE-K Loops**
- Formal Behaviors
- Property specifications
- Verification
- Conclusions

**Chapter 6: Consolidation - MAPE-K Formal Templates**
- Behavior patterns
- Property patterns
- Case Studies
- Conclusions

**Chapter 7: Conclusions and Future Work**
- Conclusions
- Lessons Learned
- Future Work

**Appendices**
- Mobile Learning Domain
- Protocol for SLR
- Protocol for Case Study
- MAPE-K Templates Documentation

**Figure 1.1: Disposition of the thesis**
Chapter 2
Research Approach

This Chapter explains the methodological approaches we have applied during the research presented in this dissertation. The Chapter starts with an overview of the research methods that we applied. Then, we discuss the different methods in more detail. For each method, we provide a short description of the method, motivate its selection in connection to the challenges the method addresses, discuss the results that we obtained, and point to validity threats. The Chapter concludes with a summary.

2.1 Overview

In order to realize the research goals presented in Chapter 1 and provide underpinning and validity of the results, we used different scientific methods. The choices for the methods were driven by the research questions to answer and specific needs we encountered in the different stages of the research [165].

The two primary challenges that we aim to tackle in this research are:

1. to understand the state of the art in formal methods for self-adaptive systems;
2. to consolidate design expertise in the form of reusable templates that designers can use for the specification and verification of behaviors for self-adaptive systems in a target domain.

For the first challenge, we applied a Systematic Literature Review (SLR). Reflecting upon the results from this study, we identified a lack of consolidation of design knowledge for self-adaptive systems, in particular regarding behavior models. This led to the second challenge that aimed at consolidating design knowledge for a target domain of self-adaptive systems. To tackle the second challenge, we applied three methods and tactics. First, we applied formal methods for the specification and verification of behaviors for a number of distributed self-adaptive systems in the target domain. Second, we analyzed the formal designs of the systems in order to consolidate patterns that support designers of new self-adaptive designs for the target domain. Third, we applied a number of case studies to assess the reusability of the consolidated patterns.

Fig. 2.1 shows the four-stage research process we followed in this thesis. From top to bottom, the figure presents the main research question, a refinement of the research question into more fine-grained challenges, the scientific methods applied to address the challenges, and the results that we obtained through them.


Chapter 2

Research Approach

This Chapter explains the methodological approaches we have applied during the research presented in this dissertation. The Chapter starts with an overview of the research methods that we applied. Then, we discuss the different methods in more detail. For each method, we provide a short description of the method, motivate its selection in connection to the challenges the method addresses, discuss the results that we obtained, and point to validity threats. The Chapter concludes with a summary.

2.1 Overview

In order to realize the research goals presented in Chapter 1 and provide underpinning and validity of the results, we used different scientific methods. The choices for the methods were driven by the research questions to answer and specific needs we encountered in the different stages of the research [165].

The two primary challenges that we aim to tackle in this research are:

(1) to understand the state of the art in formal methods for self-adaptive systems;
(2) to consolidate design expertise in the form of reusable templates that designers can use for the specification and verification of behaviors for self-adaptive systems in a target domain.

For the first challenge, we applied a Systematic Literature Review (SLR). Reflecting upon the results from this study, we identified a lack of consolidation of design knowledge for self-adaptive systems, in particular regarding behavior models. This led to the second challenge that aimed at consolidating design knowledge for a target domain of self-adaptive systems. To tackle the second challenge, we applied three methods and tactics. First, we applied formal methods for the specification and verification of behaviors for a number of distributed self-adaptive systems in the target domain. Second, we analyzed the formal designs of the systems in order to consolidate patterns that support designers of new self-adaptive designs for the target domain. Third, we applied a number of case studies to assess the reusability of the consolidated patterns.

Fig. 2.1 shows the four-stage research process we followed in this thesis. From top to bottom, the figure presents the main research question, a refinement of the research question into more fine-grained challenges, the scientific methods applied to address the challenges, and the results that we obtained through them. The
2 Research Approach

dotted arrows from left to right illustrate the path that we took during the research process.

![Diagram](image)

**Figure 2.1: Context in the evolution of the research contributions**

2.2 Systematic Literature Review

A Systematic Literature Review (SLR) [115, 191] is a well-defined approach to identify, evaluate, and interpret all studies regarding a particular research question, topic area or phenomenon of interest. A SLR applies a set of well-defined steps to avoid bias and support replicability. The concrete process of a SLR is documented in a protocol that must be strictly followed during the execution of the review. The protocol defines the research questions of the study; the criteria for the selection of secondary studies, which includes venues, keywords and filtering criteria; data items to be collected; and processes to be followed for the data analysis.
One major challenge in self-adaptive systems is to provide evidence that required quality properties are assured [127, 128]. Formal methods provide the means to rigorously specify and verify the behavior of self-adaptive systems. In order to study the state of the art in formal methods for self-adaptive system design from a software engineering point of view, we carried out a SLR. We followed established guidelines from Kitchenham and Charters [115] to perform the SLR. The study collected data from studies of all major peer-reviewed venues in the field of self-adaptive systems between January 2000 to December 2011.

2.2.1 Results

The results we obtained from the SLR provide a synthesized overview on current approaches of formal methods for self-adaptation. Concrete aspects include: aims for self-adaptation, self-adaptation properties that are studied, languages and tools used for the formalization process, among others. In addition, the study identifies a number of open challenges in the field of formal methods for self-adaptive systems.

Reflecting upon results of this SLR, we identified a lack of consolidation of design expertise with formalizing self-adaptive behaviors.

The efforts carried out in the SLR resulted in:

R1 Systematic overview of formal methods used in self-adaptive systems

The result provides an answer to research question RQ1 "Which formal methods have been used in self-adaptive systems?" with the following concrete content:

- A description of the state of the art in the use of formal methods for self-adaptive systems.
- A zone-map that represents different types of behaviors of a self-adaptive system.
- Properties of interest for self-adaptive behaviors that map to transitions between zones in the zone-map.

These results were presented at the track on Formal Methods for Self-Adaptive Systems (FMSAS) of C3S2E 2012 and published in the conference proceedings [188]. The description of the SLR with detailed results are presented in Chapter 4. The systematic review protocol can be found in Appendix B.

2.2.2 Validity of the Results

The validity of the results obtained from the SLR are dependent on the strategies applied during the study. We selected secondary studies in the review from major conferences and journals in self-adaptive systems and software engineering in general. The SLR comprises studies published between January 2000 and December 2011. The choice for the starting was motivated by the observation that self-adaptive systems became subject of active research around that time. The SLR was performed at the beginning of 2012, which explains why only studies until December 2011 were included. The results of this study are valid for the venues and period that are covered. A further study could be performed to update these
2 Research Approach

results by including recent publications. The results of the SLR may be biased by the selection of keywords that were used for the automatic selection of studies. To minimize the bias, we selected keywords based on different sources, including the extraction of standard terms from Baier and Katoen [7]. The keywords were evaluated through a focused pilot study. Four experts were involved in the data collection and the analysis process in order to reduce validity threats with respect to bias of researchers.

2.3 Formal Methods

Formal methods are mathematical based techniques that can be used to design software systems. Concretely, formal methods provide the means for rigorous specification of software behaviors and verification of required properties. For this reason, formal methods are commonly used in the field of self-adaptive systems for modeling and reasoning of self-adaptation and for model-checking of system properties [188]. Through formal specification and verification, designers can study the correctness of system behaviors in order to provide assurances with respect to desired adaptation goals. In particular, in this research we focused on formalizing the design of the behaviors of a family of self-adaptive software and providing evidence to guarantee robustness and openness properties based on resources used in the system.

We applied formal methods for the rigorous design of a number of distributed self-adaptive systems [81, 82]. We applied timed-automata (TA) [4] as language for the specification of behaviors. A TA is a state machine extended with clock variables that can be used to synchronize two or more behaviors. As a language for property specification and verification of the correctness of behaviors, we used timed computation tree logic (TCTL) [3]. With TCTL we can describe state and path formulae over the state space of the behavior models that can be verified by a tool. In our research, we used UPPAAL [15] as a tool for the design and verification of the formal models. UPPAAL is one of the most prominent available tools for modeling and verification with TA and TCTL.

2.3.1 Results

We formalized distributed self-adaptive systems from two different domains.

The first domain is mobile learning, a branch within the technology-enhanced learning discipline. We selected a distributed collaborative mobile learning application in which we applied formal methods in order to provide assurances regarding the robustness of the application to support collaboration in the learning activities [81]. The formalization of a collaborative mobile learning system is described in Chapter 5.

The second domain in which we applied formal methods is robotics. We designed the behaviors for a multi-robot application in which robots have to perform transportation tasks in a warehouse. In this second case, we specified the behaviors and properties to guarantee robustness and openness of the system regarding the availability of lanes, locations and destinations in the layout that the robots can use
to drive in the warehouse [82]. A description of the formalization of this system is presented in Chapter 6.

Through the formalization of the self-adaptive behaviors we obtained rigorous descriptions of the systems’ behaviors. The models were complemented with specifications of properties of the system that we used to analyze the behaviors and provide guarantees for the desired system goals.

We followed the MAPE-K architecture-based approach [113] for the formalization of the self-adaptive behaviors and specified distinct modules for each of the adaptation functions of the MAPE loops (Monitor, Analyze, Plan and Execute). Using distinct modules for each of the adaptation functions allowed to rigorously study the behavior of the adaptation components, one at a time or in interaction. Thus, we could benefit from the clear separation of roles in the self-adaptive system and reason about behaviors within and between MAPE loops in order to provide guarantees for the desired adaptation properties.

The results from this phase of the research are:

- **R2** Descriptive instances of formal models to rigorously specify and verify robustness and openness concerns for a distributed self-adaptive system.

The results provide initial answers to the research questions RQ2 “How to model the behavior of MAPE-K components to guarantee robustness and openness properties?” and RQ3 “What properties must be verified to provide assurances with respect to the self-adaptive behaviors?”, that is:

- Rigorously specified behaviors for concrete self-adaptive applications with robustness and openness concerns.
- Property specifications for the verification of adaptation goals and correctness of behaviors for the self-adaptive applications.

The results from the collaborative mobile learning application were published at the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2013) [81].

### 2.3.2 Validity of the Results

Self-adaptive systems have the ability to adapt dynamically with changing conditions of the context in which they operate. Therefore, to verify the correctness of a self-adaptive behavior, its context must also be formalized (i.e. models are required of the managed system, of its environment and of the adaptation goals). Consequently, the guarantees provided for self-adaptation depend on the coverage and quality of the context models. Insufficient context models may restrict the verified space-state and miss execution traces that could occur during runtime. We designed the context automata based on extensive experiences with previous empirical studies in real settings [76, 77, 84], anticipating this validity threat.

### 2.4 Consolidating Patterns

Patterns document consolidated design expertise for future reuse. Patterns provide a vocabulary for communication among designers and they can accelerate the de-
2 Research Approach

sign of new systems. In the scope of this thesis, patterns consolidate expertise from the formal design of a family of self-adaptive systems with robustness and openness requirements. These patterns can enhance the design of rigorous behaviors for new systems with similar characteristics and requirements.

Based on the formal design of different self-adaptive systems with shared characteristics, we identified common features and attributes in the specifications. Miles and Huberman [135] denote the activities of the consolidation process as noting patterns. The consolidation process is an iterative effort in which new commonalities and variants are identified and tested. The results from one iteration are used in new design efforts. These efforts in turn provide input to the process, until a relatively stable set of patterns is derived and documented.

2.4.1 Results

This consolidation process resulted in a set of MAPE-K Formal Templates [82] for self-adaptive systems in the target domain (described in Section 1.3). The MAPE-K Formal Templates comprise patterns for rigorous design and patterns for verification of self-adaptive systems. The former include behavior specification patterns for internal MAPE-K components (i.e. Monitor, Analyze, Plan and Execute components) together with different interaction alternatives. The later include property specification patterns to specify adaptation goals, intra-component properties and inter-component properties. This consolidation of the MAPE-K Formal Templates is presented in Chapter 6.

In summary, the consolidation process resulted in:

- **R3** MAPE-K Formal Templates that comprise:
  - Behavior Specification Patterns
  - Property Specification Patterns

The MAPE-K Formal Templates provide an answer to research questions RQ2 "How to model the behavior of MAPE-K components to guarantee robustness and openness properties?" and RQ3 "What properties must be verified to provide assurances with respect to the self-adaptive behaviors?", respectively with:

- A definition of a set of behavior specification patterns to assist the design of MAPE-K self-adaptive behaviors;
- A definition of a set of property specifications patterns for the verification of MAPE-K self-adaptive behaviors.

2.4.2 Validity of the Results

The MAPE-K Formal Templates were derived through a consolidation process based on experiences in formalizing different self-adaptive systems in a target domain. Some of the characteristics of the target domain may not be well covered by the variability across these applications, which may hamper the use of the templates for the design of new self-adaptive systems in the target domain. We anticipated this threat to validity by designing multiple use case scenarios from different applications domains.

16
2.5 Case Studies

The identification of patterns from experiences is not free of subjectivism (threat to internal validity) [52]. During the consolidation process, the researcher that identified the templates may have identified and synthesized parts of the formal designs in a particular manner that may be biased to his specific criteria of interest. To anticipate this threat to validity, the templates were reviewed by two experts, and extensively discussed until an agreement was reached, which should minimize bias.

Another potential threat to validity relates to the granularity of the behavior and property specification patterns. During the consolidation process, we had to balance generality with usability. More specific templates are more expressive and provide more fine grained elements for modeling and verification. However, more specific templates introduce more complexity which may hamper the usability of the templates. It may also limit the applicability of the templates in terms of the target domain. In order to find the right balance, we derived the templates from different applications and tested the results to different scenarios.

2.5 Case Studies

A case study allows to understand "a real-world phenomena (the 'case') in depth and within its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident" [194]. Through multiple case studies, it is possible to enhance qualitative evidence [1,135,194], by relying on multiple sources, including interviews, documents, observations and recordings, among others [194].

In the last phase of the research, we carried out four case studies to assess and provide qualitative evidence regarding the level of reuse of the MAPE-K Formal Templates. The participants of the case studies, final year Master students in Software Technology, were asked to design and implement self-adaptive systems using the MAPE-K Formal Templates. We followed the guidelines of Runeson et al. [155] to design and execute the case studies. Concretely, we defined a protocol for the case studies. This protocol was reviewed by an external expert and after minor modifications used during the case study activities. The protocol is included in the thesis in Appendix C.

2.5.1 Results

The analysis of the study provides qualitative evidence for the reusability of the MAPE-K Formal Templates [82]. Particularly, the case studies demonstrate that the templates can be used in additional applications that fit the target domain. The participants successfully used the behavior and property specification patterns for the design of the different self-adaptive systems. One of the applications was a simple smart home system, another one was a fire monitoring system and the other two were vehicular traffic systems. Both robustness and openness concerns were considered. The case studies are described in Chapter 6.
2 Research Approach

The primary result of the case studies is:

**R4** Qualitative evidence of the reusability of the MAPE-K Formal Templates.

The result underpins and complements previous answers to the research questions RQ2 "How to model the behavior of MAPE-K components to guarantee robustness and openness properties?" and RQ3 "What properties must be verified to provide assurances with respect to the self-adaptive behaviors?", in particular the case studies provide:

- Qualitative evidence of the reusability of the Behavior Specification Patterns for self-adaptive systems in the target domain.
- Qualitative evidence of the reusability of the Property Specification Patterns for self-adaptive systems in the target domain.

2.5.2 Validity of the Results

The case studies were performed with Master students, which may imply threats to conclusion validity. In practice, the use of formal methods is not well spread; formal methods are only used by software engineers in particular critical domains. Although the students do not represent expert software engineers, they are relatively close to the population of interest [116].

A key objective of the case studies was to analyze the level of reusability of the templates to be applied to multiple application domains. To anticipate bias of the selection of application domains, the participants could freely select an application domain of their interest, with the only condition that the application should match with the characteristics of the target domain of the MAPE-K Formal Templates. The participants could also select the self-adaptation goal for their application. Furthermore, the participants were not involved during previous phases of this research, as we wanted to avoid validity threats that could arise from being familiar with the templates prior to the study.

To avoid narrow interpretation, we combined multiple sources for the analysis of the results, including documents (formal designs of the applications in multiple stages of the study), recordings during discussion sessions and interviews. The results of the analysis were cross-checked with the participants during a concluding reflection session.

The results of the case studies are limited to the particular cases that we analyzed, both in terms of the number of cases and complexity of the designs. This may imply threats to external validity [52]. Additional studies with more complex application scenarios in different domains can further enhance the level of evidence for the reusability of the MAPE-K Formal Templates.

2.6 Research Methods Summary

In this chapter, we have presented four research methods that we applied during this research. The choice for the methods was performed along the specific challenges we faced for each of the research tasks to achieve the research goals.
Particularly, we applied a SLR to obtain a systematic view on the state-of-the-art in formal methods for self-adaptive systems. We applied formal methods for the rigorous design of a number of self-adaptive systems and verification of the required adaptation goals and correctness of the MAPE feedback loops. We consolidated patterns from the expertise and knowledge we gained from designing different self-adaptive behaviors. The resulting set of MAPE-K Formal Templates was further assessed through a number of case studies to provide qualitative evidence of the templates’ reusability.
Chapter 3

Research Background

In this chapter, we introduce the basic concepts that underlie the research of this thesis. We start with introducing the principles of self-adaptation. Then we discuss the quality goals that we target in this work: robustness and openness. Finally, we summarize the basics of timed automata and timed computation tree logic, which we use as formal languages in this research. Related work specific to the topics presented in Chapters 4, 5 and 6 is discussed in these chapters.

3.1 Self-Adaptation

Self-adaptive software systems originate from the need to deal with the increasing complexity of systems that need to face with aspects that are uncertain at design time. Examples of uncertainties are dynamic environments in which the system is found, future new users' needs, subsystems that come and go at will and internal system's failures, etc. These uncertainties challenge the software design in providing desired quality properties, such as performance, robustness, openness, security and others.

Self-adaptation is based on the design principle of separation of concerns. Self-adaptation aims to separate the logic that deals with certain quality concerns of interest via an adaptation engine (aka managing system) from the domain functionality provided by the underlying domain specific application (managed system).

Over the last fifteen years, the software engineering community has increased attention on researching self-adaptive systems, and as a result, a number of approaches have appeared for the design of such systems. Two recent roadmap studies [48, 127] analyze the state of the art in the field of self-adaptive systems and identify open challenges for future research. Two particular challenges that directly relate to the research of this thesis are the engineering of self-adaptive systems and assurances for self-adaptive systems.

Regarding the engineering aspects, two prominent approaches to self-adaptation can be distinguished. The first approach is architecture-based self-adaptation [70, 113, 143]. This approach focuses on software components for feedback loops, models and functions for adaptation and the connection points between managed and managing systems. The second approach is control-based self-adaptation [66, 96]. This approach applies principles from control theory for the design and analysis of feedback loops to realize self-adaptation. Our focus is on architecture-based self-adaptation.
Chapter 3
Research Background

In this chapter, we introduce the basic concepts that underly the research of this thesis. We start with introducing the principles of self-adaptation. Then we discuss the quality goals that we target in this work: robustness and openness. Finally, we summarize the basics of timed automata and timed computation tree logic, which we use as formal languages in this research. Related work specific to the topics presented in Chapters 4, 5 and 6 is discussed in these chapters.

3.1 Self-Adaptation

Self-adaptive software systems originate from the need to deal with the increasing complexity of systems that need to face with aspects that are uncertain at design time. Examples of uncertainties are dynamic environments in which the system is found, future new users’ needs, subsystems that come and go at will and internal system’s failures, etc. These uncertainties challenge the software design in providing desired quality properties, such as performance, robustness, openness, security and others.

Self-adaptation is based on the design principle of separation of concerns. Self-adaptation aims to separate the logic that deals with certain quality concerns of interest via an adaptation engine (aka managing system) from the domain functionality provided by the underlying domain specific application (managed system).

Over the last fifteen years, the software engineering community has increased attention on researching self-adaptive systems, and as a result, a number of approaches have appeared for the design of such systems. Two recent roadmap studies [48, 127] analyze the state of the art in the field of self-adaptive systems and identify open challenges for future research. Two particular challenges that directly relate to the research of this thesis are the engineering of self-adaptive systems and assurances for self-adaptive systems.

Regarding the engineering aspects, two prominent approaches to self-adaptation can be distinguished. The first approach is architecture-based self-adaptation [70, 113, 143]. This approach focuses on software components for feedback loops, models and functions for adaptation and the connection points between managed and managing systems. The second approach is control-based self-adaptation [66, 96]. This approach applies principles from control theory for the design and analysis of feedback loops to realize self-adaptation. Our focus is on architecture-based self-adaptation.
3 Research Background

Below, we briefly present a number of prominent conceptual architectures for the design and implementation of self-adaptive systems. The differences between the approaches is out of the scope of this research.

3.1 Architecture-based Self-Adaptation

In their seminal work, Oreizy et al. [144] coined the term architecture-based self-adaptation and introduced the basic principles. The authors recognized the need of creating software systems that are capable to modify autonomously at runtime, without a need to restart.

![Figure 3.1: Envision of architecture based self-adaptive software from [144]](image)

The authors identify two interacting life-cycles that separate the adaptation management concerns from the evolution management concerns (Fig. 3.1). The adaptation management life-cycle monitors and evaluates the application behavior during execution and generates plans to enact adaptations when needed. The evolution management "focuses on the mechanisms employed to change the application software" [144] during system evolution.

3.1.2 MAPE-K Reference Model

In 2003, IBM [101] introduced the MAPE-K reference model that was further refined in 2006 [102]. MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) contains five components required by a system to become self-adaptive. A Monitoring component (represented as 1 in Fig. 3.2) acquires information from the sys-
3.1 Self-Adaptation

tem and the environment in which it operates, and updates the knowledge to be transferred to an Analyze component (2). Based on a base of knowledge (5), the Analyze component is responsible of determining the correctness of the managed system state with respect to one (or multiple) desired quality goal. Subsequently, the Plan component (3) puts together a plan that comprises actions. Finally, the Execute (4) component carries out the planned changes on the managed system in order to adapt.

![Figure 3.2: MAPE-K reference model for self-adaptive systems from [102]](image)

3.1.3 Feedback Control Loop

A feedback control loop [32] (marked inside the dotted line in Fig. 3.3) is based on principles of control theory. In control theory the system’s output is fed back into the system in order to modify the system’s behavior (via a Controller). The feedback control loop distinguishes between the executing processes and the environment. The objective of the feedback control loop-based design is to leverage the visibility of the control loop in the design and development phases of adaptive systems in order to facilitate analysis. Different dimensions of analysis are the system requirements, the adaptive design and implementation. Additionally, it provides mechanisms for system’s model validation. To do so, additional components (presented in the figure as Model Reference and Adjustment Mechanism) are integrated in the control loop in order to alter the Controller behavior. In a system based on the feedback control loop approach, the system receives a set of inputs $u_p$ and determines required adaptations in the system in order to achieve desired system properties $y_p$ in spite of potential disturbances $d$. The Adjustment Mechanism works with information describing the system behavior (Model Reference), the Controller actions $u$ and the current system qualities $y_p$.

Müller, Pezzè, and Shaw [137] presented an architecture to model feedback control architectures. The model, shown in Fig. 3.4, uses sensors to gather current data from the managed system (Executing system) and the environment (Operating environment) to create an up to date model of the current state of the system and
3.1 Self-Adaptation

Garlan et al. [70] presented a pioneering framework for self-adaptive systems (Fig. 3.5). Components in Rainbow realize a feedback loop that closely connects with the MAPE-K reference model. Rainbow separates the managing and managed system in two layers, referred as Architecture layer and System layer respectively. A system’s architectural model (Model manager) is part of Rainbow and assists the system for the reasoning of the current system’s behavior (through the Constraint evaluator) in order to plan (through the Adaptation engine) and execute (Adaptation executor) required actions in the system. The combination of these hotspots realize the self-adaptation loop. Rainbow supports monitoring and adaptation of software systems that are distributed in a network. However, the control of adaptation is centralized.

the environment, as well as predictions of the future state. Based on the models and given a set of system objectives, a set of correction actions are planned and transferred (Effect) back to the system.

3.1.4 Rainbow

Garlan et al. [70] presented a pioneering framework for self-adaptive systems (Fig. 3.5). Components in Rainbow realize a feedback loop that closely connects with the MAPE-K reference model. Rainbow separates the managing and managed system in two layers, referred as Architecture layer and System layer respectively. A system’s architectural model (Model manager) is part of Rainbow and assists the system for the reasoning of the current system’s behavior (through the Constraint evaluator) in order to plan (through the Adaptation engine) and execute (Adaptation executor) required actions in the system. The combination of these hotspots realize the self-adaptation loop. Rainbow supports monitoring and adaptation of software systems that are distributed in a network. However, the control of adaptation is centralized.
3.1 Self-Adaptation

3.1.5 3-Layer Architecture Model

Based on previous efforts from Gat [71], Kramer and Magee [121] presented an structural approach for the design of self-adaptive systems, which advocates for the separation of goal, change and component concerns. The model provides a high level of abstraction in terms of involved components, rather than providing the logical behaviors to be found in each layer. This is not a weakness of the model, on the contrary, that is its strength. The approach offers the right level of abstraction and generality to design self-adaptive systems.

In their approach (see Fig. 3.6), the component control layer has the responsibility of providing the domain specific application functionalities. On the middle layer, the system is responsible of applying adaptation plans on the component layer, based on a set of plan algorithms. These plans are triggered (when required) in response to events on the underlaying layer. The top layer, Goal Management, is in charge of maintaining the set of plans present in the underlaying layer in order to achieve a set of goal concerns that may be dynamically changing at runtime.

3.1.6 Common Aspects

The above-presented studies offer different approaches to design self-adaptive systems. However, there are certain aspects that are shared among these studies.

One important common denominator is the separation of concerns between the managed and the managing systems. This principle aims to reduce the system’s complexity, by enabling software designers to focus on specific concerns and reduce concern interference that could originate if domain specific functionalities and adaptive behaviors were intertwined.

Figure 3.5: Rainbow framework for self-adaptive systems from [70]
3 Research Background

![3-Layer Architecture Model adapted from [121]](image)

Second, the studies suggest the use of layers, as a method to keep the separation of concerns and to reduce the number of interaction points between the managed and managing systems.

Finally, the studies incorporate knowledge structures that include one or more of the following aspects: goals, environment, managed system state and adaptation mechanism state [186]. MAPE-K aggregates them in the *Knowledge* representation. In the Control Feedback Loop approach, the model makes a distinction between the *Objective*, representing the system goals and *Models*, offering an abstraction of the environment and managed system states. In Rainbow, the knowledge is not explicitly represented in the design, but it is part of the hotspots that compose the adaptation layer. The 3-Layer Architecture Model divides models into two parts. *Goal Management* contains knowledge regarding adaptation goals and state of the managing system; while the state of the managed system and the environment are part of *Change Management*.

### 3.2 Quality Concerns

Self-adaptation aims to add particular quality properties to a software system. It is therefore important to provide a clear definition regarding the quality concerns that we aim to address in this thesis.

The IEEE Standard Glossary of Software Engineering Terminology defines *quality* as

"(1) The degree to which a system, component, or process meets specified requirements.

(2) The degree to which a system, component, or process meets customer or user needs or expectations." [103]

From their definition, we understand that qualities are related to specific requirements that a user expects from a system. In the context of this thesis, we aim at providing evidence that particular quality concerns are assured. Concretely, we
focus on robustness and openness. Therefore, it is important to provide an understanding on the definition of these terms. Below we provide an overview of the concepts and relevant efforts that have focused on them.

### 3.2.1 Robustness

The IEEE Standard Glossary of Software Engineering Terminology defines _robustness_ as

"The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions." [103]

Robustness is also commonly covered when referring to system’s _safety_ properties. Concretely, _behavioral safety_ [34] refers to a desired behavior of a system that would not crash or change to undesired behavior during runtime when encountering errors.

System robustness is a well-studied quality property [145, 167]. In 2002, Esfahani, Kouroshfar, and Malek [62] ran the First Workshop on Self-Healing Systems, WOSS2002. The workshop focused on the realization of software mechanisms to offer self-healing towards robustness. Since the date, there has been a vast work to realize robust systems via self-adaptation.

Based on previous work on robustness testing [134], Cámara et al. [40] propose an approach to evaluate and provide assurances regarding system robustness, by identifying adequate robustness tests for a self-adaptive system and altering probe behaviors to simulate errors to be studied. In [184], the authors focus on robustness concerns and compare different approaches for self-adaptation in distributed systems. Concretely, the authors elaborate on benefits of self-adaptation using multi-agent systems with respect to client-server and service-oriented approaches.

Despite the vast body of research, providing assurances for system robustness remains a challenge, in particular in the context of self-adaptation. There is a need to design systems that "provide some elasticity in order to be robust against some kinds of change, or [are] able to manage change by generating alternative solutions" [127].

In this research, we study robustness in distributed self-adaptive systems. Our particular focus is on handling failures of resources that are needed to satisfy the goals of the managed system.

### 3.2.2 Openness

Openness is commonly considered as type of _flexibility_ software quality. System openness has recently emerged as a desired system property, due to the high dynamism of systems’ environments and the fast evolution of software development. Zambonelli and Parunak [196] identified in 2003 _openness_ as one of the four main characteristics that, by then future software systems, would possess. The authors defined openness as follows: "openness: software systems are subject to decentralized management and can dynamically change their structure" [196].
3 Research Background

Another more general view of open systems is provided by Buckley et al. [34]: "Software systems are open if they are specifically built to allow for extensions. Open systems usually come with a framework for facilitating the inclusion of extensions". In this definition, the authors consider a system to be open when it allows all kind of extensions, such as software components, plug ins, etc., in addition to physical resources that may come and go. Still another point of view was brought by Oreizy et al. [144]. The authors considered the capability of a system to accept new behaviors coming from the outside as an openness property for certain self-adaptive systems.

In this work, we focus on openness concerns in distributed systems to allow the management of resources that come and go at will. Therefore, we refer to openness as a quality that fits into the definition of Zambonelli and Parunak [196].

Using self-adaptation to deal with openness is still a challenge [48, 156]. An open system implies a high level of system complexity [146], as entities in the system must be prepared to allow new interactions with arriving elements. Moreover, openness may affect other quality concerns, such as security and performance. Therefore, it is of high relevance to establish and follow good practices for the design of open self-adaptive systems that allow their analysis.

Some examples of research in self-adaptive distributed systems are [56, 104, 119, 146, 184]. Iftikhar and Weyns [104] performed a case study aiming to study the benefits of the MAPE-K architectural approach to provide robustness and openness properties to systems. MetaSelf [56] is a software architecture based on feedback control loops to allow, among other goals, openness in a system via self-organization and dependability rules. Weyns and Georgeff [184] elaborated on the benefits of using multiagent systems, as a loosely coupled solution, in order to offer openness in a system to adapt to changing business needs. Parunak and Brueckner [146] executed a survey on the self-adaptation field and elaborated on the use of self-organizing systems to offer, among others, openness quality to a system. Kota, Gibbins, and Jennings [119] presented an agent-based self-adaptive solution for finding, incorporating and sharing resources in a platform in order to improve performance in the system. In their study, the authors compare this solution with previous agent solutions and claim relevant improvements with respect to previous agent-based studies. However, this study does not apply separation of concerns through a self-adaptive layer, but integrates self-adaptation logics into the agent behavior. This trend is quite common in the autonomous agent community.

In our work, we address openness via self-adaptation following the MAPE-K architecture-based approach in order to manage the complexity of the system.

3.3 Assurances

The main reason that motivates introducing a self-adaptive layer into a system is to provide desired system qualities. However, designing self-adaptation for complex systems, such as distributed and high-intense processing systems increases the complexity in designing self-adaptive solutions, as adaptation may require the involvement of multiple nodes and the cooperation of their behaviors. As a re-
result, it is still a challenge to provide assurances that desired quality properties are achieved in the system [48, 127, 128].

The recent Dagstuhl seminar on "Software Engineering for Self-Adaptive Systems: Assurances" [128] focused on this topic. In certain domains, such as e-health, vehicular transportation and other safety-critical systems, it is critical to be able to adapt and correct the system when failures happen, but it is as imperative to provide guarantees that the adaptations are correctly performed and that the system satisfies the desired quality goals.

One widely-accepted approach to provide assurances regarding adaptation is by means of formal methods.

3.4 Formal Methods

With formal methods, we refer to design methods that have mathematical underpinning. The core benefit of applying formal methods is they provide the means for the rigorous specification of software behaviors and the verification of desired software properties. For this reason, formal methods are commonly used for modeling and reasoning with respect to systems’ behavior and for model-checking and proving particular system properties [188]. Formal models are commonly used during design time. However, recently, an increasing number of studies have been investigating the application of formal methods during runtime in order to capture systems’ behaviors and dynamically generate models of the systems for reasoning purposes, e.g. [37, 38].

Systems’ behaviors can be specified with different formal languages. Transition systems and automata are commonly used by the software engineering community for the design of self-adaptive systems. In our research we use timed-automata (TA) as the language to specify the system’s behavior, as graphical representations and state transitions are easily understood by humans [47]. We use timed computational tree logic (TCTL) as the formal language to specify system properties. Below we offer an overview of TA and TCTL languages.

3.4.1 Timed Automata

A timed automaton (TA) is a state machine extended with clock variables that models a behavior. Clock variables are used to synchronize behaviors. Additionally, automata can communicate through channels, where the sender behavior $x!$ synchronizes with the receiver behavior $x?$.

Formally, a TA is defined as a tuple

$$(N, l_0, T, \text{Label}, C, \text{clock}, \text{guard}, \text{invariant}) \ [19]$$

in which:

- $N$ is a non-empty, finite set of locations (or nodes) with an initial location $l_0 \in N$;
- $T \subseteq L \times L$ is a set of transitions;
- $\text{Label} : N \to 2^{AP}$ is a function that assigns to each location $l \in N$ a set $\text{Label}(l)$ of atomic positions;
3 Research Background

\[
C \quad \text{a finite set of clocks;}
\]

\[
clock : T \rightarrow 2^C \quad \text{a function that assigns to each transition } t \in T \text{ a set of clocks } \text{clocks}(t);
\]

\[
guard : T \rightarrow \Psi(C) \quad \text{a function that labels each transition } t \in T \text{ with a clock constraint } \text{guard}(t) \text{ over } C;
\]

\[
inv : N \rightarrow \Psi(C) \quad \text{a function that assigns to each node an invariant.}
\]

In this work, we use UPPAAL [15] to model TA. In UPPAAL behavior specifications can be complemented with expressions specified in a C-like language to define data structures (\textit{struct} concept) and functions. We follow the UPPAAL notation conventions for the description of the behavior templates, presented in Table 3.2.

**Table 3.2:** Conventions in Timed Automata figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State A</td>
<td>State</td>
<td>States of behaviors are represented by circles and (optionally) annotated in red on top of the associated state.</td>
</tr>
<tr>
<td></td>
<td>Committed State</td>
<td>Committed states are represented with circles containing the c character. Committed states must be left without time consumption.</td>
</tr>
<tr>
<td></td>
<td>Urgent State</td>
<td>Urgent states of behaviors are represented with circles containing the u character. Urgent states must be left as soon as exiting conditions are found (normally defined by conditions on outgoing transitions).</td>
</tr>
<tr>
<td></td>
<td>Initial State</td>
<td>Initial states of behaviors are represented by double-lined circles. There must be one unique Initial State per automaton, specifying the behavior state when the system starts.</td>
</tr>
<tr>
<td></td>
<td>Invariants</td>
<td>Invariants that need to be satisfied in certain states are annotated in purple under the related state.</td>
</tr>
<tr>
<td></td>
<td>Transition</td>
<td>Transitions between two states are represented by directional arrows, showing the origin and destination of the transition.</td>
</tr>
<tr>
<td></td>
<td>Conditions</td>
<td>Conditions to enable firing of transitions between states are annotated in green under the related transition.</td>
</tr>
<tr>
<td></td>
<td>Signal</td>
<td>Signals used for communication between behaviors are annotated dark blue over the associated transition.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>Functions associated with behaviors are annotated in light blue under the associated transition.</td>
</tr>
</tbody>
</table>

In timed automata, transitions between states can be taken via event-triggering or time-triggering.

With an event-triggering mechanism, one automaton triggers another via a signal sent through a channel. This case is illustrated in Fig. 3.7-left, where an automaton with a behavior \textit{B1} fires a signal in order to trigger a transition on the
behavior $B2$. In this case, we say that the second behavior $B2$ is dependent on $B1$, as this would not be able to execute transitions without the corresponding signal from the former. Optionally, data generated in a behavior $B1$ may be transferred to a behavior $B2$ using a knowledge repository.

With time-triggering mechanisms, a transition is fired based on state invariants and time conditions. This case is shown in Fig. 3.7-right, where a behavior $B2$ autonomously executes transitions in the automaton on a time-based approach ($\text{Tick}$). Due to the autonomy of the behavior with respect to another $B1$ behavior, time triggering requires a data repository to store shared knowledge whenever information needs to be transferred between behaviors. In general, time-triggering is less efficient in terms of verification (as it implies more execution threads).

![Figure 3.7: Event- (left) and time- (right) triggering approaches for component behaviors](image)

### 3.4.2 Timed Computational Tree Logic

Timed computational tree logic (TCTL) is a formal language for property specification based on CTL extended with clock variables. Expressions in TCTL describe state and path formulae allowing the verification of properties of interest, such as reachability (a system should/can/cannot reach a particular state or states), liveness (something eventually will hold), etc. The syntax supported by UPPAAL for property specification is defined as follows [97]:

$$\phi ::= p \mid !\phi \mid \phi \lor \phi \mid EX \phi \mid E[\phi \lor \phi] \mid A[\phi \lor \phi] \mid z.\phi$$

Where:

- $\phi$ is a property to be specified;
- $p$ is an atomic proposition or a clock constraint;
- $EX$ is an expression applied on a property;
- $E$ expresses the existence of a path that fulfills a property;
- $A$ expresses the invariant fulfillment of a property;
- $z$ expresses a state predicate.
Chapter 4

Systematic Literature Review on Formal Methods for Self-Adaptive Systems

One of the current challenges in the field of self-adaptive systems is to provide evidence that the adaptation goals are assured. Formal methods provide the means to rigorously specify and verify the behavior of self-adaptive systems. In this chapter, we present the results from a Systematic Literature Review (SLR) on formal methods for self-adaptive systems.

The goal of the SLR is to answer the research question "RQ1: Which formal methods have been used in self-adaptive systems?". Concrete objectives are:

• to identify trends in the use of formal methods in self-adaptive systems;
• to identify for what purpose have these methods been used;
• to assess for what type of self-adaptive applications formal methods have been applied;
• to identify which tools have been used to apply formal methods in self-adaptive systems.

The SLR shows that the attention for self-adaptive software systems is gradually increasing, but the number of studies that employ formal methods remains low. The main focus of formalization is on modeling and reasoning. Model checking and theorem proving have gained limited attention. Model checking and theorem proving are of high importance to provide evidence of systems' self-adaptation, therefore more efforts should be focused on this direction.

Regular algebra, automata and transition systems are the predominant formal languages for the specification of formal behaviors. 36.0% of the studies use some logic as property specification language. In this thesis, we use timed automata and timed computation tree logic as specification languages for self-adaptive systems. This choice is in line with the state of the art approaches that use formal methods in self-adaptive systems.

The main concerns of interest in formalization of self-adaptation are efficiency/performance and reliability. Important adaptation concerns, such as security and scalability, are hardly considered. To verify the concerns of interest, a set of new properties are defined, such as interference freedom, responsiveness, mismatch, and loss-tolerance. The SLR results underpin that more research is required on self-adaptation to deal with robustness and openness concerns.

3 Research Background

Below, we explain the different symbols that can be used in the TCTL syntaxes.

\[ E[p] \] There is a path in which \( p \) will always hold. For example, there is a path where a GPS module has always NULL coordinates. This is a broken during manufacturing GPS module.

\[ E <> p \] It is possible to reach a state in which \( p \) is satisfied. For example, a GPS module will eventually acquire position coordinates.

\[ A[p] \] \( p \) holds invariantly. For example, location coordinates are always on earth.

\[ A <> p \] \( p \) is inevitable. It will eventually happen. For example, a GPS will eventually fail in gathering accurate positions.

\[ A \rightarrow B \] If \( A \) becomes true, then \( B \) will inevitably be true. For example, if a GPS module turns on, it will eventually collect GPS coordinates.

\[ A \text{ imply } B \] If \( A \) becomes true, \( B \) will become true at the same time. For example, if a GPS acquires location coordinates, then some accuracy values with respect to the location coordinates are obtained.
Chapter 4

Systematic Literature Review on Formal Methods for Self-Adaptive Systems

One of the current challenges in the field of self-adaptive systems is to provide evidence that the adaptation goals are assured. Formal methods provide the means to rigorously specify and verify the behavior of self-adaptive systems. In this chapter, we present the results from a Systematic Literature Review (SLR) on formal methods for self-adaptive systems.

The goal of the SLR is to answer the research question "RQ1: Which formal methods have been used in self-adaptive systems?". Concrete objectives are:

- to identify trends in the use of formal methods in self-adaptive systems;
- to identify for what purpose have these methods been used;
- to assess for what type of self-adaptive applications formal methods have been applied;
- to identify which tools have been used to apply formal methods in self-adaptive systems.

The SLR shows that the attention for self-adaptive software systems is gradually increasing, but the number of studies that employ formal methods remains low. The main focus of formalization is on modeling and reasoning. Model checking and theorem proving have gained limited attention. Model checking and theorem proving are of high importance to provide evidence of systems’ self-adaptation, therefore more efforts should be focused on this direction.

Regular algebra, automata and transition systems are the predominant formal languages for the specification of formal behaviors. 36.0% of the studies use some logic as property specification language. In this thesis, we use timed automata and timed computation tree logic as specification languages for self-adaptive systems. This choice is in line with the state of the art approaches that use formal methods in self-adaptive systems.

The main concerns of interest in formalization of self-adaptation are efficiency/performance and reliability. Important adaptation concerns, such as security and scalability, are hardly considered. To verify the concerns of interest, a set of new properties are defined, such as interference freedom, responsiveness, mismatch, and loss-tolerance. The SLR results underpin that more research is required on self-adaptation to deal with robustness and openness concerns.

33

Abstract
One major challenge in self-adaptive systems is to assure the required quality properties. Formal methods provide the means to rigorously specify and reason about the behaviors of self-adaptive systems, both at design time and runtime. To the best of our knowledge, no systematic study has been performed on the use of formal methods in self-adaptive systems. As a result, there is no clear view on what methods have been used to verify self-adaptive systems, and what support these methods offer to software developers. As such insight is important for researchers and engineers, we performed a systematic literature review covering 12 main software engineering venues and 4 journals, resulting in 75 papers used for data collection. The study shows that the attention for self-adaptive software systems is gradually increasing, but the number of studies that employ formal methods remains low. The main focus of formalization is on modeling and reasoning. Model checking and theorem proving have gained limited attention. The main concerns of interest in formalization of self-adaptation are efficiency/performance and reliability. Important adaptation concerns, such as security and scalability, are hardly considered. To verify the concerns of interest, a set of new properties are defined, such as interference freedom, responsiveness, mismatch, and loss-tolerance. A relevant part of the studies use formal methods at runtime, but the use is limited to modeling and analysis. Formal methods can be applied to other runtime activities of self-adaptation, and there is a need for light-weight tools to support runtime verification.

4.1 Introduction
Self-adaptation has been widely recognized as an effective approach to deal with the increasing complexity and dynamicity of modern software systems [113, 121, 143]. A self-adaptive system comprises two parts: the managed system (also called system layer [70], managed resources [113], core function [156], base-level subsystem [187]) that deals with the domain functionality, and the managing system (or architecture layer [70], autonomic manager [113], adaptation engine [156], reflective subsystem [187]) that deals with the adaptations of the managed system to achieve particular quality objectives.

Reflecting upon the results of the SLR, we observed that despite the steady increase in the use of formal methods in the field, there is little reuse of behavior models and properties for specific self-adaptive systems. There is a lack of consolidation of design knowledge that uses formal models for self-adaptive systems. This observation was a key motivating driver for the research performed in this thesis.

Abstract

One major challenge in self-adaptive systems is to assure the required quality properties. Formal methods provide the means to rigorously specify and reason about the behaviors of self-adaptive systems, both at design time and runtime. To the best of our knowledge, no systematic study has been performed on the use of formal methods in self-adaptive systems. As a result, there is no clear view on what methods have been used to verify self-adaptive systems, and what support these methods offer to software developers. As such insight is important for researchers and engineers, we performed a systematic literature review covering 12 main software engineering venues and 4 journals, resulting in 75 papers used for data collection. The study shows that the attention for self-adaptive software systems is gradually increasing, but the number of studies that employ formal methods remains low. The main focus of formalization is on modeling and reasoning. Model checking and theorem proving have gained limited attention. The main concerns of interest in formalization of self-adaptation are efficiency/performance and reliability. Important adaptation concerns, such as security and scalability, are hardly considered. To verify the concerns of interest, a set of new properties are defined, such as interference freedom, responsiveness, mismatch, and loss-tolerance. A relevant part of the studies use formal methods at runtime, but the use is limited to modeling and analysis. Formal methods can be applied to other runtime activities of self-adaptation, and there is a need for light-weight tools to support runtime verification.

4.1 Introduction

Self-adaptation has been widely recognized as an effective approach to deal with the increasing complexity and dynamicity of modern software systems [113, 121, 143]. A self-adaptive system comprises two parts: the managed system (also called system layer [70], managed resources [113], core function [156], base-level subsystem [187]) that deals with the domain functionality, and the managing system (or architecture layer [70], autonomic manager [113], adaptation engine [156], reflective subsystem [187]) that deals with the adaptations of the managed system to achieve particular quality objectives. The key underlying idea of self-adaptation is complexity management through separation of concerns. One major challenge in self-adaptive systems is to provide guarantees about the required quality properties. Formal methods provide the means to rigorously specify and verify the behavior of self-adaptive systems. Formal methods have been applied during sys-
tem development, but also during runtime to provide guarantees about the required properties of self-adaptive systems [130, 174, 176, 187, 198].

In 2004, Bradbury et al. [26] surveyed 14 formal specification approaches for self-adaption based on graphs, process algebras, and logic formalisms. The survey evaluated the ability of each approach to specify self-managing systems, and concluded that existing approaches need to be enhanced to address issues regarding expressiveness and scalability. [48, 57, 100, 127, 156] summarize achievements of the field and outline challenges for future work, including challenges with respect to the application of formal methods for verification and validation of self-adaptive systems.

These studies provide insight in the potential use of formal methods in self-adaptive systems. However, to the best of our knowledge, no systematic study has been performed on the actual use of formal methods in self-adaptive systems. As a result, there is no clear view on what methods have been used to specify and verify self-adaptive systems, and what support these methods actually offer to software developers. However, such an insight is important for researchers and engineers.

The overall objective of this paper is to identify which formal methods have been used in self-adaptive systems and for what purpose these methods have been used. We also aim to assess for what type of self-adaptive applications formal methods have been applied, and which tools authors have used. To that end, we have performed a systematic literature review. From the study, we derive conclusions concerning state of the art formal methods in self-adaptive systems, and suggest ideas for future research. All the material that was used for the study together with the extracted data is available at http://homepage.lnu.se/staff/daweaa/SLR-FMSAS.htm.

Paper Overview.
In section 4.2 we give an overview of the method we used in our study. In section 4.3 we explain study planning. We explain the research questions we address, define search strategy and scope, and summarize the data items that were collected. In section 4.4 we present the data extracted from the primary studies (i.e., the selected studies after filtering), and interpret this data answering the research questions. Section C.9 discusses limitations of our study. We conclude with a discussion of opportunities for future research in section 4.6.

4.2 Research Method

Our study followed the principles of a systematic literature review [115], which is a well-defined approach to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest. Figure B.1 shows an overview of the three-phased method we applied in the study.

Four researchers were involved in the literature study. In review planning (Phase 1), the review protocol was defined, which includes the definition of research questions, the search strategy and scope, and the data items that had to be collected. The protocol was defined interactively by the team of reviewers. The research ques-
4.2 Research Method

Figure 4.1: Overview of the systematic review process (adapted from [28]).

Our study followed the principles of a systematic literature review [115], which is a well-defined approach to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest. Figure B.1 shows an overview of the three-phased method we applied in the study.

Four researchers were involved in the literature study. In review planning (Phase 1), the review protocol was defined, which includes the definition of research questions, the search strategy and scope, and the data items that had to be collected. The protocol was defined interactively by the team of reviewers. The research questions express the research topics of interest in this literature review. The scope of the review was based on the identification of the main workshops, conferences, and journals in the field. As search strategy, we combined automatic with manual search. Automatic search was defined as a two-step process for which two search strings were defined. The first string aims to select the studies on self-adaptive systems, and the second string aims to filter the studies on formal methods. For the manual search, inclusion and exclusion criteria were defined. Next, data items were identified and for each item a set of options was defined. The definition of data items was based on information derived from literature sources and from experiences with a preceding literature review [189]. For some of the data items, additional options were introduced during the review process, in particular for the set of languages used for modeling systems and specifying properties, and the set of tools. The protocol was crosschecked by an external reviewer and the feedback was used to make small adaptations.

Subsequently, the four researchers conducted the review (Phase 2). Studies were automatically selected based on the search criteria defined in Phase 1. One reviewer was responsible for automatic search. Manual search was performed by two researchers that checked each paper independently based on inclusion/exclusion criteria, to select the studies for answering the research questions of the study. Conflicts were resolved and if no consensus could be reached, the other researchers
were involved to come to a decision. Once the primary studies were selected, the
data items were collected by the four reviewers, which was obviously a manual
process. Collected data items were crosschecked and in case of disagreement,
conflicts were resolved. Finally, the data derived from the the primary studies was
collated and summarized to answer the research questions.

One of the reviewers coordinated the writing of this review report (Phase 3), in
close consultation with the other reviewers.

4.3 Planning Review

During planning, the protocol for the review is defined, which includes three main
steps: specify research questions, define search strategy and scope, and define data
items.

4.3.1 Research Questions

We formulated the general goal of the study through Goal-Question-Metric
(GQM) perspectives (purpose, issue, object, viewpoint) [11]:

- **Purpose**: Understand and characterize
- **Issue**: the use of formal methods
- **Object**: in self-adaptive software systems
- **Viewpoint**: from a researcher’s and engineer’s viewpoint.

The general research question translates to four concrete research questions:

- **RQ0**: Which are the trends in applying formal methods in self-adaptive systems?
- **RQ1**: Which formal methods have been used in self-adaptive systems?
- **RQ2**: For which adaptation concerns have formal methods been used?
- **RQ3**: How have formal methods been used to deal with concerns of self-adaptive systems?

With RQ0, we want to get insight in the use of formal methods by researchers on
self-adaptive systems both in time and space. RQ1 is motivated by the need to get
insight in what kind of formal methods have been used for self-adaptive systems.
This question aims to assess which languages have been used for modeling sys-
tems, verifying properties, and which tools have been used for this. With RQ2,
we aim to understand why formal methods have been used in self-adaptive sys-
tems. Concretely, we aim to assess for which concerns formal methods have been
used (reliability, performance, functional correctness of adaptations, etc.), which
properties have been verified for that (safety, liveness, etc.), and for which type of
applications these methods have been used. Finally, RQ3 aims to access how for-
mal methods have been used to provide guarantees about concerns in self-adaptive
systems (reasoning, model checking, at design time, runtime, etc.).

38
4.3 Planning Review

4.3.2 Search Scope and Strategy

The scope of the search is defined in two dimensions: time and space. Regarding time, we searched studies published from Jan. 2000 until Dec. 2011. 2000 was used as starting year since self-adaptive systems became subject of active research around that time. Regarding space, we included the primary venues for publishing research results on self-adaptive systems, as well as the major conferences and journals on software engineering. The selected venues are listed in Table B.1. The Rank is based on the Australian Research Council ranking (www.arc.gov.au/era/era_2010/).

<table>
<thead>
<tr>
<th>ID</th>
<th>Venue</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASE</td>
<td>International Conference on Automated Software Engineering</td>
<td>A</td>
</tr>
<tr>
<td>Adaptive</td>
<td>Adaptive and Self-adaptive Systems and Applications</td>
<td>n/a</td>
</tr>
<tr>
<td>DEAS</td>
<td>Design and Evolution of Autonomic Application Software</td>
<td>n/a</td>
</tr>
<tr>
<td>FSE</td>
<td>Foundations of Software Engineering</td>
<td>A</td>
</tr>
<tr>
<td>ICAC</td>
<td>International Conference on Autonomic Computing</td>
<td>A</td>
</tr>
<tr>
<td>ICSE</td>
<td>International Conference on Software Engineering</td>
<td>A</td>
</tr>
<tr>
<td>ICSEM</td>
<td>International Conference on Software Maintenance</td>
<td>A</td>
</tr>
<tr>
<td>ISSTA</td>
<td>International Symposium on Software Testing and Analysis</td>
<td>A</td>
</tr>
<tr>
<td>SASO</td>
<td>Self-Adaptive and Self-Organizing Systems</td>
<td>n/a</td>
</tr>
<tr>
<td>SEAMS</td>
<td>Software Engineering for Adaptive and Self-Managing Systems</td>
<td>n/a</td>
</tr>
<tr>
<td>WICSA</td>
<td>Working International Conference on Software Architecture</td>
<td>A</td>
</tr>
<tr>
<td>WOSS</td>
<td>Workshop on Self-Healing</td>
<td>n/a</td>
</tr>
<tr>
<td>JSS</td>
<td>Journal of Systems and Software</td>
<td>A</td>
</tr>
<tr>
<td>TAAS</td>
<td>Transactions on Autonomous and Adaptive Systems</td>
<td>n/a</td>
</tr>
<tr>
<td>TOSEM</td>
<td>Transactions on Software Engineering and Methodology</td>
<td>A*</td>
</tr>
<tr>
<td>TSE</td>
<td>Transactions on Software Engineering</td>
<td>A*</td>
</tr>
</tbody>
</table>

Table 4.1: Searched conferences and journals

Our search strategy combines automatic with manual search. Automatic search comprised of two steps: first we selected the studies that are relevant for self-adaptive systems, and then we filtered the studies that use formal methods. We used the following search string in the first step:

\[
((\text{Title:adaptive OR Title:adaptation OR Title:self OR Title:autonomic OR Title:autonomous}) \text{OR} (\text{Abstract:adaptive OR Abstract:adaptation OR Abstract:self OR Abstract:autonomic OR Abstract:autonomous}))
\]

The keywords provide the main terms that different communities use to refer to self-adaptive systems. We applied pilot searches on the set of studies from SEAMS, ICAC and TAAS to ensure that the keywords provide the right scope.

In the second step, we used the following search string:

"Transition system" OR automata OR Markov OR Petri OR "state machine" OR calculus OR algebra OR grammar OR CSP OR "temporal logic" OR LTL OR "tree logic" OR CTL OR "first order logic" OR "reaction rule" OR probabilistic OR stochastic OR "graph rewriting" OR "graph representation"
This set of keywords defines key terms that refer to the use of formal methods. We derived the terms from four sources: Baier and Katoen’s book on model checking [7], the survey of Bradbury et al. on formal methods in self-adaptive systems [26], Villegas’ paper in quality evaluation of self-adaptive systems [177], and Tamura et al.’s recent roadmap paper on verification of self-adaptive systems [174]. For the second filtering, we again applied pilot searches to ensure that the keywords provide the right scope of studies.

We further refined the studies resulting from automatic search using a manual search step. The goal of this step is to identify the primary studies that are directly related to the research questions. To that end, we defined the following inclusion/exclusion criteria:

- **Inclusion criterion 1**: The study formalizes (at least a part of) the system as well as properties of the system. Rationale: we might have included studies which employ some formal terms, but do not actually employ formal methods for a particular purpose related to self-adaptation.

- **Inclusion criterion 2**: The study separates the domain logic and the adaptation logic, that is, the mechanisms to realize adaptation are not (or only partially) interwoven with the regular functionality. Rationale: the focus of our study is on self-adaptive systems that separate the adaptation concerns from the domain functionality.

- **Exclusion criterion 1**: A study that is an editorial, abstract or a short paper. Rationale: these studies do not provide a reasonable amount of information.

- **Exclusion criterion 2**: A study that focuses on formal techniques themselves, rather than on the use of formal methods for self-adaptation concerns. Rationale: these studies do not provide information regarding the research questions.

A study was selected when it met both inclusion criteria and eliminated if it met any of the exclusion criterion.

### 4.3.3 Data Items

Items were defined to collect the necessary data from the studies to answer the research questions. For each primary study, the data items shown in Table B.2 were collected. Each study was read in detail by two reviewers, each reviewer extracting the data in a form. This data were used during a discussion to resolve conflicts and reach consensus for the data items. This discussion involved the other reviewers when no agreement could be reached. Note that the lists of options of some of the data items were extended during the review.

The data items author(s), and title (F1-F2) were used for documentation. Year (F3) and Venue (F4) are used for answering RQ0.

For modeling language (F5) we defined the following options: regular algebra (basic math, equations, set theory, etc.), state machines, transition systems, automata, Markov models (inc. Markov chains, Markov decision processes, etc.),
process calculi (CSP, \(\pi\)-calculus, etc.), Petri nets, graphs, Z notation, ADL (formally founded architecture description languages). This list was extended while the data was collected.

Property specification/verification (F6) options are: modeling language (the same language is used for modeling the system and specifying properties), first-order logic (FOL), linear temporal logic (LTL), computation tree logic (CTL), timed computation tree logic (TCTL), probabilistic computation tree logic (PCTL), and graph grammar. This initial list was extended while data was collected from the studies.

Concerns subject of formal verification (F7) refers to the self-adaptive aspects of interest of formalization. The options (based on IEEE 9126 and ISO/IEC 25012) are:

- reliability (fault tolerance, recoverability): capability of software to maintain its level of performance under stated conditions for a stated period of time
- availability: the degree to which the software is in a functioning condition, i.e. capable to perform its intended functions
- usability (ease of learning, communicativeness): effort needed to use the system
- efficiency/performance (time behavior, resource utilization): efficiency of the software by using the appropriate resources under stated conditions and in a specific context of use
- maintainability (analyzability, changeability, stability, testability): effort needed to make specified modifications
- portability: ability of software to be transferred from one environment to another
- security: ability of the system to protect against misuse
- accuracy: the extent to which the software realizes the intended behavior in a specific context of use

<table>
<thead>
<tr>
<th>Item</th>
<th>Field</th>
<th>Research question</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Author(s)</td>
<td>Documentation</td>
</tr>
<tr>
<td>F2</td>
<td>Title</td>
<td>Documentation</td>
</tr>
<tr>
<td>F3</td>
<td>Year</td>
<td>RQ0</td>
</tr>
<tr>
<td>F4</td>
<td>Venue</td>
<td>RQ0</td>
</tr>
<tr>
<td>F5</td>
<td>Modeling language</td>
<td>RQ1</td>
</tr>
<tr>
<td>F6</td>
<td>Property specification/verification</td>
<td>RQ1</td>
</tr>
<tr>
<td>F7</td>
<td>Concerns subject of formal verification</td>
<td>RQ2</td>
</tr>
<tr>
<td>F8</td>
<td>Verified properties</td>
<td>RQ3</td>
</tr>
<tr>
<td>F9</td>
<td>Use of formalization</td>
<td>RQ3</td>
</tr>
<tr>
<td>F10</td>
<td>Offline vs. runtime use of formal methods</td>
<td>RQ3</td>
</tr>
<tr>
<td>F11</td>
<td>Tools used for modeling and verification</td>
<td>RQ1</td>
</tr>
<tr>
<td>F12</td>
<td>Types of software systems</td>
<td>RQ2</td>
</tr>
</tbody>
</table>

Table 4.2: Data collection form
4.4 Results

Studies were selected for final review during manual search. The list of selected studies is added in Appendix A.

Fig. 4.2 shows the number of studies on formal methods in self-adaptation over time. While the absolute numbers are remarkably low, there is a clear progression in the number of studies that use formal methods. 2005 is clearly a pivotal year, which is obviously connected with the creation of ICAC, SEAMS, and TAAS around that time.

We also looked at relative numbers. Between 2000 and 2005 on average 6% of the total number of studies focused on self-adaptation, while this number increased to 17% in the period 2006 - 2011. Within the studies that focus on self-adaptation, between 2000 and 2005, on average, 3% use formal methods, while this number increased to 8% between 2006 and 2011.

Fig. 4.3 shows the number of studies on formal methods in self-adaptation per venue. SEAMS represents 20.0% of the studies, ICAC 12.0%, and SASO 6.6%. 14.7% of the studies are published in JSS, 10.7% in TSE and 8.0% in TAAS. Note that none of studies on self-adaptive systems of ISSTA, ICSM and TOSEM use formal methods (from 4, 19, and 8 studies on self-adaptation respectively). The top software engineering conferences FSE, ICSE, and ASE represent 17.3% of the studies.

In summary, we notice a good coverage of research results from different conferences and journals as well as increasing attention for the use of formal methods in self-adaptive systems over time. Nevertheless, the absolute numbers give a strong indication that there is a dearth of approaches that employ formal methods.

4.4 Results

We now give an overview of the study results based on the research questions we defined for our study.

RQ0: Which are the trends in applying formal methods to self-adaptive systems?

Trends are derived from the year (F3) and venue (F4). From the 6353 studies in total, 1027 were selected after applying the first search string (i.e., potential studies related to self-adaptation). Applying the second search string resulted in 489 studies (i.e., potential studies related for formal methods). From these, 75
4.4 Results

studies were selected for final review during manual search. The list of selected studies is added in Appendix A.

Fig. 4.2 shows the number of studies on formal methods in self-adaptation over time. While the absolute numbers are remarkably low, there is a clear progression in the number of studies that use formal methods. 2005 is clearly a pivotal year, which is obviously connected with the creation of ICAC, SEAMS, and TAAS around that time.

We also looked at relative numbers. Between 2000 and 2005 on average 6% of the total number of studies focused on self-adaptation, while this number increased to 17% in the period 2006 - 2011. Within the studies that focus on self-adaptation, between 2000 and 2005, on average, 3% use formal methods, while this number increased to 8% between 2006 and 2011.

Fig. 4.3 shows the number of studies on formal methods in self-adaptation per venue. SEAMS represents 20.0% of the studies, ICAC 12.0%, and SASO 6.6%. 14.7% of the studies are published in JSS, 10.7% in TSE and 8.0% in TAAS. Note that none of studies on self-adaptive systems of ISSTA, ICSM and TOSEM use formal methods (from 4, 19, and 8 studies on self-adaptation respectively). The top software engineering conferences FSE, ICSE, and ASE represent 17.3% of the studies.

In summary, we notice a good coverage of research results from different conferences and journals as well as increasing attention for the use of formal methods in self-adaptive systems over time. Nevertheless, the absolute numbers give a strong indication that there is a dearth of approaches that employ formal methods...
for assuring that self-adaptive software systems satisfy user requirements and meet their expected quality attributes. [174] argues that adoption of self-adaptation by industry is still limited due to a lack of validation and verification methods. The results of our study confirm that research and development in assurance techniques is a necessity to unlock the potential of self-adaptation beyond academic settings.

**RQ1: Which methods have been used in self-adaptive systems?**

To answer this question, we used data extracted from modeling languages (F5), property specification/verification (F6), and tools used for modeling and verification (F11).

Fig. 4.4 shows the distribution of the modeling languages used for formalization.

The main observation is that regular algebra is by far the most popular modeling language (27.6% of the studies). The use of traditional formal modeling languages such as transition systems, automata, state machines, Markov models, graphs, and process algebras is about equally distributed (each representing 6 to 10%). We could not identify any trend in the use of the different modeling languages over time.

Fig. 4.5 shows the distribution of the property specification languages used for formalization.

The majority of the studies (42.7%) formulate the properties of interest in the modeling language they use for modeling. In particular, 87.5% of the studies that use algebra as modeling language use the same language for property specification. 36.0% of the studies use some logic as property specification language. Logic is combined with different kind of modeling languages.
4.4 Results

From the data derived from data item F11, we found that 40.0% of the studies use tools for formal modeling or verification. 30.0% of the studies that use tools employ it for model checking (actually all studies that do model checking use a tool). The remaining studies that use tools employ them for modeling systems, and only one study uses a tool for proving. The most used tools are PRISM (3 studies), SPIN, LTSA, CZT, and ACME studio (each 2 studies). However, no standard tools have been emerged for formal modeling and verification of self-adaptive systems.

![Modeling Language F5](image)

**Figure 4.4:** Modeling languages used for formalization

![Property specification/verification F6](image)

**Figure 4.5:** Property specification languages used for formalization
4 Systematic Literature Review on Formal Methods for Self-Adaptive Systems

In summary, we notice that the majority of researchers employ regular algebraic notations both for modeling and property specification of self-adaptive systems. The formal notations are mostly used to provide a level of rigor in the explanations and discussion. However, the use of formal methods for providing evidence of system properties, e.g., by using model checking or theorem proving, remains limited. We further notice that only a few researchers make their models and study results publicly available. This indicates that current research efforts on formal methods in self-adaptive systems are not well integrated. Finally, we observe that a number of authors reuse tools that are developed for offline analysis to perform runtime analysis; a typical example is [37]. While this is surely valid in the context of an academic case study, such an approach is often not appropriate from a practical point of view. This observation indicates that there is a need for light-weight pluggable tools that support formal verification at runtime.

**RQ2: For which adaptation concerns have formal methods been used?**

To answer this question, we drew on data extracted from concerns subject of formalization (F7), and application domains (F12).

Fig. 4.6 shows the data derived for concerns of formalization.

The distribution of the concerns of self-adaptation in this study confirms the distribution we found in a previous study [189] (the scope of that study was limited to SEAMS). Top concerns of self-adaptation for which formalization is used are efficiency/performance (32.0%), reliability (26.7%), guaranteeing functionality (22.7%), and flexibility (18.7%).

![Figure 4.6: Adaptation concerns](image)
Fig. 4.7 shows the distribution of types of software systems for which formal methods have been used.

![Type of software system distribution](image)

**Figure 4.7:** Types of software systems used for formalization

About half of the software systems (46.7%) for which formal methods have been used are embedded systems, followed by service-based systems (26.7%).

The use of formal methods for self-adaptation in embedded systems relates to requirements on resource constraints and real time behavior of these systems. Service-based systems have gained an increasing attention during the last five years. In particular dynamic composition of services is an active area of research in self-adaptive system, with particular attention for resolving behavioral mismatches between services and dynamic selection of services to guarantee service level agreements. Nevertheless, the data extracted for RQ2 shows that formal methods are hardly considered for guaranteeing important concerns of modern software systems such as security and scalability. Furthermore, data intensive domains, such as telecommunication, and scientific domains such as climate research and bioinformatics, have gained little attention. These concerns and domains offer areas for future research on formal methods and self-adaptation.

RQ3: How have formal methods been used to deal with concerns of self-adaptive systems?

To answer this question, we used data from verified properties (F8), type of verification (F9), and offline vs. runtime use of formal methods (F10).

Fig. 4.8 shows the distribution of the properties that have been verified in the studies.

Traditional properties, including safety (18.7%), liveness and reachability (each 10.7%), and deadlock (8.0%) make up half of the verified properties. However, the other half consists of a variety of different properties, including consistency, fitness, determinism, interference freedom, responsiveness, mismatch, and loss-tolerance. This set of properties seems to be specific to the area of self-adaptation. Further research is required to obtain a solid understanding of the nature of these properties.

Fig. 4.9 shows the type of formal approach that are used in the study.
4.4 Results

<table>
<thead>
<tr>
<th>NORMAL BEHAVIOR</th>
<th>ADAPTATION BEHAVIOR</th>
<th>INVALID BEHAVIOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Livelock-free</td>
<td>Reachability</td>
<td>Fairness, Reachability, Stability</td>
</tr>
<tr>
<td>[6]</td>
<td>[20]</td>
<td>[6]</td>
</tr>
<tr>
<td>[632]</td>
<td>[20]</td>
<td>[593]</td>
</tr>
<tr>
<td>[593]</td>
<td></td>
<td>[593]</td>
</tr>
<tr>
<td>[595]</td>
<td></td>
<td>[593]</td>
</tr>
<tr>
<td>[35]</td>
<td></td>
<td>[593]</td>
</tr>
<tr>
<td>[64]</td>
<td></td>
<td>[35]</td>
</tr>
<tr>
<td>[35]</td>
<td></td>
<td>[35]</td>
</tr>
<tr>
<td>[35]</td>
<td></td>
<td>[35]</td>
</tr>
<tr>
<td>[35]</td>
<td></td>
<td>[35]</td>
</tr>
</tbody>
</table>

**LEGEND**

- LivenessZone
- Initial state
- Final state
- State Transition
- Ref.
- Property

Figure 4.10: Zones in the state-space that represent different behaviors of a self-adaptive system with properties of interest.

Design of a self-adaptive system. Model checking makes up 16.0% of the studies, and proving 14.7%. It is remarkable to see that in total, only 23 studies employ formal methods to actually provide evidence for the self-adaptive concerns of interest.

For data item F10, we found that 66.7% of the studies use formal methods for offline activities, while 33.3% use formal methods at runtime. Almost all studies use formal methods in one particular phase in the life cycle. One exception that employs formal results from design to check the implementation is [198].

As model checking is one of the prominent approaches to provide evidence for system properties, we analyzed the studies that employ model checking techniques, including [10, 33, 37, 59, 67, 69, 94, 133, 198]. We also included the recent publication [39]. From this analyzes, we derived an interesting model that maps the different types of behaviors of self-adaptive systems to zones of the state space. Fig. 4.10 shows an overview of the model.

Figure 4.8: Verified properties of formalization

Figure 4.9: Formal approach used in the study

The majority of the studies use formal methods for reasoning purpose, i.e., 60.0%. In particular, most authors use a formal specification to reason about the
4.4 Results

The design of a self-adaptive system. Model checking makes up 16.0% of the studies, and proving 14.7%. It is remarkable to see that in total, only 23 studies employ formal methods to actually provide evidence for the self-adaptive concerns of interest.

For data item F10, we found that 66.7% of the studies use formal methods for offline activities, while 33.3% use formal methods at runtime. Almost all studies use formal methods in one particular phase in the life cycle. One exception that employs formal results from design to check the implementation is [198].

As model checking is one of the prominent approaches to provide evidence for system properties, we analyzed the studies that employ model checking techniques, including [10, 33, 37, 59, 67, 69, 94, 133, 198]. We also included the recent publication [39]. From this analyzes, we derived an interesting model that maps the different types of behaviors of self-adaptive systems to zones of the state space. Fig. 4.10 shows an overview of the model.
4.6 Future Research Directions

Particularly relevant for self-adaptive systems is that work products of formalization are exploited throughout the software life cycle. This enables the transfer of quality assurances of self-adaptive systems obtained during design to the implementation and the running system, enhancing the validity of the required qualities [189]. Some studies that directly transfer formalization results over different phases of the software life cycle are [45, 175, 183, 198].

Researchers and engineers use standard languages for formal modeling and property specification. However, a set of new properties emerge that are subject of reasoning and verification, such as consistency, fitness, determinism, interference freedom, responsiveness, mismatch, and loss-tolerance. These properties appear to be specific to self-adaptation and are crucial to provide guarantees about concerns of self-adaptive systems. Developing a solid understanding and underpinning of these properties is an important subject for future research. The zone-based model presented in this paper offers an interesting starting point to get a better understanding of the specific nature of model-checking of self-adaptive systems.

We conclude with a final remark concerning the underpinning of formal papers in the area of self-adaptive systems. In many studies, authors introduce custom modeling language constructs. Often, the mathematical underpinning and soundness of these languages is assumed for granted (but usually not provided). This aspect requires attention as mathematical underpinning is the foundation of any formal method.
Particularly relevant for self-adaptive system is that work products of formalization are exploited throughout the software life cycle. This enables the transfer of quality assurances of self-adaptive systems obtained during design to the implementation and the running system, enhancing the validity of the required qualities [189]. Some studies that directly transfer formalization results over different phases of the software life cycle are [45, 175, 183, 198].

Researchers and engineers use standard languages for formal modeling and property specification. However, a set of new properties emerge that are subject of reasoning and verification, such as consistency, fitness, determinism, interference freedom, responsiveness, mismatch, and loss-tolerance. These properties appear to be specific to self-adaptation and are crucial to provide guarantees about concerns of self-adaptive systems. Developing a solid understanding and underpinning of these properties is an important subject for future research. The zone-based model presented in this paper offers an interesting starting point to get a better understanding of the specific nature of model-checking of self-adaptive systems.

We conclude with a final remark concerning the underpinning of formal papers in the area of self-adaptive systems. In many studies, authors introduce custom modeling language constructs. Often, the mathematical underpinning and soundness of these languages is assumed for granted (but usually not provided). This aspects requires attention as mathematical underpinning is the foundation of any formal method.
### 4.7 Selected Studies

<table>
<thead>
<tr>
<th>ID</th>
<th>Year</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEAMS[38]</td>
<td>2009</td>
<td>Behavioural Self-Adaptation of Services in Ubiquitous Computing Environments</td>
<td>Camara et al.</td>
</tr>
<tr>
<td>SEAMS[45]</td>
<td>2010</td>
<td>Synthesizing adapters for conversational web-services from their WSDL interface</td>
<td>Cavallaro et al.</td>
</tr>
<tr>
<td>SEAMS[168]</td>
<td>2010</td>
<td>Autonomic Computing Control of Composed Web Services</td>
<td>Solomou et al.</td>
</tr>
<tr>
<td>SEAMS[124]</td>
<td>2011</td>
<td>Awareness Requirements for Adaptive Systems</td>
<td>Sozza</td>
</tr>
<tr>
<td>SEAMS[94]</td>
<td>2009</td>
<td>A Case Study in Goal-Driven Architectural Adaptation</td>
<td>Heaven et al.</td>
</tr>
<tr>
<td>SEAMS[44]</td>
<td>2008</td>
<td>An Approach to Adapt Service Requests to Actual Service Interfaces</td>
<td>Cavallaro &amp; Nitti</td>
</tr>
<tr>
<td>SASO[118]</td>
<td>2009</td>
<td>A Completely Evolvable Genotype-Phenotype Mapping for Evolutionary Robotics</td>
<td>König &amp; Schimeck</td>
</tr>
<tr>
<td>FSE[183]</td>
<td>2001</td>
<td>A Graph Based Architectural (Re)configuration Language</td>
<td>Wermelinger et al.</td>
</tr>
<tr>
<td>JSS[141]</td>
<td>2009</td>
<td>Autonomic QoS control in enterprise Grid environments using online simulation</td>
<td>Nou et al.</td>
</tr>
<tr>
<td>JSS[160]</td>
<td>2006</td>
<td>Developing adaptive systems with synchronized architectures</td>
<td>Seele &amp; Garlan</td>
</tr>
<tr>
<td>JSS[199]</td>
<td>2006</td>
<td>Using temporal logic to specify adaptive program semantics</td>
<td>Zhang &amp; Cheng</td>
</tr>
<tr>
<td>JSS[143]</td>
<td>2008</td>
<td>An architectural approach to the correct and automatic assembly of evolving component-based systems</td>
<td>Pellecione et al.</td>
</tr>
<tr>
<td>JSS[31]</td>
<td>2008</td>
<td>A self-stabilizing autonomic recoverer for eventual Byzantine software</td>
<td>Brukman et al.</td>
</tr>
</tbody>
</table>
### 4.7 Selected Studies

<table>
<thead>
<tr>
<th>ID</th>
<th>Year</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSS [131]</td>
<td>2005</td>
<td>An agent-based adaptive bandwidth allocation scheme for multimedia applications</td>
<td>Manvi &amp; Venkataram</td>
</tr>
<tr>
<td>JSS [46]</td>
<td>2007</td>
<td>A comprehensive approach to model and use context for adapting applications in pervasive environments</td>
<td>Chaari et al.</td>
</tr>
<tr>
<td>ICSE [61]</td>
<td>2009</td>
<td>Model Evolution by Run-Time Parameter Adaptation</td>
<td>Episani</td>
</tr>
<tr>
<td>TSE [41]</td>
<td>2008</td>
<td>Model-Based Adaptation of Behavioral Mismatching Components</td>
<td>Canal et al.</td>
</tr>
<tr>
<td>ASE [34]</td>
<td>2010</td>
<td>Integrating Model Verification and Self-Adaptation</td>
<td>Borges et al.</td>
</tr>
<tr>
<td>ASE [175]</td>
<td>2005</td>
<td>Model-Based Self-Monitoring Embedded Programs With Temporal Logic Specifications</td>
<td>Tan</td>
</tr>
<tr>
<td>Adaptive [67]</td>
<td>2011</td>
<td>A Formal Orchestration Model for Dynamically Adaptable Services with COWS</td>
<td>Fex</td>
</tr>
<tr>
<td>TAAS [85]</td>
<td>2009</td>
<td>Self-Stabilizing Robot Formations over Reliable Networks</td>
<td>Gilbert et al.</td>
</tr>
<tr>
<td>TAAS [195]</td>
<td>2008</td>
<td>An Adaptive Automatically Tuning Intrusion Detection System</td>
<td>Yu &amp; Tsai</td>
</tr>
<tr>
<td>TAAS [172]</td>
<td>2011</td>
<td>Cooperative Evolution of Services in Ubiquitous Computing Environments</td>
<td>Tacconi</td>
</tr>
</tbody>
</table>
Chapter 5

Guaranteeing Robustness using Formally Verified MAPE-K Loops

In this Chapter, we study the use of formal methods for the specification and verification of a self-adaptive application. Concretely, we study a collaborative mobile learning application in which we apply self-adaptation to deal with robustness requirements. The focus is on outdoor learning activities, where groups of three or four students have to measure and calculate properties of geometrical figures, such as circles, rectangles and triangles. To support the tasks, each student uses a GPS-enabled mobile device that allows the calculation of distances. Depending on the activity, a group may require two or more GPS services, e.g. three GPS are required for triangulation. The activities take place in outdoor settings and due to the dynamic environment conditions, the GPS accuracy is not always guaranteed. Consequently, this undermines the results of the learning activity. To provide robustness to the system, we added two MAPE loops that deal with two concerns of robustness: the first loop deals with managing the availability of the GPS service based on the actual GPS service quality; the second loop deals with managing the required number of GPS services of the current task for the group.

The objectives of this study are:

- to formally specify the mobile learning application (i.e., the managed system in its environment);
- to formally specify self-adaptation behaviors that follow the MAPE-K loop approach;
- to formally specify the robustness requirements with respect to degrading GPS accuracy;
- to verify and assure that the properties hold for the model;
- to map the design models to an implementation.

This case contributes to answer the research questions “RQ2: How to model the behaviors of MAPE-K components to deal with robustness and openness requirements?” and “RQ3: What properties must be verified to provide evidence that the required self-adaptation behaviors are assured?”

To guarantee the required robustness requirements, we used TA to specify the behaviors of the MAPE loops, and expressed the robustness requirements as formal properties in TCTL.

<table>
<thead>
<tr>
<th>ID</th>
<th>Year</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICAC</td>
<td>2005</td>
<td>Dynamic Black-Box Performance Model Estimation for Self-Tuning Regulators</td>
<td>Karlsson &amp; Covell</td>
</tr>
<tr>
<td>ICAC</td>
<td>2007</td>
<td>Autonomic Reactive Systems via Online Learning</td>
<td>Seshia</td>
</tr>
<tr>
<td>ICAC</td>
<td>2008</td>
<td>Multi-Level Intrusion Detection System (ML-IDS)</td>
<td>Al-Nashif et al.</td>
</tr>
<tr>
<td>ICAC</td>
<td>2008</td>
<td>Generating Adaptation Policies for Multi-Tier Applications in Consolidated Server Environments</td>
<td>Jung et al.</td>
</tr>
</tbody>
</table>
Chapter 5

Guaranteeing Robustness using Formally Verified MAPE-K Loops

In this Chapter, we study the use of formal methods for the specification and verification of a self-adaptive application. Concretely, we study a collaborative mobile learning application in which we apply self-adaptation to deal with robustness requirements. The focus is on outdoor learning activities, where groups of three or four students have to measure and calculate properties of geometrical figures, such as circles, rectangles, and triangles. To support the tasks, each student uses a GPS-enabled mobile device that allows the calculation of distances. Depending on the activity, a group may require two or more GPS services, e.g., three GPS are required for triangulation. The activities take place in outdoor settings and due to the dynamic environment conditions, the GPS accuracy is not always guaranteed. Consequently, this undermines the results of the learning activity. To provide robustness to the system, we added two MAPE loops that deal with two concerns of robustness: the first loop deals with managing the availability of the GPS service based on the actual GPS service quality; the second loop deals with managing the required number of GPS services of the current task for the group.

The objectives of this study are:

- to formally specify the mobile learning application (i.e., the managed system in its environment);
- to formally specify self-adaptation behaviors that follow the MAPE-K loop approach;
- to formally specify the robustness requirements with respect to degrading GPS accuracy;
- to verify and assure that the properties hold for the model;
- to map the design models to an implementation.

This case contributes to answer the research questions "RQ2: How to model the behaviors of MAPE-K components to deal with robustness and openness requirements?" and "RQ3: What properties must be verified to provide evidence that the required self-adaptation behaviors are assured?"

To guarantee the required robustness requirements, we used TA to specify the behaviors of the MAPE loops, and expressed the robustness requirements as formal properties in TCTL.
5 Guaranteeing Robustness using Formally Verified MAPE-K Loops

We used explicit modules for each of the adaptation functions of MAPE loops in the design of a self-adaptive application and mapped these modules one-to-one to a Java implementation. Using explicit modules for each of the adaptation functions of MAPE loops makes it easier to model the self-adaptive behavior, as the designer can focus on one activity at a time; model the interaction between the managed and managing system, as the interaction points are well-defined; reason about the behavior within and between MAPE loops, as a result of the clear separation of concerns; reason about the interactions between the managed and managing system; specify required properties, and these properties can be specified at a more fine-grained level; identify problems in the design; and map design to implementation. However, there are also some tradeoffs. Some MAPE activities have a straightforward behavior, which may raise questions about the usefulness of a separate specification; the size of the design increases; and the cost for verification grows.

This Chapter provides a detailed documentation of the formal design of one of the applications that underly the MAPE-K Formal Templates that we present in the next Chapter.
Guaranteeing Robustness in a Mobile Learning Application using Formally Verified MAPE Loops

Abstract

Mobile learning applications support traditional indoor lectures with outdoor activities using mobile devices. An example scenario is a team of students that use triangulation techniques to learn properties of geometrical figures. In previous work, we developed an agent-based mobile learning application in which students use GPS-enabled phones to calculate distances between them. From practical experience, we learned that the required level of GPS accuracy is not always guaranteed, which undermines the use of the application. In this paper, we explain how we have extended the existing application with a self-adaptation layer, making the system robust to degrading GPS accuracy. The self-adaptive layer is conceived as a set of interacting MAPE loops (Monitor-Analysis-Plan-Execute), distributed over the phones. To guarantee the robustness requirements, we formally specify the self-adaptive behaviors using timed automata, and the required properties using timed computation tree logic. We use the Uppaal tool to model the self-adaptive system and verify the robustness requirements. Finally, we discuss how the formal design supported the implementation of the self-adaptive layer on top of the existing application.

5.1 Introduction

The upcoming generation of software systems will operate in environments that are open in the sense that they are only partially known at design-time. An increasing number of these systems will consist of loosely connected subsystems that run on pervasive devices and networks providing services that are not fully predictable. One emerging class of such systems are mobile learning applications. Mobile learning applications support traditional indoor lectures with outdoor activities using mobile devices. In previous work, we developed a mobile learning application to support learning activities that require student participation in groups, information sharing, and peer collaboration [84]. The pedagogical activities focus on strengthening mathematical topics taught in the classroom with additional outdoor activities using mobile devices. The focus is on groups of three or four students (with an age around 12) that have to measure and calculate properties of geometrical figures, such as circles, rectangles, and triangles. To support the tasks, each student uses a GPS-enabled mobile device. The mobile learning application provides learning services to the students that enable them to perform the tasks, such as services to represent tasks, measure the distance between selected mobile devices based on their current GPS locations, etc.

The mobile learning application is designed as a distributed agent-based system. On each mobile device, a device agent is deployed that provides the learning
services to the student. Device agents of a group of students that work on the same tasks form an organization, which we call a Mobile Virtual Device (MVD [84]). An MVD has a master-slave structure. The master receives new tasks from an activity agent that resides on a server deployed at the school, and reports the results back when a task is finished. Evidently, to avoid students making misleading conclusions, it is important that the system provides suitable measurements. As the quality of measurements directly depends on the accuracy of the GPS, the system should ensure that the GPS services provide a minimum level of quality w.r.t. the accuracy of measurements. Due to changing environmental conditions, the required level of GPS accuracy is not always guaranteed. Unfortunately, the mobile learning application was not designed to deal with insufficient GPS quality of service, which undermines the use of the application when conditions get worse.

Self-adaptation (SA) is a well recognized approach for dealing with particular runtime qualities [113, 121, 126]. SA is used for adding so called self-* properties (self-healing, self-protection, self-optimization) [113] to systems, addressing changing operating conditions of the system or its environment. SA is based on the design principle of separation of concerns. In particular, SA aims to separate the logic that deals with quality concerns of interest from the domain functionality provided by the underlying managed system. One prominent approach to realize SA is by means of a MAPE feedback loop (Monitor-Analyze-Plan-Execute) [113]. Whereas multiple studies have applied a MAPE design to realize self-adaptation, most of these studies consider MAPE as a conceptual framework that guides the engineering process [187]. Few studies have rigorously modeled and analyzed the behavior of MAPE designs, in particular for systems with multiple interacting feedback loops [190]. However, such rigor is required if we want to obtain guarantees about the required self-adaptive behavior.

In this paper, we show how we have extended the existing mobile learning application with a self-adaptation layer, making the system robust to degrading GPS accuracy. The self-adaptation layer is conceived as a set of interacting MAPE loops, distributed over the mobile devices. We have designed distinct components for the different functions of the MAPE loops, which offer benefits in terms of modeling, reasoning and mapping the design to implementation. To guarantee the robustness requirements, we formally specify the self-adaptive behaviors and verify the robustness requirements. We used the formally verified design to support the implementation of the self-adaptive layer on top of the existing application.

The rest of this paper is structured as follows. In Section 5.2 we provide background on MAPE. Section 5.3 introduces the mobile learning application, describes the problem, and outlines the architecture of the self-adaptive solution. In Section 5.4, we describe in detail the behavioral models of the self-adaptive system. Section 5.5 describes the required properties and discusses verification results. Section 5.6 briefly explains the mapping of behavioral models to implementation. We draw conclusions and outline plans for future work in Section 5.7.
5.2 Background and Related Work

MAPE was introduced as the conceptual core of an autonomic manager, which is central to IBM’s framework for Autonomic Computing [113]. The MAPE components realize the primary functions of a feedback loop. The Monitor component gathers relevant information from the underlying managed system and the environment. The Analyze component assesses the collected data to determine the system’s need to satisfy the adaptation objectives. The Plan component constructs the actions necessary to achieve the system’s objectives. Finally, Execute component carries out changes on the managed system. The additional Knowledge component maintains representations of the managed system and environment, adaptation objectives, and other relevant state that is shared by the MAPE components. MAPE is therefore also referred as MAPE-K.

Rainbow [70] offers a reusable architectural framework for building self-adaptive systems. The architectural layer that deals with self-adaptation, resembles similarities with a MAPE loop. Rainbow supports monitoring and adaptation of software systems that are distributed in a network. However, the control of adaptation is centralized. Another interesting example of a centralized feedback loop is described in [37]. The authors propose an approach to achieve QoS for service-based systems through an external MAPE loop. Formally specified requirements are automatically analyzed to identify and enforce optimal system configurations. The approach uses Markov models and probabilistic computation tree logic, and focuses on improving response time and dealing with failures.

A number of authors have studied interactions between feedback loops, which are more or less explicitly modeled as MAPE loops. [72] expresses structural constraints over an architectural specification that are used by component managers to automatically configure the system. [170] introduces a gossip protocol to make this approach scalable. [95] makes control loops explicit and present a UML profile for control loops that extends UML modeling concepts. [180] extends MAPE with support for inter-loop and intra-loop coordination. [9] introduces the concept of adaptive goal in service-based systems. Adaptive goals are responsible for adapting the goal model at runtime when needed. [178] presents a reference model for adaptive software that supports separation of concerns among feedback loops required to address control objectives over time. Finally, [190] describes patterns of interacting MAPE loops derived from implemented self-adaptive systems.

The work presented in this paper contributes to the presented background with a rigorous specification and verification of the behavior of the distinct components of MAPE loops and their interactions, for a concrete application.

5.3 Towards a Robust M-Learning Application

In this section we give a brief summary of the mobile learning application we developed, we pinpoint the robustness problem we faced with insufficient GPS accuracy, and we outline how we tackled this problem by extending the design of the legacy system with a self-adaptation layer.

59
5 Guaranteeing Robustness using Formally Verified MAPE-K Loops

5.3.1 Mobile-Learning Application

The mobile learning application supports outdoor learning activities, where students use GPS-enabled mobile devices. A learning activity takes place in the context of a lecture (of 1 or 1.5 hour) and is composed of a set of tasks (typically 4 to 8 tasks). An example of a learning activity is to measure and calculate properties of triangles, and one concrete task is to use triangulation techniques to find locations on the field given the three side measurements of a triangle, and having two of the triangle locations already marked on the field. Fig. 5.1 shows a use case scenario, where three groups of students (represented by MVDs) perform tasks of a learning activity.

The application is conceived as a distributed agent-based system. A Device Agent deployed on each mobile phone provides the learning services to the student (gathering locations, calculating distances, etc.). The device agents of a group that work on the same tasks form an MVD. Within an MVD, one of the agents is elected as master, while the others serve as slaves. The MVD Manager is responsible for the management of the MVD. E.g., a new master is elected when the master phone runs out of energy. The master communicates via 3G with the Server using the Communication Infrastructure. Management of the tasks at the server is the responsibility of the Activity Agent. The master of each MVD receives new tasks from the activity agent at the server and reports the results back when a task is finished.

5.3.2 Problem Description

Due to changing environmental conditions, the GPS sensitivity can vary over time, which affects the accuracy of the measurements and may undermine the use of the application when conditions get worse. We were aware of the fact that GPS is...
not always accurate. However, it proved to be a bigger problem than our initial assumptions, up to the point where inaccurate measurements mislead students conclusions.

There are two main variables that determine the required quality of the GPS measurements during learning activities: the current GPS accuracy and the required level of accuracy for the task at hand. Fig. 5.2 illustrates how the GPS accuracy error typically evolves over time for a mobile device.

Depending on the given task, the allowed level of GPS accuracy errors can be different. As an example, an 8 meters accuracy error in the GPS acquisition has a higher impact when used in a 20 meter distance calculation than when used in a 60 meter distance. Therefore, there is a need for dynamically updating the required level of GPS accuracy for the measurements of each task. Fig. 5.2 shows two horizontal lines representing different accuracy requirements for Task-1 and Task-2, where the first task requires a lower level of accuracy (error lower than 11 meters) than the second (error lower than 7 meters).

The combination of the GPS quality and the requirements for a given task determine whether a mobile device is suitable to perform the distance measurements for the task. We marked two time instances in Fig. 5.2 representing points where the mobile device is not providing the required quality of measurements. At time stamp 20, while running Task-1, a 14 meter accuracy error is reported, failing the 11 meters accuracy that is required for the task. Notice that the available GPS quality may fulfill the requirements for one task, but fail for another task (see e.g., time stamp 40). From our experience, we learned that the required GPS quality of the mobile devices varies substantially. However, we noticed that during a learning activity, at most 20% of GPS modules failed to provide the required quality level, and this for a duration lower than 20% of the time of the learning activity. As a rule, we can state that (worst case in practice) less than 10% of the mobile devices are in an undesired state at the same time.

![Figure 5.2: Exemplification of GPS accuracy](image)

In order to deal with failing devices, we need to take into account the requirements with respect to the required number of devices for the tasks. As explained before, a learning activity consists of a set of tasks. However, different tasks may require a different number of devices. For example, two mobile devices are suffi-
cient to measure the diameter of a circle, while triangulation requires three mobile devices. Therefore, we need to take into account the number of required GPS devices per group (MVD) when handling failing devices. Typically, between 10 and 20% redundant phones are available.

Summarizing, the GPS accuracy of phones can degrade making them invalid for distance measurements. As a result, the number of mobile devices in a group may be insufficient to complete tasks successfully. Currently, the application does not support students with identifying the lack of sufficient quality of the GPS module and solving the problem by dynamically integrating available phones. To deal with this problem, we aim to enhance the current system with self-adaptation mechanisms that guarantee the robustness of the system with respect to decreasing GPS quality of mobile devices.

### 5.3.3 Adding a Self-Adaptive Layer

To realize the required robustness, we added a self-adaptive layer on top of the exiting system. We realized the self-adaptive layer using MAPE [113] loops, as shown in Fig. 5.3. Concretely, to provide robustness to the system we added two MAPE loops that deal with two concerns of robustness: the first loop deals with managing the availability of the GPS service based on the actual GPS service quality (left-side MAPE in Fig. 5.3); the second loop deals with managing the required number of GPS services of the current task for the MVD (right-side MAPE in Fig. 5.3). Probes and effectors enable the MAPE loops gathering the relevant information of the underlying managed system and applying the planned adaptations actions.

The first MAPE loop (GPS Service Concern) is local to each mobile device. This loop monitors the quality of the GPS module, compares it with the required quality, and based on that, activates or deactivates the GPS service. When a GPS service is deactivated, it can trigger the second MAPE loop to start a self-healing process, that is, find a new device and add this to the MVD. We say can trigger, because there may be redundant phones in the MVD, so that no replacement is required.

The second MAPE loop (MVD Concern) is distributed over the devices of the MVD. This MAPE loop uses a master-slave pattern [190]. Fig. 5.4 shows the distribution of MAPE components for three phones. The master-slave pattern enables coordination of self-adaptation among nodes in a distributed system. Devices have similar roles (master and slave) both with respect to adaptation in the second MAPE loop and the functionality provided by mobile learning application (i.e., the managed system). All devices of an MVD (master and slaves) monitor the mobile learning application and execute adaptation actions on it, but only the master is responsible for analysis and planning adaptations.

If the master detects that the number of GPS services in the MVD is not sufficient for the current task, it looks for an additional service. If there is a free GPS service available, the device that provides that service is dynamically added to the MVD, if not, the master periodically re-checks.
The master role can be performed by any of the phones in an MVD, making the organization robust in case of a master failure. In this paper, we abstract from the mechanisms to elect a new master. We refer the interested reader to [104] for self-healing mechanisms to deal with failures of a master in a master-slave organization deployed in a distributed application.

5.4 Behavioral Design

The structural models of the self-adaptive layer described in the previous section show the primary building blocks of the MAPE loops and there interactions. These models are useful for explaining the adaptation mechanisms at a high-level of abstraction, and defining course-grained modules to implement the self-adaptive layer. However, to guarantee the robustness requirements, we need a rigorous specification of the self-adaptive behaviors, together with the properties that express the robustness requirements. This specification allows then to verify whether the self-adaptive behaviors comply to the properties. To that end, we formally specify the behavioral design of the self-adaptive layer.

In this research, we use Uppaal [15], a model checking tool that supports modeling of behaviors (also called processes) using timed automata and verification.
of the robustness properties expressed in timed computation tree logic (TCTL). Timed automata and TCTL provide an accessible formalism. Concretely, a timed automaton is a finite-state machine extended with clock variables, which are used to synchronize behaviors. The automata represent states in which a behavior can be found and define actions to be performed on the transition between states. Behaviors can communicate through channels by signal passing, where the sender process \( x! \) synchronizes with the receiver process \( x? \). The automata can be complemented with expressions specified in a C-like language to define data structures (\textit{struct} concept) and functions. Expressions in TCTL describe state and path formulae allowing the verification of properties of interest, such as \textit{reachability} (a system should/can/cannot etc. reach a particular state or states), \textit{liveness} (something eventually will hold), etc.

In the rest of this section, we describe the behaviors of the self-adaptive layer in three parts. We start by presenting the processes of the external world. Then, we present the behaviors of first MAPE loop (GPS Service Concern) and conclude with the behaviors of the second MAPE loop (MVD Concern). For the managed system, we only model the essential aspects that are required with respect to self-adaptation.

### 5.4.1 External World Processes

The need for self-adaptation is triggered by changes in the external world. To that end, it is necessary to formally specify an abstraction of the external world. In our case, the external world consists of three behaviors: the Activity Agent, the Context, and the GPS Module. Fig. 5.5 shows the behaviors in relation to the MAPE loop for GPS service self-adaptation (which we discuss below).
5.4 Behavioral Design

First MAPE loop (GPS Service Concern):

- **SA (Self-Adaptation)**:
  - **GPS SA Monitor**
  - **GPS SA Analyze**
  - **GPS SA Plan**
  - **GPS SA Execute**

Second MAPE loop (MVD Concern):

- **MS (Managed System)**
  - **GPS Requirement Probe**
  - **GPS Quality Probe**
  - **GPS Service Effector**
  - **GPS Groups Effector**

**CI (Communication Infrastructure)**:

- **COM**: Mobile Learning Application
- **GPS Module**
- **GPS Groups**

**ENV (Environment)**:

- **Context**
- **Communication Infrastructure**

**Figure 5.5**: Process mapping for GPS service self-adaptation

---

5.5 Guaranteeing Robustness using Formally Verified MAPE-K Loops

### Image 3.1: Master/Slave MVD self-healing pattern (for 3 phones)

Timed automata and TCTL (Timed Computation Tree Logic) provide an accessible formalism. Concretely, a timed automaton is a finite-state machine extended with clock variables, which are used to synchronize behaviors. The automata represent states in which a behavior can be found and define actions to be performed on the transition between states. Behaviors can communicate through channels by signal passing, where the sender process $x!$ synchronizes with the receiver process $x?$. The automata can be implemented with expressions specified in a C-like language to define data structures (struct concept) and functions. Expressions in TCTL describe state and path formulae allowing the verification of properties of interest, such as reachability (a system should/can/cannot etc. reach a particular state or states), liveness (something eventually will hold), etc.

In the rest of this section, we describe the behaviors of the self-adaptive layer in three parts. We start by presenting the processes of the external world. Then, we present the behaviors of the first MAPE loop (GPS Service Concern) and conclude with the behaviors of the second MAPE loop (MVD Concern). For the managed system, we only model the essential aspects that are required with respect to self-adaptation.

5.4.1 External World Processes

The need for self-adaptation is triggered by changes in the external world. To that end, it is necessary to formally specify an abstraction of the external world. In our case, the external world consists of three behaviors: the Activity Agent, the Context, and the GPS Module. Fig. 5.5 shows the behaviors in relation to the MAPE loop for GPS service self-adaptation (which we discuss below).
5 Guaranteeing Robustness using Formally Verified MAPE-K Loops

An **Activity Agent**, located at the activity server, is in charge of setting the requirements for the GPS accuracy to perform the tasks, and the number of mobile devices that are required per group. Fig. 5.6 shows the automaton of the Activity Agent\(^1\). A first step initializes the distributed application, defining an initial deployment of phones to MVDs. Next, the Activity Agent is in charge to control the activity flow. On a periodic basis\(^2\) (**Time**-**Activity**), the activity agent sends new tasks in the activity with new requirements (**SubmitTask** state), until the tasks in the activity (**TotalLoops**) are completed (**Final** state). Task requirements define the desired minimal accuracy necessary for the GPS modules and the number of GPS modules in each MVD (represented by **MASactivity.min_accuracy** and **MASactivity.number_GPS**) (see Fig. 5.5).

**Figure 5.6:** Activity Agent

The environment influences the GPS module quality, potentially bringing a GPS service to an undesired state. The environment is abstractly modeled by the **Context** automaton (Fig. 5.7), which describes states of the environment with respect to GPS interference: **Clean** or **Noisy**, and transitions at defined time instances (provided by the **getNextNoise** and **getNextRecover** functions). **Pid** refers to the phone ID. One context automaton is instantiated for each mobile device, allowing us to model the influence of the environment on each GPS module.

**Figure 5.7:** Context

The **GPS Module** behavior, which is part of the Communication Infrastructure layer, is modeled by the automaton shown in Fig. 5.8. This behavior gets quality signals from the context (via **qualityGPSModuleUp** and **qualityGPSModuleDown**) that are used to update the representation in the system, represented by **MASPhoneStruct.GPS_Quality** (see Fig. 5.5).

\(^1\)Transitions between states fire based on conditions and/or received signals (we place these above transition arrows) and can perform actions or send signals to other processes (we place these below transition arrows).

\(^2\)The model abstracts the Activity Agent behavior by sending new requirements on a period basis. In practice, there is an activity flow between server and device agents based on the assignment and completion of tasks.
5.4 Behavioral Design

As mentioned above, we abstracted the managed system to its essentials required to deal with self-adaptation. Concretely, the managed system is represented using structures in Uppaal. Snippet 5.1 illustrates how different elements of the mobile learning application are represented: the current information \( \text{w.r.t.} \) the GPS quality \( (\text{GPS} \_\text{Quality}) \) and the service \( (\text{GPS} \_\text{Service}) \) state, the participation in organizations \( (\text{status}) \), and temporary information used to determine changes on the GPS quality \( (\text{change} \_\text{NotTreated}, \text{prev} \_\text{quality}) \).

### Snippet 5.1: MASPhoneStruct

```c
struct {
    int GPS_Quality; // Undesired or OK
    int GPS_Service; // Deactivated or Active
    int status; // inMVD or Free
    bool change_NotTreated; // internal (MAPE loop sync)
    int prev_quality; // internal (GPS quality)
}
```

Once we have a formal model of the external world and an abstraction of the managed system, we can model the self-adaptation processes.

#### 5.4.2 GPS Service Self-Adaptation Processes

The GPS service self-adaptation processes model the first MAPE loop that deals with activating and deactivating GPS services based on the quality of the GPS signals. Fig. 5.5 shows the mapping of the behaviors of the MAPE loop to the components of the MAPE loop, shown in Fig. 5.3. There are two variables that can affect the suitability of a GPS module: the current task requirements and the GPS quality. Therefore, we model two probe processes that gather system information. Fig. 5.9 shows the automaton that represents the behavior of the GPS Quality Probe. The automaton contains a state in which the GPS quality is being sensed \( (\text{Probing}) \), and two additional states where the quality is Increasing and Decreasing. In case the GPS quality is modified, the GPS Service Monitor is notified by sending a signal \( (\text{SAqualityGPSIncreased} \text{ and } \text{SAqualityGPSDecreased}) \).

Similarly, the GPS Requirement Probe captures changes in the activity requirements \( (\text{MASactivity.min\_accuracy}) \) to notify the monitor component (automaton not shown).

The GPS SA Monitor process (Fig. 5.10) is in charge of monitoring the underlying managed system and updating the knowledge repository \( (\text{SAPhoneStruct} \text{ in } \text{Fig. 5.5}) \), supporting analysis and planning of self-adaptive actions. The automaton monitors two separated variables. On the left hand side, changes on the GPS quality change the GPS module to an undesired state. The environment is abstractly modeled by the automaton (Fig. 5.7), which describes states of the environment with respect to an undesired state. The environment is abstractly modeled by the automaton (Fig. 5.7), which describes states of the environment with respect to an undesired state. The environment is abstractly modeled by the automaton (Fig. 5.7), which describes states of the environment with respect to an undesired state. The environment is abstractly modeled by the automaton (Fig. 5.7), which describes states of the environment with respect to an undesired state.

\(^3\) The \( C \) represent committed states where a behavior cannot delay.
5 Guaranteeing Robustness using Formally Verified MAPE-K Loops

The **GPS SA Analyze** process (Fig. 5.11) waits in the **Waiting** state for a trigger from the Monitor process to make a transition to the **Analyzing** state. One of four possible states can be reached (**KeepGood**, **KeepBad**, **ChangedGood**, **ChangedBad**), depending on the current GPS quality and the requirements to accomplish the current task. In case changes are identified (**ChangedGood**, **ChangedBad**), the Plan process is notified via a **SA_GPS_degraded/recovered** signal. Snippet 5.2 illustrates how the analyze functions to determine transitions to potential undesired states are specified in Uppaal.

The **GPS SA Monitor** process (Fig. 5.10) waits in the **Waiting** state for a trigger from the Monitor process to make a transition to the **Analyzing** state. One of four possible states can be reached (**KeepGood**, **KeepBad**, **ChangedGood**, **ChangedBad**), depending on the current GPS quality and the requirements to accomplish the current task. In case changes are identified (**ChangedGood**, **ChangedBad**), the Plan process is notified via a **SA_GPS_degraded/recovered** signal. Snippet 5.2 illustrates how the analyze functions to determine transitions to potential undesired states are specified in Uppaal.

The **GPS SA Plan** process (Fig. 5.12) is responsible for planning adaptation actions with respect to the GPS service. That is, deciding whether to turn on/off the GPS service provided by the phone (**ChangeToGood**, **ChangeToBad**).

**Snippet 5.2: changesToGPSBad() function**

```c++
bool changesToGPSBad ( phone_id Pid )
{
    if ( SAphoneStruct[Pid].GPS_Quality <
```
5.4 Behavioral Design

The GPS SA Execute process (Fig. 5.13) is in charge to apply the planned actions to the managed system. From a waiting state, it is triggered by the GPS SA Plan to make a transition and modify the GPS service via one of the states SetGPSServiceUp or SetGPSServiceDown.

To support the Execute process, two effectors are provided that perform the actual adaptations to the managed system. The GPS Service Effector process (Fig. 5.14) is in charge of activating/deactivating the GPS service on the managed system. This is represented by the MASPhoneStruct.GPS_Service variable (see Fig. 5.5). Additionally, a GPS Group Effector process is designed to remove a phone from an MVD in case the GPS Service is deactivated (automaton not shown).

Figure 5.12: GPS SA Plan

Figure 5.13: GPS SA Execute

Figure 5.14: GPS Service Effector
5.4 Behavioral Design

5.4.3 MVD Self-Healing Processes

The MVD self-healing processes model the second MAPE loop that deals with recovery of undesired MVD states. Fig. 5.15 shows the mapping of the behaviors of the MAPE loop to the components of the MAPE loop, shown in Fig. 5.3

The MVD Requirement Probe (Fig. 5.16) shows a behavior that gathers information about the group requirements for the current task (\texttt{MASactivity.number\_GPS}). Changes that are detected are communicated to the corresponding Monitor process.

Mid refers to the MVD ID.

\texttt{SAMVDReqChanged[Mid]! didRequirementsChange(Mid) Changed updatePreviousRequirement(Mid)Probing}

Figure 5.16: MVD Requirement Probe

The MVD Requirement Monitor process (Fig. 5.17) is in charge of monitoring changes in the task requirements and, when signaled by the Requirement Probe, it updates the knowledge repository for the self-healing process (modeled as the \texttt{SAMVDStruct}, see Fig. 5.15). As shown in the Fig. 5.15, the probe and monitor processes that deal with the MVD requirements are only instantiated at the master device.

\texttt{Monitoring SAMVDReqChanged[Mid]? updateMVDReq()}

Figure 5.17: MVD Requirement Monitor

If a mobile device deactivates the GPS service resulting from the GPS service self-adaptation, a MVD Membership Probe communicates the change to the MVD Membership Monitor, represented by the process in the Fig. 5.18.

Unlike the MVD Requirement Probe and MVD Requirement Monitor processes, the MVD Membership Probe and the MVD Membership Monitor processes are instantiated at all mobile devices to gather the distributed information w.r.t. the MVD composition. The MVD Membership Monitor identifies whether an MVD is affected or not by a GPS service that is turned off (\texttt{myMVD != NOGROUP}).

\texttt{removing[Pid] myMVD = NOGROUP myMVD = determineMyMVD(Pid) removing_Phone(myMVD)}

Figure 5.18: MVD Membership Monitor

---

5 Guaranteeing Robustness using Formally Verified MAPE-K Loops

---

**Figure 5.15:** Process mapping for MVD self-healing
5.4.3 MVD Self-Healing Processes

The MVD self-healing processes model the second MAPE loop that deals with recovery of undesired MVD states. Fig. 5.15 shows the mapping of the behaviors of the MAPE loop to the components of the MAPE loop, shown in Fig. 5.3.

The MVD Requirement Probe process (Fig. 5.16) shows a behavior that gathers information about the group requirements for the current task (MASactivity.number_GPS, see Fig. 5.15). Changes that are detected are communicated to the corresponding Monitor process. Mid refers to the MVD ID.

![Figure 5.16: MVD Requirement Probe](image)

The MVD Requirement Monitor process (Fig. 5.17) is in charge of monitoring changes in the task requirements and, when signaled by the Requirement Probe, it updates the knowledge repository for the self-healing process (modeled as the SAMVDSStruct, see Fig. 5.15). As shown in the Fig. 5.15, the probe and monitor processes that deal with the MVD requirements are only instantiated at the master device.

![Figure 5.17: MVD Requirement Monitor](image)

If a mobile device deactivates the GPS service resulting from the GPS service self-adaptation, a MVD Membership Probe communicates the change to the MVD Membership Monitor, represented by the process in the Fig. 5.18. Unlike the MVD Requirement Probe and MVD Requirement Monitor processes, the MVD Membership Probe and the MVD Membership Monitor processes are instantiated at all mobile devices to gather the distributed information w.r.t. the MVD composition. The MVD Membership Monitor identifies whether an MVD is affected or not by a GPS service that is turned off (myMVD != NOGROUP).

![Figure 5.18: MVD Membership Monitor](image)
5 Guaranteeing Robustness using Formally Verified MAPE-K Loops

The **MVD SH Analyze** process (Fig. 5.19) is triggered by the **MVD SH Monitor**\(^4\). The Analysis process identifies three possible scenarios, that is, the MVD is **Complete** (the number of GPS services covers the requirements), **Incomplete** (GPS services are missing) or **Redundant** (there is redundancy of GPS devices). The Analyze process communicates with the Plan process to start recovering the MVD that is missing GPS services (via **SH_MVD_Incomplete**) or to stop a possible search otherwise (via **SH_MVD_Redundant/Complete**).

When triggered by the Analyze process, the **MVD SH Plan** process (Fig. 5.20) initiates a search to locate a free phone (**found_Phone**) that offers a GPS service (**LookForFreeGPS**). In case the resource cannot be found, the process stays in the **NoFreeGPS** state and repeats the search until it finds a service and its goal is achieved (**AllFine**). The phone that provides the service (**found_Phone**) is notified to get integrated in the MVD. Only the master phone executes the Plan process.

The **MVD SH Execute** process (Fig. 5.21) is in charge of applying the planned decisions for MVD self-healing. One Execute process is instantiated in each phone, in order to allow changing the phone state and integrating it into the corresponding MVD. The integration is performed through a **MVD Effector**, and results in changing the **MASPhoneStruct.status** (see Fig. 5.15).

### 5.5 Verification of Self-Adaptation

Once we have modeled processes, we can formulate the self-adaptation requirements as logical expressions over the models. Uppaal uses a subset of TCTL

\(^4\)The Analyze process is logically triggered by Monitor processes; actually the processes interact indirectly via the knowledge repository of the SA layer.
5.5 Verification of Self-Adaptation

(timed computation tree logic) to specify state and path formulae that can be verified. We discuss four groups of properties: functional correctness, GPS service adaptation, MVD self-healing, and MAPE loop interference.

5.5.1 Functional Correctness

To verify functional correctness, we check the absence of deadlock in the system (F1), and we check that for all tasks the required number of GPS services are available in each group. Deadlock is directly supported in Uppaal. F2 presents a concrete scenario that checks the required GPS services for group 1.

F1: A[] not deadlock
F2: A[] ServerAgent1.SubmitTask imply
    MASmvdStruct[1].nMembers >=
    MASactivity1.number_GPS

5.5.2 GPS Service Self-Adaptation

We define three robustness properties that allow verification of self-adaptation of the GPS service. R1 specifies that a GPS service, which provides an insufficient quality, will eventually be recognized by the Analyze process. R2 specifies that the GPS service of a GPS module with insufficient quality will actually be deactivated. Finally, R3 specifies that such GPS service will eventually be removed from any MVD. R1 and R2 are exemplified by defining an instance that analyzes cases on the phone number 1. R3 studies the scenario in which 3 groups (1, 2 and 3) do not contain the deteriorated phone 1.

R1: GPSModule(1).Deteriorating -->
    GPSSAAnalyze(1).ChangeBad
R2: GPSModule(1).Deteriorating -->
    MASPhoneStruct[1].GPS_Service == DEACTIVATED
R3: GPSModule(1).Deteriorating -->
    MASmvdStruct[1].member[1] == NOT_USED &
    MASmvdStruct[3].member[1] == NOT_USED

5.5.3 MVD Self-Healing

We define three properties that allow verification of self-healing of MVDs. R4 specifies that when an MVD is incomplete, eventually a search of a replacing GPS service will be initiated. R5 verifies that a search will eventually be completed successfully, and R6 that this will lead to the MVD being in a Complete (or Redundant) state. R4 to R6 illustrate the rules applied to group 1.

1 The correctness of R5 and R6 relies on the assumption that only a fraction of the available GPS services can go down at the same time, and redundant services are available, as described in Section 5.3.

73
5 Guaranteeing Robustness using Formally Verified MAPE-K Loops

R4: MVDSHAnalyze(1).Incomplete -->
    MVDSHPlan(1).LookForFreeGPS
R5: MVDSHPlan(1).LookForFreeGPS -->
    MVDSHPlan(1).AllFine
R6: MVDSHAnalyze(1).Incomplete -->
    MVDSHAnalyze(1).Complete ||
    MVDSHAnalyze(1).Redundant

5.5.4 MAPE Loop Interference

Finally, we verify that there is no interference between the MAPE loops. R7 specifies that the deactivation of a GPS service (as a result from adaptations in the first MAPE loop), is correctly handled by not including the undesired GPS service in any MVD (by the MVD self-healing process in the second MAPE loop). R8 specifies another required interference property that concerns the integration of a phone in a group (second MAPE loop) while the GPS service of this phone is deactivated (first MAPE loop) because it can no longer provide the required quality. In this case, the service of the failing phone should be replaced by another available service. R8 shows a concrete scenario where group 1 (MVDSHPlan(1) and MASmvdStruct[1]) has initially selected phone 2. In case the GPS service of phone 2 becomes DEACTIVATED during the integration process, phone 2 will be NOT_USED and a replacing service (MVDSHPlan(1),found_Phone != 2) will be selected. Finally, R9 specifies that, as a result of self-healing processes, a GPS service will not belong to two different MVDs at a time.

R7: GPSInternalEffector(1).Inactive -->
    MASmvdStruct[1].member[1] == NOT_USED &&
    MASmvdStruct[3].member[1] == NOT_USED
R8: MASphoneStruct[2].GPS_Service == DEACTIVATED &&
    MVDSHPlan(1).found_Phone == 2 -->
    MVDSHPlan(1).found_Phone != 2
R9:A[] forall(Pid:phone_id) forall(Mid1:MVD_id)
    forall(Mid2:MVD_id)
    MASmvdStruct[Mid1].member[Pid] == USED &&
    Mid1 != Mid2
    imply MASmvdStruct[Mid2].member[Pid] == NOT_USED

5.5.5 Verification and Results

We instantiated different scenarios with increasing number of phones and MVDs. Learning activities consisted of 6 tasks. Measurements confirm that the processing cost for verification grow exponentially with the complexity of the scenarios (number of phones and number or MVDs). In this particular domain, a scenario with 3 phones and 1 MVD required 393 ms to verify the deadlock property (F1), while 6 phones and 2 MVDs required around 21 minutes, and 12 phones and 3 MVDs required several hours to analyze. The analysis results show only small differences in terms of cost for verifying the other properties for the same scenario. For example, for a scenario with 1 MVD and 3 phones, the verification cost varied between 167 ms to 178 ms for properties F2 and the robustness properties R1 to R9.
5.6 From Design to Implementation

The original mobile learning application was implemented using JADE [16]. JADE is a platform that provides facilities to develop distributed multi-agent systems, including services for message communication, service registration and discovery, etc. We briefly explain how we implemented the self-adaptive layer on top of the legacy system.

We mapped one-to-one the behavioral processes presented in Section 5.4 to Java classes. Fig. 5.22 illustrates the classes that implement the MAPE loop of the GPS service concern. The probe processes are implemented as Jade behaviors, which are executed by the agent on each mobile device. Snippet 5.3 shows the implementation of the GPS Quality Probe. The code that is executed on a periodic basis (TickerBehaviour) has been granted access to the current GPS accuracy data via the getAccuracy interface offered by the LocationManager. Effector classes implement effectors that have access to the underlying system to adapt it when needed. For example, the GPSServiceEffector class implements an effector that consumes the setGPSService() method of the PhoneManager to activate (or deactivate) the GPS service when demanded.

Snippet 5.3: GPSQualityProbeBehaviour class

```java
public class GPSQualityProbeBehaviour extends TickerBehaviour {
    [...]
    public GPSQualityProbeBehaviour(Agent agent, long period, float threshold) {
        super(agent, period);
    }

    @Override
    protected void onTick() {
        float accuracy = LocationManager.getInstance().getMyLocation().
            getAccuracy();
        myLogger.log(Logger.INFO, "Tick! Check GPS accuracy");
        if (accuracy - prev_accuracy > Threshold)
            myLogger.log(Logger.WARNING, "GPS accuracy changed");
        GPSMonitor.getInstance().update("accuracy", accuracy);
        pre_accuracy = accuracy;
    }
}
```

Analogously, we mapped the processes of the MVD self-healing loop to Java classes. For the communication between the masters and the activity agent on the one hand, and between the agents of MVDs on the other hand, we used ACLMessages (Agent Communication Language Messages) provided by JADE. These messages offer high-level communication primitives and supporting protocols, such as request-confirm, inform, etc.

5.7 Conclusions and Future Work

In this paper, we explained how we have extended a legacy mobile learning application with a self-adaptation layer, making the system robust to degrading GPS accuracy. We designed the self-adaptive layer as a set of interacting MAPE loops.
5 Guaranteeing Robustness using Formally Verified MAPE-K Loops

Distibuted over the mobile devices. To guarantee the required robustness requirements, we used timed automata to specify the behaviors of the MAPE loops, and expressed the robustness requirements as formal properties in TCTL. The Uppaal tool allowed us to verify the properties. The behavioral design was then mapped to Java implementation.

While MAPE is widely recognized in the community, it is often only used as a conceptual guidance for the design of self-adaptive systems. In this paper, we used explicit modules for each of the adaptation functions of MAPE loops in the design of a self-adaptive application and mapped these modules one-to-one to an implementation. We report a number of lessons learned from this effort. Using explicit modules for each of the adaptation functions of MAPE loops makes it easier to:

• model the self-adaptive behavior, as the designer can focus on one activity at a time;
• model the interaction between the managed and managing system, as the interaction points are well-defined;
• reason about the behavior within and between MAPE loops, as a result of the clear separation of concerns;
• reason about the interactions between the managed and managing system;
• specify required properties, and these properties can be specified at a more fine-grained level;
• identify problems in the design;
• map design to implementation.

However, there are also some tradeoffs:

• some MAPE activities have a straightforward behavior, which may raise questions about the usefulness of a separate specification;
• the size of the design increases;
• the cost for verification grows.

Regarding the generalization of the approach, we remark that in the presented application, we could easily separate the adaptation behavior from the business logic. From other work in our team [104], it is clear that this separation is not always so easy to realize. This calls for more attention to the separation of managed and managing system and the study of their interactions, including formal verification.

As the next step, we first aim to further verify that the required properties hold in the implemented mobile learning application. To that end, we plan to check the compliance of traces derived from the verification of the design with traces obtained from the running implementation. Second, we plan to study the overhead implied by adding MAPE loops to the system at runtime, including cost in terms of resources, and communication due to interactions among MAPE loops.
distributed over the mobile devices. To guarantee the required robustness requirements, we used timed automata to specify the behaviors of the MAPE loops, and expressed the robustness requirements as formal properties in TCTL. The Uppaal tool allowed us to verify the properties. The behavioral design was then mapped to Java implementation.

While MAPE is widely recognized in the community, it is often only used as a conceptual guidance for the design of self-adaptive systems. In this paper, we used explicit modules for each of the adaptation functions of MAPE loops in the design of a self-adaptive application and mapped these modules one-to-one to an implementation. We report a number of lessons learned from this effort. Using explicit modules for each of the adaptation functions of MAPE loops makes it easier to:

- model the self-adaptive behavior, as the designer can focus on one activity at a time;
- model the interaction between the managed and managing system, as the interaction points are well-defined;
- reason about the behavior within and between MAPE loops, as a result of the clear separation of concerns;
- reason about the interactions between the managed and managing system;
- specify required properties, and these properties can be specified at a more fine-grained level;
- identify problems in the design;
- map design to implementation.

However, there are also some tradeoffs:

- some MAPE activities have a straightforward behavior, which may raise questions about the usefulness of a separate specification;
- the size of the design increases;
- the cost for verification grows.

Regarding the generalization of the approach, we remark that in the presented application, we could easily separate the adaptation behavior from the business logic. From other work in our team [104], it is clear that this separation is not always so easy to realize. This calls for more attention to the separation of managed and managing system and the study of their interactions, including formal verification.

As the next step, we first aim to further verify that the required properties hold in the implemented mobile learning application. To that end, we plan to check the compliance of traces derived from the verification of the design with traces obtained from the running implementation. Second, we plan to study the overhead implied by adding MAPE loops to the system at runtime, including cost in terms of resources, and communication due to interactions among MAPE loops.
Chapter 6
MAPE-K Formal Templates

To support designers with specifying models and properties effectively and correctly, this Chapter studies the consolidation of formal designs of a family of self-adaptive systems. We documented one of the concrete applications in Chapter 5.

The family of systems define a target domain characterized as follows:

• Systems comprise software deployed on distributed nodes;
• The nodes have an explicit position in the environment (and may be mobile);
• The nodes have continual communication access;
• Dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software;
• Resources in the system may come and go.

The objectives of this study are:

• to specify a set of reusable behavior models for the design of self-adaptive systems;
• to specify a set of reusable property specifications for the verification of self-adaptive behaviors;
• to demonstrate the reuse of the reusable models and property specifications.

The results of the study are a set of formally specified templates to design MAPE-K based self-adaptive systems and verify robustness and openness properties. We call these templates MAPE-K Formal Templates. The templates address the research questions “RQ2: How to model the behaviors of MAPE-K components to deal with robustness and openness requirements?” and “RQ3: What properties must be verified to provide evidence that the required self-adaptation behaviors are assured?”

The MAPE-K Formal Templates comprise: (1) reusable behavior patterns for modeling the different components of a MAPE-K feedback loop (based on networks of timed automata), and (2) property specification patterns that support verification of the correctness of the adaptation behaviors (based on timed computation tree logic).

To demonstrate the reusability of the MAPE-K Formal Templates, we performed four case studies, in which students applied the templates in the design
Chapter 6

MAPE-K Formal Templates

To support designers with specifying models and properties effectively and correctly, this Chapter studies the consolidation of formal designs of a family of self-adaptive systems. We documented one of the concrete applications in Chapter 5. The family of systems define a target domain characterized as follows:

- Systems comprise software deployed on distributed nodes;
- The nodes have an explicit position in the environment (and may be mobile);
- The nodes have continual communication access;
- Dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software;
- Resources in the system may come and go.

The objectives of this study are:

- to specify a set of reusable behavior models for the design of self-adaptive systems;
- to specify a set of reusable property specifications for the verification of self-adaptive behaviors;
- to demonstrate the reuse of the reusable models and property specifications.

The results of the study are a set of formally specified templates to design MAPE-K based self-adaptive systems and verify robustness and openness properties. We call these templates MAPE-K Formal Templates. The templates address the research questions "RQ2: How to model the behaviors of MAPE-K components to deal with robustness and openness requirements?" and "RQ3: What properties must be verified to provide evidence that the required self-adaptation behaviors are assured?"

The MAPE-K Formal Templates comprise: (1) reusable behavior patterns for modeling the different components of a MAPE-K feedback loop (based on networks of timed automata), and (2) property specification patterns that support verification of the correctness of the adaptation behaviors (based on timed computation tree logic).

To demonstrate the reusability of the MAPE-K Formal Templates, we performed four case studies, in which students applied the templates in the design
of concrete self-adaptive applications that fit with the target domain. One of the applications was a simple smart home system, another one was a fire monitoring system and the two others were vehicular traffic systems.

The case studies demonstrate that the MAPE-K Formal Templates can be applied to applications that share the characteristics of the target domain. We observed that participants reused the specification patterns using refinement (of states, transitions, functions, etc.), reduction (only the required parts of the templates were used), and multiple instantiations of patterns. The participants in the case studies stated that the proposed set of behavior specification patterns covered their needs and the property specification patterns simplified the verification tasks.

The results of this study provide qualitative evidence that the MAPE-K Formal Templates support designers of self-adaptive systems with specifying models and properties effectively and correctly.
MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems

Abstract
Designing software systems that have to deal with dynamic operating conditions, such as changing availability of resources and faults that are difficult to predict, is complex. A promising approach to handle such dynamics is self-adaptation that can be realized by a MAPE-K feedback loop (Monitor-Analyze-Plan-Execute plus Knowledge). To provide evidence that the system goals are satisfied, regarding the changing conditions, state of the art advocates the use of formal methods. However, little research has been done on consolidating design knowledge of self-adaptive systems. To support designers, this paper contributes with a set of formally specified MAPE-K templates that document design expertise for a family of self-adaptive systems. The templates comprise: (1) behavior specification patterns for modeling the different components of a MAPE-K feedback loop (based on networks of timed automata), and (2) property specification patterns that support verification of the correctness of the adaptation behaviors (based on timed computation tree logic). To demonstrate the reusability of the formal templates, we performed four case studies in which final-year Master students used the templates to design different self-adaptive systems.

6.1 Introduction
Designing contemporary software intensive systems, such as mobile applications, multi-robot systems, and networked smart homes, and guaranteeing the system goals is complex due to their dynamic operating conditions. Examples are changing availability of resources and faults that are difficult to predict. This situation has led to the development of self-adaptive systems, i.e., systems that are able to reconfigure their structure and modify their behavior at run-time in order to adapt to changes with little or no human intervention [48, 127, 128]. The underlying principle of self-adaptation is separation of concerns. A self-adaptive system typically consists of a managed system and a feedback loop. The managed system deals with the domain concerns for the users, while the feedback loop deals with adaptation concerns of the managed system (e.g., optimize the managed system for different operating conditions, heal the managed system when a fault is detected, etc.). In this research, we focus on architecture-based self-adaptation [70, 121, 143, 187]. In architecture-based self-adaptation, the system reasons about a model of itself and its environment and adapts when needed according to some adaptation goals. The feedback loop typically consists of Monitor, Analyze, Plan, and Execute components, complemented with models (Knowledge) of the managed system, its environment and adaptation goals. This structure is referred to as MAPE-K [113].
One important challenge in engineering self-adaptive systems is to provide evidence that the system goals are satisfied, regarding the dynamically changing operating conditions [48, 127, 128]. To that end, state of the art in architecture-based self-adaptation advocates the use of formal methods. A recent systematic literature survey [188] identified a total of 75 papers that use formal methods in self-adaptive systems. Studying the state of the art reveals that designers usually specify models and required properties of the self-adaptive system for the problem at hand from scratch [188]. There is little systematic consolidation of the design knowledge for future use. Only a few efforts have been done on consolidating design expertise of self-adaptive systems, including [151] that documents twelve patterns for the design of self-adaptive systems, and [190] that presents a set of patterns for decentralized control of MAPE-K feedback loops. However, the focus is primarily on structural aspects and there is a lack of rigorous underpinning of the patterns. Other researchers have demonstrated the value of formally specified templates, e.g., [173] formally specified modeling patterns and [88] identified specification patterns for probabilistic requirements. [65] points out that the definition of such templates are an important research challenge for self-adaptive systems.

During the past years, we have been studying self-adaptation in a number of applications, with a particular focus on robustness and openness requirements. The family of systems we have studied are characterized by distributed nodes that have an explicit position in the environment and that have continual communication access. The dynamics in the system and its environment (e.g., mobility of nodes or streams of new tasks) are orders of magnitude lower than the speed of communication and the execution of the software. These characteristics represent a target domain that is representative for a large number of contemporary systems, including collaborative mobile applications, robotic systems, traffic support systems, etc. Some of the recent applications we studied are an intelligent monitoring system to detect traffic jams [104], a mobile system to support outdoor learning activities [81], and robotic transportation system [105].

In the course of designing and building these self-adaptive applications, we derived a set of MAPE-K Formal Templates for designing feedback loops of self-adaptive systems in the target domain. The formally specified MAPE-K Formal Templates document design expertise that can help designers with specifying models and properties for new self-adaptive software systems in the target domain. The templates comprise: (1) behavior specification patterns for modeling the different components of a MAPE-K feedback loop and their interactions, and (2) property specification patterns for specifying required properties of the adaptation behaviors that can be formally verified.

To demonstrate the reusability of the MAPE-K Formal Templates, we performed four evaluative case studies. During these case studies, final-year Master students applied the MAPE-K Formal Templates to design multiple self-adaptive applications that match with the characteristics of the target domain.

In this research, we use networks of timed automata (TA) as the formal language for the specification of behavior patterns, and timed computational tree logic (hereafter, TCTL) for the property specification patterns. In a recent study in the use of formal methods for self-adaptive systems [188], automata were identified as one
of the mostly used languages in the field. TA are automata extended with clock variables that comprise behavior states and actions that represent transitions between states. Clock variables are used to synchronize behaviors, together with channels that enable communication between behaviors. Therefore, a network of automata allows the specification of interacting behaviors (such as behaviors for Environment, Managed System, Monitor, Analyze, Plan and Execute components) that synchronize through clock variables and a set of channels, which altogether specify the overall behavior of a self-adaptive system.

TCTL is a formal language based on computational tree logic extended with clock variables that allows specifying properties. Expressions in TCTL describe state and path formulae enabling the verification of properties of interest, such as reachability (a system should/can/cannot etc. reach a particular state or states), liveness (something eventually will hold), safety (given a particular condition in the system, all possible system executions should hold certain desired conditions or avoid specific undesired conditions), etc. TA and TCTL can be combined to specify behaviors of self-adaptive systems and to verify that the desired properties hold.

The remainder of the paper is structured as follows. In Section 6.2, we discuss related work. Section 6.3 elaborates on the target domain that we focus on. The section introduces two self-adaptive applications from which the MAPE-K Formal Templates were derived. Section 6.4 presents the behavioral specification patterns for modeling feedback loops of self-adaptive systems. In Section 6.5, we present the property specification patterns that support the verification of self-adaptation behaviors. We illustrate the MAPE-K Formal Templates with excerpts of two self-adaptive applications. In Section 6.6, we report the results of the four case studies that demonstrate the reusability of the MAPE-K Formal Templates. Finally, we draw conclusions in Section 6.7.

## 6.2 Related Work

Over the past 15 years, a vast body of research on self-adaptive systems has been developed, and recently the exploitation of formal methods has gained increasing attention. We start the discussion of related work with a selection of approaches that use formal approaches for the design of self-adaptive systems. Then we zoom in on studies that present design patterns for self-adaptive systems. We conclude with a selection of studies on property specification patterns outside the scope of self-adaptive systems.

### 6.2.1 Formal Approaches to Self-Adaptation

Providing evidence that self-adaptation properties are satisfied is subject of active research [130, 174, 176]. During the recent Dagstuhl seminar "Software Engineering for Self-Adaptive Systems: Assurances" [128], several approaches to provide evidence for assurances of self-adaptive systems were studied. Some approaches use formal methods (e.g., model checking) at design time, others at runtime or
6 MAPE-K Formal Templates

a combination of both. Our research belongs to the methods that aim to provide evidence to assure system requirements during system design.

[198] presents a process to create formal models for adaptive systems, verify the models and automatically translate the models into executable programs. The authors use Petri Nets and linear temporal logic to provide assurances for the system goals, and model-based testing to guarantee conformance between the models and programs. In follow up work [200], the authors model a dynamically adaptive program as a collection of (non-adaptive) steady-state programs and a set of adaptations that realize transitions among steady state programs in response to environmental changes. To handle the state explosion, the authors propose a modular model checking approach. Linear Temporal Logic (LTL) to specify properties of the non-adaptive portions of the system is combined with A-LTL (an adaptation operator extension to LTL) to concisely specify properties that hold during the adaptation process.

FORMS [187] presents a comprehensive reference model, entitled FOrmal Reference Model for Self-adaptation. FORMS builds on existing frameworks and established principles of self-adaptation, such as computational reflection, MAPE-K, and architecture-based adaptation. The reference model offers a vocabulary that consists of a small number of primitives and a set of relationships among them that delineates the rules of composition. The model is formally specified, which enables engineers to precisely define the key characteristics of self-adaptive software systems, and reason about them.

[62] presents a quantitative approach for making adaptation decisions under uncertainty, called POISED. POISED builds on possibility theory (grounded in fuzzy mathematics) to assess both the positive and negative consequences of uncertainty. POISED makes adaptation decisions (reconfiguration of its customizable software components) that result in the best range of potential behavior, improving a software system’s quality of service. [74] proposes Mechatronic UML to support model-driven development and verification of safety guarantees of distributed embedded real-time systems. Mechatronic UML modeling is based on a rigorously specified subset of UML. This approach offers verification at design time to provide assurances regarding the system functional and non-functional requirements, by describing the complete system behavior.

The discussed works provide a sample of research that exploits formal methods in the design of self-adaptive systems during the past decade. The work underpins the relevance of rigorous modeling and verification. However, it also shows that there is a lack of consolidating expertise in the design of self-adaptive systems’ behaviors, allowing engineers benefiting from systematic knowledge [166, 188]. The aim of this paper is to provide templates for the specifications of MAPE-K behaviors that consolidate design expertise and provide support to designers of a family of self-adaptive systems.
6.2 Related Work

6.2.2 Design Patterns

A common approach to consolidate design knowledge is by means of documenting design patterns. In the field of architecture-based self-adaptive systems, only a few efforts have been performed on documenting design patterns.

[86] presents several software reconfiguration patterns for dynamic evolution of software architectures. The authors define a reconfiguration pattern comprising of recurring sequences of adaptation steps (e.g., stopping/starting, (un)linking, adding/removing) necessary for ensuring consistent adaptation of a software system. In [87], the authors employ reconfiguration patterns in the context of self-managed service-oriented software systems, while in [63], Esfahani et al. show their utility in the design of architecture-based middleware solutions.

[60] proposes an adaptive software architecture style consisting of: (1) a basic bottom most layer with application components that control a robot, and (2) one or more meta layers with meta-level components that implement fault-tolerance, dynamic update, resource discovery, redeployment, etc. In the proposed architecture, each layer may adapt the layer beneath. The authors use the approach for the design and implementation of self-adaptive behavior in robotics software.

[151] presents a set of 12 patterns based on a number of studies in self-adaptive systems. The patterns are documented following a classic template that describes the intent of the pattern, context, structure using UML class diagrams, and consequences of the pattern among other description items. The presented patterns focus at software design level. They aim to facilitate the design and construction of a self-adaptive software system by providing alternative solutions for adaptation in the systems’ implementations.

In another recent study [190], the authors present a set of patterns for decentralized control in self-adaptive systems. The study documents five patterns for the coordination and interaction of multiple MAPE-K control loops, including master-slave MAPE pattern, coordination pattern and information sharing pattern for large self-adaptive systems. The primary focus of the patterns is on structural aspects of the architectural design of self-adaptation logic. The patterns are illustrated with concrete application instances from which they were derived.

The presented efforts provide a number of documented patterns for architecture-based self-adaptive systems. The patterns range from a high-level architecture to concrete design level. The primary focus is on structural aspects. The work presented in this paper complements these efforts with design patterns to rigorously specify behaviors of interacting MAPE-K components.

6.2.3 Property Specification Patterns

The second part of this work is centered on property specification patterns that allow to formulate and verify required properties of MAPE-K behaviors. To the best of our knowledge, no related work exists on property patterns dedicated to self-adaptive systems. We discuss a number of general approaches that do not focus on self-adaptive systems and then zoom in on initial ideas for property specification patterns of self-adaptive systems.
[58] introduces property specifications for verification of finite-state machines. The specifications allow expressing recurring properties in a generalized form allowing to model and verify system requirements in a rigorous manner. Other examples of property specifications have been proposed for real-time systems [120] and systems with probabilistic quality requirements [88]. [20] presents the results of a study on specification templates for service-based applications. The authors performed an extensive analysis of the usage of documented specification properties for a large body of specifications of a concrete banking company. Current property patterns are not specified to be applied in self-adaptive systems, and recent efforts in property specification for self-adaptive systems have not been consolidated to create patterns for the field.

As model checking is one of the prominent approaches to provide evidence for system properties, we analyzed existing work that employ model checking techniques for verification of self-adaptive systems [188]. From this analysis, we derived an interesting model that maps the different types of behaviors of self-adaptive systems to zones of the state space: normal behavior, undesired behavior, adaptation behavior, and invalid behavior. Properties of interest with respect to self-adaptation typically map to transitions between different zones. For example, a property may express whether invalid states can be reached from a system’s normal behavior, or it can express whether a system adapts correctly from undesired behavior via adaptation behavior back to normal behavior. This zone model provides a potential basis for a systematic documentation of specification properties for self-adaptive systems. We refer the interested reader for additional information to [188].

6.3 Target Domain

The templates embody the knowledge that we gained from the formalization of adaptive behaviors for a number of self-adaptive systems in mobile learning [81], traffic control [104] and robotics [105] domains. The target domain that we have studied represents self-adaptive systems with the following characteristics:

- Systems comprise software deployed on distributed nodes;
- The nodes have an explicit position in the environment (and may be mobile);
- The nodes have continual communication access;
- Dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software;
- Resources in the system may come and go.

As we consider distributed self-adaptive systems, the managed system typically comprise multiple parts (that are deployed on different nodes) that we denote with local managed systems. These parts may be adapted by one or more feedback loops. Concretely, we study self-adaptive systems based on MAPE-K feedback loops to deal with robustness and openness requirements. It is important to notice that our particular focus is on adaptations that require adding and removing
resources in the system. With resources, we refer to controllable parts of the managed system, such as nodes, subsystems and components. Regarding robustness, we study self-adaptation to deal with parts of the system that fail. Regarding openness, we study self-adaptation to deal with parts that come and go dynamically. To that end, one or more MAPE-K feedback loops can observe local managed systems and their execution context and adapt the local managed systems by adding and removing resources.

Below we describe two example applications that we use throughout the paper to illustrate the different templates. These applications were instances of the applications from which the templates were derived. As explained in the introduction, we use TA and TCTL for the formalization of the MAPE-K templates. Appendix A provides a brief summary of the languages primitives of TA and TCTL. For the specification of models and verification of properties we used the Uppaal tool [15].

### 6.3 Target Domain

#### 6.3.1 Mobile Learning Application

One of the mobile learning applications we studied supports outdoor learning activities of pupils that have to learn geometry concepts using GPS enabled mobile devices [81]. GPS services are the resources in this domain. The activities require the pupils to work in groups, i.e., two mobile devices are required for distance calculations, three devices for triangulation, and more devices for other more advanced activities. Two MAPE-K feedback loops were designed in order to provide robustness regarding the following two concerns. First, the environmental conditions may affect the GPS reliability and consequently affect the quality of the GPS-based measurements. We specified a first MAPE-K loop to ensure that only GPS services that provide a minimum level of accuracy in their measurements are used. Second, the number of available GPS services in a group may not satisfy the required services for the activities. We specified a second MAPE-K loop to ensure that groups possess the required GPS services to carry out the required activity. The MAPE-K loops are distributed among the mobile devices that are used in the learning activities. In this paper, we focus on the second concern: ensuring that groups have the required GPS services to complete their tasks (second MAPE-K loop). We illustrate parts of the environment and managed system’s behavior for the mobile learning application.

Fig. 6.1-top shows the timed automaton that specifies the behavior of the environment in which the learning activities takes place. The behavior provides an abstraction of the sky conditions, in which the sky can either be Clear or Cloudy. Transitions between these states were modeled through specific functions (getNextCloudy, getNextClearing, isValid) and conditions on a timer.

The automaton shown in Fig. 6.1-middle specifies the part of the GPS component behavior that deals with the quality of the acquired locations. The behavior of the GPS behavior is affected by the sky conditions, that is, the GPS service is either activated or deactivated (the automata communicate through the qualityGPSModule signals). The updateGPSService() function allows activation or deactivation of the GPS service.
6 MAPE-K Formal Templates

The Probe behavior shown in Fig. 6.1-bottom senses the state of the GPS service and notifies the related self-adaptive loop when needed (via the GPSwentDown signal).

![Automaton describing the map layout of a robot](image)

**Figure 6.1:** Sky environment (top), GPS module (middle) and GPS-Probe behaviors for a mobile learning application

The combination of the automata specifies the behavior of a local managed system that, depending on the sky conditions, may activate or deactivate GPS services. Based on this behavior, a group of phones may become incomplete when services are deactivated. We focus on the completeness of required GPS services in a group as a robustness concern. To that end, we extended the domain logic of the mobile learning application with MAPE-K feedback loops to deal with these robustness properties.

6.3.2 Logistic Robotic System

We studied different adaptation scenarios where robots have to perform transportation tasks in a warehouse following a graph-based routing layout. To drive in the warehouse, the robots maintain a representation of the routing layout. For this application, map elements (such as lanes, location on the layout, etc.) represent resources in a distributed system. In this paper, we study a scenario where robots are instructed to disable or enable particular lanes of the layout (e.g., for maintenance activities in a warehouse) modify the layout by adding and removing parts of it where the robots can drive (e.g., adding a new drop location to deliver loads). We specified the layout that each robot uses for driving in a warehouse (Fig. 6.2), and the behavior of a set of interfaces that allow a manager to manipulate the map elements in the layout (Fig. 6.3-left). We defined a set of possible manipulation
actions that a manager could perform on the layout during execution time: adding \((b\text{Add})\) and removing \((b\text{Rem})\) locations \((-\text{Loc})\), edges \((-\text{Edge})\)\(^1\) and destinations \((-\text{Dest})\) in the map. For instance, \(b\text{RemLoc}\) in Fig. 6.3-left collects requests from the manager to remove a location from the layout, and notifies a specific robot \(RiD\) to update its knowledge regarding desired changes in the layout. Fig. 6.2 shows the representation of the layout. Locations and destinations that are currently not available for the robot are marked in white, and disabled edges are marked in gray. The robot is unaware of the existence of these specific locations and lanes.

![Automaton describing the map layout of a robot](image)

 Manipulating the map layout may be constrained by the current conditions, e.g., a robot cannot disable a lane it is driving on. Fig. 6.3-right specifies a behavior that controls the current robot location and targeted destination in the layout. To deal with the manipulations of the layout we extended the domain logic of the robots with a MAPE-K feedback loop.

### 6.4 Behavior Specification Patterns for Modeling MAPE-K Feedback Loops

MAPE-K [113] provides a reference model to realize self-adaption that comprise a control loop consisting of four main components that realize adaptation actions to a managed system. Fig. 6.4 shows a high level model of a self-adaptive system where a managed system is extended with a feedback loop realized through

\(^1\) Adding or removing an edge in the map translated in enabling or disabling the related lane on the layout.
6 MAPE-K Formal Templates

We now present the set of formally specified patterns for modeling the component behaviors of MAPE-K feedback loops that we derived from previous experiences designing self-adaptive systems in collaborative mobile learning [81], robotics [79, 105] and traffic system domains [54, 104]. For each pattern, we start with presenting concrete instances of the specific MAPE component for the two application scenarios presented in Section 6.3. Then, we study the common characteristics of the formalized components and identify behavior patterns that provide reusable specifications for the design of different MAPE behaviors.

Due to space constraints, we cannot provide a detailed specification of all aspects of the patterns, including time- and event-trigger mechanisms, specification of data structures, specification of all functions, etc. For a complete specification...
6.4 Behavior Specification Patterns for Modeling MAPE-K Feedback Loops

with a detailed explanation of the patterns, supported by different examples we refer to the project website\textsuperscript{1}.

6.4.1 Knowledge

To realize self-adaptation, the Monitor, Analyze, Plan and Execute components use models (Knowledge) that provide an abstract representation of relevant aspects of the managed system, its environment, and the self-adaptation goals. In line with FORMS [187], we divide knowledge in four parts (which are technically specified as four \textit{struct} definitions in Uppaal) as follows:

- \textit{ManagedSystemKnowledge} provides an abstraction of the managed system; this part of the knowledge represents relevant information regarding the resources in the managed system. This information may contain data about resources, whether resources are used or not, and other aspects, such as resource dependencies or quality properties.

- \textit{EnvironmentKnowledge} provides an abstraction of the environment, that is, this knowledge represents information about the context in which the self-adaptive system is situated and operates.

- \textit{ConcernKnowledge} describes the knowledge w.r.t. the adaptation concern of interest; this part specifies the \textit{requiredResources} to realize the goals of the self-adaptive system.

- \textit{AdaptationKnowledge} represents runtime data that is shared between the MAPE behaviors; we distinguish between \textit{flags} that are used for indirect synchronization of behaviors and \textit{workingData} that represents data that behaviors use to realize their functions (e.g., historical data for analysis, workflows for adaptation, etc.).

6.4.2 Monitor

Monitor collects information (through probes) from the managed system and possibly the environment to update the Knowledge of the feedback loop. Before updating the Knowledge, the Monitor may preprocess the collected data. Examples of preprocessing are standardization of data, filtering and aggregating data.

6.4.2.1 Mobile Learning Case

In the mobile learning scenario, an automaton was specified to describe the GPS-Monitor behavior. The GPS-Monitor is in charge of updating knowledge about the membership of phones in groups.

Concretely, the Monitor behavior: (1) identifies whether GPS services go down, (2) checks whether a GPS going down was already reported, (3) identifies the group in which the mobile device of the GPS service was used, and (4) updates the knowledge regarding the identified group. The four steps are represented by four states of the Monitor automaton (Fig. 6.5). In the \textit{Monitoring} state, the behavior is

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6_5.png}
\caption{Behavior specifying interfaces that managers use to update the map layout (left) and updating the robot's current location and destination (right).}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6_4.png}
\caption{MAPE-K. Model for self-adaptive systems [113]}
\end{figure}

\footnotesize{\textsuperscript{1} http://homepage.lnu.se/staff/daweaa/MAPE-K-Templates.htm [79]}
6 MAPE-K Formal Templates

waiting to be notified by the Probe (GPSwentDown[Pid]?). If it is a new notification (isNew(caseID)), the group identification starts at CollectingGroup (through determineMyMVD(Pid)), and finally, after GroupIdentified, the behavior updates the knowledge regarding phone memberships via the remove_Phone(myMVD) function.

6.4.2.2 Robot Transportation Case

In the robot transportation scenario, we designed a monitor behavior that updates knowledge regarding: (1) desired modifications in the layout, and (2) the current position and destination of the robot. Therefore, the behavior monitors two data sources that are required by the adaptation logic. First, the behavior captures signals that identify the different type of modifications that should be applied on the layout (signals can be sent via the sAddEdge, sRemEdge, sAddLoc, sRemLoc, sAddDest and sRemDest channels shown in the top automata in Fig. 6.6). Second (Fig. 6.6-bottom), the behavior monitors robot’s contextual information needed to determine whether an adaptation can take place, or should be postponed until required conditions are satisfied (e.g. the robot is driving to a destination that needs to be removed). This was specified in the behavior through two channels that identified updates regarding the current robot location (sPosUpdate) and targeted destination (sDestUpdate). The monitor behavior triggers the analyze process via the analyze[RiD]! signal.

6.4.2.3 Monitor Design Pattern

We now present a Monitor design pattern that consolidates our knowledge in the design of monitor behaviors.

Triggering events, such as GPSwentDown[Pid]? in the mobile learning scenario and sDestUpdate[Rid]? in the robot transportation scenario, are generalized into a triggering mechanism represented to initiate the collection of data. Fig. 6.7 shows an event triggering mechanism specified via the Monitor[Node_ID]? signal. In the mobile learning application, this was specified through the determineMyMVD() function; in the robot application, the data to monitor was self-contained in the signals. We generalize this step in a getData() function, which is in charge of collecting the values of the data items that the behavior is expected to monitor. We include a preprocessing step in the monitor for such behaviors that may require data standardization, aggregations, etc. (if isPreprocessingRequired() then preprocess()). In the mobile learning case, a preprocessing step was required to avoid reporting repeated requests.
Finally, after new values are found (NewChange), the monitor behaviors update the knowledge and notify the related Analyze components. In the mobile learning scenario this was modeled with the remove_phone() function that updated specific knowledge of the group and set the conditions for an analyze behavior to process new data. For the robot transportation application this was modeled via the different update functions (rem_loc[Rid] = RECEIVED, update_dest(), etc.) followed by the analyze[Rid]! signal. We specify this process in the Monitor design pattern with the updateKnowledge() function, followed by a signal that triggers Analysis (Analyze[Node_ID]!).

In summary, the Monitor behavior is divided in the following steps: monitor triggering, collecting data, preprocessing data, updating working data, and signaling analyze behavior(s).

Alternatively, we specify a time-triggered patterns (all the pattern variations are documented on the project website). Here we present a time-triggered Monitor (Fig. 6.8) that periodically initiates its behavior to collect sensed data via getData(). The monitor frequency is defined by a time condition (t == Period) and a state invariant (t <= Period).
6.4.3 Analyze

Analyze is responsible for determining whether adaptation actions are required based on the state of the managed system, the environment and the adaptation concern of interest.

6.4.3.1 Mobile Learning Case

For the mobile learning scenario, we designed an Analyze behavior that determines whether groups have enough GPS services to perform the learning activities. Fig. 6.9 shows the automaton of the analyze behavior with time triggering (on a 5-time-units frequency basis). The behavior uses the activity requirements and the number of used GPS resources in the group to perform analysis. The getRequired() and getUsed() functions provide this information by accessing ConcernKnowledge and ManagedSystemKnowledge. This knowledge is then used to evaluate whether the available GPS services in the group are sufficient to realize the activity goals (using the comparison functions). The analysis results are used to coordinate with the Plan to take action when needed (e.g. SH_MVD_Incomplete[Mid]!).

Figure 6.9: Analyze behavior for the mobile learning application

6.4.3.2 Robot Transportation Case

For the robot transportation case, the Analyze behavior needs to identify whether the representation of the layout in the robot corresponds to the real layout defined by the administrator. We specified three functions in charge to determine whether the current layout representation was correct (noNeedForAdaptation()), required new lanes, locations or other resources (needAdaptationAdd()), or resources should be removed from the layout (needAdaptationRemove()). Fig. 6.10 shows the event-triggered analyze behavior for the robot scenario.
Based on the functions, the behavior could take three different transitions from the Waiting state. A transition via ElementsMissing is taken in case elements in the map should be enabled or added. A transition via ElementsExtra is taken when one (or more) elements in the map should be disabled or removed. For these two cases, the Analyze behavior communicated with a Plan behavior in the same robot (RiD) through signaling (planning*[RiD]!). In the remaining case, the Analyze behavior directly returns to the Waiting state and waits until new analysis are required.

6.4.3.3 Analyze Design Pattern

Now, we present the Analyze pattern (Fig. 6.11) that consolidates the expertise we acquired from designing analyze behaviors for different applications as illustrated above.

The Analyze behavior matches the required resources with the resources in use, taking into account the current context and working data (such as GPS services or lanes in a robot layout). We equipped the Analyze behavior with methods to analyze the needs for adaptation. Functions such as comparing quantities of GPS services and checking that the robot works with an updated layout (needAdaptationAdd()) are generalized into a matchResources() function. We support the analysis with a set of functions in charge of collecting data from different knowledge representations (get*( ) functions). For example, we observed this type of functions in the getRequired() in the mobile learning application, and similar functionality was internally specified in the three analysis functions designed in the robots application. We provide an example (needAdaptationAdd) in Declaration 6.1.

```
1  bool needAdaptationAdd(){
2      return add_loc[RiD] == RECVD || enable_lane[RiD] == RECVD ||
3          add_dest[RiD] == RECVD;
4  }
```

Snippet 6.1: Analyze.needAdaptationAdd function for robot traffic application

Analysis comprise three primary states. The result is Satisfied when the managed system possesses the resources it requires to realize its goals. The Complete state in the mobile learning, and the free transition in the robots (noNeedForAdaptation()) show this part of the behavior. The analysis result is Undersatisfied when the system lacks resources or these are not satisfying the current context and goals, and Oversatisfied when the system has redundant resources. These states directly
match with *Redundant/Incomplete* and *ElementsExtra/ElementsMissing* from our illustrative scenarios. The results of the analysis are then communicated to the Plan behavior (e.g.*Plan_UnSatisfied*[Node_ID]*)! Notice that an Analyze can notify *Satisfied* states to the Plan behavior, in case adaptation plans may be cancelled. Fig. 6.11 shows the generic template for an analyze behavior with event triggering (*Analyze*[Node_ID]*)!.

**Figure 6.11:** Analyze behavior pattern with event-triggering

To summarize, we define the following steps in a Analyze behavior: analyze triggering, data collection, analysis process and signaling the related plan behavior(s).

**6.4.4 Plan**

The Plan behavior is responsible for planning mitigation actions to adapt the managed system when needed.

**6.4.4.1 Mobile Learning Case**

Given the possible outcomes of the analysis process, we designed a Plan behavior to correct incomplete groups (*SH_MVD_Incomplete*[Mid]!). Fig. 6.12 shows the specification for the Plan behavior. On the right hand side of the automaton, we specified the adaptation plan that must be carried out in order to recover the system from an undesired state. We defined a set of tasks in the *updateActions*(Mid) function, that specify that an available substitute mobile device must be found and integrated into the group.

We designed the Plan behavior to coordinate with the Execute behavior in order to apply the adaptation actions. On the right side of the automaton, the Plan coordinates with the Execute via the *addPhone*[Mid]! signal. On the left side of the automaton, we designed the Plan behavior to interrupt adaptation actions for particular cases in which the managed system has recovered from undesired states (i.e., when the activity requires less resources than before). Therefore, the Plan includes a branch that can cancel previous requested adaptation actions (*Execute_cancel*[Mid]!).
6.4 Behavior Specification Patterns for Modeling MAPE-K Feedback Loops

6.4.4.2 Robot Transportation Case
The plan behavior for the robotic case comprises two automata. The first automaton (Fig. 6.13-left) is in charge of composing a plan with the required adaptation actions to deactivate and remove items from the layout. The second (Fig. 6.13-right), composes a plan to activate and add items. The Plan behaviors are triggered by signals originated in the Analyze behavior. Then, the behavior identifies the type of the require actions (e.g. `enableLane()` identified the need to add a new lane in the layout) and plan the necessary adaptation tasks (`planEnabling()`). This process can be repeated multiple times until all the necessary adaptation actions are planned. Once planning is finished (`planned()` condition), Plan notifies the Execute behavior to execute the planned actions to the managed system.

**Figure 6.12:** Plan behavior for the mobile learning application

**Figure 6.13:** Plan behaviors for the robot transportation application

6.4.4.3 Plan Design Pattern
We consolidate the concrete plan specifications into the reusable Plan pattern as shown in Fig. 6.14. The Plan behavior may require two types of mitigation actions depending on the results of an Analyze behavior (in case of a satisfied situation, no adaptation plan is required). Either there is a need to add resources (in case the system state is `unsatisfied`) or resources may be released (when the system state is `oversatisfied`). There are cases, such as the mobile learning case, in which oversatisfied scenarios may not require adaptations.
6 MAPE-K Formal Templates

6.4 Behavior Specification Patterns for Modeling MAPE-K Feedback Loops

In the first case, (right part of the behavior pattern in Fig. 6.14), the Plan is triggered to compose a set of plan actions in order to add resources to the managed system (via the addResource() function). In some particular cases, different planning behaviors to add resources may be instantiated in order to specify particular algorithms for different cases, such as enabling lanes or adding items to the robot layout. Afterwards, the Plan behavior notifies the Execute behavior to execute the adaptation actions to the managed system (Execute_Add[Node_ID]!). In the second case, resources can be released (left part of the behavior template in Fig. 6.14). The procedure to release a resource is similar as for adding a resource and it is specified in the releaseResource() function.

In summary, we define the following steps of a Plan behavior: plan triggering, identification of the required type of plan, plan creation and signaling the related execute behavior(s).

6.4.5 Execute

The Execute behavior is responsible of executing plans of adaptation actions to the managed system.

6.4.5.1 Mobile Learning Case

We designed the Execute behavior to execute the actions defined by the Plan behavior. In this concrete scenario, only Incomplete groups required a mitigation plan. Planning includes looking for a mobile device and integrating it in the group (Mid). When triggered (addPhone[Mid]?), the Execute behavior searches for an available phone based on the criteria defined in the plan (found_Phone=getFreePhone()). In case the search was unsuccessful (found_Phone==NOPHONE), the step is repeated every 2-time-units. Otherwise, the found phone will be integrated into the group (Use[found_Phone]!). We specified the behavior to be able to cancel an execute process during the search process if required (Execute_cancel[Mid]?).

6.4.5.2 Robot Transportation Case

The Execute behavior for the robot transportation application is modeled with two automata. Fig. 6.16-top shows an automaton of the Execute behavior for enabling lanes and adding elements to the layout representation. For these cases, the behavior directly executes the adaptation plans. The Execute behavior identifies the type of adaptation actions to be performed (enable lanes, add a location and add

![Figure 6.14: Plan behavior pattern with time-triggering](image)

In the first case, (right part of the behavior pattern in Fig. 6.14), the Plan is triggered to compose a set of plan actions in order to add resources to the managed system (via the addResource() function). In some particular cases, different planning behaviors to add resources may be instantiated in order to specify particular algorithms for different cases, such as enabling lanes or adding items to the robot layout. Afterwards, the Plan behavior notifies the Execute behavior to execute the adaptation actions to the managed system (Execute_Add[Node_ID]!). In the second case, resources can be released (left part of the behavior template in Fig. 6.14). The procedure to release a resource is similar as for adding a resource and it is specified in the releaseResource() function.

In summary, we define the following steps of a Plan behavior: plan triggering, identification of the required type of plan, plan creation and signaling the related execute behavior(s).

6.4.5 Execute

The Execute behavior is responsible of executing plans of adaptation actions to the managed system.

6.4.5.1 Mobile Learning Case

We designed the Execute behavior to execute the actions defined by the Plan behavior. In this concrete scenario, only Incomplete groups required a mitigation plan. Planning includes looking for a mobile device and integrating it in the group (Mid). When triggered (addPhone[Mid]?), the Execute behavior searches for an available phone based on the criteria defined in the plan (found_Phone=getFreePhone()). In case the search was unsuccessful (found_Phone==NOPHONE), the step is repeated every 2-time-units. Otherwise, the found phone will be integrated into the group (Use[found_Phone]!). We specified the behavior to be able to cancel an execute process during the search process if required (Execute_cancel[Mid]?).

6.4.5.2 Robot Transportation Case

The Execute behavior for the robot transportation application is modeled with two automata. Fig. 6.16-top shows an automaton of the Execute behavior for enabling lanes and adding elements to the layout representation. For these cases, the behavior directly executes the adaptation plans. The Execute behavior identifies the type of adaptation actions to be performed (enable lanes, add a location and add

![Figure 6.16: Robot transportation Execute process](image)
6.4 Behavior Specification Patterns for Modeling MAPE-K Feedback Loops

Figure 6.15: Mobile learning Execute process

destination). Based on that, functions are defined (e.g. `enable_edge()`) and signals (`addEdge[Rid]!`) that allow the Execute behavior to interact with an Effector (shown in Fig. 6.17 and that specifies the functions to be applied on the managed system under the Execute request).

Figure 6.16: Robot transportation Execute process

Fig. 6.16-bottom shows the execute behavior to remove and disable elements of the layout representation. The automaton identifies the type of actions to be executed, coordinates with the managed system to prepare for the adaptation actions, and applies the adaptation actions to the managed system. In these cases, the second step is required to assure that no map elements are removed while the robot would still require them (e.g. driving on a lane that will be disabled).
source to the managed system. Preparation typically requires steps that are re-
quired to apply the adaptation actions, e.g., lock certain resources (as in the mobile
learning scenario), ensure that the managed system is in a safe (quiescent) state,
etc. Such preparation may require multiple preparation steps.

Once preparation is completed, the behavior moves to the
DoActionsAdd
state, where
actions such as integrate a new phone, add a location and activate a lane are per-
formed, via the (invokeNextAction()) function and the Effector_Add[Node_ID]! sig-
nal.

Adding resources may require
PostExecutionAdd
tasks that are dealt with by postAction(). Examples are unlocking resources of the managed system and up-
dating internal information in the knowledge repositories. Finally, the Execute be-
havior checks whether the plan is completed (actionsPending()). If more actions
are Pending, the behavior returns to the PrepareAdd state from where it repeats the
process to execute the next actions to the managed system.

6.5 Property Specification Patterns for Verifying
MAPE-K Feedback Loops

Timed computational tree logic (TCTL) allows designers to specify properties
of the MAPE-K behaviors modeled in TA and verify whether the models comply to
the properties. TCTL allows defining path formulate that a behavior may or may
not take. Path formulae quantify over paths of the state space of the model and can
in general be classified into reachability, safety, and liveness properties.

During the design of the different self-adaptive systems we developed, we de-
defined a variety of properties. Below, we illustrate some of the properties we speci-
fied for the mobile learning and the robot transportation scenarios.

In order to verify properties of the design of a self-adaptive system, it is impor-
 tant to note that the behavioral models of the MAPE-K feedback loop need to be
integrated with models of the managed system and its environment. Section 6.3
provides some excerpts of the models for the mobile learning and the robot trans-
portation scenarios; however, we do not further elaborate on the behaviors of the
managed system and the environment as they are domain-specific and out of scope
of this paper.

6.4.5.3 Execute Design Pattern

Now we present the Execute patterns that consolidates our expertise of designing
Execute behaviors for different self-adaptive systems. We identified the need of
two branches in the Execute behavior: one branch that deals with adding resources
to the managed system; the other branch, to release resources. Fig. 6.18 shows the
"add branch" of the Execute behavior pattern. The "remove branch" follows an
equivalent logic.

We identified three phases in an Execute behavior: preparation, execution
(doAction) and post-execution. First, the Execute behavior is triggered to take
action (Execute_Add[Node_ID]?). Execute checks whether the managed system
is ready to invoke the planned actions (isSystemReady(Prepare)), as we observed
with the waitRequired() in the robot case. If this is the case, Execute performs
preparation tasks (PrepareAdd state) before executing an action for adding a re-
source.
source to the managed system. Preparation typically requires steps that are required to apply the adaptation actions, e.g., lock certain resources (as in the mobile learning scenario), ensure that the managed system is in a safe (quiescent) state, etc. Such preparation may require multiple preparation steps.

Once preparation is completed, the behavior moves to the DoActionsAdd, where actions such as integrate a new phone, add a location and activate a lane are performed, via the (invokeNextAction) function and the Effector_Add[Node_ID]! signal.

Adding resources may require PostExecutionAdd tasks that are dealt with by postAction(). Examples are unlocking resources of the managed system and updating internal information in the knowledge repositories. Finally, the Execute behavior checks whether the plan is completed (actionsPending()). If more actions are Pending, the behavior returns to the PrepareAdd state from where it repeats the process to execute the next actions to the managed system.

6.5 Property Specification Patterns for Verifying MAPE-K Feedback Loops

Timed computational tree logic (TCTL) allows designers to specify properties of the MAPE-K behaviors modeled in TA and verify whether the models comply to the properties. TCTL allows defining path formulate that a behavior may or may not take. Path formulae quantify over paths of the state space of the model and can in general be classified into reachability, safety, and liveness properties.

During the design of the different self-adaptive systems we developed, we defined a variety of properties. Below, we illustrate some of the properties we specified for the mobile learning and the robot transportation scenarios.

In order to verify properties of the design of a self-adaptive system, it is important to note that the behavioral models of the MAPE-K feedback loop need to be integrated with models of the managed system and its environment. Section 6.3 provides some excerpts of the models for the mobile learning and the robot transportation scenarios; however, we do not further elaborate on the behaviors of the managed system and the environment as they are domain-specific and out of scope of this paper.

6.5.0.4 Mobile Learning Case

We discuss three properties that we specified for verifying the correctness of the self-adaptive behavior of the mobile learning scenario.

G1: SAAnalyze(X).Incomplete --> SAAnalyze(X).Complete || SAAnalyze(X).Redundant
G2: SAAnalyze(X).Incomplete --> SAExecute(X).LookForFreeGPS

Property G1 states that if an Analyze behavior detects that a group is incomplete and needs a GPS service to cover the current requirements (state Incomplete), eventually the MAPE behaviors will solve the problem by adding a GPS service.
to the group (state Complete or Redundant) (the latter state may result for example when a GPS device recovered during the adaptation process). This property focuses on verifying that the adaptation goals for the mobile learning system are achieved.

We specified property G2 to verify that whenever the Analyze detects that a group is incomplete, the Execute behavior will eventually start looking for a GPS service (LookForFreeGPS). With this property, we focus on guaranteeing the correct coordination and interactions between components in the MAPE loop.

Finally, property G3 allows verifying the required behavior of an internal MAPE component. In this particular case, we want to provide guarantees that when an Analyzer finds that a group is complete, but one of the used resources (ManagedSystemKnowledge[X].member[Y]==USED, being Y the resource) is down (EnvironmentKnowledge[X].GPS[Y].state = DOWN), eventually the Analyzer will detect this failure and mark the Incomplete state. This, in turn, will trigger Plan to look for a replacement of the GPS service.

6.5.0.5 Robot Traffic Case

For the robot traffic system, we present a subset of the properties we specified and verified and describe the rationale behind them.

\[ \text{R1}: \text{Analyze}(X).\text{ElementsMissing} \rightarrow \text{PlanEnableOrAdd}(X).\text{EnableOrAdd} \]
\[ \text{R2}: \text{Monitor}(X).\text{RequestToDisableLane} \land \text{disabledLane}[X] = \text{Lane}_Z \rightarrow \text{Execute}(X).\text{DisableLane} \land \text{disabledLane}[X] = \text{Lane}_Z \]

Property R1 states that when the Analyze detects that map elements are missing in the robot knowledge, the Plan will eventually add (PlanEnableOrAdd state in the Plan behavior) the missing parts of the map layout. This property focuses on guaranteeing correct interactions between Analyze and Plan behaviors, as a critical aspect for the correct self-adaptive behavior.

With property R2, we aimed to verify a domain specific property that focuses on the correctness of the self-adaptation goals. Concretely, the property states that when the Monitor behavior receives a request to disable a lane Z, the particular lane will eventually be disabled by the Execute process\(^1\). This property guarantees the correct behavior of the MAPE behaviors involved in disabling lanes.

6.5.1 Property Specification Patterns

Now we present the consolidation of the property specifications for MAPE-K feedback loops of the different self-adaptive systems we have developed. The property specification patterns can be classified in three groups (see Fig. 6.19) based on the scope of the property that is being defined. Group 1, Adaptation Goal Specification Patterns, embraces properties with a scope that focuses on the realization of the adaptation goal by the MAPE-K feedback loop. Groups 2, Intra-Behavior Specification Patterns, specifies properties of internal MAPE component behaviors. Group 3, Inter-Behavior Specification Patterns, focuses on properties that specify the interaction between multiple behaviors of a MAPE-K loop.

\(^1\)\text{disabledLane}[X] \text{is a global variable used to identify one (or more) specific lane(s) to be disabled}
6.5 Property Specification Patterns for Verifying MAPE-K Feedback Loops

Figure 6.19: Groups of property specification patterns

We provide a representative set of property specification patterns of each group. The property specification patterns refer to state and path formulate based on the previously presented behavior specification patterns. For additional properties, we refer the interested reader to the project website.

6.5.1.1 Adaptation Goal Specification Patterns

The adaptation goal specification patterns focus on providing evidence for assurances regarding self-adaptation actions. Therefore, they allow verifying that when a self-adaptive system requires adaptation, eventually the adaptation will be carried out. Examples of this type of patterns are G1 and R2 in the illustrative scenarios.

P1: Analyze(X).Unsatisfied \rightarrow Analyze(X).Satisfied || Analyze(X).Oversatisfied

P1 defines when an Analyze behavior (Fig. 6.11) at node X detects that the managed system is in an unsatisfied state, the MAPE behaviors will eventually adapt the managed system bringing it in a satisfied or oversatisfied state. Property P1 is based on the assumption that eventually resources will become available to realize the adaptation goal.

6.5.1.2 Intra-Behavior Specification Patterns

Intra-behavior specification patterns allow specification and verification of properties of individual MAPE behaviors, independently of the other MAPE behaviors.

P2: ConcernKnowledge[X].required_resources > ManagedSystemKnowledge[X].used_resources \rightarrow Analyze(X).Unsatisfied
P3: Plan(X).AddResource \rightarrow Plan(X).Satisfied
P4: Plan(X).ReleaseResource \rightarrow Plan(X).Satisfied
P5: Execute(X).DoActionAdd \rightarrow Execute(X).PlanCompleted
P6: Execute(X).DoActionRelease \rightarrow Execute(X).PlanCompleted

We defined P2 as an abstraction of properties such as G3. P2 defines that when resources used by the managed system are insufficient according the requirements defined in the Knowledge, the Analyze behavior will eventually detect this unsatisfied situation. The greater than symbol is an abstract operator that allows checking whether the used resources satisfy the required resources; this operator needs to be
6 MAPE-K Formal Templates

 instantiated for the domain at hand. Intra-behavior property verification is particularly relevant for complex behaviors. E.g., in complex adaptation scenarios, it is important to verify that the Plan creates adequate plans that will adapt the managed as required. P3 and P4 are two property specification patterns for a Plan behavior (Fig. 6.14) that allow to verify whether adding (P3) and releasing a resource (P4) from the managed system leads to a Satisfied state. These properties aim at specifying that the behavior satisfies liveness when planning adaptation actions. Similarly, P5 and P6 specify patterns for an Execute behavior (Fig. 6.18) that allow to verify whether the plan for adding a resource to the managed system (P5) and releasing a resource from the managed system (P6) are properly modeled.

6.5.1.3 Inter-Behavior Specification Patterns

 We identified several patterns that focus on providing guarantees about the interactions between components of the MAPE loop; these patterns are generalizations from properties such as R1 and G2. We consolidate these properties into a set of property specification patterns that can be applied to MAPE-K systems.

\[
P7: \text{Analyze}(X).\text{Unsatisfied} \rightarrow \text{Plan}(X).\text{AddResource}
\]

\[
P8: \text{Plan}(X).\text{ReleasePlanReady} \rightarrow \text{Execute}(X).\text{DoActionRelease}
\]

\[
P9: \text{Plan}(X).\text{AddPlanReady} \rightarrow \text{Execute}(X).\text{DoActionAdd}
\]

P7 defines a property to verify the correct collaboration between the Analyze (Fig. 6.11) and Plan (Fig. 6.14) behaviors. The property states that when Analyze detects an unsatisfactory state of the managed system, eventually the Plan behavior will create a plan to add a new resource; a concrete instance of this pattern is R1. P8 and P9 describe property patterns to verify the correct interaction between Plan and Execute (Fig. 6.18) behaviors. P8 specifies that when Plan has generated a plan to release a resource, this plan is eventually executed by the Execute behavior. P9 specifies the correct execution of a plan to add a resource.

6.6 Assessment of the MAPE-K Formal Templates

To demonstrate the reusability of the MAPE-K Formal Templates for new applications in the target domain, we performed four case studies in the context of a course in a Software Engineering Master program at Linnaeus University. A case study allows to understand "a real-world phenomena (the 'case') in depth and within its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident" [194]. Through multiple case studies, it is possible to obtain qualitative evidence [1, 135, 194] by exploiting multiple data sources, such as interviews, documents, observations, recordings, among others [194]. Therefore, running multiple case studies offered us the means to provide qualitative evidence for the reuse of the MAPE-K Formal Templates, by analyzing the level of reuse of the templates (phenomena) without controlling or manipulating the participants [194]. In this section, we describe the settings and results from the case studies. The complete materials of the study are available at the project website [79], including the protocol for the study, course materials, session recordings, and the detailed results from the study.

104
6.6 Assessment of the MAPE-K Formal Templates

6.6.1 Study Settings

The study was performed in the context of the 4DV108: Advanced Software Design Master course during a period of 6 weeks. The course included different phases on the design of a self-adaptive system through the application of MAPE-K loops using formal methods.

We carried out four case studies. Each case study comprises a participant that designs and implements a self-adaptive system from a freely selected application domain that fits the characteristics of the target domain of our research. The selected application domains were smart homes, security system, and vehicular traffic systems. These applications comply with the characteristics of the target domain presented in Section 6.3. Each case study is treated as a holistic unit with no independent embedded units [194]. Thus, we take an holistic study approach with multiple-cases in order to study the data that is produced during the whole design and development process. Fig. 6.20 shows an overview of the course with the design of the study.

![Overview of 6-weeks course in which the study took place](image)

**Figure 6.20:** Overview of 6-weeks course in which the study took place

The study included the following tasks that the participants had to perform:

1. Select an application (managed system). The participants were not limited in the selection of the application domain (e.g., robotics, logistics, aeronautics, etc.). The only restriction was that the application should comply to the characteristics of the target domain (described in Sec. 6.3)

2. Design and specify the managed system behavior and required domain properties, and verify the properties.

3. Select a self-adaptive requirement for the managed system that focused on robustness or openness. The participants were not restricted in the specifics of the self-adaptation goal.
6 MAPE-K Formal Templates

4. Design and specify a self-adaptive solution for such managed system (design and required properties), and verify the correctness of properties of self-adaptive system.

5. Participate in semi-structured interviews.

In Week 1, the participants were introduced TA and TCTL and UPPAAL, the tool used for designing and verifying the formal models. In Task 1, the participants proposed cases for distributed systems that fit with the characteristics of the target domain, and in Task 2 the students had to design the system and verify key properties.

In Week 4, the participants were introduced to the concept of self-adaptive systems, the MAPE-K reference model and the MAPE-K Formal Templates. For Task 3, the participants were requested to select two concerns for self-adaptation that apply to their designed managed systems. We collected the managed systems’ designs, in order to study the participants progression through learning the use of formal languages and their domain with UPPAAL tool. Then, in Task 4, the participants were requested to specify the behavior for one MAPE-K loop and to verify self-adaptation properties. The students had three weeks to complete this phase. During this period, the students were followed up closely by the course holders.

At the end of the study, we carried out individual semi-structured interviews (Task 5) of around 30 minutes. The goal of the interview was to collect data regarding the participant experiences during the application design and the use of the templates in their domains.

6.6.1.1 Research Questions

The aim of the study was to answer a set of research questions (RQ) that address reusability from different viewpoints.

**RQ1** "Can the MAPE-K Templates be applied to applications that share the characteristics of the target domain?"

**RQ2** "Do the Behavior Specification Patterns allow the behavior specification of MAPE-K components for the specific self-adaptive systems, and if so, how?"

**RQ3** "Do the Property Specification Patterns allow the property specification for MAPE-K loops for the specific self-adaptive systems, and if so, how?"

With RQ1, we wanted to provide initial evidence that the templates can be applied in additional application domains (under the umbrella of our target domain) and applications that were not used as a basis from which the MAPE-K Formal Templates were derived. With RQ2, we wanted to analyze the reusability of the MAPE-K templates and to investigate the types of reuse of the behavior patterns during the study. Similarly, with RQ3 we wanted to analyze the reuse regarding MAPE-K property patterns.

6.6.2 Analysis of the Results

Now, we present the results we obtained through the analysis of the data collected in the study. We studied the participants’ models in order to identify the use of different parts of the behavior patterns (states, transitions, functions, data structures)
and property patterns in the different application domains. Finally, we used the interviews to cross-check the results from the analysis process and to extend them with first-hand experiences and feedback from the participants.

We present the results following the three research questions. First we provide results regarding the reusability of the MAPE-K Formal Templates in different application domains. Second, we provide results w.r.t. reuse of the Behavior specification patterns and we conclude with results for reuse of the Property specification patterns.

6.6.2.1 Application Domains

During the study, self-adaptive systems from multiple application domains were designed, including one case of smart homes system, one security system and two vehicular traffic systems.

We select two cases to illustrate the results from the case studies.

In Case-1, the system is a smart home application that controls the heating system. The controller can sense the temperature conditions outside the house, and process this information in order to adapt the heating system when needed. The controller was designed to work with one temperature sensor and one heating system regulator. The controller is programmed to connect with the temperature sensor and communicate with it using a wireless channel. Due to the managed system design, when a second temperature sensor is added into the system, the controller would read data only from the last introduced sensor. In order to avoid biased data from a single temperature sensor, the home owner wanted the system to take into account temperature measures from multiple sensors around the house. To that end, a self-adaptation solution was applied to the managed system that adds openness to the system, that is, allow the system to handle multiple temperature sensors.

Case-2 is a vehicular traffic application that controls traffic-lights that handle potential obstructions on the road; the obstructions can be sensed through traffic cameras. The quality of the cameras that identify obstacles in roads depend on the scope of their view. Cameras have a higher precision with a narrow scope covering only one road. Therefore, two traffic cameras were placed on each crossing, in order to observe the traffic conditions from west and south sides providing high degree of accuracy. When obstructions were found, the system was designed to notify drivers to take an alternative way ("turn right") using the traffic-lights. The goal of the self-adaptation in this case was to maintain a complete coverage of the roads when a camera failed by modifying the remaining camera settings in order to cover west and south ways, with a penalization in the accuracy. This case specified a managing system that deal with system robustness to failing cameras.

We can answer research question RQ1, Can the MAPE-K Templates be applied to applications that share the characteristics of the target domain?, positively as the templates were applied to four different applications that fit the target domain, including smart homes, security systems and vehicular traffic systems. Below we provide a deeper analysis on the levels of reuse of MAPE-K Formal Templates that the participants realized within the cases.
6 MAPE-K Formal Templates

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Instances</th>
<th>Triggering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probe</td>
<td>8</td>
<td>E</td>
</tr>
<tr>
<td>Monitor</td>
<td>6</td>
<td>E &amp; T</td>
</tr>
<tr>
<td>Analyze</td>
<td>4</td>
<td>E &amp; T</td>
</tr>
<tr>
<td>Plan</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>Execute</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>Effector</td>
<td>4</td>
<td>E</td>
</tr>
</tbody>
</table>

Table 6.1: Behavior Specification Pattern selection

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Total States</th>
<th>Selected (%)</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>30</td>
<td>15 (50%)</td>
<td>2</td>
</tr>
<tr>
<td>Analyze</td>
<td>20</td>
<td>15 (75%)</td>
<td>4</td>
</tr>
<tr>
<td>Plan</td>
<td>20</td>
<td>9 (45%)</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>56</td>
<td>11 (19.64%)</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>126</td>
<td>50 (39.68%)</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 6.2: State selection per MAPE Behavior

### 6.6.2.2 Reuse Behavior Specification Patterns

The participants specified automata to design Monitor, Analyze, Plan and Execute behaviors. In all the cases studied, the participants instantiated distinct behaviors for Probe, Monitor, Analyze, Plan, Execute and Effector behaviors. In all the cases, one automaton was specified for Analyze, Plan, Execute and Effector behaviors. For Monitor behaviors, more than one automaton in average were designed. Self-adaptive system usually require multiple data sources to monitor the managed system and its environment. We observed that multiple instances of Probe behaviors were used to collect data (we found an average of 2 probes per case). The data is summarized in Table 6.1.

We observed that event-triggered behaviors (E in Table 6.1) were selected in all the cases, and complemented in some particular cases with time-triggering mechanisms (T in Table 6.1) in order to specify time-out behaviors (e.g. Monitor and Analyze behaviors for Case-2). This is the case shown in Fig. 6.21, where the behavior implements a time-out function using a time-triggering approach \((b==3)\). We will focus later on some specifics of this behavior.

![Diagram of Analyze behavior for a vehicular traffic system](image)

**Figure 6.21:** Analyze behavior for a vehicular traffic system

The selection of event-triggering was explained as a particular requirement for the application domains. However, the complexity in modeling time-triggered behaviors was another factor for the selection. One of the participants stated: "Event-trigger is [the approach] I used the most. If I want to perform some actions, it is easy for me to send a signal. But time-triggering is hard to control."

Regarding the reuse of the behavior specifications, one participant stated: "I used all the templates, but I simplified them. I took away some states that were not necessary for the behavior, [but still] maintained the same logic". Reduction
of the templates as a refinement approach was common to all the cases. All the participants acknowledged having based the design of their behaviors on the Behavior Specification Patterns. Generally, the self-adaptation solutions that were required for the application domains did not require the levels of complexity that the Behavior Specification Patterns allow. Therefore, the participants selected part of the states in the patterns. This is particularly clear in the Plan and Execute behaviors, where only add branches were used, and no preparation and postactions states were needed. However, the participants confirm having used the Execute pattern for inspiration in their design. Table 6.2 shows the level of reuse of states in the patterns. For example, from the potential 30 states in Monitor specifications (5 states/Monitor x 6 Monitor behavior specified), we count a total of 15 reused states, and 2 new states required for the behavior specification. For numbers regarding Plan and Execute specifications, only "add branches" have been considered.

We illustrate behavior specification pattern reuse and refinement with two examples. Fig. 6.22-left shows the Plan behavior for the vehicular traffic application. In this case, the participant refined and reduced the Plan behavior in order to cover the addition of a resource in the system (UseCameraSouth) when the west camera had failed (Plan_Unsatisfied[id]?). In Fig. 6.22-right, we observe a refinement of the Execute behavior for the smart house application. In this particular case, the Execute behavior has been specified to calculate the average of the multiple temperature sensors via calculateAV() (which refines the invokeNextAction() function) and feed the controller of the heating system in order to work with the updated values (signal updateValue! to synchronize with the Effector behavior).

Figure 6.22: Plan behavior for a vehicular traffic system system (left). Execute behavior for a smart house system (right)

In a small number of cases, the participants extended the behaviors with additional states. We found two reasons for these behavior extensions. The first, the automaton has included intermediate states in order to refine parts of the existing behavior. Fig. 6.23 shows an instance of state extension. In the Analyze behavior for the Case-1, the Unsatisfied state has been extended into two substates (PD and PS) in order to differentiate different types of undersatisfaction (when temperature sensors have different and equal values).

The second reason we found for adding states is the need for intermediate states in the behavior. This is the case presented in the Analyze behavior in Fig. 6.21. In this case, the role of the matchResources() function has been specified through an expected sequence of signals. When a camera is requested to provide the traffic...
6.6 Assessment of the MAPE-K Formal Templates

We observed that reuse of behavior patterns was achieved via refinement, reduction, and the use of multiple instances of patterns. We observed different types of refinement, including refinement of states, transitions, functions (specification and logic) and data structures.

6.6.2.3 Reuse Property Specification Patterns

The participants used property specification patterns of the three groups to verify properties of adaptation goals, intra-behavior relations and inter-behavior relations. In total, 16 properties were specified in the different case studies to verify the self-adaptive behaviors (see Tables 6.4, 6.5 and 6.6).

Involved Behavior Instances

<table>
<thead>
<tr>
<th>Involved Behavior Instances</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>0</td>
</tr>
<tr>
<td>Analyze</td>
<td>3</td>
</tr>
<tr>
<td>Plan</td>
<td>2</td>
</tr>
<tr>
<td>Execute</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 6.4: Adaptation Goal

Property selection

<table>
<thead>
<tr>
<th>Scope Instances</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor-Analyze</td>
<td>1</td>
</tr>
<tr>
<td>Analyze-Plan</td>
<td>1</td>
</tr>
<tr>
<td>Plan-Execute</td>
<td>3</td>
</tr>
<tr>
<td>Execute-Effector</td>
<td>1</td>
</tr>
<tr>
<td>Plan-Effector</td>
<td>1</td>
</tr>
<tr>
<td>TOTAL</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 6.5: Intra-behavior Property selection

Three properties were used to verify the overall adaptation goals. One of these properties was specified using the Analyze behavior only; the two others were specified using the Analyze and Plan behaviors. Six properties were used to verify the correctness of internal MAPE component behaviors. We observed particular attention for verification of the correctness of the Analyze behavior (five cases; the sixth case related to the correctness of the Plan behavior). Finally, seven properties were used to verify the interaction between MAPE behaviors. Five of these properties involved the Plan behavior. In summary, we observe that the Analyze and Plan behaviors are considered as the most critical behaviors for self-adaptation by the participants. We illustrate each group of property specification patterns with concrete instances.

The following properties are instances of Adaptation Goal Specification pattern P1:

\[
P1: \text{Analyze}(X).\text{Unsatisfied} \rightarrow \text{Analyze}(X).\text{Satisfied} \lor \text{Analyze}(X).\text{Oversatisfied}
\]

\[
\text{Pr1: Plan}(1).\text{UseCameraSouth} \rightarrow \text{Analyze}(1).\text{ObstacleCameraWorking}
\]

This leads us to the analysis of function reuse in the behavior patterns. We observed that only 20 from a total of 76 functions in the behavior patterns were used (Table 6.3) for the specification of the MAPE behaviors. This selection aligns with the reduction of states (and transitions) used for specifying the behaviors. The selected functions were refined in order to specify the domain-specific logic of the self-adaptation. Commonly, refinement was done via the code of the functions. However, we identified one particular case in which a function was replaced by a specified sequence of expected signals (time-out example in Fig. 6.21). We did not identify cases in which new functions were introduced to model MAPE-K behaviors.

The automata and the language primitives used to define functions and data structures can be considered as simplified C-like code. One of the interviewees indicated that the code used for the behavior specification may be highly reusable for the actual software implementation. However, another participant pointed to difficulties to transfer the design of signals and channels to software code. This confirms common sense that timed automata to formalize the MAPE-K behaviors have limitations in terms of the expressiveness, and some behavior abstractions (such as communication among distributed nodes) may require more complex implementations.

Based on the results of our analysis, we can answer the second research question RQ2 “Do the Behavior Specification Patterns allow the behavior specification of MAPE-K components for the specific self-adaptive systems, and if so, how?” pos-
itively. We observed that reuse of behavior patterns was achieved via refinement, reduction, and the use of multiple instances of patterns. We observed different types of refinement, including refinement of states, transitions, functions (specification and logic) and data structures.

6.6.2.3 Reuse Property Specification Patterns

The participants used property specification patterns of the three groups to verify properties of adaptation goals, intra-behavior relations and inter-behavior relations. In total, 16 properties were specified in the different case studies to verify the self-adaptive behaviors (see Tables 6.4, 6.5 and 6.6).

<table>
<thead>
<tr>
<th>Involved Behavior</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>0</td>
</tr>
<tr>
<td>Analyze</td>
<td>3</td>
</tr>
<tr>
<td>Plan</td>
<td>2</td>
</tr>
<tr>
<td>Execute</td>
<td>0</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>3</strong></td>
</tr>
</tbody>
</table>

Table 6.4: Adaptation Goal Property selection

<table>
<thead>
<tr>
<th>Scope</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor-Analyze</td>
<td>1</td>
</tr>
<tr>
<td>Analyze-Plan</td>
<td>1</td>
</tr>
<tr>
<td>Plan-Execute</td>
<td>3</td>
</tr>
<tr>
<td>Execute-Effector</td>
<td>1</td>
</tr>
<tr>
<td>Plan-Effector</td>
<td>1</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>7</strong></td>
</tr>
</tbody>
</table>

Table 6.5: Intra-behavior Property selection

<table>
<thead>
<tr>
<th>Scope</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>0</td>
</tr>
<tr>
<td>Analyze</td>
<td>5</td>
</tr>
<tr>
<td>Plan</td>
<td>1</td>
</tr>
<tr>
<td>Execute</td>
<td>0</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>6</strong></td>
</tr>
</tbody>
</table>

Table 6.6: Inter-behavior Property selection

Three properties were used to verify the overall adaptation goals. One of these properties was specified using the Analyze behavior only; the two others were specified using the Analyze and Plan behaviors. Six properties were used to verify the correctness of internal MAPE component behaviors. We observed particular attention for verification of the correctness of the Analyze behavior (five cases; the sixth case related to the correctness of the Plan behavior). Finally, seven properties were used to verify the interaction between MAPE behaviors. Five of these properties involved the Plan behavior. In summary, we observe that the Analyze and Plan behaviors are considered as the most critical behaviors for self-adaptation by the participants. We illustrate each group of property specification patterns with concrete instances.

The following properties are instances of Adaptation Goal Specification pattern P1:

P1: Analyze(X).Unsatisfied --> Analyze(X).Satisfied || Analyze(X).Oversatisfied

Pr1: Plan(1).UseCameraSouth --> Analyze(1).ObstacleCameraWorking
6 MAPE-K Formal Templates

\[
\text{Pr2: } \text{AlarmAnalyze}(0).\text{NotWorked} \rightarrow \text{AlarmAnalyze}(0).\text{Worked}
\]

\[
\text{Pr3: } \text{Room}.\text{SmokeIncreased} \text{ and } \text{Alarm}(0).\text{Broken} \rightarrow \text{Alarm}(1).\text{Red\_loud} \text{ and } \text{Alarm}(2).\text{Red\_loud}
\]

*Pr1* specifies a property for a vehicular traffic system that allows verifying that when the West camera failed and the planner plans to use the South camera to detect obstacles (Plan(1).UseCameraSouth) the Analyzer will eventually reach a Satisfied state (as a result of the adaptation process), that is, the obstacle camera is working properly (Analyze(1).ObstacleCameraWorking). *Pr2* and *Pr3* belong to a third case study with that focuses on a security system for fire detection. Property *Pr2* specifies a property for a fire detection system that allows verifying that when the Alarm Analyzer detects that the detection system is not working properly (AlarmAnalyze(0).NotWorked), this problem will eventually be resolved (AlarmAnalyze(0).Worked). Property *Pr3* allows verifying that when increasing smoke is detected and an alarm in one of room is broken, the alarms in the other rooms are directly put in highest alarm (Alarm(x).Red\_loud).

The following properties are instances of Intra-Behavior Specification pattern P2:

\[
\text{P2: } \text{ConcernKnowledge}[X].\text{required\_resources} > \text{ManagedSystemKnowledge}[X].\text{used\_resources} \rightarrow \text{Analyze}(X).\text{Unsatisfied}
\]

\[
\text{Pr4: } A[] (\text{Analyzer}.\text{NoNeedSA} \text{ imply } \text{numberOfSensors} == 1)
\]

\[
\text{Pr5: } A[] (\text{Analyzer}.\text{NeedSA} \text{ imply } \text{numberOfSensors} > 1)
\]

*Pr4* and *Pr5* specify properties for a smart home temperature control system that allow verifying that the Analyzer correctly detects the need for self-adaptation. Concretely, if only a single temperature sensor is registered (numberOfSensors == 1), there is no need for adaptation (Analyzer.NoNeedSA), but if more sensors are registered (numberOfSensors > 1) adaptation is required (Analyzer.NeedSA).

The following properties are instances of an Inter-Behavior Specification pattern, in particular pattern P9:

\[
\text{P9: } \text{Plan}(X).\text{AddPlanReady} \rightarrow \text{Execute}(X).\text{DoActionAdd}
\]

\[
\text{Pr6: } \text{Planner}.\text{PlanReady} \rightarrow \text{Executor}.\text{CalculateAverage}
\]

\[
\text{Pr7: } \text{Execute}(1).\text{InformCameraSouth} \rightarrow \text{Effector}(1).\text{ChangeCameraAngle}
\]

*Pr6* specifies specifies a property for a temperature control system in a smart home that allows verifying that when the planner is ready to incorporate a new sensor (Planner.PlanReady) the executor will eventually calculate the average of the sensed values of the sensors (Executor.CalculateAverage) and adapt the managed system accordingly (see Fig. 6.22-right).

*Pr7* specifies a property that involves the Execute and Effector behavior for a vehicular traffic system. The property allows verifying that when the alternative camera south is activated to extend its monitoring (Execute(1).InformCameraSouth), the effector will expand the scope-view of the camera (Effector(1).ChangeCameraAngle) to cover west traffic (in addition to the south traffic covered in normal conditions).
The case studies demonstrate the reusability of the Property Specification Patterns. During the interviews, the participants acknowledged that they directly used or based their property specifications on the Property Specification Patterns. The patterns assisted the participants to identify properties that were suitable and relevant to verify the correct behavior of their self-adaptive system.

Based on the results of our analysis, we can answer the third research question RQ3 "Do the Property Specification Patterns allow the property specification for MAPE-K loops for the specific self-adaptive systems, and if so, how?" positively. The Property Specification Patterns provide guidance to the designer to identify concrete properties that should hold for the self-adaptive system at hand. The participants stated that the proposed set of patterns covered their needs and the property patterns simplify the verification tasks. One of the participants stated: "The properties [aka. property specification patterns] saved time". The participants used property specification patterns of the three groups to verify the correctness of their self-adaptive systems. The selection was not equally distributed; Inter- and Intra-behavior properties were more frequently selected than Adaptation Goal Specification properties. This observation underpins that designers verify not only the realization of the self-adaptation goals, but put significant emphasis on the correctness of MAPE behaviors and their interactions.

6.6.2.4 Reflection on the Study Results

We run four case studies in order to collect qualitative evidence regarding the reusability of the MAPE-K Formal Templates. The successful application of the MAPE-K Formal Templates to different applications that match the target domain demonstrates the reusability of the templates.

The participants commonly supported the benefits that the templates provide to reason about the managing system. One of the participants stated it as follows: "It makes the process clear. I know what the components must do. It also helped to understand the process expected from managing systems." Additionally, we observed strong indications that the use of the templates has a positive effect on the required efforts and time to design the managing systems. The specification of distinct MAPE behaviors allows easier comprehension of the component roles. During the interviews one of the participants stated: "The way [the MAPE-K Formal Templates] are written, they almost work for every type of system. They need some small customization. The whole idea was clear in front of me and it saved a lot of time. [Also] it help understanding how the system would work".

The participants mentioned their concerns to transfer formal specifications into code implementation. Aspects such as communication between components are highly abstracted in the Behavior Specification Patterns, which allows powerful reusability, but are more difficult to transfer into code.

In general, the participants perceived the use of the MAPE-K Formal Templates for the design of a self-adaptive system as a satisfactory experience. One of the participants phrased it: "[I would use the templates again] if I want to design a self-adaptive system, because it makes things easier just to follow the process.". Although more systematic inquires are required to get additional evidence for the MAPE-K Formal Templates, the study results provide initial evidence that defin-
6.7 Conclusions

A recent SLR [188] identified a lack of consolidation on the formal design of self-adaptive systems. Additionally, very few studies use formalizations to specify and verify self-adaptation, which hampers the maturity of the field. The MAPE-K Formal Templates presented in this paper consolidate our research efforts on the use of formal methods for the design and verification of a family of self-adaptive systems that define our target domain. The templates provide reusable design knowledge, which can be considered as a domain specific language that allows rigorous modeling and verification of the behaviors of MAPE-based feedback loops for a target domain. The target domain are distributed applications in which self-adaptation is used for managing resources for robustness and openness requirements via adding and removing resources in the system.

When defining the behavior and property specification patterns we had to balance between generality and usability. More specific templates are potentially more expressive and provide designers more fine grained elements for modeling and verification. However, more specific templates introduce more complexity which may hamper the usability of the patterns. It may also limit the applicability of the patterns in terms of the target domain. To find the right balance we derived the patterns from different applications and validated the results to new scenarios.

Following the essentials of the MAPE-K reference model, the behavior patterns support the design of the MAPE functions as distinct behaviors. The main benefits of designing the different adaptation functions in an independent manner are: flexible support for fine grained verification, and increased understandability and reduced complexity; the latter was demonstrated in a recent empirical experiment [185]. The case studies that we reported in this paper support these claims, we identified a reduction in efforts required for the design of the managing system, both in terms of time to understand the MAPE behaviors and their design for concrete applications.
6.7 Conclusions

However, our experiences have also identified some limitations in terms of expressiveness of modeling. Timed automata provide an intuitive modeling language, but, the language primitives primarily support modeling of behavioral aspects of designs. Furthermore, aspects such as messaging and communication protocols, which are commonly used in distributed systems, can be difficult to model with the language primitives of timed-automata. Therefore, in some cases it may become challenging to model the behavior of a distributed self-adaptive system.

In our future efforts, we consider the application of the MAPE-K Formal Templates to more complex self-adaptive systems. This will allow us to further test the appropriateness of both the behavior and property specification patterns, and moreover their applicability to larger-scale applications. We plan to focus on the knowledge part of the MAPE-K loop and define reusable model templates for particular sub-domains. In this effort, we plan to incorporate support for modeling and verifying probabilistic properties, which are essential to model uncertainties in self-adaptive systems. Finally, we also plan to study interacting MAPE-K loop which are essential to deal with multiple concerns and self-adaptation in decentralized settings.
Chapter 7

Conclusions

Engineering contemporary software systems and guaranteeing the required system goals is complex due to the dynamic conditions under which these systems have to operate. Self-adaptation is a well-recognized approach to deal with the complexity and dynamics of software systems. A current challenge in self-adaptive systems is to provide evidence that self-adaptation guarantees functional and quality requirements. State of the art advocates the use of formal methods as one prominent approach to rigorously specify and verify the behavior of self-adaptive systems. However, there is a lack of research on consolidating design knowledge for self-adaptive systems, which hampers reuse of expertise.

In this thesis, we present a set of formally specified templates to support engineers with designing MAPE-K-based self-adaptive systems. The templates consolidate design expertise for a family of self-adaptive systems. The target domain that we have studied represents self-adaptive systems with the following characteristics:

• Systems comprise software deployed on distributed nodes;
• The nodes have an explicit position in the environment (and may be mobile);
• The nodes have continual communication access;
• Dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software;
• Resources in the system may come and go.

In the last chapter of this dissertation, we draw conclusions from our research. We reflect on lessons learned from our experience with the formalization of self-adaptive behaviors. We outline suggestions for further research building upon the research presented in this thesis. We conclude with a closing reflection on our work.

7.1 Contributions

In this thesis, we presented a formal approach for the design and verification of distributed self-adaptive systems. Whereas a vast body of work has focused on the structural aspects of self-adaptive systems, our focus is on the behavioral aspects for such systems.

The research contributes to the state of the art with the following contributions:

Chapter 7

Conclusions

Engineering contemporary software systems and guaranteeing the required system goals is complex due to the dynamic conditions under which these systems have to operate. Self-adaptation is a well-recognized approach to deal with the complexity and dynamics of software systems. A current challenge in self-adaptive systems is to provide evidence that self-adaptation guarantees functional and quality requirements. State of the art advocates the use of formal methods as one prominent approach to rigorously specify and verify the behavior of self-adaptive systems. However, there is a lack of research on consolidating design knowledge for self-adaptive systems, which hampers reuse of expertise.

In this thesis, we present a set of formally specified templates to support engineers with designing MAPE-K-based self-adaptive systems. The templates consolidate design expertise for a family of self-adaptive systems. The target domain that we have studied represents self-adaptive systems with the following characteristics:

- Systems comprise software deployed on distributed nodes;
- The nodes have an explicit position in the environment (and may be mobile);
- The nodes have continual communication access;
- Dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software;
- Resources in the system may come and go.

In the last chapter of this dissertation, we draw conclusions from our research. We reflect on lessons learned from our experience with the formalization of self-adaptive behaviors. We outline suggestions for further research building upon the research presented in this thesis. We conclude with a closing reflection on our work.

7.1 Contributions

In this thesis, we presented a formal approach for the design and verification of distributed self-adaptive systems. Whereas a vast body of work has focused on the structural aspects of self-adaptive systems, our focus is on the behavioral aspects for such systems.

The research contributes to the state of the art with the following contributions:
7 Conclusions

- We provide a systematic overview of the state of the art on using formal methods in self-adaptive systems [188]. The overview identifies which formal methods have been used in self-adaptive system and for what purpose these methods have been used, and helps identifying open problems for future research. We created a zone-map that shows a state-space representation with the different states and behaviors in which a self-adaptive system can be found and defines properties of interest for behaviors in and between states of self-adaptive systems.

- We offer a set of MAPE-K Formal Templates [82] that support designers with the specification and verification of self-adaptive systems for the target domain [82]. The MAPE-K Formal Templates comprise two complementary parts:
  - Behavior Specification Patterns [81, 82], which support modeling Monitor, Analyze, Plan and Execute components as distinct, interacting behaviors. The patterns support reasoning about separate adaptation functions and modeling specific interaction mechanisms between different MAPE behaviors, tailored to the problem at hand. Modeling distinct component behaviors also offers flexibility regarding different deployments for the self-adaptive behaviors in a distributed system setting [81, 83].
  - Property Specification Patterns [81, 82], which comprise three types of properties for self-adaptive systems’ behaviors: Adaptation Goal, Intra-Behavior properties and Inter-Behavior properties. The properties support fine grained verification of the correctness of the adaptation behaviors.

We provide qualitative evidence for the reusability of the MAPE-K Formal Templates based on four case studies in which the templates were used for the design and verification of different self-adaptive systems that match the target domain.

7.2 Lessons Learned from Applying Formal Methods in Self-Adaptive Systems

In the course of the research of this thesis, we gained extensive experience with applying formal methods for modeling and verifying self-adaptive systems. We report some lessons learned from this experience.

Separation of Concerns. In this research, we have consequently applied the principles of separation of concerns. At the overall system level, we strictly separate the logic of the managed from the managing system. At the level of the managing system, the MAPE-K Formal Templates separate the different functions of the adaptation logic. This systematic separation of concerns increases understandability and productivity, which is in line with the results of a recent empirical
experiment [185]. On the other hand, strict separation between the domain logic and the adaptation concerns can be hard. In some of the cases, we noticed that enabling support for robustness and openness required invasive instrumentation of the managed systems. In other cases, a strict separation between MAPE functions may be too costly (e.g., for performance of verification). Integrating functions in one behavior may then be an option, at the costs of some tradeoffs, e.g., reduced extensibility.

**Scalability in Model-Checking.** Model-checking is known to suffer from the state-space explosion problem, which makes the approach not scaling well with the size of systems (with current technology). A set of grained detailed models allows analyzing and verifying more specific behavior properties for a self-adaptive system. However, strict separation of the adaptation behaviors results in an increase of the number of automata and, consequently, a blow-up of the state-space. Time-triggered behaviors add to this effect. Finding the correct level of abstraction is an iterative task where the designer needs to find a suitable compromise between verification processing and the level of detail for property specifications. Two interesting techniques to anticipate this problem are abstraction (verification of the system at the level of higher-level architectural design) combined with isolation (verification of individual and combinations of behaviors equipped with stubs).

**From Formalizations to Code Implementation.** In this thesis, we have centered our focus on applying formal methods for designing and verifying distributed self-adaptive systems. The models for designing self-adaptive behaviors provide the basis for code development (transferring verified guarantees to implementation). Some challenges appeared during this transfer. The functions and data structures used in TA are specified in a C-like language. However, this language has some limitations in terms of expressiveness to realize particular behavioral aspects. For example, in this work, we typically model messaging and communication protocols using global variables and channels in TA. However, code implementations required multiple classes and appropriate communication middleware. From our experiences, a relevant part of the behavior specification could be transferred to Java implementation. Nonetheless, complex additional implementations were required to provide communication and service discovery aspects. It is therefore important to take into account limitations of expressiveness in the specifications of the behaviors.

### 7.3 Future Work

The research results presented in this thesis open several opportunities for future research. We present potential lines for future research efforts ordered from more narrow to broad-scoped lines of research.
7 Conclusions

Complex Applications. An interesting track for future research is to apply the MAPE-K Formal Templates to more complex self-adaptive systems, including industrial self-adaptive systems. This will allow to further test the appropriateness of both the behavior and property specification patterns, and their applicability to larger-scale applications.

Behavior Reference Model for Self-Adaptive Systems. One interesting result of the SLR [188] study that zoomed in on model-checking techniques for self-adaptive systems is a zone-model that maps the different types of behaviors of self-adaptive system to zones of its state-space. In this model, properties of interest correspond to transitions between zones. The zone-model offers an interesting starting point to get a better understanding of the specific nature of model-checking of self-adaptive systems. This may ultimately lead to a behavioral reference model for self-adaptive systems, complementing reference models that focus on structural aspects such as [121, 187].

Probabilistic Properties. Another interesting line of research, in line with work from [37, 73, 88], is to focus on the knowledge part of the MAPE-K loop and define reusable model templates for particular sub-domains. In this effort, it would become interesting to incorporate support for modeling and verifying probabilistic properties, which are essential to model uncertainties in self-adaptive systems.

MAPE-K Formal Templates for Different Domains and Qualities. The templates presented in this thesis have been designed to target a particular family of self-adaptive systems and with robustness and openness concerns in focus. Future work may consider self-adaptive systems with different characteristics and different adaptation requirements.

Run-time Verification. As elaborated in this thesis, property verification is crucial to provide evidence for required properties of self-adaptation. The verification of properties at design-time may become either too expensive at runtime or unfeasible due to uncertainty at design time. Therefore, a promising line of research is to study online verification [37, 127, 136]. Runtime property verification could benefit from sensing real and current environment and managed system conditions. Incremental verification can reduce the number of states to evaluate, limiting the space-state.

MAPE-K Loop Interaction. The MAPE-K Formal Templates allow formalization of one or more MAPE loops. As we showed in Chapter 5, the templates allow designing multiple MAPE loops in a system to deal with one or more concerns [81]. In this case, adaptation activities in different MAPE loops may interfere. Weyns et al. [190] propose a set of patterns for decentralized control in self-adaptive systems, focusing on the structural perspective of the multiple MAPE-K feedback loops. An interesting line of future research is to extend that study with the behavioral aspects from the viewpoint of MAPE-K loop interactions [127]. Some of the issues to be considered in such an effort are: "what are the
effects of different deployments of the MAPE components in a distributed setting?", "how to coordinate behaviors to avoid undesired MAPE loop interference?" and "what are the effects of different deployments of the knowledge in such scenarios?"

**Formal Models at Run-time.** In this thesis, we have focused on the design of formal models for self-adaptive systems. These models are abstractions of a system that are typically used for implementation using a traditional programming paradigm (i.e. object-oriented programming). Preserving assurances during the implementation of verified formal models is not trivial. An interesting line of future research is to use the verified models at runtime and realize adaptation by directly executing the models [105]. This would preserve the guarantees from verification. Additionally, the models could be used at runtime to support runtime verification.

### 7.4 Closing Reflection

The number, size and integration of distributed software systems is steadily increasing and, with that, their complexity and our dependency on them. The traditional focus of providing functionality has shifted to assuring quality. Therefore, self-adaptation will increasingly become more common in future software intensive systems and, consequently, formal methods to support providing assurance will become inevitable.

The research presented in this thesis demonstrates how self-adaptive system behaviors can rigorously be modeled and its properties verified with formal templates. It is our belief that formal templates for designing self-adaptive behaviors provide a promising approach for assuring the qualities of future system designs and developments.
The efforts presented in this thesis originate from the concerns that were part of a project with a focus on mobile learning, which has been carried out in the CeLeKT research group. The CeLeKT research group focuses its efforts on projects that are multidisciplinary and use new Information and Communication technologies (ICT) tools and methods to support the implementation of novel learning activities. These projects aim to deploy and analyze the use of new ICT tools and methods to support teachers and students in order to provide innovative opportunities for education.

GEM (GEometry Mobile) [75, 77, 78, 84], is a CeLeKT project in which we examine the use of mobile technologies to design and deploy educational activities for the learning of Mathematics. In GEM, students (12 and 14 years old) work in groups and estimate and calculate figure measurements (i.e. lengths, areas, volumes etc.) in outdoor settings. Measurement calculations are supported via mobile devices with GPS technology. A particular mobile application has been developed in order to allow students to measure distances between members in the group based on the students' relative locations.

The nature of the dynamic outdoor environment introduces a set of uncertainties into the system that may disrupt the activity goals. We focus on exploring the dynamically changing conditions of the environment and developing different mechanisms to mitigate those problems that may arise. These conditions can influence the behavior of (parts of) the system and the features to support the learning activity. For example, meteorological conditions can detriment the GPS accuracy.

During a period of three years (2009-2012), I have been exhaustively exploring the field of mobile learning in order to understand the different concerns that may influence and play a role in supporting collaboration in mobile learning activities. These efforts resulted in a Licentiate thesis publication [76].

The following list summarizes the characteristics of the GEM activities:

- The learning activity is performed in outdoor settings;
- The participants carry out the activity in groups;
- The activity requires collaboration in order to achieve the learning goals;
- Achieving the learning goals require geometry-related calculations;

1 More information regarding the CeLeKT research group can be found at http://www.celekt.info.
Appendix A

Mobile Learning Domain

The efforts presented in this thesis originate from the concerns that were part of a project with a focus on mobile learning, which has been carried out in the CeLeKT research group\(^1\). The CeLeKT research group focuses its efforts on projects that are multidisciplinary and use new Information and Communication technologies (ICT) tools and methods to support the implementation of novel learning activities. These projects aim to deploy and analyze the use of new ICT tools and methods to support teachers and students in order to provide innovative opportunities for education.

GEM (GEometry Mobile) [75, 77, 78, 84], is a CeLeKT project in which we examine the use of mobile technologies to design and deploy educational activities for the learning of Mathematics. In GEM, students (12 and 14 years old) work in groups and estimate and calculate figure measurements (i.e. lengths, areas, volumes etc.) in outdoor settings. Measurement calculations are supported via mobile devices with GPS technology. A particular mobile application has been developed in order to allow students to measure distances between members in the group based on the students’ relative locations.

The nature of the dynamic outdoor environment introduces a set of uncertainties into the system that may disrupt the activity goals. We focus on exploring the dynamically changing conditions of the environment and developing different mechanisms to mitigate those problems that may arise. These conditions can influence the behavior of (parts of) the system and the features to support the learning activity. For example, meteorological conditions can detriment the GPS accuracy. During a period of three years (2009-2012), I have been exhaustively exploring the field of mobile learning in order to understand the different concerns that may influence and play a role in supporting collaboration in mobile learning activities. These efforts resulted in a Licentiate thesis publication [76].

The following list summarizes the characteristics of the GEM activities:

- The learning activity is performed in outdoor settings;
- The participants carry out the activity in groups;
- The activity requires collaboration in order to achieve the learning goals;
- Achieving the learning goals requires geometry-related calculations;

\(^1\) More information regarding the CeLeKT research group can be found at http://www.celekt.info
A Mobile Learning Domain

- Mobile technologies are used as tools that assist the students during the learning activity;
- The dynamism of the system and the environment can affect the activity goals.

In this appendix I elaborate on the design and implementation of a self-adaptive solution to support collaborative mobile learning activities. This study presents a distributed solution created on top of a multi-agent system and extended with self-adaptive mechanisms that are realized through MAPE-K feedback loops. The study presented in this appendix provides two assessment processes in lab settings in order to study functional and non-functional requirements, including resource consumption, availability, complexity, communication overhead and robustness, as well as self-adaptation overheads.
Architectural Concerns for the Design and Implementation of Distributed Collaborative Mobile Learning Systems

Abstract

Mobile technologies have emerged as facilitators in the learning process, extending traditional activities conducted in classroom contexts. Moving outside traditional learning contexts supported by mobile technologies brings elements of uncertainty, which may place the pedagogical activities at risk. Technical aspects related to real-time resource-sharing, system robustness and reliability of services are more difficult to manage in mobile settings and they may affect individual and collaborative performance. Despite of significant research carried out in the field of mobile learning, very few efforts have focused on and provided evidences of covering mobile collaborative learning requirements from a software engineering perspective.

This work focuses on the software architecture aspects aiming at addressing the challenges related to resource-sharing aspects for collaborative mobile learning activities, such as resource limitations on mobile devices and mobile-to-mobile resource access. Additionally, we present the use of self-adaptation as a solution to mitigate risks of resource unavailability and organization failures that arise from environment and system dynamism. The results from this work define a step towards a reference architecture for collaborative mobile learning applications. Our evaluation provides indications regarding the system correctness with respect to resource-sharing and collaboration concerns and offers evidences of self-adaptation benefits for collaborative mobile learning applications.

A.1 Introduction

Mobile technologies have emerged as facilitators in the learning process, offering new ways to access and use pedagogical materials and defining the mobile learning paradigm [163]. Kukulska-Hulme [122] states that learners should be able to engage in educational activities without being bound to a tightly-delimited physical location. In this context, mobile technologies have the capability to provide multiple resources that meet a subset of the new learning needs with a single device [164]. Mobile technologies can enrich learning activities and satisfy requirements for individual and group activities. More specifically, these technologies might foster user interactions based on access to rich content across locations and at any time using portable equipments such as wireless laptops, personal digital assistants (PDAs) and smart phones [159].
A Mobile Learning Domain

In addition, mobile technologies can assist the development of collaborative learning activities. Ogata [142] mentions the advantages related to the use of mobile technologies to organize and mediate social interactions, regardless of time and location in which the pedagogical activities take place. Zurita and Nussbaum’s [202] work extends the list of potential benefits mentioning that mobile technologies can be used to facilitate information sharing, moderate the tasks to be completed, facilitate the management of rules and participant roles, and mediate the acquisition of new knowledge. Moreover, nowadays learning activities do not only take place in traditional learning environments (such as classrooms, lecture halls, etc.) but also in non-traditional environments (such as outdoor settings, learning in public places, museums, exhibits, etc.) and there is a need to better integrate learning goals across these contexts. Mobile technologies play a crucial role in supporting these new developments by enabling users’ active participation in these new learning landscapes without taking into account the limitations imposed by space and time.

In a recent study, Lucke & Resing [129] perform an extensive survey in the field of mobile learning, in order to identify current trends and challenges in the field. The authors identify challenges that are spread along two dimensions, which involve the educational settings (consisting of Formalized settings, Physical settings, Collaborative settings and others) and different levels of context-awareness (consisting of Learner identity, Activity, Location, Object, Device and Network). Even though the authors mention the Software and Hardware dimension in their classification, the importance of tackling Software Engineering aspects that are necessary to support mobile learning is not stressed enough. Initial efforts to focus on software related aspects for Mobile Learning are described by [23, 148]. Concerns such as platforms for scalability, software engineering methods for interoperability and strategies for deploying robust scalable applications were identified as challenges that the field needs to address.

The collaborative nature of many Technology-Enhanced Learning (TEL) activities seem to require a technological platform\(^1\) that support resource sharing, including data, learning objects, and resources. This has recently prompted the usage of distributed solutions on mobile devices. It is a first challenge to provide a software architecture in order to support mobile learning activities where resources and data must be shared during execution.

Mobile software applications generally take into consideration ideal cases of execution, based on a set of correctness assumptions. However, this view is not aligned with real environment deployment, where scenarios found at runtime do not confront assumptions taken during design and implementation [80, 139]. In mobile learning activities, there is a high level of uncertainty regarding the system and the environment that may diminish the fulfillment of the activities’ goals [55, 179, 197]. These uncertainties can occur due to variables in the environment itself. Outside the classroom settings, the environment involves several uncertainties itself. A first classical example of uncertainty concerns the mobile device battery,

\(^1\) We define technological platform as a combination of hardware and software solutions for a specific application domain.
which its availability can be difficult to guarantee in the absence of electrical outlets. A second example of environment uncertainty concerns the network data access, which may get interrupted affecting to complete the collaborative tasks using mobile devices. Uncertainties can also be related to variables in the technological platform. For example, there could be uncertainties regarding the technological resources management to cover activity requirements. In certain scenarios, the activity may require more resources than the resources that can be found in one single mobile device. This could occur when the mobile device lacks a specific resource (i.e. a barometric pressure), or because a requirement demands the combination of multiple devices. The list of uncertainties can increase extensively. Therefore, it is a second challenge to design software architectures for collaborative mobile learning activities that can offer robustness to achieve the activity goals, despite certain uncertainties.

In this work, we center our efforts in addressing the challenges of resource-related limitations for collaborative mobile learning activities, and to study software mechanisms to mitigate risks that arise from environment and system uncertainties. We focus on collaborative mobile learning activities making use of distributed architectures, and facing a wide variety of uncertainties and limitations to answer the following general research question:

"How should we design a robust software system to support collaborative mobile learning activities with shared resource capabilities?"

The rest of the paper is structured as follows: in Section A.2 we present related efforts in this research area to bring up current challenges in the state-of-the-art. Section A.3 presents two learning scenarios with requirements that serve as a basis for design, analysis and deployment. In Section A.4, we propose an architecture solution that addresses the identified requirements, which is evaluated in Section A.5. Section A.6 extends the proposed solution to address environmental uncertainties by applying self-adaptation techniques. In Section A.7 the conclusions of this work are presented and compared with related efforts on the field, together with a description of future efforts.

A.2 Related Efforts

In the field of collaborative TEL activities, the technological platform supporting the activity must provide spaces for individual and collaborative learning, and to promote interactions among peers [202]. One of the main objectives of a collaborative learning activity is to provide individual support in the activity and to encourage collaboration in order to strengthen the learning outcomes. Current mobile technologies possess a set of capabilities that can offer the means to enhance collaborative learning activities [159], through multimedia and other technological resources, positioning sensors, wireless connectivity, and other sensors. However, in certain applications one can easily envision requirements that cannot be covered by features offered by single device. High processing and memory demanding tasks such as image and voice recognition with extensive databases, and
A Mobile Learning Domain

real-time collaborative tasks are some examples that illustrate this limitation. In the software engineering field, this have been addressed by service composition approaches, which requires the combination of multiple nodes. In our previous work, we address this challenge by the use of Mobile Virtual Devices (MVD) [77], where multiple mobile devices form an organization that is provided with the mechanisms for offering and consuming resources located in this organization. One approach adopted in the mobile computing community is to consider mobile devices like thin clients that consume services located in the Cloud, also known as cloud-computing services [153]. This approach extends the capabilities of mobile devices, which can consume local resources as well as services offered in the Cloud, a virtual environment of distributed resources on large-capacity servers [99, 149]. One example of client and Cloud approach is presented in [99], that provides a middleware layer to connect Cloud-based services with mobile client through SMS services; and in [152], where the SMS channel is substituted with GPRS/WIFI connections. These solutions allow extendibility of the mobile device capabilities. However, these solutions are limited to interactions between mobile devices and cloud services, which restricts desired interactions between users as well as real-time sharing of resources that are located in mobile devices.

In order to allow and promote interaction among users, some efforts have explored the use of peer-to-peer approaches with mobile technologies. Coco [109] and Mico [108] are suites of collaboration applications that use peer-to-peer technology to enable spontaneous collaborations and share computing resources across a network. The aim of these platforms is to develop a peer-to-peer platform using XML-based protocols, achieving scalability, some level of flexibility and ad-hoc networking formation. Additional efforts, with document sharing approaches towards information collaboration, have applied ad-hoc network infrastructures among mobile devices [30, 140]. These solutions focus on content sharing, but ignore more complex resources sharing and coordination-related issues.

More recent solutions have employed distributed systems in their mobile learning activities through software agents in order to strengthen the individual and group tasks [14, 68]. These platforms receive the name of multi-agent systems, also known as MAS [192]. The agents in a MAS are autonomous, which allow them to complete tasks individually, but also possess the necessary mechanisms to enable communication among agents, thus facilitating group activities [107]. The agent autonomy provides a chance to offer some levels of adaptation to fit the learning activities into environment context and users’ settings [5]. Even though resources are distributed among the MAS platform, the current approaches are focused on sharing static resources (like multimedia content, documents, etc.), and not on sharing other more complex resources (e.g. hardware resources like camera, GPS sensors, processing capabilities, etc.). This still becomes a limitation to use specific hardware components that are not locally present in a device, and restricts the selection of mobile devices.

The above-presented studies offer an overview of the current trend related to the design of collaborative mobile learning technologies with regard to information and resource sharing aspects. One particular approach uses multi-agent systems, to provide the functions for resource sharing within the platform and autonomy to
allow individual activity assessment. However, none of the previous work completely covers the emerging technological requirements for collaborative learning activities, that include orchestration, real-time collaboration and resource sharing on mobile settings. New mobile learning technological platforms should offer resource-sharing mechanisms to consume local and external resources, in order to avoid mobile device hardware limitations and to support the set requirements mentioned above.

The following research questions refine the general research question in order to tackle the above identified challenges:

**RQ1:** Which are the most suitable characteristics that software architectures should possess to offer resource sharing for collaborative mobile learning activities?

**RQ2:** Which software elements are necessary in a technological solution for collaborative mobile learning applications, in order to satisfy resource-sharing requirements for individual performance and collaborative interactions?

**RQ3:** How do these software elements affect the system response-time, as a quality requirement that is necessary for the collaborative activities?

**RQ4:** How to guarantee robustness with respect to supporting collaboration through the mobile applications in dynamic environments?

### A.3 Learning Scenarios

This section introduces two learning scenarios that illustrate the challenges mentioned above. The scenario descriptions will result on a list of generic Use Cases and related requirements that suggest the future of collaborative mobile learning applications. The list of requirements will offer the base for the elaboration of architectural characteristics and challenges to be addressed in this work.

#### A.3.1 English Numbers Sorting

Infante et al. [106] presented a scenario that uses a single computer with multiple-mouses to support collaborative learning activities. Based on these efforts, a new concept of collaborative language laboratories was developed [35] using multiple-headsets. Enhancing English pronunciation and understanding are the main goals of this learning activity. The system assists the activity via a speech-recognition solution, which evaluates participants’ pronunciation and provides automatic feedback.

The English pronunciation variant presented above has two main limitations, namely the user mobility and a personal physical space [142]. Having a shared physical space, the students could peek on their colleagues’ information, which can diminish the learning outcomes in some cases. Including a personal physical space to the learning application can benefit the learning process, where each student could be provided with the suitable information for his/her tasks in the activity. To address these limitations a new version of this solution has been created.
A Mobile Learning Domain

In English Number Sorting the students are still organized in groups of three. The activity can be performed in a shared lecture room during the classes and from the students’ home during homework. The activity has two main phases: first an individual and second a collaborative. In the first phase, each student is assigned a number that must be correctly pronounced (pronounced numbers are recorded to be used in the second phase). On the second phase, the students must propose and agree on an incremental sequence of numbers. For this phase, the students do not visualize the numbers in the group, but they can hear their peers’ recordings from the phase one. Fig. A.1 shows a group of three participants in the first phase of the English Numbers Sorting activity. The left-side member must still pronounce “seven” to activate the assigned number; the center participant has correctly pronounced the number “four” (“Right!” feedback), and the right-handed participant has incorrectly pronounced the number “six” (“Wrong. Try again” feedback). All the students are required to correctly pronounce their assigned number in order to proceed with the second phase.

This scenario exemplifies the need for a system that supports individual performance and collaboration for a learning activity. Based on the scenario description, a supporting system must offer multiple nodes (mobile phones) for the participants in the activity, and these need to be connected through a communication infrastructure in order to cover resource-sharing requirements. In particular, a supporting system should offer mechanisms to distribute the assigned numbers among the students’ devices (in phase one), share recorded audio files between participants in a group (in phase two).

Due to current technological limitations, mobile devices may not be capable of locally perform speech-recognition processes with the desired level of accuracy. Therefore, an external server is required to operate this process. In order not to interfere with the activity flow, we define 3 sec. as an acceptable lapse for system feedback, not interrupt the students performance in the activity.

A.3.2 The Hidden Treasure

This scenario, "The Hidden Treasure", provides a second example of collaborative mobile learning activity with resource-sharing characteristics and participants mobility in outdoor settings. The Hidden Treasure is based on the previous work on fostering the understanding of geometric concepts for students between the age
of 10 and 12 assisted by mobile technologies [76]. During the activity, groups of
participants perform distance measurements on the field in order to perform dif-
ferent geometric calculations (studying distances, perimeters, area, and volume).
Mobile devices equipped with GPS are used as a tool for distance measurement,
being able to obtain the existing distance between the phones in a group.

In The Hidden Treasure, the participants must apply triangulation techniques to
find a hidden treasure.

In order to complete the activity, each participant receives a mobile device with
a customized application for the calculation of distances. Mobile devices can in-
teract with an activity server, that specifies the set of tasks to be carried out in the
activity and collects activity responses for each group. Additionally, participants
receive a treasure map on paper (see Fig. A.2-left side). Points A and B on the
map are present on the field as well, to make possible a mapping between the real
world and the map representation. Given these two initial points, the distances
between points (represented as A-G points and treasure in the map) and trigonom-
etry concepts, the participants must determine the physical location of the treasure.
Finding the treasure requires the discovery of the intermediate points, which can
be incrementally obtained through triangulation. Triangulation tasks require the
involvement of at least three participants. Thus, the participants must collaborate
and coordinate their actions in order to determine the points in the map. Under the
user request (attempt), the application must provide feedback in terms of current
distances between members in the group. The feedback can then be compared
with expected distances in the map, in order to determine if the attempt success-
fully corresponds with a desired location (Fig. A.2-right side).

This scenario requires a system that uses mobile devices with GPS capabilities
in order to locate positions on the field. With respect to resource-sharing aspects,
the participants must share their current location to perform the triangulation tasks.
Additionally, the system must offer a communication channel between participants
in a group to notify about the discovery of a targeted point, in order to provide
A Mobile Learning Domain

coordination between individual and group performance. This activity requires locations to be shared in a moderately high frequency (every 10 sec. in a group), in order to provide feedback with respect to inter-participant distance. In order not to disturb the activity flow, the system is required to offer distance-calculation functionality when required and to have low latency delays for the feedback. We defined 10 seconds as the time that can be tolerated to present a distance-calculation functionality and 1 second as the accepted latency for the system response to the measurements.

A.3.3 Requirements Overview

Fig. A.3 offers a context diagram of the use cases identified in the two previous learning scenarios, which will offer key points for design and validation of a system solution for the collaborative mobile learning activities. Based on the use cases, we provide the following list as a set of requirements that a software system should have (ENS stands for English Number Sorting, THT stands for The Hidden Treasure):

- Offer one device per participant.
  - ENS & THT: to Perform Individual Task

- Offer activity flow management
  - ENS & THT

- Allow local resource reading.
  - ENS: to Access Microphone
  - THT: to Access GPS location

- Allow remote resource access. Resource-sharing capabilities.
  - ENS: to Share audio recording, and Check pronunciation, Validate sequence of numbers
  - THT: to Share GPS coordinates, Validate treasure location

- Offer tools for coordination and collaboration
  - ENS & THT: to Calculate distance, Sort numbers

- Offer mechanisms for service composition
  - ENS: to Check pronunciation
  - THT: to Calculate distance

- Offer a defined QoS in terms of response time.
  - ENS & THT: to Perform Individual tasks, Perform collaborative task, Share local resource, Manage group

- Offer a defined QoS in terms of service robustness.
  - ENS & THT: to Perform Individual tasks, Perform collaborative task, Share local resource, Manage group (covered in Section A.6)
A.3 Learning Scenarios

Figure A.3: Use Cases for the described learning scenarios.

1. **Offer one device per participant.**
   - ENS & THT: to **Perform Individual Task**

2. **Offer activity flow management.**
   - ENS & THT

3. **Allow local resource reading.**
   - ENS: to **Access Microphone**
   - THT: to **Access GPS location**

4. **Allow remote resource access. Resource-sharing capabilities.**
   - ENS: to **Share audio recording**, and **Check pronunciation**, **Validate sequence of numbers**
   - THT: to **Share GPS coordinates**, **Validate treasure location**

5. **Offer tools for coordination and collaboration**
   - ENS & THT: to **Calculate distance**, **Sort numbers**

6. **Offer mechanisms for service composition**
   - ENS: to **Check pronunciation**
   - THT: to **Calculate distance**

7. **Offer a defined QoS in terms of response time.**
   - ENS & THT: to **Perform Individual task**, **Perform collaborative task**, **Share local resource**, **Manage group**

8. **Offer a defined QoS in terms of service robustness.**
   - ENS & THT: to **Perform Individual task**, **Perform collaborative task**, **Share local resource**, **Manage group** (covered in Section A.6)
A Mobile Learning Domain

A.4 System Architecture

In this section we propose an architecture to support collaborative mobile learning activities. The architecture has been designed following the requirements identified in the use cases presented in the previous section. We describe the software decisions taken for architecture definition. Then, we describe the whole architecture, showing their relation with respective requirements and use cases.

A.4.1 Software Decisions

According to the use cases presented in Section 3, we require a distributed architecture that provides the infrastructure for individual assignment of devices and offers one device per participant in the activity. Distributed architectures can facilitate the distribution of resources among the participants in the activity and can offer capabilities for sharing resources within the activity.

In particular, we advocate for the use of multi-agent systems as an approach of implementing a distributed system. MAS are composed of multiple devices with autonomous capabilities for decision-taking [107, 192]. In a MAS, each agent holds a set of behaviors that describe the logic that specifies the actions of the different autonomous agents and communication protocols. These autonomous capabilities are highly beneficial for the definition of individual tasks (to determine the individual performance) and to define the considerations for the collaborative tasks [50]. Also, multi-agent systems provide mechanisms to improve communication quality, increasing message transmission performance and reducing emerging bottleneck issues [64] when compared with center-based distributed approaches. In our architecture, each mobile device becomes an agent of the MAS.

In the context of this work, autonomous agents provide autonomous decision-making on tasks that should be completed locally (e.g., choosing a service among various service providers), and facilitate delegation of certain atomic responsibilities with respect to the entire system (i.e., numbers sorting and calculation of distances). These features give autonomy to the nodes, in comparison to a centralized coordinator solution. A decentralized solution reduces response times in large scenarios where partial and located knowledge is sufficient, as it avoids potential bottlenecks. Therefore, an architecture based on autonomous agents provides scalability, allowing more nodes with autonomy in described processes. For instance, with the definition of specific behaviors for the mobile devices it has been possible to define the autonomous logic to use the phone microphone during the first phase of the “English Number sorting” scenario, to define when GPS resource should be accessed in “the Hidden treasure” scenario, and to provide user-related feedback locally and autonomously on the mobile device.

Multi-agent systems are enabled to deploy a communication model, using protocols and other mechanisms for data exchange between agents. This supports the communication needed for a collaborative activity. For instance, this feature has been used to offer communication between peers in the activities, between the mobile devices and activity servers and for resource sharing capabilities.
Resource sharing mechanisms are required to satisfy the lack of some resources in the user nodes. Particularly, a MAS enables devices to access remote resources, when those resources are not available locally, using standard approaches, e.g. yellow pages services. In our previous work [77], we presented an approach for resource-sharing in organizations with mobile devices that receives the name of mobile virtual device (hereafter, MVD). A MVD consists of the aggregation of multiple mobile devices in order to create a virtual entity that shares resources located on mobile devices within the MVD. The MVD also allows the elaboration of complex services via the service composition of existing services and resources located on the devices. Accordingly, the combination of a MAS with a MVD middleware provides the capabilities for the use of local and remote resources within the work group, which covers the necessary technological needs in order to satisfy the previous use cases. This combination of MAS and MVD is a unique feature of work in collaborative mobile learning applications, when compared with the study performed on the field and presented in Section A.2. Table A.1 summarizes and relates Use Cases with taken software design decisions.

### Table A.1: Mapping learning scenarios requirements with software decisions

<table>
<thead>
<tr>
<th>Concern</th>
<th>Requirement</th>
<th>Software Decisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual work</td>
<td>Access resource:</td>
<td>Autonomous agents</td>
</tr>
<tr>
<td></td>
<td>Microphone, GPS</td>
<td>deployed on mobile devices</td>
</tr>
<tr>
<td></td>
<td>Perform individual task</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Activity flow management</td>
<td></td>
</tr>
<tr>
<td>Collaboration (communication)</td>
<td>Perform collaborative task</td>
<td>Use of a distributed system through Multiagent systems</td>
</tr>
<tr>
<td></td>
<td>Share local resource</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Provide complex service</td>
<td></td>
</tr>
<tr>
<td>Collaboration (organization management)</td>
<td>View group members</td>
<td>Agents on devices self-organize dynamically as MVD</td>
</tr>
<tr>
<td></td>
<td>Manage group</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Provide complex service</td>
<td></td>
</tr>
<tr>
<td>Resource Sharing</td>
<td>Provide complex service</td>
<td>Multiagent system, MVD</td>
</tr>
<tr>
<td>Adequate Time Response</td>
<td>Perform individual task</td>
<td>Communication via</td>
</tr>
<tr>
<td></td>
<td>Perform collaborative task</td>
<td>a Multiagent system</td>
</tr>
<tr>
<td></td>
<td>Share local resource</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Provide complex service</td>
<td></td>
</tr>
</tbody>
</table>

### A.4.2 Architecture Definition

Our proposed solution for collaborative mobile learning activities combines the use of MAS solutions with the concepts of MVD. First, we introduce our suggested architecture through a layer diagram that provides a global understanding of the solution and offers a high level of abstraction of the deployment of the distributed system (Fig. A.4). Second, we enter in a deeper level of detail with a component diagram in order to identify the basic components in the system and understand their roles and interactions within the system (Fig. A.4).

Three main layers have been defined in the model in order to construct the distributed system (see Fig. A.4). The first layer, User Interface (UI) offers the points of interactions with the user, such as windows, buttons, videos and feedbacks re-
A Mobile Learning Domain

![Architecture described through a layers diagram.](image)

**Figure A.4:** Architecture described through a layers diagram.

With regard to the activity. This layer interacts with the underlying layer, multi-agent system (MAS), which is in charge of providing domain specific functionalities that are required for the activity in place. The MAS layer contains an Agent and an Organization Middleware, based on the MVD concepts [77]. Agents are in charge of executing the roles that are necessary to perform tasks in the activity, such as accessing mobile device resources (both local and remote), managing the activity logic, providing interaction between mobile devices and servers and interaction between members in the collaborative group. In order to manage group-related aspects, such as the identification of members in the group, the MAS layer gets support from the MVD. Together, the Agent and MVD middleware are capable of providing functionalities for collaborative tasks, such as distance calculation between two mobile devices in a group. Finally, the Communication Infrastructure (CI) layer provides mechanisms that are necessary for communication within the platform, such as channels and protocols.

Additionally, Fig. A.5 shows the participation of multiple nodes in the platform creating a distributed system. In a first analysis, we classify the nodes in four different typologies depending on their role in the distributed system. These are (1) the mobile devices, (2) activity servers, (3) proxies for external services and (4) infrastructure servers.

Typically, mobile devices are the point of interaction for participants in the activity, as these provide the application interface, some local resources and mobility features that are required for the mobile learning activity. The activity server has the role to control the activity flow of the participants in the activity. This node manages the groups defining the participants in each of them, registers their performance (in terms of the group response to each task in the activity) and handles the activity flow by providing the proper feedback to the participants. In some cases, the distributed platform may lack resources that can be relevant for the ac-
Figure A.5: Architecture described through a components diagram.
A Mobile Learning Domain

tivity. Under these conditions, an optional service proxy can extend the platform functionalities by providing a linkage to external service providers. One example for service proxy usage, related to the previously presented scenario, can be applied to access Google Speech API services. Proxy nodes can as well be used for concerns such as pedagogical content provision through a Learning Management System (such as Moodle), and remote information access (such as access to news RSS feeds). Finally, an infrastructure node is required to provide functionalities for the infrastructure, such as agent- and service-registration services, messaging channels and logging repositories.

Fig. A.5 shows the top-level components of the architecture, in order to introduce the key components that are necessary for this MAS-MVD distributed solution. Additionally, the figure incorporates the environment entity, which represents components interacting with the system (such as GPS satellites communicating with GPS devices) and channels for communication among devices in the distributed system.

The Activity Manager and Activity Client components are responsible for offering the features to follow the activity flow, such as a student-teacher interaction manner. The responsibilities of these components may include the control of the activity flow, presentation of task announcements, feedback and complementary materials needed for the activity at hand. Therefore, goals can be defined, and resources that are required to achieve these goals.

In order to participate in the MAS, all nodes require a software agent. This component is responsible of executing the node’s behavior. This behavior can contain the global activity logic (Server Agent in the activity server), individual behaviors for tasks within the activity (Device Agent in mobile devices) and additional behaviors for managing the MAS (Agent Manager in the infrastructure server) and connecting to remote services (Proxy Agent). Agents in mobile devices have the autonomy to execute local behaviors. This feature covers the requirement of performing individual tasks in the pedagogical activities.

The agents in the MAS possess the capabilities to exchange messages. This feature is used for the collaborative aspects that are required in the groups. However, an additional component MVD Middleware is necessary to support this collaboration. The MVD middleware component manages knowledge with respect to the members belonging to the groups, so meaningful communications can be established only between the relevant agents in the groups. Additionally, nodes that offering complex services (service composition), may use the MVD middleware component to determine the location of services in the MVD that are required for such service composition. For example, distance-calculation services may require of 2 or more GPS devices in a MVD to be offered.

The Resources Manager provides the required mechanisms for accessing local resources in a node and resources in the rest of the platform. Among its responsibilities, the resource manager should be able to locate needed resources for the activity and register local resources (as services) in the Yellow Pages component so other nodes in the platform can make use of them. The resource manager can as well be used to provide information about services offered from outside the
platform, making use of a Proxy agent in Service Proxy nodes. Finally, the communication infrastructure should contain a component that is specific for the communication requirements in the distributed system such as communication among agents (Communication Middleware), and to communicate with the environment, such as reading the GPS coordinates (Context).

A.5 Architecture Implementation and Assessment

Two software implementations have been created following the architecture design presented in Section A.4, for the learning scenarios presented in Section A.3. In this section we offer the assessment review performed on the implemented solutions. We start providing a description of the technologies used in the implemented applications. Afterwards, we describe the metrics used for the assessment, describe the settings used for the assessment (deployment and relevant environmental conditions) and reason about the results and link them to the research questions. Log traces and software implementations can be downloaded from the project website for review and replication purposes¹.

A.5.1 Implementation Description

We make use of the JADE platform [17], a Java multi-agent system solution. JADE provides a framework for creating software agents and declaring the desired agent behaviors. JADE is available for multiple platforms, including Android mobile devices. Using JADE we define specific behaviors for agents that run on the mobile devices and on the activity server. On the mobile devices, we implement behaviors for audio recording, position gathering and other domain specific functionalities. On the activity server, we implement a behavior to manage groups of participants, control the activity flow for the groups, and register group performance.

JADE provides mechanisms for agent communication, agent discovery and registration (provided by an Agent Manager component), and service discovery and registration. A Directory Facilitator component offers the yellow pages’ role. Some specific functionalities are not feasible deployable on the mobile devices, such as a speech recognition service with the required accuracy. For these cases, the system is complemented with proxy agents running on servers that provide different service roles. One example is the LSR Agent (local speech recognition) that provides Microsoft SAPI services and is deployed on a local server in the MAS. The Microsoft SAPI has a limitation that allows only processing one recording at a time. Therefore, a second speech recognition service agent is included in the platform. This agent, RSR Agent (remote speech recognition), provides access to the Google Speech API.

Organization concerns are deployed among nodes in the MAS using our previous efforts in the MVD design [77], including mobile devices and servers involved in the platform. The MVD allows local management of groups and identification

¹http://homepage.lnu.se/staff/digmsi/SA-MAS/
of existing resources in the platform. On the Activity Server node, the MVD component is used to declare the number of groups that participate in the activity and the members to be assigned in each group. On the mobile device nodes, the MVD components are used for two complementary purposes. First, the MVD identifies participants in the group, in order to determine the collaboration boundaries. Second, it determines the physical mobile devices that are used by each participant in the group, in order to determine the resources that can be shared in the group. More detailed information about this particular software architecture can be found in the project web.

On Section A.4 we have presented the structural architecture design in order to answer the research questions with respect to the technological characteristics and the software elements that the solution should posses. Next, it is necessary to perform an assessment on the software architecture performance in collaborative mobile learning scenarios. Using the requirements presented in Section A.3, we assessed the systems architecture with respect to two different domains: functional requirements and non-functional requirements, including the system’s availability, performance and complexity.

### A.5.2 Functionality Availability Assessment

We studied the behavior of the functional requirements in the "English Number Sorting" activity with the involvement of 6 participants divided in two groups. We assessed the functionalities presented in the Use Case diagram checking their correct behavior and availability. The activity took place in Santiago of Chile and had 40 minutes of duration. We studied the microphone recording (access microphone resource), the audio playing (access audio recording) and the speech recognition feedback functionalities, using the mobile devices nodes and checking that the functionalities were present when required. The MAS deployment included one dedicated computer (Intel Core I5, 4GB RAM) to offer the Activity Server role and offer Microsoft SAPI service (LSR), a service proxy computer (Intel Core I3, 2GB RAM) to access Google Speech API services (RSR) and 6 Lenovo tablet devices with a 2.1GHz processor, 1GB of RAM and running Android 3.1 as user terminals.

Functional requirement issues were only found in the Check Pronunciation use case. The Microsoft SAPI service was defined as the preferred source for speech recognition, and the Google Speech API service as an alternative option when the first was not accessible (blocked processing another audio file). During an initial test (in Table A.2), the speech recognition process was initiated 49 times (instances). 2 instances were not correctly initiated on the mobile device due to mobile client issues accessing the microphone. The remaining 47 instances fired a request to consume the local speech recognition resource (Internal in MAS), from which 32 could be directly processed, indicating a 68.09% of availability of Microsoft SAPI resource. However, 15 needed to be forwarded to a remote speech recognition resource (External to MAS) because the Microsoft SAPI was blocked by another ongoing request. From these 15 instances submitted to the External to MAS resource, 12 were successfully processed (i.e. 80% of availability in Google
Speech API resource), while the remaining 3 resulted in a time-out error from the External to MAS resource.

As mentioned in the Introduction, 100% availability of resources and services is far from common in mobile device applications. Beside the issues in the English Number Sorting, we also suffered of unavailability of resources in The Hidden Treasure. In a previous experiment of “The Hidden Treasure” scenario, 12 mobile devices were used in outdoor settings, in order to use GPS’s in a real environment. The activity took place in Växjö, south of Sweden and had 55 minutes of duration. The mobile devices run Android 2.3 with a 600MHz processor. During the execution process, the participants performed distance calculations between members in the group [76]. However, in some conditions, the mobile devices could not provide accurate measurements for the distance calculations. The errors became present in 2 mobile devices and lasted for a few minutes. Human involvement was required to recover from the errors, by restarting the mobile devices. Based on our logs, these failures represented a 1,51% of failures in the distance calculation functionality.

A.5.3 Performance and Complexity Assessment

Additionally, concerning non-functional requirements, we focused on how the implementation fulfills domain requirements and how architecture software decisions satisfied the desired system response time. We studied performance and complexity of the system measuring the following two types of metric: resource selection complexity and communication overhead.

To assess the system performance and overhead on the English Number Sorting implementation, we studied three different cases for resource consumption, whether they were Local to the node, Internal in MAS or External to MAS. Local resource consumption provides a point of reference for the measurements. The last two cases describe situations in which the resource to be consumed is not located in the same node, but it is present in another node in the MAS (Internal in MAS) or it is external to the MAS and accessed through a Proxy Agent (External to MAS). We studied Local resource consumption by requesting speech recognition services from the LSR agent itself. Internal in MAS is represented by requests initiated on a mobile device to consume Microsoft SAPI services in the LSR agent. External to MAS is represented by requests initiated on a mobile device to consume Google Speech Recognition services in the RSR agent.

During the test, we analyzed the resource selection complexity with respect to the location of the selected resources. The local resource consumption for speech recognition service requires the involvement of 7 components, including the following: (1) Activity Client, (2) Activity Manager, (3) Device Agent, (4) Server Agent, (5) MVD Middleware, (6) Communication Middleware and (7) Resource Manager. The first six enumerated components are involved during the process of acquiring the assigned number to pronounce. The seventh, offers access to the local resource.

For an Internal in MAS resource consumption, the number of components involved increases to 10. In addition to the previous six components for number acquisition, four more components are used to access external resources. (7) Agent

141
A Mobile Learning Domain

Table A.2: Results of functional and non-functional requirements analysis on the English Number Sorting application

<table>
<thead>
<tr>
<th>Metric</th>
<th>Local (Microsoft SAPI)</th>
<th>Internal in MAS (Microsoft SAPI)</th>
<th>External to MAS (Google Speech API)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Selection Test1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Involved Components</td>
<td>7</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Resource selection</td>
<td>N/A</td>
<td>47</td>
<td>15</td>
</tr>
<tr>
<td>Resource availability</td>
<td>N/A</td>
<td>32 (68.09%)</td>
<td>12 (80%)</td>
</tr>
<tr>
<td>Communication Overhead Test2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of samples</td>
<td>100</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>Resource selection (avg)</td>
<td>10 ms</td>
<td>18 ms</td>
<td>18 ms</td>
</tr>
<tr>
<td>Resource usage (avg)</td>
<td>23 ms</td>
<td>23 ms</td>
<td>24 ms</td>
</tr>
<tr>
<td>Resource response time (avg)</td>
<td>33 ms</td>
<td>42 ms</td>
<td>480 ms</td>
</tr>
<tr>
<td>Total time (avg)</td>
<td>43 ms</td>
<td>65 ms</td>
<td>498 ms</td>
</tr>
<tr>
<td>Overhead (avg)</td>
<td>N/A</td>
<td>51.16%</td>
<td>1058.14%</td>
</tr>
<tr>
<td>Re-transmissions (%)</td>
<td>N/A</td>
<td>2.7%</td>
<td>2.9%</td>
</tr>
</tbody>
</table>

Manager, (8)Yellow Pages, (9)LSR: Proxy Agent and (10)LSR: Resource Manager. The Agent Manager communicated with the Yellow Pages component in order to identify the location of the desired resource. Once this is located, the resource is consumed through the Proxy Agent.

The consumption of speech recognition resources External to the MAS requires two additional components, (11)RSR Proxy Agent and (12)RSR Resource Manager, in order to access Google Speech API services. External to MAS resources are only requested if Internal to MAS resources are not available, therefore, components (9)LSR: Proxy Agent and (10)LSR: Resource Manager are also involved in this type of instances.

The access to speech recognition services require the discovery of resource location, the usage of the network infrastructure that supports the MAS and the use of Service Proxies. When compared with local resource consumptions, speech recognition service consumption evidenced an additional overhead. This overhead is due to required internal communication to consume services in the MAS, and additional Internet communications to access External to MAS resources. The comparison between the three latency measurements made possible to determine the efficiency of the system in its searches for, and provision of, resources, which will enable future comparisons in terms of performance with systems providing equivalent functionalities.

The communication overhead was studied measuring the time involved in the selection of a specific external resource (Internal in MAS and External to MAS columns in Table A.2) and compared with time required to consume an equivalent resource local to a node (Local column in Table A.2). For the communication overhead analysis a Test2 was executed. The audio files to be processed had a weight ranging between 90-100KB, and the network settings consisted of a dedicated 6MB/s WIFI connection and a 0.3MB/s Internet connection. In order to measure the communication overhead variables, 250 speech recognition requests were performed (125 using the Internal in MAS resource and 125 using the External to MAS resource - Microsoft SAPI). Additionally, we measured 100 executions of service consumptions that were locally originated in the LSR agent (server of-
A.5 Architecture Implementation and Assessment

ferring the Microsoft SAPI service - Google Speech API), in order to study the overhead implications of using distributed services in the platform in comparison to local resource consumption.

We measured the elapsed time in four different points during the service consumption process. The first comprises the time required to determine the location of the speech recognition service (Resource selection in the table). When this was local to the agent the discovery was faster (10ms) than when it was external (18ms). This overhead can be explained by the need of using the Yellow Pages services in order to determine the location of the desired resource. The second metric offers a view on the time used by the service provider to process the speech recognition (Resource usage), and evidences the equality of the two solutions.

We defined the total time for a service delivery (Resource response time) as the time spent for the service execution plus the time spent during communication processes. It is not surprising to identify that this lapse of time substantially increased when resources External to MAS were accessed. When using the local Microsoft SAPI, only internal node calls are required (33ms) to consume the service. When accessing resources Internal in MAS, the communication overhead (42ms) is limited to a local network usage both used for service discovery through the Yellow Pages and for the service consumption. However, when requiring resources External to MAS, the overhead values are increased (480ms), due to the use of Internet connections, which severely impacts latency measurements. Through these numbers we can observe that the usage of a distributed infrastructure based on a MAS can affect the system’s performance in terms of delivery times. However, when services are located in the MAS, these increases are negligible in terms of absolute numbers (22ms) and due to Yellow Pages requests and WIFI communications. However, the latency increases considerably when services need to be found outside the MAS, and Internet communications are required. In this case, the latency has a growth factor of 10x to consume resources External to MAS for our scenario.

When consuming MAS services, an alternative approach is to consider direct connections between agents, avoiding Yellow Pages services. Such approach would improve performance and latency measurements, but would affect the system’s scalability and extensibility, as resources should be known in beforehand by the agents involved in the platform.

The use of External to MAS resources could have more implications besides performance and latency. Reliability of the services could be not guaranteed, which becomes a threat with respect to system robustness, and can lead to risks for the learning activity. One example is the use of Google Speech API in our learning scenario. In this case, services are not managed by the platform, but external actors are involved (Google) and it becomes an uncertainty the availability of the service (due to high request load, maintenance processes, system failures, among others). Additionally, for "The Hidden Treasure" scenario, we evidenced that GPS resource cannot be considered reliable, which may influence the activities’ outcomes. In the following section we present an extension of the platform that provides mitigation mechanisms to provide guarantees for certain quality concerns, robustness in our case.

143
A Mobile Learning Domain

Robustness, reliability and other software quality concerns, such as the concerns described in this study, have been deeply studied in software engineering field. One approach to provide features towards quality concerns is through the application of self-adaptive systems, a software solution capable of adapting its behavior and structure in order to adapt to dynamic operating conditions.

A.6 Self-Adaptation

The assessment study evidenced a lack of resource availability in particular cases, mostly due to uncertainty regarding the resource and service state. System robustness and resource availability are critical concerns that must be provided in collaborative mobile learning applications, and widely present in multiple other domains. One way to deal with quality concerns is to modify the system’s design and implementation to address the concerns. Another way is through the application of self-adaptation mechanisms [70], which principle is based on the separation of concerns. A self-adaptive system is divided into two subsystems: the managed system (which supports functions that are specific to the domain) and the managing system (responsible for contributing with quality properties to the system) [121].

The self-adaptation mechanisms are placed on top of a managed system, and have access to read system states (Probe) and to touch (Effector) determined system points so adaptations can be performed with the aim of providing selected quality concerns. A well-established approach to realize self-adaptation with through MAPE-K feedback loops [113]. A MAPE-K loop is composed of four main roles that Monitor the environment and managed system states, Analyze the completeness of the system with respect to the desired goals, Plan mitigation actions in order to address identified errors, and Execute the selected plan. These four activities are supported on a Knowledge base that provides a level of abstraction of the activity, managed system, environment and goals.

In our particular case, robustness is achieved when the groups involved in an activity include the number of resources that are required for the activity goals, and these resources provide the desired service quality. For example, in The Hidden Treasure scenario, our system is robust when the groups in the activity have the required number of GPS resources (which can vary depending on the task) and provide location under a specified accuracy measurement (accuracy tolerance may vary depending on the task).

In Fig. A.6 we illustrate an upgrade on the architecture created on the previous designed system, in order to guarantee robustness through a self-adaptive layer. In our particular case, the managed system represents the distributed system that we described in Section A.5. The managing system is responsible of analyzing the current behavior of the managed system and, in case of service failures, adopting the necessary measures to maintain system’s robustness.

Two self-adaptation loops are designed in the solution in order to provide robustness for two separate concerns. A first MAPE-K loop is concerned on the consistency of the accuracy that GPS resources provide. In other words, the first
A.6 Self-Adaptation

feedback loop is concerned with managing the GPS service availability based on the current GPS service quality. The second MAPE-K loop is concerned with managing the number of GPS resources in a group, so it can achieve the activity goals.

Below, we provide a description of components that are required to provide the self-adaptation behaviors in a managing system, and the modifications that are required in a legacy system in order to provide points for monitoring and transferring the self-adaptation actions into the managed system.

![Self-adaptation components on mobile nodes.](image)

**Figure A.6:** Self-adaptation components on mobile nodes.

### A.6.1 The Managed System

The managed system requires specific components in order to provide the necessary functions for the domain field and the necessary behaviors for the activity, which are domain specific. In addition to these system’s components, it is necessary to include points of interaction into the managed system so that the self-adaptation mechanisms can be applied. There are two types of components that are necessary to allow interactions between the managed and managing systems: the *Probes* and *Effectors*. The Probe components focus on enabling information gathering from the system, in order to allow reasoning for the self-adaptation mechanisms. In particular, for our "The Hidden Treasure" scenario, probe components
have been implemented to allow data acquisition with respect to the GPS accuracy in the mobile devices, the current state of the organizations in the groups (represented in the MVDs) and the activity requirements. Therefore, probes must have access to read certain parameters from the Device agent component (current acquired GPS accuracy) and from the MVD middleware component (current organization deployment).

Analogously, the Effector components deal with transmitting the adaptation tasks, decided by the managing system, into the managed system. For The Hidden Treasure scenario, Effectors have been implemented to manage the GPS service in accordance to the GPS accuracy, deactivating or activating the GPS service when required; and to modify the group compositions when required. Those two types of components are the only communication points between the managed and managing system, which enable the system monitoring and the carrying out of adaptation decisions to address the quality concerns.

### A.6.2 The Managing System

The managing system focuses on the provision of quality properties that are desired in the managed system. Our particular quality concern in the collaborative mobile learning activities has focused on providing robustness of the system in front of resource failures.

In order to provide self-adaptation for such concern, it is necessary, first, to Monitor the necessary system and environment parameters that can have an influence in our concern. In the case of The Hidden Treasure scenario, it becomes necessary to monitor the current state of the managed system in terms of accuracy of each of the GPS resources and the number of GPS resources in a group, and monitor the current activity requirements in terms of the required GPS accuracy and required number of members in a group to fulfill the activity goals. Second, it becomes necessary to Analyze the correctness of the managed system behavior. For instance, analyze the GPS service quality with respect to the activity requirements; or analyzing the completeness of a group, in terms of GPS resources. Third, a Plan needs to be designed to mitigate detected issues. For instance, deactivate a GPS service if it performance is not accurate, or determine an additional GPS resource so integrate in a group if this is found incomplete. And fourth, Execute the planned actions to transfer the mitigation to the managed system.

Given the nature of our scenarios, the suggested solution is based on a distributed system. Therefore, it becomes necessary that the components of the managing system are distributed among the nodes of the MAS. For example, in order to self-adapt GPS services, it is necessary that nodes in the MAS locally contain the Monitor, Analyze, Plan and Execute components. For concerns that refer to distributed nodes, it becomes necessary to deploy the self-adaptation components in a distributed manner as well. For instance, for group self-adaptation, it becomes necessary to locate Monitor and Execute components in each of the members of the organization, while the Analyze and Plan components can be located on specific member of the group, following a master-slave pattern approach [190].
In our studies, aspects such as the service quality in the use of GPS coordinates and the availability of audio and microphone service are some of the parameters monitored on mobile device nodes and the quality (in terms of availability) of speech recognition service are parameters monitored on the server side. The self-adaptive mechanisms may be customized to address the quality concerns that are desired for the system at hand. Thus, self-adaptation mechanisms for GPS accuracy may be different than the self-adaptation mechanisms used for microphone recording services.

A.6.3 Self-Adaptation Evaluation

We present the evaluation process of the self-adaptive mechanisms applied to The Hidden Treasure scenario to provide robustness for GPS service reliability and group completeness in front of GPS inaccuracy.

A first set of self-adaptation mechanisms have been designed to manage the GPS services. The self-adaptation is based on GPS accuracy, deactivating the GPS service and unpublishing it (from the Yellow Pages) if the GPS quality is not satisfying or activating and publishing the GPS service if the quality recovered. The GPS service self-adaptation is deployed in all the mobile devices involved in the activity and it is locally and individually managed in the nodes. In other words, the self-adaptation mechanisms are locally controlled and implemented on each of the mobile devices, providing autonomy and decentralization to the self-adaptive solution. The GPS service self-adaptation mechanism is represented in Fig. A.6, by the Resource concern MAPE loop on the mobile devices SA layer.

The deactivation of GPS services, as a result, can lead to groups having lack of resources necessary to cover the MVD requirements. Therefore, a second MAPE-K loop is designed to monitor the completeness of the groups in the activities and, to mitigate potential issues, to incorporate additional phones to the group when required (Organization concern MAPE loop on the mobile devices). This self-adaptation loop is shown in Fig. A.6 with the Organization concern MAPE-K loop. One device in each organization is selected as a master device to be in charge of the self-adaptation decisions. This device is then responsible to gather information with respect to the organization state, the activity requirements (in terms of number of GPS resources) and to determine mitigation plans when required. Due to the distributed deployment of a group, the Monitor role is distributed among the nodes in the organization, which means that all the mobile devices are in charge of monitoring the current GPS service state and to notify the master device when changes are monitored. Equally, the Execute role is distributed in the organization, in order to transfer the mitigation actions across the members in the organization. The interested reader can find more information with respect to this self-adaptation behavior in [81].

The evaluation is performed in lab settings based on The Hidden Treasure scenario, with an activity that requires 3 GPS resources per group, and a minimum accuracy of 10 meters for the GPS service.

We evaluate the GPS self-adaptation through a Test1 with one mobile device running Android 2.3 on a 1.2GHz processor and 1GB RAM. We use a dedicated
3MB/s WIFI connection for the communication aspects. In order to study the self-adaptation mechanisms in detail, we include an additional component that emulates the GPS behavior. This component simulates GPS locations and accuracies, which let us provoke GPS inaccuracies on devices and to study the self-adaptive mechanisms in place.

High stress scenarios (in terms of GPS failures) are simulated to study the GPS self-adaptation in detail. A total of 200 failures and recovers are emulated on the GPS services. 100 of these failures emulate the behavior of a GPS module affected by cloudy conditions, giving inaccurate measurements (errors >10meters) during 3 seconds. This behavior is played in a loop that lasts 42sec. The other 100 errors emulate a flapping GPS device behavior having inaccuracy measurements during 0.5s every 22 seconds.

The self-adaptation mechanisms incur with a processing overhead, not only during the adaptation processes (to mitigate risks when a failure has been detected), but also during desired system’s behavior due to the periodic monitoring process. An initial cost of self-adaptation is originated by a GPS probe component that would periodic monitor the GPS service quality, with a frequency of $f = 500ms$ (milliseconds). The figures represented in GPS columns in Table A.3 offer an overview of the additional overhead produced by the implemented self-adaptive layers and the efficiency of the self-adaptation mechanisms’ behaviors with respect to self-healing. For local GPS service self-adaptation, the processing overhead does not result in a considerable increase. However, it is interesting to notice that, in this first self-adaptation experiment, all the failures and recovers were correctly treated by the implemented self-adaptive mechanism. One reason for this success lays on the fact that, in the worst-case scenario, the self-adaptation mechanism will require 411ms to adapt the GPS service in the system. This number was always lower than the changes in the environment, which updated the GPS state in a 500ms frequency-basis.

Table A.3: Results of the self-adaptation mechanisms.

<table>
<thead>
<tr>
<th>Variable</th>
<th>GPS Test1</th>
<th>MVD Test2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Adaptation overhead (%)</td>
<td>1.38% 3.15%</td>
<td>3.15%</td>
</tr>
<tr>
<td>Failure detection (ms)</td>
<td>14 63</td>
<td>511 973</td>
</tr>
<tr>
<td>Alternative plan identification (ms)</td>
<td>19 78</td>
<td>1338 15994</td>
</tr>
<tr>
<td>Correction application (ms)</td>
<td>75 411</td>
<td>1435 16197</td>
</tr>
<tr>
<td>Effectiveness of failure correction (%</td>
<td>100% 97.50%</td>
<td></td>
</tr>
<tr>
<td>of resolved failures)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With respect to the organization robustness concern, the self-healing mechanisms were expected to identify lack of resources in a group and heal these states by autonomously integrating a new device in the group. In our lab settings, a lack of GPS resources in a group can be originated by GPS service failures. A Test2 is performed to evaluate the organization self-adaptation. 5 mobile devices are involved in this test, emulating a scenario with groups of three participants and two additional spare devices for contingency issues. In the Test2, the GPS behavior has been defined to have a 10% failure rate (in terms of accuracy) during periods of resolved failures.)
between 10s and 20s. The MVD columns in Table A.3 describe the results of this test.

Based on the scenario requirements defined in Section A.3, we consider the self-adaptation mechanisms to be successful when a group can be recovered in less than 10 seconds. During our test, it occurred 40 failures of GPS resources involved in the group. 39 group incompleteness were recovered in less than 2s. However, the remaining required around 16s to be recovered. For our experiment, this implies that the self-adaptation mechanisms have an effectiveness of 97.50%.

The network infrastructure plays an important role in the self-adaptation process due to the distributed deployment of the nodes, which implies higher periods for managing organizational failures that when compared to local self-adaptation. For example, the detection of a failure in a slave (a GPS being turned off in one member of the organization) requires a communication process between the slave and the master and it took up to 973ms.

The self-adaptation mechanisms presented in this section have been designed to mitigate potential risks that the system may face at runtime. Even the mechanisms have demonstrated a high level of effectiveness in front of failures, not all the failure instances are correctly addressed. Therefore, it is necessary to invest further efforts in analyzing the self-adaptation mechanisms and understanding the environmental conditions that can lead to the not-addressed particular cases. This aspect demands the use of rigorous methods to specify the system and self-adaptive mechanisms behaviors.

A.7 Conclusions and Future Efforts

One critical aspect in collaborative mobile learning scenarios is the autonomy of actions taken by the participants in the activity combined with resource sharing aspects for collaboration. The RQ1 "Which are the most suitable characteristics that a software architecture should possess to offer resource sharing for collaborative mobile learning activities?" concerns the software architecture aspects that are necessary to support such activities. We propose the use of a distributed system architecture composed by mobile devices and servers to support the activity. Based on a multi-agent system (MAS) architecture, we propose the design of a software architecture that offers one device per participant with a certain level of autonomous behavior (for individual performance aspects) extended with communication features (in order to support collaboration with the rest of the participants in the activity). Additionally, a distributed architecture solution, in which each participant carries a personal mobile device, offers scalability to the system, in terms that new mobile devices can be integrated to the system by registering them into the MAS. In terms of resource access, our approach allows that nodes in the system can access local and remote resources, as soon as they are offered by nodes in the distributed system.

With respect to the second research question "Which software elements are necessary in a technological solution for collaborative mobile learning applications, in order to satisfy resource-sharing requirements for individual performance and
A Mobile Learning Domain

collaborative interactions?”, we want to provide a more detailed level of description of the structural architecture design for a collaborative learning application. We suggest the use of a set of components for local and distributed resource management, node behavior management and organization management. Efforts like Huang & Yin [99], and Rao et al., [152] have tackled the resource management issue by extending mobile device capabilities with cloud-computing approaches. Even these efforts can extend the set of functionalities present on a mobile device, these still lack features with respect to resource-sharing between mobile devices. Johnson’s group [108, 109] provides an innovative solution in the mobile learning community, that brings a peer-to-peer based solution to offer resource-sharing between devices. However, this solution does not provide instant communication between peers, but all mobile devices are supported by a server-instance where resources (mainly data) is stored and pulled in a periodic basis. Our solution is one step forward with respect to Johnson’s approach. We suggest the use of a MAS based solution that allows direct and real-time communication between nodes in the system, and that accomplishes the response-time restrictions that are set for our mobile learning scenarios. This particular aspect is the concern expressed in the RQ3 "How do these software elements affect the system response-time, as a quality requirement that is necessary for the collaborative activities?". Through our experiments, we identified that a distributed architecture based on mobile MAS can cover response-time contain that are necessary in order not to restrain collaborative mobile learning applications. Moreover, due to the MAS based approach, this solution suggests good scalability properties in the scope of learning scenarios. Future research will focus on providing evidences in medium and large scale scenarios.

Finally, we have focused on robustness concerns for collaborative mobile learning applications to answer the question "How to guarantee the support of collaboration through the mobile applications?". We have presented an extension of our MAS solution with self-adaptation capabilities towards resource and organization robustness aspects. Additionally, we studied the processing overhead that self-adaptation mechanisms can imply on a legacy system, identifying that these are neglectable. On the contrary, we have presented that the implemented self-adaptation mechanisms (for The Hidden Treasure scenario) could self-adapt service state in around 75ms and organization issues in 1300ms as average values. This contrasts to our previous experiences, where human involvement was necessary, and service and organization could require up to 10 minutes to be fixed.

To the best of our knowledge, there have not been additional relevant efforts that have focused on failure recovering in collaborative mobile learning applications. More studies should emphasize providing quality properties to learning activities.

The designed self-adaptation mechanisms imply changes in the behaviors of the nodes in the MAS. This aspect becomes more relevant when working in a distributed environment, as it may affect the behavior of groups in the system or, in the worse case scenario, even the overall system. Formal methods should be used to evaluate the correct design of a system of this fashion, and verify that the desired goals and the required property qualities are achieved through the self-adaptation processes [188].
A.7 Conclusions and Future Efforts

In our future efforts, we consider the evaluation of the proposed self-adaptive architecture within the "English Numbers Sorting" scenario and study its generalization by extending the study to additional mobile learning scenarios.
Appendix B


This section describes the protocol that was followed for the execution of the systematic literature review process presented in Chapter 4. Within this protocol, we specify:

• the research questions to be addressed through the study;
• the sources to be used during the review, defined in time and space dimensions;
• the search strings that we define for the study selection;
• a list of the data items we collect from the selected studies;
• and internal settings and work strategy that we apply during the systematic literature review.
Appendix B


This section describes the protocol that was followed for the execution of the systematic literature review process presented in Chapter 4. Within this protocol, we specify:

- the research questions to be addressed through the study;
- the sources to be used during the review, defined in time and space dimensions;
- the search strings that we define for the study selection;
- a list of the data items we collect from the selected studies;
- and internal settings and work strategy that we apply during the systematic literature review.
(Version 1.0—2012-03-19)

Abstract
One major challenge in self-adaptive systems is to provide guarantees about the required runtime quality properties. Formal methods provide the means to rigorously specify and reason about the behavior of self-adaptive systems, both at design time and runtime. To the best of our knowledge, no systematic study has been performed on the use of formal methods in self-adaptive systems. As a result, there is no clear view on what methods have been used to specify and verify self-adaptive systems, and what support these methods actually offer to software developers. As such insight is important for researchers and engineers, we performed a systematic literature review to aggregate empirical evidence about the use of formal methods in self-adaptive systems. An essential element in conducting a systematic literature review is to establish a protocol for the study. The protocol defines in advance how the systematic review is to be conducted. Such definition is necessary to structure the work and important to avoid bias. This document describes the protocol for the systematic literature review on formal methods for self-adaptive systems.

Overview
The protocol description is organized in 6 sections as follows. In section B.1 we discuss background and motivate our study. Section B.2 gives a general overview of the research method. In section B.3 we explain the research questions we address. Section B.4 defines the search strategy and scope. In section B.3 we describe the data items that need to be collected. Finally, we conclude with a brief description of data analysis in section B.6.

B.1 Background and Justification
Self-adaptation has been widely recognized as an effective approach to deal with the increasing complexity and dynamicity of modern software systems [113, 121, 143]. A self-adaptive system comprises two parts: the managed system that deals with the domain functionality and the managing system that deals with the adaptations of the managed system to achieve particular quality objectives. The key underlying idea of self-adaptation is complexity management through separation of concerns. One major challenge in self-adaptive systems is to provide guarantees about the required runtime quality properties. Formal methods provide the means to rigorously specify and verify the behavior of self-adaptive systems. Formal
methods have been applied during system development, but also during runtime to provide guarantees about the required properties of self-adaptive systems [130, 174, 176, 187, 198].

In 2004, Bradbury et al. [26] surveyed 14 formal specification approaches for self-adaptation based on graphs, process algebras, and logic formalisms. The survey evaluated the ability of each approach to specify self-managing systems, and concluded that existing approaches need to be enhanced to address issues regarding expressiveness and scalability. [48, 57, 100, 127, 156] summarize achievements of the field and outline challenges for future work, including challenges with respect to the application of formal methods for verification and validation of self-adaptive systems.

These studies provide some insight in the potential use of formal methods in self-adaptive systems, however, to the best of our knowledge, no systematic study has been performed on the actual use of formal methods in self-adaptive systems. As a result, there is no clear view on what methods have been used to specify and verify self-adaptive systems, and what support these methods actually offer to software developers. However, such an insight is important for researchers and engineers.

The overall objective of this systematic literature review is to identify which formal methods have been used in self-adaptive systems, and for what purpose these methods have been used. We also aim to access for what type of self-adaptive applications formal methods have been used, and which tools authors have used. From the study, we derive conclusions concerning support provided by the state of the art formal methods in self-adaptive systems, and areas for future research.

B.2 Overview of Research Method

A systematic literature review [115] is a well-defined approach to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest. Figure B.1 shows an overview of the main phases of our study.

Four researchers are involved in this literature study. In review planning (Phase 1), the review protocol is defined by the researchers, as described in this document. The review protocol includes the definition of research questions, the search strategy and scope, the data items that had to be collected, and the approach for data analysis and presentation of the results. Subsequently, the researchers have to conduct the review (Phase 2). Studies have to be selected based on the search criteria and data has to be collected as specified in the protocol defined in Phase 1. Next, the data derived from the the primary studies has to be collated and summarized to answer the research questions defined in the protocol. Finally, the review report has to be produced (Phase 3). The final report will be cross checked by an independent researcher. His feedback will be used to improve the description and correct minor issues. All the material of the study will be available at http://homepage.lnu.se/staff/daweaa/SLR-FMSAS.htm.
B.4 Search Scope and Strategy

The scope of the search is defined in two dimensions: time and space. Regarding time, we search papers published from Jan. 2000 until Dec. 2011. 2000 is used as start date since self-adaptive systems have become subject of active research around that time. Regarding space, we include the primary venues for publishing research results on self-adaptive systems, as well as the major conferences and journals on software engineering. The selected venues are listed in Table B.1. The Rank is based on the evaluation published by the Australian Research Council. 1

Table B.1: Searched venues

<table>
<thead>
<tr>
<th>ID</th>
<th>Venue</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASE</td>
<td>International Conference on Automated Software Engineering</td>
<td>A</td>
</tr>
<tr>
<td>DEAS</td>
<td>Design and Evolution of Autonomic Application Software</td>
<td>n/a</td>
</tr>
<tr>
<td>ICAC</td>
<td>International Conference on Autonomic Computing</td>
<td>B</td>
</tr>
<tr>
<td>ICSE</td>
<td>International Conference on Software Engineering</td>
<td>A</td>
</tr>
<tr>
<td>ICSM</td>
<td>International Conference on Software Maintenance</td>
<td>A</td>
</tr>
<tr>
<td>ISSTA</td>
<td>International Symposium on Software Testing and Analysis</td>
<td>A</td>
</tr>
<tr>
<td>SASO</td>
<td>Self-Adaptive and Self-Organizing Systems</td>
<td>n/a</td>
</tr>
<tr>
<td>SEAMS</td>
<td>Software Engineering for Adaptive and Self-Managing Systems</td>
<td>n/a</td>
</tr>
<tr>
<td>WICSA</td>
<td>Working International Conference on Software Architecture</td>
<td>A</td>
</tr>
<tr>
<td>WOSS</td>
<td>Workshop on Self-Healing</td>
<td>n/a</td>
</tr>
<tr>
<td>FSE</td>
<td>Foundations of Software Engineering</td>
<td>A</td>
</tr>
<tr>
<td>JSS</td>
<td>Journal of Systems and Software</td>
<td>A</td>
</tr>
<tr>
<td>TAAS</td>
<td>Transactions on Autonomous and Adaptive Systems</td>
<td>n/a</td>
</tr>
<tr>
<td>TOSEM</td>
<td>Transactions on Software Engineering and Methodology</td>
<td>A*</td>
</tr>
<tr>
<td>TSE</td>
<td>Transactions on Software Engineering</td>
<td>A*</td>
</tr>
</tbody>
</table>

Our search strategy combines automatic with manual search. Automatic search comprised of two steps: first we select the studies that are relevant for self-adaptive systems, and then we filter the studies that use formal methods. We use the following search string in the first step:

(( Title:adaptive OR Title:adaptation OR Title:self OR Title:autonomic OR Title:autonomous ) OR

The keywords provide the main terms that different communities use to refer to self-adaptive systems. We will apply pilot searches on the set of papers from

1 http://www.arc.gov.au/era/era_2010/archive/era_journal_list.htm

B.3 Research Questions

We formulate the general goal of the study through Goal-Question-Metric (GQM) perspectives (purpose, issue, object, viewpoint) [11]:

- **Purpose**: Understand and characterize
- **Issue**: the use of formal methods
- **Object**: in self-adaptive software systems
- **Viewpoint**: from a researcher’s and engineer’s viewpoint.

The general research question translates to four concrete research questions:

- **RQ0**: Which are the trends in applying formal methods in self-adaptive systems?
- **RQ1**: Which formal methods have been used in self-adaptive systems?
- **RQ2**: For which adaptation concerns have formal methods been used?
- **RQ3**: How have formal methods been used to provide to deal with concerns of self-adaptive systems?

With RQ0, we want to get insight in how formal methods have been used by researchers both in time and space. RQ1 is motivated by the need to get insight in what kind of formal methods have been used for self-adaptive systems. This question aims to assess which languages have been used for modeling systems, verifying properties, and which tools have been used for this. With RQ2, we aim
to understand why formal methods have been used in self-adaptive systems. Concretely, we aim to assess for which concerns formal methods have been used (reliability, performance, functional correctness of adaptations, etc.), which properties have been verified for that (liveness, deadlock, etc.), and for which type of applications these methods have been used. Finally, RQ3 aims to assess how formal methods have been used to provide guarantees about concerns in self-adaptive systems (reasoning, model checking, proof, etc.).

B.4 Search Scope and Strategy

The scope of the search is defined in two dimensions: time and space. Regarding time, we search papers published from Jan. 2000 until Dec. 2011. 2000 is used as start date since self-adaptive systems have become subject of active research around that time. Regarding space, we include the primary venues for publishing research results on self-adaptive systems, as well as the major conferences and journals on software engineering. The selected venues are listed in Table B.1. The Rank is based on the evaluation published by the Australian Research Council.

Table B.1: Searched venues

<table>
<thead>
<tr>
<th>ID</th>
<th>Venue</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASE</td>
<td>International Conference on Automated Software Engineering</td>
<td>A</td>
</tr>
<tr>
<td>Adaptive</td>
<td>Adaptive and Self-adaptive Systems and Applications</td>
<td>n/a</td>
</tr>
<tr>
<td>DEAS</td>
<td>Design and Evolution of Autonomic Application Software</td>
<td>n/a</td>
</tr>
<tr>
<td>ICAC</td>
<td>International Conference on Autonomic Computing</td>
<td>B</td>
</tr>
<tr>
<td>ICSE</td>
<td>International Conference on Software Engineering</td>
<td>A</td>
</tr>
<tr>
<td>ICSM</td>
<td>International Conference on Software Maintenance</td>
<td>A</td>
</tr>
<tr>
<td>ISSTA</td>
<td>International Symposium on Software Testing and Analysis</td>
<td>A</td>
</tr>
<tr>
<td>SASO</td>
<td>Self-Adaptive and Self-Organizing Systems</td>
<td>n/a</td>
</tr>
<tr>
<td>SEAMS</td>
<td>Software Engineering for Adaptive and Self-Managing Systems</td>
<td>n/a</td>
</tr>
<tr>
<td>WICSA</td>
<td>Working International Conference on Software Architecture</td>
<td>A</td>
</tr>
<tr>
<td>WOSS</td>
<td>Workshop on Self-Healing</td>
<td>n/a</td>
</tr>
<tr>
<td>FSE</td>
<td>Foundations of Software Engineering</td>
<td>A</td>
</tr>
<tr>
<td>JSS</td>
<td>Journal of Systems and Software</td>
<td>A</td>
</tr>
<tr>
<td>TAAS</td>
<td>Transactions on Autonomous and Adaptive Systems</td>
<td>n/a</td>
</tr>
<tr>
<td>TOSEM</td>
<td>Transactions on Software Engineering and Methodology</td>
<td>A*</td>
</tr>
<tr>
<td>TSE</td>
<td>Transactions on Software Engineering</td>
<td>A*</td>
</tr>
</tbody>
</table>

Our search strategy combines automatic with manual search. Automatic search comprised of two steps: first we select the studies that are relevant for self-adaptive systems, and then we filter the studies that use formal methods. We use the following search string in the first step:

(( Title:adaptive OR Title:adaptation OR Title:self OR Title:autonomic OR Title:autonomous ) OR

The keywords provide the main terms that different communities use to refer to self-adaptive systems. We will apply pilot searches on the set of papers from

1 http://www.arc.gov.au/era/era_2010/archive/era_journal_list.htm

157

SEAMS, ICAC and TAAS to ensure that the keywords provide the right scope of papers.

In the second step, we use the following search string:

"Transition system" OR automata OR Markov OR Petri OR "state machine" OR calculus OR algebra OR grammar OR CSP OR "temporal logic" OR LTL OR "tree logic" OR CTL OR "first order logic" OR "reaction rule" OR probabilistic OR stochastic OR "graph rewriting" OR "graph representation"

This set of keywords define key terms that refer to the use of formal methods. We derived the terms from three sources: Baier and Katoen’s book on model checking [7], the survey of Bradbury et al. on formal methods in self-adaptive systems [26], and Tamura et al.’s recent roadmap paper on verification of self-adaptive systems [174]. For the second filtering, we will again apply pilot searches to ensure that the keywords provide the right scope of papers.

We further refine the studies resulting from automatic search using a manual search step. The goal of this step is to identify the primary studies that are directly related to the research questions. To that end, we define the following inclusion/exclusion criteria:

- **Inclusion criterion:** The study is directly related to RQ1 and RQ2. Rationale: we might have included studies which employ some formal terms, but do not actually employ formal methods for a particular purpose.
- **Exclusion criterion:** A paper that is an editorial, abstract or a short paper. Rationale: these papers do not provide a reasonable amount of information.

A study is selected when it met the inclusion criterion and eliminated if it met the exclusion criterion.

### B.5 Data Items

For each primary study, the data items shown in Table B.2 have to be collected. Two reviewers read each study in detail and extract the data in a form. This data is used during a discussion to resolve conflicts and reach consensus for the data items. This discussion involves the other reviewers when no agreement could be reached.

The data items author(s), and title (F1-F2) were used for documentation. Year (F3) and Venue (F4) are used for answering RQ0. Options for modeling language (F5) are:

- algebra
- set theory
- state machines
- transition systems
- timed automata
- Markov chains
- Markov decision processes

158
B.5 Data Items

Table B.2: Data collection form

<table>
<thead>
<tr>
<th>Item ID</th>
<th>Field</th>
<th>Concern / research question</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Author(s)</td>
<td>Documentation</td>
</tr>
<tr>
<td>F2</td>
<td>Title</td>
<td>Documentation</td>
</tr>
<tr>
<td>F3</td>
<td>Year</td>
<td>RQ0</td>
</tr>
<tr>
<td>F4</td>
<td>Venue</td>
<td>RQ0</td>
</tr>
<tr>
<td>F5</td>
<td>Modeling language</td>
<td>RQ1</td>
</tr>
<tr>
<td>F6</td>
<td>Property specification/verification</td>
<td>RQ1</td>
</tr>
<tr>
<td>F7</td>
<td>Concerns subject of formal verification</td>
<td>RQ2</td>
</tr>
<tr>
<td>F8</td>
<td>Verified properties</td>
<td>RQ3</td>
</tr>
<tr>
<td>F9</td>
<td>Use of formalization</td>
<td>RQ3</td>
</tr>
<tr>
<td>F10</td>
<td>Offline vs. runtime use of formal methods</td>
<td>RQ3</td>
</tr>
<tr>
<td>F11</td>
<td>Tools used for modeling and verification</td>
<td>RQ1</td>
</tr>
<tr>
<td>F12</td>
<td>Application domain</td>
<td>RQ2</td>
</tr>
</tbody>
</table>

- process calculi (CSP, π-calculus)
- Petri nets
- Z notation
- Alloy
- other.

Property specification/verification (F6) options are:

- modeling language (the language used for modeling the system is also used for specifying/verifying properties)
- first-order logic
- linear temporal logic
- computation tree logic
- timed computation tree logic
- probabilistic computation tree logic
- other.

Concerns subject of formal verification (F7) refers to the self-adaptive aspects of interest of formalization. The options (based on IEEE 9126 and ISO/IEC 25012) are:

- reliability (fault tolerance, recoverability): capability of software to maintain its level of performance under stated conditions for a stated period of time
- availability: the degree to which the software is in a functioning condition, i.e. capable to perform its intended functions
- usability (ease of learning, communicativeness): effort needed to use the system
- efficiency/performance (time behavior, resource utilization): efficiency of the software by using the appropriate resources under stated conditions and in a specific context of use
- maintainability (analyzability, changeability, stability, testability): effort needed to make specified modifications.
- portability: ability of software to be transferred from one environment to another
- security: ability of the system to protect against misuse

159

- accuracy: the extent to which the software realizes the intended behavior in a specific context of use
- flexibility in use: capability of the software to provide quality in the widest range of contexts of use, incl. dealing with unanticipated change and uncertainty
- functional correctness of the self-adaptive system
- other.

Verified properties (F8) refers to the concrete properties that are subject of reasoning or verification. The options are:

- liveness
- persistence
- deadlock
- safety
- fairness
- reachability
- stability
- other.

Use of formalization (F9) options are:

- modeling
- reasoning
- proving
- model checking
- other.

Offline versus runtime use of formal methods (F10) options are:

- offline (formal methods are applied to design models)
- runtime (formal methods are used by the system itself at runtime).

Tools used for formal modeling and verification (F11) options are:

- SPIN (general tool for verifying the correctness of distributed software models, spinroot.com/spin/)
- PRISM (probabilistic symbolic model checker, www.prismmodelchecker.org/)
- Uppaal (integrated tool environment for real-time systems modeled as networks of timed automata, www.uppaal.org/)
- CZT (tools for formal modeling in the Z, czt.sourceforge.net/)
- CPN Tools (tool for editing, simulating, and analyzing Colored Petri nets, cpntools.org/)
- other.

Applications domain (F12) for which self-adaption is used in the study. Possible application domains are:

- parallel computing (grid, parallel computing, cloud computing etc.)
- service-based systems (websites, business applications, e-commerce, etc.)
B.6 Data Analysis and Reporting

- client-server systems
- embedded systems, including robotic systems
- traffic and transportation
- other.

**B.6 Data Analysis and Reporting**

The data derived from the studies will be collated and summarized to answer the research questions. The synthesis includes the following:

1. Listing of findings,
2. Reaching consensus among reviewers in case of conflicting opinions,
3. Analysis of findings, and
4. Answering research questions and interpretation of the results.

Based on the synthesis, conclusions and recommendations for future research in the field will be derived, and limitations of the review have to be identified. Finally, the results of the review are presented in a review report.
Appendix C: Protocol for Case Studies: Reusability of MAPE-K Formal Templates

This section describes the case study protocol that we followed during the execution presented in Chapter 6. Within this protocol, we specify:

• the research questions to be addressed through the study;
• the hypothesis and propositions we formulated;
• the research team involved in the design and execution of the case study;
• details regarding the design of the case study;
• the selection of cases to be included in the study;
• data items to be collected;
• the calendar of study execution;
• and the limitations of the study.
### Appendix C

**Protocol for Case Studies: Reusability of MAPE-K Formal Templates**

This section describes the case study protocol that we followed during the execution presented in Chapter 6. Within this protocol, we specify:

- the research questions to be addressed through the study;
- the hypothesis and propositions we formulated;
- the research team involved in the design and execution of the case study;
- details regarding the design of the case study;
- the selection of cases to be included in the study;
- data items to be collected;
- the calendar of study execution;
- and the limitations of the study.
Protocol for Case Studies.
MAPE-K Formal Templates to Rigorously Design Behaviors of Self-Adaptive Systems
(Version 1.0—2014-01-22)

Abstract
To provide evidence that the self-adaptive behaviors satisfy system’s goals, regarding the changing conditions, state of the art advocates the use of formal methods. We have designed a set of MAPE-K Formal Templates (hereafter MFT) that consolidate our knowledge in the design of formal models for a family of self-adaptive software systems.

Through the use of the MFT, designers can specify self-adaptive systems that realize the MAPE-K architectural approach in order to specify the MAPE-K components behaviors and to study desired self-adaptation properties.

In this work, we study the MFT with the aim of offering evidence of their reusability and benefits for the self-adaptive community. We carry out a multiple-case study where the MFT are applied in a number of domains (specifying and verifying the behavior of self-adaptive systems) in order to analyze the level of reusability of the templates.

This document describes the protocol for the study to analyze the reusability of the suggested MFT for a family of self-adaptive systems. We carry out a holistic multiple-case study with four cases. The study holds four cases of self-adaptive systems applied to different domains. The participants involved during the study perform a number of design iterations that include (1) design of a managed system, (2) design, specify and verify the correctness of a self-adaptive solution for the previous managed system. During this second part, the participants are offered the MFT as a tool to support the managing system design process.

We collect data from three main sources in order to provide evidence of reusability regarding the MFT.

C.1 Background
Designing contemporary software intensive systems, such as mobile applications and multi-robot systems, and guaranteeing the system goals is complex due to their dynamic operating conditions. Examples are changing availability of resources and faults that are difficult to predict. This situation has led to the development of self-adaptive systems, i.e., systems that are able to reconfigure their structure and modify their behavior at run-time in order to adapt to changes with little or no human intervention [48, 127, 128]. The underlying principle of self-adaptation is separation of concerns. A self-adaptive system typically consists of a managed system and a control (or feedback) loop. The managed system deals with the
domain concerns for the users, while the control loop typically deals with some
adaptation concern of the managed system (e.g., optimize the managed system for
different operating conditions, heal the managed system when a fault is detected,
etc.). In this study, we focus on architecture-based self-adaptation [70, 121, 143,
187]. In architecture-based self-adaptation, the system reasons about an abstract
model of itself and adapts when needed according to some adaptation goals. The
feedback loop typically consists of Monitor, Analyze, Plan, and Execute compo-
ments, complemented with models (Knowledge) of the managed system, its envi-
ronment and adaptation goals. This structure is referred to as MAPE-K [113].

One important challenge in engineering self-adaptive systems is to provide
evidence that the system goals are satisfied, regarding the dynamically changing op-
erating conditions [48, 127, 128]. To that end, state of the art in architecture-based
self-adaptation advocates the use of formal methods. A recent systematic litera-
ture survey [188] identified a total of 75 papers that use formal methods in self-
adaptive systems. Studying the state of the art reveals that existing approaches
define models and required properties of the self-adaptive system for the prob-
lem or domain at hand from scratch. There is no systematic consolidation of the
design knowledge for future use. Other researchers have demonstrated the value
of reusable templates for the specification and verification of formal models, e.g.,
Taibi and Ngo [173] formally specify models or reusable patterns, and Grunske
[88] defines reusable properties for the specification of probabilistic requirements.
Filieri, Ghezzi, and Tamburrelli [65] point out that the definition of such reusable
formal templates are an important research challenge for self-adaptive systems.

C.1.1 MAPE-K Formal Templates

During the past five years, we have been studying various self-adaptive systems,
with a particular focus on self-adaptation to realize robustness and openness. Some
of the recent applications we studied are an intelligent monitoring system to detect
traffic jams [104], a mobile system to support outdoor learning activities [81], and
robotic systems to perform transportation tasks.

In the course of designing and building the self-adaptive applications, we de-

erived a set of MAPE-K Formal Templates (MFT for short) for designing MAPE-K
based feedback loops of self-adaptive systems (see Appendix D) that targets the
following family of self-adaptive systems:

C.1.1.1 Target domain characteristics

• The system comprise software deployed on distributed nodes;
• The nodes have an explicit position in the environment (and may be mobile);
• The system has continual communication access;
• Dynamics in the system and the environment are orders of magnitude lower
  than the speed of communication and the execution of the software;
• Resources in the system may come and go.

Such MFT are important to advance on the state of the art in the design of self-
adaptive systems, by providing tools to support the design of formally specified
C Protocol for Case Studies: Reusability of MAPE-K Formal Templates


C.2 Terminology

In this document, a set of terms regarding the formal specification of system’s behavior are presented. Here we provide the definitions for the terms to be found during the protocol. Notice that we use capital letters at the beginning of terms to refer to MAPE-K templates (e.g. Monitor), which are the objects under study, and non-capital letters for instantiations of the templates for specific domains (such as trafficMonitor).

MAPE-K Formal Templates (MFT) is a set of templates with a high level of abstraction for self-adaptive systems that fit with the criteria presented in Section C.1.1.1-”Target domain characteristics”. The MFT are composed by a set of Behavior Specifications Patterns and a set of Property Specification Patterns. The MAPE-K Formal Templates can be found in Appendix D.

Behaviors Specifications Pattern (BSP) is a high level formal model that specifies the behavior of a component in a feedback control loop. The RFBS can be refined to specify concrete formal behaviors for particular self-adaptive systems at hand. BSP are defined through timed-automata (finite-state machine extended with clock variables, which are used to synchronize behaviors). The following Fig. C.1 provides one example of a BSP for an Analyze component.

Property Specification Pattern (PSP) is a high level formal property specification that describes desired properties that a self-adaptation (or a component in a self-adaptation) system must hold. Property Specification Patterns are defined using timed-computational tree logic (TCTL). TCTL describe state and path formulas allowing the verification of properties of interest, such as reachability (a system should/can/cannot reach a particular state or states), liveness (something eventually will hold), etc. The following TCTL property shows a case for a Property Specification Pattern.

**Figure C.1: Example of Behaviors Specifications Pattern**
A behavior specification is an instance of a Behaviors Specifications Pattern, using timed-automata, for a feedback control loop component in a particular self-adaptive system. Therefore, the behavior specifications are concrete for systems at hand in a particular domain, and describe the specific behavior that such component implements. The following Fig. C.2 provides one example of a behavior specifications for the Analyze component for a Mobile Device application with GPS services.

A property specification is an instance of a Property Specification Pattern, using TCTL, to describe (part of) the expected behavior of a (component of a) concrete control loop. Therefore, the property specifications are concrete for systems at hand in a particular domain, and describe specific properties that such component must hold. The following TCTL property shows a property specification for a property specification to study an analyze behavior applied to a collaborative learning application.

C.3 Research Questions

We consolidated knowledge gained through our previous experience, with the creation of MFT. We present the MFT as a base of work for future self-adaptive system designers, that will assist them in the development of their designs.

The purpose of this study is to provide evidence of the reusability of the proposed MFT. We formulate set of hypothesis with respect to the templates reusability, and perform four case studies that aims confirm or refute such hypothesis.
C.3.1 Question Formulation

Based on the previous background presentation, and on the experiences through the development of the RFM, we formulate the following research questions (RQ). We elaborate on the rationale for each one of the research questions under their statement.

**RQ1:** "Can the MAPE-K Templates be applied to applications that share the characteristics of the target domain?"

With RQ1 we want to provide evidence that the templates can be applied in additional domains and applications that were not studied in detail during the process of their (the MAPE-K Formal Templates) design. The MFT are composed by a set of behavioral specification templates and property specification templates. Therefore, RQ1 can be more deeply formulated in order to study the level of reusability of each one of these units.

**RQ2:** "Do the Behaviors Specifications Patterns allow the behavior specification of MAPE-K components for the specific self-adaptive systems?"

With RQ2, we want to analyze that the BSP can be applied for the specification of self-adaptation behaviors for different applications that fit into our category of family of systems (see Target domain characteristics in Sec. C.1.1.1). Therefore, we want to study the expressiveness of the Behaviors Specifications Patterns to identify if they can positively be used to design self-adaptive behaviors given an existing managed system.

**RQ3:** "How do the Behaviors Specifications Patterns support the design of MAPE-K loops?"

Considering that the BSP allow the behavior specification of components for self-adaptive systems, we want to study how the BSP are used to specify the components’ behaviors. Therefore, with RQ3, we want to further study the types of reuse and refinement applied during the case studies. In particular, we want to identify the different types of reuse and refinement that the Behaviors Specifications Patterns can allow for the self-adaptation design. Initial points considered for reuse include (1) multiple behavior instantiation and (2) behavior states, (3) transitions, (4) functions reuse and (5) refinement.

**RQ4:** "Do the Property Specification Patterns allow the property specification and verification of the MAPE-K loops for the specific self-adaptive systems?"

Analogously, with RQ4 we want to determine whether the templates for property specification can be used to define internal MAPE-K behavior properties, properties for interactions between behaviors of MAPE-K components and to specify properties of the self-adaptive behavior as a whole.

**RQ5:** "How do the Property Specification Patterns support the design of MAPE-K loops?"

RQ5 focuses on understanding the different refinement and reuse approaches that can be applied for the property specification of the self-adaptive systems at hand.
C.3.2 Hypotheses

Following Basili’s framework for hypothesis scoping [12], and from the research questions presented above, we provide the following constituents that helps us to frame the hypothesis to be studied in this study.

**Analyze** a set of MAPE-K Formal Templates for the design of a family of self-adaptive systems

for the **purpose of** evaluating their potentials

with respect to their reusability

from the viewpoint of researchers and formal behavior designers,

in the context of final-year master students in software engineering that add different self-adaptation properties to the design of a distributed software application.

Based on our previous experiences in the design of formal specification of self-adaptive systems behaviors and properties, we generate the following set of hypothesis (H):

**H1:** "The MAPE-K Formal Templates can be applied in domains that qualify for the family of self-adaptive systems (target domain) different than mobile learning and robotics"

**H2:** "The Behaviors Specifications Patterns can be reused and refined (in total or in part) for the specification of the self-adaptive systems behavior for a managed system in a specific domain at hand"

**H3:** "The Property Specification Patterns can be reused and refined (in total or in part) for the property specification and verification of the self-adaptive systems behavior for a managed system in a domain at hand"

**H4:** "The MAPE-K Formal Templates can support the design of self-adaptive system behaviors through instantiation, multi-instantiation and refinement of the following items: behavioral specifications, states, transitions, data structures and functions."

C.4 Design

To provide evidence of the MFT reusability, or to refute the previous presented hypotheses’ propositions, we execute a four case studies where the MFT use will be in focus.

The study is presented as four "holistic case studies" [194], that will take place during a period of 40 working days. We run the case studies within the Master course: "4DV108: Advanced Software Design", which focuses on the software design and engineering of self-adaptive systems. Four students in the course become

---

1 One goal of the study will be to identify additional items of reusability
the participants of the study. The design of the study is presented as four holistic case studies [194].

Within these case studies, we take an holistic study approach as opposed to an embedded approach with multiple units of analysis. The rationale in this decision come from the following conditions. The studies are performed within a master course¹, where the participants are required to (1) describe and design a managed system that complies with the characteristics given in Sec. C.1.1.1 and (2) design, specify and verify the correctness of a self-adaptive solution for such managed system. Therefore, the course is presented as a design and development process with no independent embedded units, that results in the creation of a self-adaptive system, for each one of the students. The process generated by each student (participant) becomes a case study. Thus, for each case we take an holistic study approach in order to study the data that is produced during the whole development process.

C.4.1 Propositions

The following are research propositions (RP) that will assist on the study of the hypothesis (H) validity. The research propositions are founded on the MFT and their documentation presented in Appendix D. We consider an open-ended list of research propositions, and it is part of this study to identify different approaches for the reuse of the MFT. We provide an initial categorization of propositions that consider:

Research propositions concerning the reusability of the Behaviors Specifications Patterns:

H2: The Behaviors Specifications Patterns can be reused and refined (in total or in part) for the specification of the self-adaptive systems behavior for a managed system in a domain at hand.

RP2.X.1 States in the BSP are instantiated to specify equivalent meaningful component states.

RP2.X.2 Transitions in the BSP are instantiated to specify equivalent behavior transitions between states.

RP2.X.3 Functions in the BSP are instantiated to specify equivalent pieces of behavior logic in transitions.

RP2.X.4 Data structures in the BSP are instantiated to specify to represent equivalent meaningful data representations.

RP2.X.5 Behavior specifications can be multiple-instantiated within a self-adaptive system.

RP2.X.6 States in the BSP are refined to specify extended/reduced/composed meaningful component states.

¹ The duration of the course is from the 20th of January to the 23rd of March 2014.
C.5 Case Selection

RP2.X.7 Transitions in the BSP are refined to specify extended/reduced/composed equivalent behavior transitions.

RP2.X.8 Functions in the BSP are refined to specify extended/reduced/composed equivalent meaningful function logic.

RP2.X.9 Data structures in the BSP are refined to specify extended/reduced/composed equivalent meaningful data structures.

Research propositions concerning the reusability of the Property Specification Patterns:

H3: The Property Specification Patterns can be reused and refined (in total or in part) for the property specification and verification of the self-adaptive systems behavior for a managed system in a domain at hand.

RP3.X.1 PSP are instantiated for the specification and verification of equivalent properties of (a component of) the self-adaptive behavior.

RP3.X.2 PSP are refined to adapt for the specification and verification of equivalent properties of (a component of) the self-adaptive behavior.

RP3.X.3 PSP are combined to adapt for the specification and verification of equivalent properties of (a component of) the self-adaptive behavior.

An initial and illustrative list of concrete instantiations of the research propositions can be found in Section C.13.

C.5 Case Selection

The selection of the participants in this study follows the hereafter criteria:

- are master students that take a Software Engineering program.
- have been taught in the field of self-adaptive systems.
- have previous knowledge with respect to quality concerns.
- have previous knowledge in the design of distributed systems.
- have not been involved in the design of the MAPE-K Formal Templates that are focus of this study.
- did not had previous formation on the use of the templates that are focus of this study.

The participants work individually, which results in four case studies.

In order to study the reusability of the MFT in a broad number of domains, we provide a list of criteria that the system should satisfy (see Section C.1.1.1- Target domain characteristics), but we do not limit the application domains where the system can be applied.

The following selection of participants, together with the open selection of domains where the MAPE-K models are applied, have been defined in order to reduce threats of internal and external validity (see Section C.9).
C Protocol for Case Studies: Reusability of MAPE-K Formal Templates

C.6 Procedures and Roles

The procedures defined for this study include a set of resources materials and involves a set of research members with different specific roles.

C.6.1 Materials Resources

Course materials. The participants are provided with course materials, including slides, lectures and research publications in the field of self-adaptive systems. These resources should assist the participants in building their knowledge required for the performance of this study.

MAPE-K Formal Templates (UPPAAL). The MAPE-K models consist of a set of behavioral models (timed-automata) to design MAPE-K-based behaviors and a set of property specifications for the verification of the correct behaviors towards desired adaptation goals.

MAPE-K Template Documentation. Documentation for the use of the MAPE-K templates is provided. The documentation provides the basics for the understanding of the behavior patterns, the property specifications patterns. The MAPE-K Template Documentation can be found in Appendix D.

C.6.2 Roles of Research Team Members

Researcher A. The first researcher’s role is to resolve questions with respect to the templates semantics. The resolving process excludes assistance or consultation with respect to the design process (See Subsection C.6.3-Phase 3).

The researcher is involved in the protocol design, template semantics elaboration, in the data analysis process and reporting tasks.

Researcher B. The researcher’s role is to teach the participants in the use of formal models, theory with respect to self-adaptation and MAPE-K architectural pattern and about the templates for the behavior design of MAPE-K behaviors (templates that are focus of this study). Additionally, the researcher is involved in the protocol design, in the data analysis process and reporting tasks.

External researcher E1. The role of the external researcher is to guarantee the "correct" design of the study under the characteristics of research approaches and the relevance of the expected contribution to the field. The external researchers should assist in the process of identifying threats to validity in the study, and propose points for improvement (under the constraints of the study). Additionally, the external researcher E1 is involved in the report review process, in order to guarantee the expected level of quality in the final report, as the main contribution of this study for the research community in the field of self-adaptive systems.

External researcher E2. The role of this external researcher is to guarantee the expected level of quality in the final report. This researcher has no previous access to the study data, but the researcher’s first contact to the study is through the report paper. The role of this researcher is to guarantee a proficient level of report understandability from a blind reviewer point of view.

\[1 \text{http://homepage.lnu.se/staff/daweaa/MAPE-K/templates/reusable_templates.zip}\]
C.6.3 Phases of the Study

Now, we present and describe the different phases in which this study is divided. A summary of the activities is presented in a calendar format in Sec. C.11. This study is performed within the master course "4DV108: Advanced Software Design", which starts on the 20th of January 2014.

C.6.3.1 Phase 0: Protocol design

During this phase, the researchers define the protocol for the study in course. The protocol is reviewed by an external researcher in the field of self-adaptive systems, in order to guarantee the correct design of the study through research methods.

C.6.3.2 Phase 1: Foundations

This phase is executed during the first two weeks of the master course. During this period of time, the students (participants of the case studies) are formed on the topic of formal methods and formal models to rigorously specify system’s behaviors and properties. With respect to formal modeling aspect, the participants are taught on the use and design of TA as a language for formal behavior specification and TCTL as a language for property specification. The participants are introduced to the UPPAAL tool [15] to learn the functionalities for the design of system behaviors (via extended timed-automata), for the specification of system properties (though TCTL) and the verification of defined properties. UPPAAL is the main tool to be used during the study for

- the design of the managing system;
- the design of the system’s environment;
- and the design of the managing system, through the MAPE-K templates, which is the focus of this study.

C.6.3.3 Phase 2.1: Designing a Managed System

This sub-phase is divided in a set of steps. In a first step, the participants are instructed regarding the first part of their design activity. Given the minimal system requirements presented in Section C.1.1.1- "Target domain characteristics".

The participants are requested to identify a managed system that fits within the target domain and to describe the system’s behavior and concerns. The students must determine the domain in which the managed system is situated. In this step, researchers A and B revise the participants selections and researcher B provides the relevant feedback to guarantee the system’s adequacy for the study with respect to the target domain.

On a second step, the participants are requested to provide formal models that specify the managed system’s behavior, a formal behavior that represents and abstraction of the environment where the system is found and a set of properties that verify that system concerns are achieved. During this step, the participants must provide versions of the formalizations on a frequency basis, and are assisted by researchers A and B, in order to guarantee a correct formalization of the managed system.

173
C Protocol for Case Studies: Reusability of MAPE-K Formal Templates

In the third step, the participants have accomplished a formal model that describes a managed system with the characteristics upper presented.

C.6.3.4 Phase 2.2: Applying MAPE-K Formal Templates
The goal of the sub-phase is to introduce the participants to the MFT for the formal behavior design for a family of self-adaptive systems that are characterized by the descriptions presented in the target domain. The researchers provide the set of MAPE-K Formal Templates (models under study) to the participants together with the templates’ documentation.

The participants are demanded to identify a set of quality concerns that are desired to be covered on the previous managed system, through the design of a self-adaptive control loop. The participants can use the MAPE-K Formal Templates in order to accomplish the adaptive behaviors.

During this sub-phase, the participants maintain periodic follow up meetings in order to see the managing system design progress and to clarify aspects from the MAPE-K Formal Template’s documentation when required. This becomes a source of data to study the models reuse in different domains. For this purpose, the participants are required to meet the main researcher in three occasions (during a two week period, between 20 and 40 minutes per session and student).

The participants deliver an extension of the models of the managed system’s behavior with the MAPE-K architectural-based self-adaptation models. This becomes a main source for data analysis with respect to the models reusability.

C.6.3.5 Phase 2.5: Debriefing
Researcher A executes semi-structured interviews in order to get first hand information from the participants with respect to their experiences in designing the formal models and the potential use of the MAPE-K Formal Templates and the levels of comprehension of the templates’ documentation. Questions in the interview are presented in Section C.12.

C.6.3.6 Phase 5.1: Data analysis
In this phase, the researchers analyze the data collected a set of sources in order to understand the use of the templates and to provide some indications with respect to the potential and soundness of the MAPE-K Formal Templates in multiple domains previously described.

The sources of data collection and their relationships with the study are further elaborated in Sec. C.7. The analysis process is described in Sec. C.8.

C.6.3.7 Phase 5.2: Debriefing
Researchers A and B meet with the participants to present the outcomes form the data analysis and to contrast the findings with them. The aim of this sub-phase is to increase the understanding of the study data and reduce potential false conclusions.

C.6.3.8 Phase 6: Reporting
In this phase, we report the outcomes from the study as part of a publication regarding the MFT. First, we elaborate on the need for formal models to specify the
behavior of self-adaptive systems, as an open challenges in order to provide assurances of self-adaptation correctness. Second, we present the MFT as a contribution that originates from the consolidation of previous experiences in the design of formal templates for self-adaptive systems, and we introduce the semantics of the MFT. Then, we elaborate on the MFT evaluation performed through the study.

The reporting of the study is generated and included in a Journal article. In order to offer a high level of quality, the Journal article is initially reviewed by one external researcher that is experienced in the field, the external researcher E2. This researcher has not prior been involved in the study, is requested to review the article. The aim of this reviewer is to identify potential flaws and aspects that may be unclear for an external reader. Given the feedback, the reporting is updated and submitted to the selected venue.

C.7 Data Collection

Through this study, the researchers expect to collect the data from the following data sources (DS):

**DS1. Focus Groups**  *Meetings with the participants during the application of the MAPE-K models for the creation of the self-adaptive solution.*

These are a first degree data sources. The meetings will be located at the university facilities (Linnaeus University), and have a duration that will vary between 20 and 40 minutes. The sessions will be recorded. During the sessions, the researcher will take notes (observations) in order to determine (metrics) behavioral models (or part of) that have been considered for use and refinement (re-instantiation of behaviors, states, transitions, invariants, functions, data structures, properties) during the design process, and issues that may arise during the sessions.

These data provide a first point of analysis to understand the complexity of use of the models, and the level of comprehension of their semantics, through the direct interaction with the participants, and by observing their evolution during the models application.

The data items from this source can be seen in Table C.1.

**DS2. Assignment Outcome** *Analysis from the final deliverables of the self-adaptive systems (behavior specifications and property specifications).*

These are second degree data sources. This data sources offer an objective source to determine the models reusability. Through this data items, it is possible to focus on parameters for template refinement, in order to determine which are the template items that have been reused (and the degree of reuse), modified, extended, reduced and other ways of refinement.

The data items from this source can be seen in Table C.2.

**DS3. Interviews** *Results from the participant interviews.*

These are second degree data sources. With this data source, we will identify the participant perceptions and experiences during the use of the templates, in
### Table C.1: Data items to be collected through the Focus group sessions

<table>
<thead>
<tr>
<th>ID</th>
<th>Data Item</th>
<th>Description</th>
<th>Proposition</th>
<th>Research Question</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Group ID</td>
<td>Code of the group</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1.2</td>
<td>Session ID</td>
<td>Code of the session. In order to determine the evolution of the template designs</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1.3</td>
<td>Domain</td>
<td>Description of the domain where the SA is being applied</td>
<td>N/A</td>
<td>RQ1</td>
<td>H1</td>
</tr>
<tr>
<td>1.5</td>
<td>Analyze Beh.</td>
<td>Template of the Analyze(s) behavior(s)</td>
<td>RP2.A.1, RP2.A.2, RP2.A.3, RP2.A.4</td>
<td>RQ2, RQ4</td>
<td>H2, H4</td>
</tr>
<tr>
<td>1.6</td>
<td>Plan Beh.</td>
<td>Template of the Plan(s) behavior(s)</td>
<td>RP2.P.1, RP2.P.2, RP2.P.3, RP2.P.4</td>
<td>RQ2, RQ4</td>
<td>H2, H4</td>
</tr>
<tr>
<td>1.7</td>
<td>Execute Beh.</td>
<td>Template of the Execute(s) behavior(s)</td>
<td>RP2.E.1, RP2.E.2, RP2.E.3, RP2.E.4</td>
<td>RQ2, RQ4</td>
<td>H2, H4</td>
</tr>
<tr>
<td>1.8</td>
<td>Global SA</td>
<td>Property that specifies a global self-adaptation behavior</td>
<td>RP3.G.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>1.9</td>
<td>Monitor Prop.</td>
<td>Property that specifies the desired behavior of the monitor component</td>
<td>RP3.M.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>1.10</td>
<td>Analyze Prop.</td>
<td>Property that specifies the desired behavior of the analyze component</td>
<td>RP3.A.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>1.11</td>
<td>Plan Prop.</td>
<td>Property that specifies the desired behavior of the plan component</td>
<td>RP3.P.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>1.12</td>
<td>Execute Prop.</td>
<td>Property that specifies the desired behavior of the execute component</td>
<td>RP3.E.1, RP3.E.2</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>1.13</td>
<td>Inter-Beh. Prop.</td>
<td>Properties that specify desired interactions between MAPE components</td>
<td>RP3.I.1, RP3.I.2</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
</tbody>
</table>
### Table C.2: Data items to be collected through the self-adaptive system deliverables

<table>
<thead>
<tr>
<th>ID</th>
<th>Data Item</th>
<th>Description</th>
<th>Proposition</th>
<th>Research Question</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Group ID</td>
<td>Code of the group</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2.2</td>
<td>Session ID</td>
<td>Code of the session. In order to determine the evolution of the template designs</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2.3</td>
<td>Domain</td>
<td>Description of the domain where the SA is being applied</td>
<td>N/A</td>
<td>RQ1</td>
<td>H1</td>
</tr>
<tr>
<td>2.4</td>
<td>Monitor Beh.</td>
<td>Template of the Monitor(s) behavior(s)</td>
<td>RP2.M.1,</td>
<td>RQ2, RQ4</td>
<td>H2, H4</td>
</tr>
<tr>
<td>2.5</td>
<td>Analyze Beh.</td>
<td>Template of the Analyze(s) behavior(s)</td>
<td>RP2.A.1,</td>
<td>RQ2, RQ4</td>
<td>H2, H4</td>
</tr>
<tr>
<td>2.6</td>
<td>Plan Beh.</td>
<td>Template of the Plan(s) behavior(s)</td>
<td>RP2.P.1,</td>
<td>RQ2, RQ4</td>
<td>H2, H4</td>
</tr>
<tr>
<td>2.7</td>
<td>Execute Beh.</td>
<td>Template of the Execute(s) behavior(s)</td>
<td>RP2.E.1,</td>
<td>RQ2, RQ4</td>
<td>H2, H4</td>
</tr>
<tr>
<td>2.8</td>
<td>Global SA property</td>
<td>Property that specifies a global self-adaptation behavior</td>
<td>RP3.G.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>2.9</td>
<td>Monitor Prop.</td>
<td>Property that specifies the desired behavior of the monitor component</td>
<td>RP3.M.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>2.10</td>
<td>Analyze Prop.</td>
<td>Property that specifies the desired behavior of the analyze component</td>
<td>RP3.A.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>2.11</td>
<td>Plan Prop.</td>
<td>Property that specifies the desired behavior of the plan component</td>
<td>RP3.P.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>2.12</td>
<td>Execute Prop.</td>
<td>Property that specifies the desired behavior of the execute component</td>
<td>RP3.E.1,</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>2.13</td>
<td>Inter-Beh. Prop.</td>
<td>Properties that specify desired interactions between MAPE components</td>
<td>RP3.I.1,</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
</tbody>
</table>
order to better understand the template’s benefits and identify aspects for deeper improvements/depuration.

The interviews aim is to clarify the rational the participants used in the process of design of the components behavior and properties specifications, in order to provide additional evidence on the use of the MAPE-K templates and to understand the actions that the participants did during the refinement phase.

The data items from this source can be seen in Table C.3.

C.8 Analysis

The analysis to be performed during this study has the aim of confirming the initial hypothesis.

C.8.1 Hypothesis confirmation

1. "The MAPE-K Formal Templates can be applied in a broad number of domains that qualify for the family of self-adaptive systems"
   Through the identification of the domains and managed system instances, we can provide indications with respect to the templates reuse in multiple domains.

2. "The Behaviors Specifications Patterns can be reused and refined (in total or in part) for the specification of the self-adaptive systems behavior for a managed system in a domain at hand"
   Through the identification, both in typology and number, of the (part of the) Behaviors Specifications Patterns that have been reused/refined during the study, we can provide indications with respect to their reuse for the design of MAPE-K behaviors for a family of self-adaptive systems.

3. "The Property Specification Patterns can be reused and refined (in total or in part) for the property specification and verification of the self-adaptive systems behavior for a managed system in a domain at hand"
   Through the identification, both in typology and number, of the (part of the) Property Specification Patterns that have been reused/refined during the study, we can provide indications with respect to their reuse for the design of MAPE-K behaviors for a family of self-adaptive systems.

C.9 Study Limitations

Here we present the limitations that this study must consider and acknowledge based on the application of rigorous research methods.
C.9 Study Limitations

Table C.3: Data items to be collected through the Interviews

<table>
<thead>
<tr>
<th>ID</th>
<th>Data Item</th>
<th>Description</th>
<th>Proposition</th>
<th>Research Question</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Group ID</td>
<td>Code of the group</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3.2</td>
<td>Session ID</td>
<td>Code of the session. In order to determine the evolution of the template designs</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3.3</td>
<td>Domain</td>
<td>Description of the domain where the SA is being applied</td>
<td>N/A</td>
<td>RQ1</td>
<td>H1</td>
</tr>
<tr>
<td>3.4</td>
<td>Monitor Desc.</td>
<td>Explanation of the Monitor behavior, its triggering mechanism, sources of data monitored and interactions with other components</td>
<td>RP2.M.1, RP2.M.2</td>
<td>RQ2, R4</td>
<td>H2</td>
</tr>
<tr>
<td>3.5</td>
<td>Preprocess</td>
<td>Explanation of the functionality of the preprocessing function. Which are the data sources used, which type of preprocessing is applied, which is the expected post-processing outcome and function output</td>
<td>RP2.M.3, RP2.M.4</td>
<td>RQ2, R4</td>
<td>H2</td>
</tr>
<tr>
<td>3.6</td>
<td>Analyze Desc.</td>
<td>Explanation of the Analyze behavior, its triggering mechanism, sources of data to be analyzed, logic in the analyze process and interactions with other components</td>
<td>RP2.A.1, RP2.A.2</td>
<td>RQ2, R4</td>
<td>H2</td>
</tr>
<tr>
<td>3.7</td>
<td>Plan Desc.</td>
<td>Explanation of the Plan behavior, its triggering mechanism, sources of data sources to define plans, logic in the plan, and interactions with other components</td>
<td>RP2.P.1, RP2.P.2</td>
<td>RQ2, R4</td>
<td>H2</td>
</tr>
<tr>
<td>3.8</td>
<td>Execute Desc.</td>
<td>Explanation of the Execute behavior, its triggering mechanism, sources of data for execution of tasks, logic in the execution process and interactions with other components</td>
<td>RP2.E.1, RP2.E.2</td>
<td>RQ2, R4</td>
<td>H2</td>
</tr>
<tr>
<td>3.9</td>
<td>Global SA properties</td>
<td>Description of the property(ies) defined to specify the global self-adaptation behavior; and the process taken to define such property(ies)</td>
<td>RP3.G.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>3.10</td>
<td>Monitor Prop.</td>
<td>Description of the property(ies) defined to specify the monitor component behavior; and the process taken to define such property(ies)</td>
<td>RP3.M.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>3.11</td>
<td>Analyze Prop.</td>
<td>Description of the property(ies) defined to specify the analyze component behavior; and the process taken to define such property(ies)</td>
<td>RP3.A.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>3.12</td>
<td>Plan Prop.</td>
<td>Description of the property(ies) defined to specify the plan component behavior; and the process taken to define such property(ies)</td>
<td>RP3.P.1</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>3.13</td>
<td>Execute Prop.</td>
<td>Description of the property(ies) defined to specify the execute component behavior; and the process taken to define such property(ies)</td>
<td>RP3.E.1, RP3.E.2</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
<tr>
<td>3.14</td>
<td>Inter-Beh. Prop.</td>
<td>Description of the property(ies) defined to specify the interactions between MAPE component behaviors; and the process taken to define such property(ies)</td>
<td>RP3.I.1, RP3.I.2</td>
<td>RQ3, RQ5</td>
<td>H3, H4</td>
</tr>
</tbody>
</table>
C.10 Reporting

Data source:
With respect to the scope of the MFT reuse, there is a threat to validity in the selection of domains where the MFT can be applied. To address this threat, the participants have freedom of choice with respect to the domain where the self-adaptive system is applied. Only a set of minimum requirements is provided describing the characteristics that such a self-adaptive system should possess.

C.10 Reporting

The reporting of this study will be included as part of a research contribution in a Journal article, preliminary to be submitted to TAAS (Transactions on Autonomous and Adaptive Systems). Elaborating on a previous study, which describes the origins of the MAPE-K templates for a family of self-adaptive systems, we will strengthen our claims by offering the documentation of the MAPE-K templates and providing some indications towards the templates reuse in other domains besides the initially focused and by external participants.

The previous study can be found at: https://www.dropbox.com/s/kr9r6ylarjrwsd4/submitted.pdf

C.11 Schedule

Below we provide the schedule of the study. The study is expected to be carried out during 6 (to 7) consecutive weeks, and performed during academic working hours (beside exceptions). In order to facilitate the relation between dates and activities, the further have been labeled with a $P_x$, where $P_x$ identifies the Phase in where the activity can be found.

C Protocol for Case Studies: Reusability of MAPE-K Formal Templates

C.9.1 Internal validity

In order to reduce threads for internal validity, this study includes a number of measures:

**Protocol:** The protocol is designed with the participation of two researchers (A and B), in order to reduce bias and improve the application of adequate research methods. In additionally, one external researcher (E1) is involved during the protocol revision to increase the study design quality.

**Selection:** The participants are master students. However, these participants are known to be the next generation of practitioners in the design of self-adaptive systems.

**Template Reuse:** Researcher A is not involved in the design of the managing systems (for example, as an assistant or consultant for the participants), but his role is to clarify aspects concerning the template semantics.

**Data Analysis 1:** Two researchers are involved during the data analysis, in order to reduce researcher bias. In case of lack or agreement, a third researcher will be introduced in the process.

**Data Analysis 2:** The results from the analysis are corroborated with the participants through an interview process that takes place during phase 6.

C.9.2 External validity

**Relevance 1:** The relevance of the study for the community is a threat to the validity of this study as a contribution to the field. Therefore, external researchers from the field are involved through the protocol revision, in order to determine the relevance of the study.

**Relevance 2:** This study involves master students and cases within a higher education program. It is important to extend this study in the future in order to involve practitioners from the field of self-adaptive systems and see the impact of the templates in a practitioner environment. However, as indicated by Filieri, Ghezzi, and Tamburrelli [65], there is a the need of these templates and it has been considered as one of the open challenges in the field.

C.9.3 Reliability

**Data Interpretation:** Bias of the researcher in data interpretation may a threat to the study validity. We plan to make the data available for external access and review.

**Data source:** One threat to the study validity is the number and type of case studies. In a near future, this should be complemented with future analysis to extend this study with additional case studies and provide more evidence that support the hypothesis that we desire to validate. It may also become a threat to the validity of the study regarding the broad use of the templates. Conclusions must be constructed taking this limitation into account.
Data source: With respect to the scope of the MFT reuse, there is a threat to validity in the selection of domains where the MFT can be applied. To address this threat, the participants have freedom of choice with respect to the domain where the self-adaptive system is applied. Only a set of minimum requirements is provided describing the characteristics that such a self-adaptive system should possess.

C.10 Reporting

The reporting of this study will be included as part of a research contribution in a Journal article, preliminary to be submitted to TAAS (Transactions on Autonomous and Adaptive Systems). Elaborating on a previous study, which describes the origins of the MAPE-K templates for a family of self-adaptive systems, we will strengthen our claims by offering the documentation of the MAPE-K templates and providing some indications towards the templates reuse in other domains besides the initially focused and by external participants.

The previous study can be found at: https://www.dropbox.com/s/kr9r6ylarjrw5d4/submitted.pdf

C.11 Schedule

Below we provide the schedule of the study. The study is expected to be carried out during 6 (to 7) consecutive weeks, and performed during academic working hours (beside exceptions). In order to facilitate the relation between dates and activities, the further have been labeled with a $P_x$, where $P_x$ identifies the Phase in where the activity can be found.
### January. Foundations and Managed System behavior and property specification

<table>
<thead>
<tr>
<th>MON</th>
<th>TUE</th>
<th>WED</th>
<th>THU</th>
<th>FRI</th>
<th>SAT</th>
<th>SUN</th>
</tr>
</thead>
</table>
| 25  | P1- Formal Models Lecture  
The Adaptive course starts.  
The students are introduced to formal modeling.  
The students get criteria for a managed system modeling. | 28  | P1- Proposals v.1  
Students provide domain examples with managed system description.  
The students get feedback concerning the adequacy of the examples to fit with the assignment criteria. | 30  | P1- Proposals v.2  
P2- F Managed Sys v.1  
Improved version of domain examples with managed system description and a formal model of the managed system and the environment’s behavior and property specification.  
The students get feedback w.r.t. to the design and assistance for formal modeling. |     |     |     |

---

### February. Use of MAPE-K templates for Self-adaptive systems

<table>
<thead>
<tr>
<th>MON</th>
<th>TUE</th>
<th>WED</th>
<th>THU</th>
<th>FRI</th>
<th>SAT</th>
<th>SUN</th>
</tr>
</thead>
</table>
| 4   | P2- F Managed Sys v.2  
Students provide a behavior and property specifications for managed system and environment. Students get feedback w.r.t. to design. | 6   | P3- F Managing Sys Lecture  
Introduction to Self-Adaptation, MAPE-K and MFT. Students formally specify the SA behaviors and properties. The assignment criteria are provided. |     |     |     |     |
| 7   | P2- F Managed Sys Submission  
Students have a completed formal model of the managed system and property specifications for their domain. | 11  | P3- F Managing Sys Proposal v.1  
Students present Managing System proposal. Clarifications w.r.t. MFT documentation. | 13  | P3- F Managing Sys Proposal v.2  
Students present Managing System proposal. Clarifications w.r.t. MFT documentation. |     |     |     |
| 18  | P3- F Managing Sys Design v.1  
Students bring version of Managing System design. Clarifications w.r.t. MFT documentation. | 20  | P3- F Managing Sys Design v.2  
Students bring version of Managing System design. Clarifications w.r.t. MFT documentation. |     |     |     |
| 25  | P3- F Managing Sys Design v.3  
Students bring version of Managing System design. Clarifications w.r.t. MFT documentation. | 27  | P4- Interview.  
The students submit their formal SAS.Interview session. |     |     |     |
| 28  | P5- Analyze Data  
Researcher studies assignments in order to analyze data w.r.t. the MFT reuse and the R.Q. Interviews are used to answer related R.Q. |     |     |     |     |     |     |
February. Use of MAPE-K templates for Self-adaptive systems.

<table>
<thead>
<tr>
<th>MON</th>
<th>TUE</th>
<th>WED</th>
<th>THU</th>
<th>FRI</th>
<th>SAT SUN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P2- F. Managed Sys v.2</td>
<td>Students provide a behavior and property specifications for managed system and environment. Students get feedback w.r.t. to design.</td>
<td>6</td>
<td>P3- F. Managing Sys Lecture</td>
<td>Introduction to Self-Adaptation, MAPE-K and MFT. Students formally specify the SA behaviors and properties. The assignment criteria are provided.</td>
</tr>
<tr>
<td>4</td>
<td>P2- F. Managed Sys v.2</td>
<td>Students provide a behavior and property specifications for managed system and environment. Students get feedback w.r.t. to design.</td>
<td>7</td>
<td>P2- F. Managed Sys Submission</td>
<td>Students have a completed formal model of the managed system and property specifications for their domain.</td>
</tr>
<tr>
<td>11</td>
<td>P3- F. Managing Sys</td>
<td>Students present Managing System proposal. Clarifications w.r.t. MFT documentation.</td>
<td>13</td>
<td>P3- F. Managing Sys Proposal v.1</td>
<td>Students present Managing System proposal. Clarifications w.r.t. MFT documentation.</td>
</tr>
<tr>
<td></td>
<td>Proposal v.2</td>
<td>Students present Managing System proposal. Clarifications w.r.t. MFT documentation.</td>
<td>18</td>
<td>P3- F. Managing Sys Design v.1</td>
<td>Students bring version of Managing System design. Clarifications w.r.t. MFT documentation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Students bring version of Managing System design. Clarifications w.r.t. MFT documentation.</td>
<td>20</td>
<td>P3- F. Managing Sys Design v.2</td>
<td>Students bring version of Managing System design. Clarifications w.r.t. MFT documentation.</td>
</tr>
<tr>
<td>25</td>
<td>P3- F. Managing Sys</td>
<td>Students bring version of Managing System design. Clarifications w.r.t. MFT documentation.</td>
<td>27</td>
<td>P4- Interview.</td>
<td>The students submit their formal SAS. Interview session.</td>
</tr>
<tr>
<td></td>
<td>Design v.3</td>
<td>Students bring version of Managing System design. Clarifications w.r.t. MFT documentation.</td>
<td>28</td>
<td>P5- Analyze Data</td>
<td>Researcher studies assignments in order to analyze data w.r.t. the MFT reuse and the R. Q. Interviews are used to answer related R. Q.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Students bring version of Managing System design. Clarifications w.r.t. MFT documentation.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C.12 Interview Questions

IQ1 - In this study you have specified a managed system that describes a solution composed of multiple nodes. Please provide a description of this managed system.

IQ2 - The managed system that you have specified in this study is in the [Domain-Name] domain. Can you confirm this?

With these two questions, we aim to collect information regarding data items that describe the domain in which the system is embedded. This data item is necessary to study the reusability of the MAPE-K Formal Templates in other domains than the domains from where the templates are originated (see RQ1).

Defining Base for comparison

IQ3 - How long (in hours) would you say that you required to define a managed system to study? (This phase is prior to specification)

IQ4 - How long (in hours) would you say that you required to specify your managed system?

With these two questions we aim to collect information regarding the level of complexity in using formal methods, as a base for comparison to study the complexity of using the MAPE-K Formal Templates (see RQ3 and RQ5).

Difficulty to Understand

IQ5 - Once you started studying the MAPE-K Formal Templates, how would you define the level of difficulty to understand the MAPE-K Formal Templates?

With IQ5 we aim to collect information regarding the level of reusability of the MAPE-K Formal Templates, and more concretely, to study the level of expressiveness of them (see RQ2, RQ4).

Difficulty of Usage (usability problems)

IQ6 - How would you define the level of difficulty to apply the MAPE-K Formal Templates to specify a Self-Adaptation solution for your managed system?

With IQ6 we aim to collect information regarding the level of reusability and utility of the MAPE-K Formal Templates, and more concretely, to study the level of expressiveness of them.
C.12 Interview Questions

Domain

IQ1 - In this study you have specified a managed system that describes a solution composed of multiple nodes. Please provide a description of this managed system.

IQ2 - The managed system that you have specified in this study is in the [Domain-Name] domain. Can you confirm this?

With these two questions, we aim to collect information regarding data items that describe the domain in which the system is embedded. This data item is necessary to study the reusability of the MAPE-K Formal Templates in other domains than the domains from where the templates are originated (see RQ1).

Defining Base for comparison

IQ3 - How long (in hours) would you say that you required to define a managed system to study? (This phase is prior to specification)

IQ4 - How long (in hours) would you say that you required to specify your managed system?

With these two questions we aim to collect information regarding the level of complexity in using formal methods, as a base for comparison to study the complexity of using the MAPE-K Formal Templates (see RQ3 and RQ5).

Difficulty to Understand

IQ5 - Once you started studying the MAPE-K Formal Templates, how would you define the level of difficulty to understand the MAPE-K Formal Templates?

With IQ5 we aim to collect information regarding the level of reusability of the MAPE-K Formal Templates, and more concretely, to study the level of expressiveness of them (see RQ2, RQ4).

Difficulty of Usage (usability problems)

IQ6 - How would you define the level of difficulty to apply the MAPE-K Formal Templates to specify a Self-Adaptation solution for your managed system?

With IQ6 we aim to collect information regarding the level of reusability and utility of the MAPE-K Formal Templates, and more concretely, to study the level
C Protocol for Case Studies: Reusability of MAPE-K Formal Templates

of usability of the templates to be applied for a specific domain, including the Formal Behavior and the Property specifications (see RQ2 and RQ4).

IQ7- Which units of the MAPE-K Formal Templates did you decide to reuse for the specification of your self-adaptation solution?

With this question we aim to collect information regarding the level of reusability and utility of the MAPE-K Formal Templates, and more concretely, to study the different units that have been considered for a particular self-adaptation specification in order to identify the different types of reuse and refinement that have been applied (see RQ3 and RQ5).

Complexity of Usage (task time)

IQ8- How long (in hours) would you say that you required to specify your managing system?

With IQ8 we aim to collect information regarding the level of reusability of the MAPE-K Formal Templates, and more concretely, to study the complexity in applying the templates by comparing the efforts to design the managed system (see IQ3 and RQ3 and RQ5).

Reusability Benefits (level of satisfaction)

9- Do you consider the MAPE-K Formal Templates to provide any benefits for the design of self-adaptive systems? Could you describe the benefits?

10- Did you find any behavior (or behaviors) templates or property (or properties) particularly useful for the design of your self-adaptive solution?

11- Based on your experience, would you apply the MAPE-K Formal Templates again in the future? For which purpose?

12- Do you consider that using the MAPE-K Formal Templates would reduce the amount of time required for code implementation?

With IQ9, IQ10, IQ11 and IQ12 we aim to collect information regarding the scope of capabilities of the MAPE-K Formal Templates to support the design for self-adaptive systems (considering reduction of implementation costs, assuring software quality, reusability, etc.) (see RQ2, RQ3, RQ4 and RQ5).
C.13 Propositions

| 13 | Which problems did you face during the usage of the MAPE-K Formal Templates? What would you change after having experienced with them? |

With IQ13 we aim to collect information regarding the scope of capabilities of the MAPE-K Formal Templates to support the design for self-adaptive systems, focusing on the negative aspects of them (see RQ2 and RQ4).

Additional Information

| 14 | Do you have additional comments that you consider relevant to mention for this study? |

With IQ14 we aim to collect additional information that otherwise may me lost through the process of the semi-structured interview.

C.13 Propositions

Research propositions concerning the reusability of the Behaviors Specifications Patterns:

H2: "The BSP contribute on the specification of the self-adaptive systems behavior for a managed system in a domain at hand."

RP2.M.1: IF the user requires to specify a monitor behavior triggered by an event, THEN the monitor model contains a signal originated from a probe component.

RP2.M.2: IF the user requires to specify a monitor behavior triggered on a period basis, THEN the monitor model contains a signal originated from a probe component.

RP2.M.3: IF the user requires to specify a monitor component behavior, THEN the monitor behavior contains a cyclic loop between Waiting state and a NewChange that can intersperse a PreProcessing state from Waiting to NewChange.

RP2.M.4: IF the user requires to specify a monitor and a preprocessing-data phase is required, THEN the user can refine the monitor template and exists a transition between a Waiting state and NewChange state that performs a preprocessing data.

RP2.A.1: IF ...

other (equivalent propositions are defined for the rest of the Behavior Specification Patterns)

Research propositions concerning the reusability of the Property Specification Patterns:
Appendix D

MAPE-K Formal Templates Documentation

In this Appendix we provide the formal documentation for a set of MAPE-K Formal Templates for the design and specification of self-adaptive systems. The MAPE-K Formal Templates target software systems that are characterized by the following properties (target domain):

- systems comprise software deployed on distributed nodes;
- the nodes have an explicit position in the environment (and may be mobile);
- the system has continual communication access;
- dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software;
- resources in the system may come and go at will.

Within the MAPE-K Formal Templates, we provide formal specifications for the behavior of different components in the MAPE-K control loop (Monitor, Analyze, Plan, and Execute, complemented with Probes and Effectors) and formal property specifications to support the verification of the correctness of the adaptation behaviors.

H3: "The PSP contribute on the property specification and verification of the self-adaptive systems behavior for a managed system in a domain at hand."

RP3.G.1: IF the user requires to specify a property to describe a correct transition from Unsatisfied concern to Satisfied, THEN the user can refine the property P1 for such purpose.

RP3.A.1: IF the user requires to specify a property to describe the correct detection of Undesired system states, THEN the user can refine the property P2 to verify the Analyze correctness for such purpose.

RP3.P.1: IF the user requires to specify the correctness of the Plan behavior in order to satisfy a self-adaptation concern, THEN the user can refine the properties P3 and P4 for this purpose.

RP3.E.1: IF the user requires to specify a property that describes the execution of a Plan process of adding a resource in the system, THEN the user can refine the property P5 for such purpose.

RP3.E.2: IF the user requires to specify a property that describes the execution of a Plan process of removing a resource from the system, THEN the user can refine the property P6 for such purpose.

other (equivalent propositions are defined for the rest of the Property Specification Patterns)
Appendix D

MAPE-K Formal Templates Documentation

In this Appendix we provide the formal documentation for a set of MAPE-K Formal Templates for the design and specification of self-adaptive systems. The MAPE-K Formal Templates target software systems that are characterized by the following properties (target domain):

- systems comprise software deployed on distributed nodes;
- the nodes have an explicit position in the environment (and may be mobile);
- the system has continual communication access;
- dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software;
- resources in the system may come and go at will.

Within the MAPE-K Formal Templates, we provide formal specifications for the behavior of different components in the MAPE-K control loop (Monitor, Analyze, Plan and Execute, complemented with Probes and Effectors) and formal property specifications to support the verification of the correctness of the adaptation behaviors.
MAPE-K Formal Templates for Self-Adaptive Systems: Specifications and Descriptions

Abstract

In this document we provide the semantics for a set of MAPE-K Formal Templates for the design and specification of self adaptive systems. The MAPE-K Formal Templates target software systems that are characterized by the following properties. (i) Systems comprise software deployed on distributed nodes; (ii) The nodes have an explicit position in the environment (and may be mobile); (iii) The system has continual communication access; (iv) Dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software.

The MAPE-K Formal Templates allow the design of feedback control loops that follow the MAPE-K architectural approach. Therefore, the MAPE-K Formal Templates make distinction between the different behaviors of the Monitor, Analyze, Plan and Execute components. The MAPE-K Formal Templates consider self-adaptive systems that make use of time-triggered and event-triggered behavior approaches and multiple instantiation of MAPE-K components. The MAPE-K Formal Templates comprise: (1) Behavior Specifications Patterns for modeling the different components of a control loop, and (2) Property Specifications Patterns that support verification of the correctness of the adaptation behaviors.

We use timed-automata for the Behavior Specifications Patterns and timed-computational tree logic (TCTL) as the formal language for Property Specification Patterns. We use the UPPAAL graphical representation for the templates described in this document, and as a tool for the design of the behaviors of self-adaptive systems.

D.1 Introduction

We focus on architecture-based self-adaptation [70, 121, 143, 187]. In architecture-based self-adaptation, the system reasons about an abstract model of itself and adapts when needed according to some adaptation goals. The control loop typically consists of Monitor, Analyze, Plan, and Execute components, complemented with models (Knowledge) of the managed system, its environment and adaptation goals. This structure is referred to as MAPE-K [113].

One important challenge in engineering self-adaptive systems is to provide evidence that the system goals are satisfied, regarding the dynamically changing operating conditions [48, 127, 128]. To that end, state of the art in architecture-based self-adaptation advocates the use of formal methods. A recent systematic
D.1 Introduction

literature review [188] identified a total of 75 papers that use formal methods in self-adaptive systems. Studying the state of the art reveals that there is little reuse on defining behavior models and required properties for the self-adaptive system for specific problem or domains. There is no systematic consolidation of the design knowledge for future use. Other researchers have demonstrated the value of reusable templates for the specification and verification of formal models, e.g., [173] formally specifies models or reusable patterns, and [88] defines reusable properties for the specification of probabilistic requirements. [65] points out that the definition of such reusable templates are an important research challenge for self-adaptive systems.

During the past few years, we have been studying various self-adaptive systems, with a particular focus on self-adaptation to realize robustness and openness. Some of the recent applications we studied are an intelligent monitoring system to detect traffic jams [104], a mobile system to support outdoor learning activities [81], and robotic systems to perform transportation tasks. In the course of designing and building the self-adaptive applications, we derived a set of reusable templates for designing MAPE-K based control loops of self-adaptive systems.

To help designers with formulating formal models and properties effectively and correctly, this paper contributes with a set of reusable templates that document the design expertise for a family of self-adaptive software systems. The templates comprise: (1) generic behavior templates for modeling the different components of a control loop and their interactions (the behavior templates are specified in networks of timed automata), and (2) abstract specification properties that support verification of the correctness of the adaptation behaviors (the properties are specified in timed computation tree logic). We illustrate the use of the templates with excerpts of architectural designs of two applications.

The templates are derived from designing a family of self-adaptive systems (target domain) with the following characteristics:

- Systems comprise software deployed on distributed nodes;
- The nodes have an explicit position in the environment (and may be mobile);
- The system has continual communication access;
- Dynamics in the system and the environment are orders of magnitude lower than the speed of communication and the execution of the software;
- Resources in the system may come and go.

We study self-adaptive systems based on MAPE-K control loops to realize robustness (i.e., the system deals autonomously with particular types of faults) and openness (the system deals autonomously with parts that come and go dynamically). To that end, local control loops can observe local managed systems and their execution context and adapt the managed systems by adding and removing resources. Resources are abstractly defined and refer to any type of controllable feature of the managed system, such as components, functions, variables, devices, etc.
D.2 Preliminaries

Here, we present preliminary information that provides the underpinnings to understand the provided templates. The section is divided in three parts. In the first part, we provide an overview of the MAPE-K control loop, in order to set the bases in which the formal templates are constructed. In the second part, we present relevant information with respect to the formal languages we use for the specification of the MAPE-K Formal Templates. In a third part, we elaborate on the different types of behavior interactions that can be specified and that relate to the behaviors found in a feedback control loop using MAPE-K.

D.2.1 MAPE-K

The MAPE-K structural design for self-adaptive systems was introduced by IBM in 2003 [101] and further developed in 2006 [102]. MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) contains five components for the design of managing systems, and that provide the adaptive capabilities that a self-adaptive system possess. A Monitoring component (represented as the circle 2 in Fig. D.1) acquires information from the system and the environment in which this is found. The Monitor serializes this information, via updating the basis of knowledge represented in (1), to be transferred to an Analyze component (3). Based on this base of knowledge (1), the Analyze component is responsible of determining the correctness of the managed system state with respect to one or more desired quality goals. Subsequently, the Plan component (4) designs a set of action plans to Execute (5) in order to direct the managed system to a known desired behavior state.

A MAPE-K control loop may use the knowledge base to provide representations of the managed system, the environment where the system is located, goals that the system is desired to achieve and information with respect to the self-adaptation mechanism itself (in order to save working data and to allow learning mechanisms or self-adaptation).

![Figure D.1: MAPE-K. Model for self-adaptive systems [102]](image)

The Fig. D.1 shows two additional components, sensors and actuators, that become the interfaces for the managing system to interact with the managed system.
In this work, we use the term "probes" to refer to sensor components and "effectors" to refer to actuator components. Probes allow access to (sense) relevant information from the environment and the managed system, in order to feed the required knowledge for self-adaptation decisions. On the other side, the Effectors are in charge of applying (actuate) the planned self-adaptation actions into the managed system, in order to direct the system towards a desired state.

D.2.2 Formal Languages

Formal languages allow the rigorous specification of system’s behavior using a mathematical representation. The MAPE-K Formal Templates, that are focus of this work, are use formal languages for the specification of the behavior of the self-adaptive mechanisms. In this subsection we focus on two specific formal languages. We use timed-automata for behavior specification and timed-computational tree logic for property specification.

D.2.2.1 Timed Automata

In a recent study in the use of formal methods for self-adaptive systems [188], automata was identified as one of the most used languages. One type of automata is known as timed automata. A timed automaton is a finite-state machine extended with clock variables that models a behavior. Clock variables are used to synchronize behaviors.

A timed automaton is a tuple \((N, l_0, T, Label, C, clock, guard, invariant)\) [19, 123] with:

\[
\begin{align*}
N &= \text{a non-empty, finite set of locations (or nodes) with an initial location } l_0 \in N; \\
T &\subseteq L \times L \quad \text{a set of transitions;} \\
Label : N &\rightarrow 2^{AP} \quad \text{a function that assigns to each location } l \in N \text{ a set Label}(l) \text{ of atomic positions;} \\
C &= \text{a finite set of clocks;} \\
clock : T &\rightarrow 2^C \quad \text{a function that assigns to each transition } t \in T \text{ a set of clocks clocks}(t); \\
guard : T &\rightarrow \Psi(C) \quad \text{a function that labels each transition } t \in T \text{ with a clock constraint guard}(t) \text{ over } C; \\
inv : N &\rightarrow \Psi(C) \quad \text{a function that assigns to each node an invariant.}
\end{align*}
\]

We use networks of timed-automata to model the component behaviors of control loops. The network of automata allows the specification of multiple behaviors (such as behaviors for Environment, parts of a Managed System, Monitor, Analyze, Plan and Execute components) independently, in order to provide a rigorous description of the overall self-adaptive system behavior. The multiple behaviors specified for a self-adaptive system may require certain level of synchronization. Automata can communicate through channels, where the sender behavior can send signals \(S\) (represented by the exclamation sign: \(channel\)) in order to synchronize
with the receiver behavior listening for signal $S$ (represented by the question mark sign: $\text{channel}?$).

In this work, we use UPPAAL [15] to model the timed-automata. In UPPAAL behavior specifications can be complemented with expressions specified in a C-like language to define data structures (struct concept) and functions.

We follow UPPAAL symbol conventions for the description of the behavior templates, presented in Table D.2.

**Table D.2: Conventions in Timed Automata figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State A</td>
<td>State</td>
<td>States of behaviors are represented by circles and (optionally) annotated in red on top of the associated state.</td>
</tr>
<tr>
<td></td>
<td>Committed State</td>
<td>Committed states are represented with circles containing the $c$ character. Committed states must be left without time consumption.</td>
</tr>
<tr>
<td></td>
<td>Urgent State</td>
<td>Urgent states of behaviors are represented with circles containing the $u$ character. Urgent states must be left as soon as exiting conditions are found (normally defined by conditions on outgoing transitions).</td>
</tr>
<tr>
<td></td>
<td>Initial state</td>
<td>Initial States of behaviors are represented by double-lined circles. There must be one unique Initial State per automaton, specifying the behavior state when the system starts.</td>
</tr>
<tr>
<td>$t&lt;=\text{Period}$</td>
<td>Invariants</td>
<td>Invariants that need to be satisfied in certain states are annotated in purple under the related state.</td>
</tr>
<tr>
<td></td>
<td>Transition</td>
<td>Transitions between two states are represented by directional arrows, showing the origin and destination of the transition.</td>
</tr>
<tr>
<td></td>
<td>Conditions</td>
<td>Conditions to enable transitions between states are annotated in green under the related transition.</td>
</tr>
<tr>
<td></td>
<td>Signal</td>
<td>Signals used for communication between behaviors are annotated dark blue over the associated transition.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>Functions associated with behaviors are annotated in light blue under the associated transition.</td>
</tr>
</tbody>
</table>

Fig. D.2 shows a timed automaton illustrates the usage of the specification items presented in Table D.2. The automaton specifies one part of the behaviors for a GPS-based system. In particular, this behavior specification describes an abstraction of the context where the system is found, a "sky behavior simulation".

In the automaton, we can identify a cyclic behavior that loops between Clear and Cloudy sky conditions, that may influence the quality of signal that a GPS resource could receive. Beside these two main states, an additional main state Initializing is presented in the behavior. The Initializing state describes an opening state of the system where system parameters and functions are being initialized. This process is (only) required at start time. The Initializing state is represented as an initial state (double-lined), and can only be left when the isSystemReady() function()
This process is (only) required at start time. The Initializing state is represented by the two main states Clear and Cloudy. In the automaton, the first step is performed via a function that gathers the time stamp for the next Clear and Cloudy sky states. This is achieved through the getNextCloudy() and getNextClearing() functions, that set the nextCloud and nextClearing variables $^1$ respectively. The second step is represented by the two main states Clear and Cloudy. In the third step, the automaton defines the state outgoing condition, in order to allow changes between Clear and Cloudy states only after the defined time-values is reached. This is achieved by the timer $\geq$nextCloud condition to leave the Clear state.

The process described above specifies the behavior of the sky conditions for a concrete system. This behavior interacts Sky conditions can affect the level of accuracy in a GPS module. Therefore, the automaton that represents the sky behavior interacts with the automaton for a GPS module behavior (represented in Fig. D.3). This interaction is achieved via channels. Two specific channels are defined to allow signaling for both improving sky conditions (qualitySkyUp) and diminish sky conditions (qualitySkyDown). This automaton sends a signal through the corresponding channel when the sky state changes (i.e. qualityGPSModuleUp[Pid]!).

![Figure D.2: Example of automaton with sender behavior. Sky environment](image)

**Figure D.2:** Example of automaton with sender behavior. Sky environment

condition is satisfied, which specifies the completion of initialization processes. Notice that the initial state has been specified as urgent, in order to start a "sky behavior simulation" as soon as the system becomes ready. The behavior illustrates a loop mechanism that is time-based. To implement this mechanism, three main steps are required. First, we gather the time-value for the next event; second, we wait until the time-value is reached; and third, we perform a transition towards the next sky state. In the automaton, the first step is performed via a function that gathers the time stamp for the next Clear and Cloudy sky states. This is achieved through the getNextCloudy() and getNextClearing() functions, that set the nextCloud and nextClearing variables $^1$ respectively. The second step is represented by the two main states Clear and Cloudy. In the third step, the automaton defines the state outgoing condition, in order to allow changes between Clear and Cloudy states only after the defined time-values is reached. This is achieved by the timer $\geq$nextCloud condition to leave the Clear state.

The process described above specifies the behavior of the sky conditions for a concrete system. This behavior interacts Sky conditions can affect the level of accuracy in a GPS module. Therefore, the automaton that represents the sky behavior interacts with the automaton for a GPS module behavior (represented in Fig. D.3). This interaction is achieved via channels. Two specific channels are defined to allow signaling for both improving sky conditions (qualitySkyUp) and diminish sky conditions (qualitySkyDown). This automaton sends a signal through the corresponding channel when the sky state changes (i.e. qualityGPSModuleUp[Pid]!).

![Figure D.3: Example of automaton with receiver behavior. GPS module](image)

**Figure D.3:** Example of automaton with receiver behavior. GPS module

Fig. D.3 shows the specification of the receiver behavior using timed automata for a GPS module description. The model has been designed to listen for the two distinct signals that determine whether the GPS module is Improving or Deteriorating its GPS accuracy, based on the behavior described for the "sky behavior simulation". In this case, the behavior listens (i.e. qualityGPSModuleUp[Pid]?)

$^1$ In the Declarations, we define variables with small letters and Constants starting with a capital letter.

195
for signals arriving through the (qualitySkyUp) and (qualitySkyDown) channels, and updates an internal representation of the GPS quality through the increaseQuality() and decreaseQuality() functions.

### D.2.2.2 Timed Computational Tree Logic

UPPAAL supports the design of property specification using timed-computational tree logic (TCTL). TCTL is a formal language for property specification based on CTL extended with clock variables.

The syntax supported by UPPAAL for property specification is defined as follows:

$$\phi ::= p \mid !\phi \mid \phi \wedge \phi \mid EX \phi \mid E[\phi \cup \phi] \mid A[\phi \cup \phi] \mid z.\phi$$

Where:

- $\phi$ is a property to be specified;
- $p$ is an atomic proposition or a clock constraint;
- $EX$ is an expression applied on a property;
- $E$ expresses the existence of a path that fulfills a property;
- $A$ expresses the invariant fulfillment of a property;
- $z$ expresses a state predicate.

Below, we explain the different symbols that can be used in the TCTL syntax.

- $E[p]$ There is a path in which $p$ will always hold. For example, there is a path where a GPS module has always NULL coordinates. This is a broken during manufacturing GPS module.
- $E <> p$ It is possible to reach a state in which $p$ is satisfied. For example, a GPS module will eventually acquire position coordinates.
- $A[p]$ $p$ holds invariantly. For example, location coordinates are always on earth.
- $A <> p$ $p$ is inevitable. It will eventually happen. For example, a GPS will eventually fail in gathering accurate positions.
- $A \rightarrow B$ If $A$ becomes true, then $B$ will eventually be true. For example, if a GPS module turns on, it will eventually collect GPS coordinates.
- $A \rightarrow B$ If $A$ becomes true, $B$ will become true at the same time. For example, if a GPS acquires location coordinates, then some accuracy values with respect to the location coordinates are obtained.

More information with respect to the property specification language, and TCTL related aspects can be found at [19, 123].

### D.2.3 Behavior Interactions

Commonly, components in a system may have dependencies, therefore their behaviors may require certain interactions to be specified. In this subsection we describe different types of interactions that can be specified using formal languages.
D.2 Preliminaries

D.2.3.1 Time and Event Triggering approaches

In timed automata, transitions between states can be taken via event-triggering or time-triggering.

With an event-triggering mechanism one automaton triggers another via a signal sent through a channel. This case is illustrated in Fig. D.4-left, where an automaton with a behavior $B1$ fires a signal in order to trigger a transition on the behavior $B2$. In this case, we say that the second behavior $B2$ is dependent on $B1$, as this would not be able to execute transitions without the corresponding signal from the former. Optionally, data generated in a behavior $B1$ may be transferred to a behavior $B2$ using a knowledge repository.

![Diagram of Event- (left) and Time- (right) triggered interaction between components](image)

**Figure D.4:** Event- (left) and Time- (right) triggered interaction between components

For example, a Monitor behavior may directly trigger an Analyze behavior via a signal (see Fig. D.5-left). Event-triggering is in general efficient with respect to verification.

With time-triggering mechanisms, a transition is fired based on state invariants and time conditions. This case is shown in Fig. D.4-right, where a behavior $B2$ autonomously executes transitions in the automaton on a time-based approach (Tick). Due to the autonomy of the behavior with respect to another $B1$ behavior, time triggering requires a data repository to store shared knowledge whenever information needs to be transferred between behaviors. Fig. D.5-right presents an example of a time-triggering mechanism for an Analyze behavior. In this case, a Monitor behavior updates the Knowledge repository and the Analyze behavior periodically reads the Knowledge to determine whether there is a need for adaptation actions.

Time-triggered approaches may be interesting to apply in a behavior $B2$ for cases where:

1. Relevant number of events are being processed in a behavior $B1$ and do not need to be immediately processed in $B2$ (such as number of "B1-options sold", and "B2-price per option recalculation" on a stock exchange day), and
2. When a behavior $B2$ requires to perform a set of actions in a number of steps (see an example in Analyze-Plan interactions in Sections D.3.4 and D.3.5)

In general, time-triggering is less efficient in terms of verification (as it implies more execution threads).
D MAPE-K Formal Templates Documentation

The interactions presented in Fig. D.5 illustrate the two type of triggering approaches applied to an interaction between two MAPE behaviors. However, these approaches can as well be applied to the remaining components in the MAPE-K control loop.

D.2.3.2 Active and Reactive Approaches

We find the following pair of behavior interactions in a MAPE-K control loop: Probe-Monitor, Monitor-Analyze, Analyze-Plan, Plan-Execute and Execute-Effector (see Fig. D.5). Within the interaction between two MAPE-K components, one can have an active role (with respect to the interaction), which would be responsible to initiate interactions between the two involved components, while the second will have a reactive role for the same interaction. An interaction is **Active-Reactive** when the first behavior has an active role in the initiation of the interaction and the second behavior has a reactive role. Fig. D.6-left side shows an Active-Reactive interaction illustrated by an arrow from the Probe to the Monitor components. In this particular case, the Probe component is in charge of initiating a communication towards the Monitor component and provide new sensed data in the managed system. An interaction is **Reactive-Active** when the second behavior has an active role in the initiation of the interaction and the first behavior has a reactive role. Fig. D.6-right side shows an Reactive-Active interaction illustrated by an arrow from the Monitor to the Probe component. In this particular case, the Monitor component, is in charge of initiating a communication towards the Probe component (for example given a time-triggered approach implemented on the Monitor) and requests the Probe to sense current data values in the managed system and to communicate them back to the Monitor component.

We use Active-Reactive interactions for the design of the **MAPE-K Formal Templates**. However, the templates may be refined in order to design Reactive-Active interactions between components, in order to request probe actions from a Monitor, or demand additional monitoring from an Analyze behavior, to provide some examples.

198
D.3 Behavior Specification Patterns

We now present the Behavior Specification Patterns that support the formal specification of behaviors for the MAPE components. We differentiate between the Monitor, Analyze, Plan and Execute components, as the main components of a MAPE-K control loop. Additionally, we provide a simple formalization for knowledge representations, as a basis required for multiple behavior process, such as data transfer between behaviors and analysis processes.

MAPE components of the control loop require points of interaction with the managed system. These are commonly realized through Probe and Effector components. Therefore, we additionally provide a high level of abstraction of behavior specifications for Probe and Effector components. It is important to mention that Probe and Effector components are highly dependent on the managed system design, therefore they become domain dependent and may require deep customizations for the targeted system.

We describe the Behavior Specification Patterns by elaborating on the states, transitions, functions and data structures and signals for synchronization that compose them.

D.3.1 Knowledge

Relevant efforts have studied a formal representation of Knowledge for self-adaptive systems [70, 128, 187]. The MAPE-K Formal Templates presented in this work do not elaborate on formalizations for Knowledge representations. However, the self-adaptation behaviors require knowledge representations to provide an abstraction of the environment, managed system, goals and the managing system in order to perform the adequate adaptation actions. We provide a basic specification for knowledge.

D.3.1.1 Typologies of Knowledge

In line with We follow the four type of knowledge representations defined in FORMS [187]. Knowledge representations are technically specified as four struct definitions.
D.3 Behavior Specification Patterns

- **ManagedSystemKnowledge** provides an abstraction of the managed system. ManagedSystemKnowledge represents the relevant information regarding the resources in the managed system. This information may contain data i.e. to define whether the resource is used or not and other resource parameters, such as probabilistic models. The following Declaration D.1 is an snippet for the declaration of the Managed System Knowledge representation. The declaration considers a system with multiple Nodes that can belong to organizations and that provide certain resources that may be used for the application at hand. For example, Nodes may be mobile devices with GPS resources. For each resource, relevant information may be considered, such as the typology of the resource and the quality of the service offered. Notice that the declaration allows the specification of a number of nodes.

```c
const int Nodes = <define>; //number of nodes to be found in the System
const int NRes = <define>; // number of existing resources
typedef struct {
    int resourceType;
    int resourceQuality;
}Resource;
Resource resources[NRes];

struct {
    int usedResources; //Summary of used resources
    int availableResource[NRes]; // Detail of used (and not used) resources
    int usedNode; //does the node belong to a group or not
}ManagedSystemKnowledge[Nodes];
```

**Declaration D.1:** Code for Managed System Knowledge specification

- **EnvironmentKnowledge** provides an abstraction of the environment. That is, this knowledge represents information about the context of the self-adaptive system. The following Declaration D.2 offers a snippet of the Environment-Knowledge. The environment declaration is domain specific, and my require different dimensions for each system at hand. Therefore, it may be required to define multiple environment properties and parameters that may be required for self-adaptation purposes. We allow the declaration of an Environment for each one of the nodes in a system.

```c
typedef struct {
    int param1;
    int param2;
}Property;
typedef struct {
    Property prop1;
    int paramA;
    int paramB
}Environment;
```

**Declaration D.2:** Code for Environment Knowledge specification
D.3 Behavior Specification Patterns

```c
struct {
    Environment environment;
} EnvironmentKnowledge[Nodes];
```

**Declaration D.2:** Code for Environment Knowledge specification

- **ConcernKnowledge** describes the knowledge w.r.t. the adaptation concern of interest; this part specifies the `requiredResources` to realize the goals of the self-adaptive system. The following Declaration D.3 describes the structure for `ConcernKnowledge`. For each concern, two parameters are defined, to specify the number of resources that are required for that specific concern, and the quality of the services that the selected resources should hold. Notice that the structure may allow the declaration of 0 or more concerns.

```c
const int NConcerns = <define>; //Number of concerns to be handled by the Self-Adaptive System
typedef struct {
    int requiredResources;
    int requiredQuality;
} Concern;
typedef struct {
    Concern concern[Concern_ID];
} ConcernKnowledge[NConcerns];
```

**Declaration D.3:** Code for Concern Knowledge specification

- **AdaptationKnowledge** represents runtime data that is shared between the MAPE behaviors; we distinguish between `flags` that are used for indirect synchronization of behaviors and `workingData` that represents data that behaviors use to realize their functions (e.g., historical data for analysis, workflows for adaptation, etc.).

```c
typedef struct {
    bool relevant;
    int monitorID; //ID for related Monitor(s)
    int param1;
    int param2;
} ProbeWD; //Probe related Working Data
typedef struct {
    bool MonitorPreProcessNeeded;
    int param1;
} MonitorWD; //Monitor related Working Data
typedef struct {
    int param1;
} AnalyzeWD; //Analyze related Working Data
const int Pending = -1;
const int Requested = 0;
const int Performed = 1;
```
D.3 Behavior Specification Patterns

87
88
89
90
91
92
93
94
95
96
97
98
99
100
101

// Setter to add an action in a Plan
void addAction(int node, int plan, int actionType, int param1, int param2){
    int action;
    AdaptationKnowledge[node].planWD.taskInPlan ++;
    action = AdaptationKnowledge[node].planWD[plan].taskInPlan
    AdaptationKnowledge[node].planWD[plan].action[action].type = actionType;
    AdaptationKnowledge[node].planWD[plan].action[action].param1 = param1;
    AdaptationKnowledge[node].planWD[plan].action[action].param2 = param2;
}

// Setter to reset the Actions in a Plan
void resetActions(int node, int plan){
    AdaptationKnowledge[node].planWD[plan].taskInPlan=0;
}

Declaration D.5:
Code for Concern Knowledge specification

D.3.2 Probe behavior pattern

The Probe is in charge of sensing specific items from the managed system, the goal definitions or the environment (such as quality parameters, service values, current deployment values, etc.) and to transfer this knowledge to the corresponding Monitor behavior. The Probe may require a domain specific logic to perform the sensing role. For example, some sensing criteria may be defined in order to determine when the Monitor component must be notified, such as thresholds for notification of changes in the managed system and some other filtering criteria.

Figure D.7:
Probe behaviors patterns with event- (top) and time- (bottom) triggering approaches

D.3.1.2 Functions

Given the Knowledge data structure formal representations, we define a set of functions that allow the manipulation of their content. Knowledge management functions are classified based on their functionality between getters and setters. The getters are responsible of setting knowledge content for the multiple data structures in each of in the four knowledge typologies.

Below, we present an reduced number of functions that illustrate the possible data manipulations for knowledge representations.

// Setter for a Resource module
void setResourceUsed(int node, int resource, int status){
    ManagedSystemKnowledge[node].availableResource[resource] = status;
}

// Getter for a Resource module
int getResourceUsed(int node, int resource){
    return ManagedSystemKnowledge[node].availableResource[resource];
}

Declaration D.4: Code Adaptation Knowledge specification
D.3 Behavior Specification Patterns

D.3.2 Probe behavior pattern

The Probe is in charge of sensing specific items from the managed system, the goal definitions or the environment (such as quality parameters, service values, current deployment values, etc.) and to transfer this knowledge to the corresponding Monitor behavior. The Probe may require a domain specific logic to perform the sensing role. For example, some sensing criteria may be defined in order to determine when the Monitor component must be notified, such as thresholds for notification of changes in the managed system and some other filtering criteria.

Figure D.7: Probe behaviors patterns with event- (top) and time- (bottom) triggering approaches
D.3.2.1 Instantiation

A Probe automaton specifies the behavior for a probe component. We design a probe component to be in charge of sensing one managed system or one environment variable or parameter. Therefore, it may become necessary to specify multiple Probe behaviors in order to sense all the relevant sources that are required for self-adaptation (one probe per source).

D.3.2.2 States

There are four main states in a Probe behavior. These are Waiting, Probing, Filtering and Notifying states.

Waiting. In a Waiting state, the Probe behavior waits for a triggering mechanism to initiate the process of sensing the required data. An event-triggered Probe is activated by an specific behavior in the managed system via the Probe[Red_ID] signal\(^1\). A time-triggered Probe periodically \((t=\text{Period})\) senses the current value of the variable under probe.

Probing represents states in which the probe component is in charge of sensing a specific source from the related managed system, goal definitions or environment source.

Filtering represents states in which the Probe does a pre-analysis on the sensed data before this is (potentially) notified to the related Monitor component. Filtering states may contain multiple criteria, such as predefined thresholds to obtain meaningful data (maximum and minimum variation in data) or relevant changes in the data mean or variance. Filtering becomes domain specific and requires specific logic implementation for each case.

Notifying represents states in which the Probe component transfers information with respect to the sensed data to the corresponding Monitor component, in order to evaluate and apply (if needed) required adaptation actions into the managed system.

D.3.2.3 Functions and Data Structures

The Probe automaton is complemented with a set of data structures and functions that are necessary for the design of the probe logic. The functions to be defined are domain specific and will require specific logic based on the managed system at hand. Below we present the rational to be implemented in each of the functions and the data structures that are involved in them.

probe() is in charge of sensing a specific source from the managed system, the goal specifications or the environment. This function is domain specific, as it requires access to specific data structures from the source. The probe() stores sensed data into the probe working data structure (probeWD), for further processing before it is notified to the related Monitor component.

\(^1\) \text{Node}_{\text{ID}}\) identifies a node in the system where the Monitor behavior is found.

204
D.3 Behavior Specification Patterns

void probe()
{
    //Define the domain specific functions to sense the variable under Probe
    AdaptationKnowledge[Node_ID].probeWD[Res_ID].param1 = ...;
}

Declaration D.6: Probe.probe() function

filterData() is in charge of preprocessing the sensed data in order to filter out values that are not relevant or are negligible for the self-adaptation process. The filtering data process is domain specific, and must consider aspects such as minimum and maximum boundaries for sensed data relevance and other domain specific criteria. In case the sensed data is considered to be meaningful for the self-adaptation process\(^2\), a flag (probeWD.relevant parameter) in the probe working data (probeWD) is required to be activated, in order to assist the preAnalyzeData() function.

const bool Relevant = true;
const bool Irrelevant = false;
void filterData () {
    //Define the domain specific functions to preprocess the sensed data
    //Set an internal flag to Relevant in order to identify the existence of
    //relevant data to be notified
    AdaptationKnowledge[Node_ID].probeWD[Res_ID].relevant = Relevant;
    //Set an internal flag to Irrelevant if no meaningful data was sensed
    AdaptationKnowledge[Node_ID].probeWD[Res_ID].relevant = Irrelevant;
}

Declaration D.7: Probe.probe() function

preAnalyzeData() determines the need of notifying the new sensed data to the related Monitor behavior. This function must return Relevant if the sensed data has successfully passed the filtering process. Otherwise, Irrelevant should be returned.

const bool Relevant = true;
const bool Irrelevant = false;
bool preAnalyzeData(){
    return AdaptationKnowledge[Node_ID].probeWD[Res_ID].relevant;
}

Declaration D.8: Probe.probe() function

getMonitorID() determines the Monitor behavior(s) that must be notified concerning the new sensed data. A Probe behavior may be related to one or more self-adaptation control loops, therefore a broadcast channel ID may be required in some cases.

int getMonitorID(){
    int ID;
}

\(^2\)Meaningful data for self-adaptation process may or may not imply self-adaptation actions to be executed.
D.3.3 Monitor behavior pattern

The Monitor collects information from the managed system and possibly the environment in order to update the Knowledge of the control loop. Before updating the Knowledge, the Monitor may preprocess the collected data. Preprocessing tasks may include conversions (e.g. boolean values to integer values), standardization of information (to transform the data into a suitable format for further analysis, such as conversions from absolute values to percentages), data aggregation (to aggregate data from multiple sources, aggregate historical data for statistical purposes, etc.) and data filtering (to avoid incorrect monitored data values or redundant data).

It is important to emphasize that, in our Behavior Specification Patterns, the Monitor is not in charge of interpreting collected data, as this role must be designed in the Analyze component.

Fig. D.8 shows the automata to specify the Monitor behavior with event- and time-triggering.

D.3.3.1 Instantiation

The Monitor is in charge of monitoring one single data item from either the managed system, the environment or the established requirements (we consider system with dynamic goals, such as changing system quality requirements). Therefore, one or more instantiations of a Monitor are required depending on the number of data items considered as sources for self-adaptation.
D.3.3 Monitor behavior pattern

D.3.3.1 Instantiation

Analyze

isPreProcessRequired()

preProcess()

Preprocessing

NewChange

Waiting

Monitor[Node_ID]

getData()

isPreProcessRequired()

Preprocessing

setMonitorFlag()

updateKnowledge()

Figure D.8: Monitor behaviors with event (top) and time (bottom) triggering approaches

D.3.3.2 States

The Monitor behavior can be in three main states: Waiting, Preprocessing and NewChange states.

Waiting. In a Waiting state, the Monitor waits for a trigger to gather data. An event-triggered Monitor is activated by the related Probe via the Monitor[Node_ID]? signal. A time-triggered Monitor periodically (t == Period) monitors the data provided by the probe component. When leaving the Waiting state, the monitored data is gathered through the getData() function.

Preprocessing. The Monitor includes a preprocessing data phase in case it is necessary. The preprocessing phase starts after having collected the monitored data, and prior to notification to an Analyze behavior. The Monitor behavior can visit the Preprocessing state one or more times for a single monitoring loop. This is, the behavior allows multiple iterations for data preprocessing. A preProcess() function is in charge of treating the monitored data and prepare it (such as standardization, data aggregation, etc.) for analyze purposes when required (evaluated by the isPreprocessingRequired() function).

NewChange. Finally, the Monitor notifies the Analyze with respect to the monitored updates. The notification mechanism can be specified either following an event-triggered approach (through the Analyze[Node_ID]! signal) or using the appropriate flags (through the setAnalyzeFlag() function). Depending on the Analyze logic, flags may not be required for time-triggered approaches.

D.3.3.3 Functions and Data Structures

The Monitor automaton is complemented with a set of data structures and functions that are necessary for the design of the monitoring logic. Below we present

1 Node_ID identifies a node in the system where the Monitor behavior is found.
the rational to be implemented in each of the functions and the data structures that are involved in them.

**getData()** is in charge of gathering the monitored data from the corresponding data sources. This function must collect data processed by the related Probe. Below we present an example of the `getData` function.

```java
void getData(){
    //Collects the data to be monitored from the corresponding Probe behavior
    Probe_ID = Monitor_ID; //We consider a 1-to-1 relation between Probe and Monitor
    AdaptationKnowledge[Node_ID].monitorWD[Monitor_ID].data1 = AdaptationKnowledge[Node_ID].probeWD[Probe_ID].data1;
    //Multiple sources may be required to gather monitored data.
}
```

**Declaration D.12:** `Monitor.getData()` function

**isPreProcessRequired()** is in charge of determining the need for preprocessing tasks, such as needs for data aggregation, filtering and standardization, for all the data sources that are under the domain of this monitoring behavior.

```java
bool isPreProcessRequired(){
    //Implement adequate functions for preProcessing need detection
    if (...) {
        flag = true;
    }
    return flag;
}
```

**Declaration D.13:** `Monitor.isPreProcessRequired()` function

**preProcess()** is in charge of preparing the monitored data for analyze purposes. The `preProcess` works with data inside the monitor working data (`monitorWD`) in the `AdaptationKnowledge`. This function does not require input and output data structures, but only manipulates information in the `monitorWD`.

```java
void preProcess(){
    //Do filtering if needed
    if (data1...) { data1 = ...}
    //Do data aggregation if needed
    if (data1...) { data1 = ...}
}
```

**Declaration D.14:** `Monitor.preProcess()` function

**updateKnowledge()** is in charge of transferring the monitored data to the Analyze. To do so, this function works with both the monitor and analyze working data (`monitorWD` and `analyzeWD`). This update should consider all relevant monitored data to be required by the Analyze.
D.3 Behavior Specification Patterns

setAnalyzeFlag() is in charge of notifying the related Analyze with respect to the need for an analysis, via rising the related flag and only for time-triggered Analyze behaviors. Depending on the Analyze logic, flags may not be required for time-triggered approaches.

reset() is in charge of resetting the timer counter \( t \) prior to enter a state with a time-based invariant.

D.3.4 Analyze behavior pattern

The Analyze behavior is responsible to determine whether adaptation actions are required. The need for adaptation is based on the current state of the managed system, the environment and the adaptation concerns of interest. Concretely, an Analyze behavior matches the required resources with the resources in use, taking into account the current context and working data. Analysis can result in three primary conclusions. The current situation (behavior and/or deployment of the managed system for current environment and goals to achieve) is Satisfied when the managed system possesses the resources it requires to realize its goals. The situation is Undersatisfied when the system lacks resources, and Oversatisfied when the system has redundant resources. Fig. D.9 shows the Behavior Specification Patterns for Analyze behaviors with event-triggering (Analyze[Node_ID]?) and time-triggering (\( t==\text{Period} \)) approaches.

D.3.4.1 Instantiation

The Analyze behavior is in charge of analyzing the completeness of the system with respect to a set of desired system requirements and to apply achieve the needed adaptation concern. We consider the need of one Analyze behavior that focuses on one specific adaptation concern.
D.3.4.2 States

We identify four main states in an Analyze behavior. These are the Satisfied, Oversatisfied, Undersatisfied and Analyzing states.

Satisfied specifies states in which the managed system fulfills the requirements that are defined for a specific moment (we consider dynamic requirements). In other words, the system is analyzed as Satisfied if the number of required resources matches the number of existing available resources in the system.  

Oversatisfied specifies states in which the managed system possesses resources that surpass the defined requirements to achieve the system’s goals.  

Undersatisfied specifies states in which resources in the managed system do not cover the defined requirements to achieve the system’s goals.  

Analyzing specifies states in which the behavior is processing the working data in order to determine the fulfillment of the managed system with respect to the current requirements. The Analyzing state requires a prior collection of the knowledge required for analysis (getRequired(), getUsed(), and getContext()). Next, an analysis is performed (matchResources()). The results of the analysis are then communicated to the Plan (Plan_OverSatisfied[Node_ID]!, Plan_Satisfied[Node_ID]!, Plan_UnSatisfied[Node_ID]!).
Plan_UnSatisfied[Node_ID]!}. Communication with the Plan can also be done via flags, as presented in the Monitor behavior pattern.

D.3.4.3 Functions and Data Structures

The Analyze automaton is complemented with a set of data structures and functions that are necessary for the design of the analyze logic. The analyze logic process may become domain specific, however, below we present the rational to be specified in each of the functions and the data structures that are involved in the Analyze.

getRequired() retrieves the current software requirements with respect to the resources that are required for the system’s completeness. This function must access the analyze working data (analyzeWD). We provide a snippet for the specification of the getRequired() function, which gathers the defined number of required resources that the system should have in order to achieve its goals.

```c
int getRequired(Concern &required[N]){
    //Specify the logic of the getRequired function
    required = AdaptationKnowledge[Node_ID].analyzeWD[Analyze_ID].required;
}
```

**Declaration D.18: Analyze.getRequired() function**

getUsed() retrieves the current software requirements with respect to the number of resources that are required for the system’s completeness. This function must access the ManagedSystemKnowledge. We provide a snippet for the specification of the getUsed() function, which gathers the current number of used resources in the system.

```c
int getUsed(Resource &used[N]){
    int i;
    Resource res;
    for (i = 0; i < Resources_ID; i++){
        if (ManagedSystemKnowledge[Node_ID].availableResource[i] == 1){
            res = resources[i];
            include(used, res);
        }
    }
}
```

**Declaration D.19: Analyze.getUsed() function**

getContext() retrieves the current information with respect to the context. One particular case is to obtain information regarding the current state of used resources (represented in the ManagedSystemKnowledge). This function must access the EnvironmentKnowledge and (optionally) the ManagedSystemKnowledge. We provide an snippet for the specification of the getContext() function, which gathers the current state (&context[])) of the used resources in the system.
D.3 Behavior Specification Patterns

states branches may imply further scoping of the system of interest. Fig. D.10 shows the Behavior Specification Pattern for the Plan behavior based on time-triggering and event-triggering approaches.

execute_release[Node_ID]

plan_overSatisfied[Node_ID]?

plan_satisfied[Node_ID]?

execute_add[Node_ID]!

addResource() releaseResource()

plan_unSatisfied[Node_ID]?

releaseResource addResourceSatisfied

releasePlanReady addPlanReady

SatisfiedRes == planToDo()

t==Period

releaseResource

SatisfiedRes == planToDo()

t<=Period

targetedNode= Node_ID,

addResource()

setExecuteFlag(releaseResource)

Figure D.10: Plan behavior patterns with event- (top) and time- (bottom) triggering approaches

D.3.5.1 Instantiation

The Plan performs a set of specific mitigation tasks. Multiple Plans may be instantiated to perform different mitigation strategy plans, based on different rationales. For instance, multiple Plans can be instantiated for different type of resources to be managed in the system.

D.3.5.2 States

We identify a total of five main states in the Plan. Below we present the rationale for each one of the five main states.

Satisfied represents states in which the system does not require mitigation plans for adaptation. This is a state where the Plan is on hold. This state can be left based on event-triggered approach (through the Plan_Satisfied[Node_ID]! signal from an Analyze) or under a time-triggered approach (t==Period).

AddResource. In case additional resources are required to be added to the managed system (right part of the behavior templates in Fig. D.10), the Plan will go to the AddResource state. The Plan in this state will coordinate with the Execute to look for a resource to be integrated in the system and to integrate them into the managed system.

D.3.5 Plan behavior pattern

The Plan is responsible for planning mitigation actions when needed to bring the managed system in a desired state (i.e. Satisfied state). Based on the results of the analysis, two types of mitigation actions may need to be planned (in case of a satisfied situation, no adaptation plan is required). Either there is a need to add resources (in case the system state is unsatisfied) or resources may be released (when the system state is oversatisfied). Notice that AddResource and ReleaseResource
states branches may imply further scoping of the system of interest. Fig. D.10 shows the Behavior Specification Pattern for the Plan behavior based on time-triggering and event-triggering approaches.

**Figure D.10:** Plan behavior patterns with event- (top) and time- (bottom) triggering approaches

### D.3.5.1 Instantiation

The Plan performs a set of specific mitigation tasks. Multiple Plans may be instantiated to perform different mitigation strategy plans, based on different rationales. For instance, multiple Plans can be instantiated for different type of resources to be managed in the system.

### D.3.5.2 States

We identify a total of five main states in the Plan. Below we present the rationale for each one of the five main states.

**Satisfied** represents states in which the system does not require mitigation plans for adaptation. This is a state where the Plan is on hold. This state can be left based on event-triggered approach (through the Plan_Satisfied[Node_ID]! signal from an Analyze) or under a time-triggered approach ($t==\text{Period}$).

**AddResource.** In case additional resources are required to be added to the managed system (right part of the behavior templates in Fig. D.10), the Plan will go to the AddResource state. The Plan in this state will coordinate with the Execute to look for a resource to be integrated in the system and to integrate them into the managed system.
**AddPlanReady.** When the tasks (for adding a resource) to be carried out in the adaptation plan have been defined, the Plan visits the AddPlanReady state. In this state, the behavior has completed the plan for mitigation and must notify the Execute to transfer the mitigation actions into the managed system. The notification may be performed through a Execute_Add[Node_ID] signal or by activating the corresponding flag (though the setExecuteFlag() function with the AddRes parameter to determine the typology of the plan actions).

**ReleaseResource.** In case resources must be released from the managed system (left part of the behavior template in Fig. D.10), the Plan will go to the ReleaseResource state. The procedure to release a resource is similar as for adding a resource.

**ReleasePlanReady.** When the tasks (for releasing a resource) to be carried out in the adaptation plan have been defined, the Plan visits the ReleasePlanReady state. In this state, the behavior has completed the plan for mitigation and must notify the Execute behavior to transfer the mitigation actions into the managed system. The notification may be performed through a Execute_Release[Node_ID] signal or by activating the corresponding flag (though the setExecuteFlag() function with the ReleaseRes parameter to determine the typology of the plan actions).

### D.3.5.3 Functions and Data Structures

The Plan automaton is complemented with a set of data structures and functions that are necessary for the design of the plan logic. The plan logic process may become domain specific, however, below we present the rational to be implemented in each of the functions and the data structures that are involved in the plan behavior.

plantToDo() determines the typology of action plans that are required according to the results from the Analyze. This function can be used to support time-triggering approaches, and mainly reads the values set in the plan flag (flagP). An initial set of options that the function can return are "add" (represented by AddRes), "release" (ReleaseRes) and "satisfied" (SatisfiedRes).

```c
int planToDo(){
    return AdaptationKnowledge[Node_ID].flagP;
}
```

**Declaration D.24:** Plan.planToDo() function

**addResource()** is in charge of updating the set of plan actions that are required for the adaptation process, concretely to include a resource (still to be identified) into the managed system. Below, we present an abstraction of the function specification. This function manipulates information in the execute working data (executeWD).

```c
void addResource(){
    int i;
}
```

**Declaration D.28:** Plan.addResource() function
D.3 Behavior Specification Patterns

releaseResource() is in charge of updating the set of plan actions that are required for the adaptation process, concretely to remove a resource (still to be identified) from the managed system. Below, we present an abstraction of the function specification. This function manipulates information in the execute working data (executeWD).

```c
void releaseResource() {
    int i;
    AdaptationKnowledge[Node_ID].executeWD.actions++;
    i = AdaptationKnowledge[Node_ID].executeWD.actions;
    AdaptationKnowledge[Node_ID].executeWD.action[i].type = ReleaseRes;
    ...
    AdaptationKnowledge[Node_ID].executeWD.action[i].status = Requested;
}
```

Declaration D.26: Plan.releaseResource() function

setExecuteFlag() is in charge of setting the flag (flagE) to notify the Execute to transfer the adaptation plan into the managed system.

```c
void setExecuteFlag(int executeAction) {
    AdaptationKnowledge[Node_ID].flagE = executeAction;
}
```

Declaration D.27: Plan.setExecuteFlag() function

reset() is in charge of resetting the timer counter t prior to enter a state with a time-based invariant. This function is specified for time-triggered approaches, ensuring the correct triggering mechanisms.

```c
int reset() {
    return Reset;
}
```

Declaration D.28: Plan.reset() function

### D.3.6 Execute behavior pattern

The Execute is responsible of executing the mitigation actions that have been defined by the Plan. These plans can be either to release or add resources, and can contain multiple set of actions to be invoked into the managed system for adaptation purposes. Fig. D.11 and Fig. D.12 show the Behavior Specification Patterns for Execute with event-triggering and time-triggering approaches.
Figure D.11: Execute behavior pattern with event-triggering approach
Figure D.12: Execute behavior pattern with time-triggering approach
The behavior is divided in two sections, aligned with the structure of the Plan. On the left part of the model, we find the execution of plans to release resources. On the right part of the mode, we find the execution of plans to add resources to the managed system.

D.3.6.1 Instantiation

An Execute behavior can be designed to perform a specific set of mitigation tasks, following the design of the Plan behavior. Multiple Execute behaviors may be instantiated to transfer different mitigation strategy plans, based on different rationales. For instance, multiple Execute can be instantiated for different type of resources to be managed in the managed system.

D.3.6.2 States

We identify a total of eight main states in the Execute. Below we present the rationale for each one of them.

Waiting represents states in which the system does not require mitigation actions to be transferred to the managed system. This is a state where the Execute is waiting to start the execution of adaptation actions. The Execute leaves this state on event-triggered (through the Execute_Add[Node_ID]! and Execute_Release[Node_ID]! signals from the Plan) or based on time-triggered (t==Period).

PrepareAdd describes states in which the Execute is performing tasks that are pre-requisite for applying the planned actions for adding a resource into the managed system. Notice that an action to be executed by the Execute may require multiple preparation steps.

DoActionsAdd describes states in which the Execute performs the actions specified by the previous Plan in order to include a resource into the managed system. The Execute may visit this state one or more times, depending on the logic of the action to be performed. In this state, the Execute interacts with the Effector, in order to transfer the actions into the managed system. This interaction is performed through the Effector_Release[Node_ID]! signal or the setEffectorFlag() function (to notify the Effector) and the isSystemReady(Do) function (in order to get Effector’s feedback).

PostExecuteAdd describes states in which the Execute is performing tasks that are necessary after applying the planned actions for adding a resource into the managed system. Notice that an action to be executed by the Execute may require multiple post-execution steps. After performing a PostExecuteAdd tasks, the component behavior may require processing additional tasks (if planned).

PrepareRelease describes states in which the Execute is performing tasks that are pre-requisite for applying the planned actions for releasing a resource from the managed system. Notice that an action to be executed by the Execute may require multiple preparation steps.
**DoActionsRelease** describes states in which the Execute performs the actions specified by the previous Plan in order to release a resource from the managed system. The Execute may visit this state one or more times, depending on the logic of the action to be performed. In this state, the Execute interacts with the Effector, in order to transfer the actions into the managed system. This interaction is performed through the `Effector_Release[Node_ID]!` signal or the `setEffectorFlag()` function (to notify the Effector) and the `isSystemReady(Do)` function (in order to get Effector’s feedback).

**PostExecuteRelease** describes states in which the Execute is performing tasks that are necessary after applying the planned actions for releasing a resource from the managed system. Notice that an action to be executed by the Execute may require multiple post-execution steps. After performing a `PostExecuteRelease` tasks, the component behavior may require processing additional tasks (if planned).

**PlanCompleted** represents states in which the plans have been successfully transferred to the managed system with adaptation purposes. This state is a transitional state towards the Waiting state. The PlanCompleted facilitates property specifications regarding the completeness of mitigation actions.

### D.3.6.3 Functions and Data Structures

The `Execute` automaton is complemented with a set of data structures and functions that are necessary for the design of the execute logic. The execute logic process is domain specific, however, below we present the rational to be implemented in each of the functions and the data structures that are involved in the execute behavior.

**getExecuteFlag()** determines the typology of action plans that are required according to a defined plan. This function is used for time-triggering approaches, and mainly reads the values set in the execute flag (`flagE`). An initial set of options that the function can return are `AddRes`, `ReleaseRes` and `SatisfiedRes`.

```c
int getExecuteFlag(int executeAction) {
    int action = AdaptationKnowledge[Node_ID].flagE;
    AdaptationKnowledge[Node_ID].flagE = complete;
    return action;
}
```

**Declaration D.29:** `Execute.getExecuteFlag()` function

**isSystemReady()** is in charge of evaluating whether the managed system is ready for transferring the next of the adaptive actions. This function may be required for the evaluation of the current managed system state (or part of it) in order to determine the do-ability to apply a desired adaptation action. This function can become domain specific and may require interactions with the Effector. Below we present a snippet for the implementation of the function, where the Execute checks the current state of actions that the Effector may have completed, prior to begin with new adaptations.

---

Adaptation actions are transferred to the managed system through the interaction of the Effector components.
const int Pending = -1;
const int Requested = 0;
const int Performed = 1;

const int Prepare = 0;
const int Do = 1;
const int Post = 2;

bool isSystemReady(int type){
    //Determine whether the system is ready to apply the coming task
    if (type == Prepare){
        //Check whether the system is ready to apply preparation actions
        if (...) return false;
    }
    if (type == Do){
        //Check whether the effector is ready to apply an action
        if (AdaptationKnowledge[Node_ID].effectorWD.status != Performed)
            return false;
    }
    if (type == Post){
        //Check whether the system is ready to apply postexecution actions
        if (...) return false;
    }
    return true;
}

Declaration D.30: Execute.isSystemReady() function

prepare() is in charge of performing preparation actions prior to adaptation. This function may include the search for a resource (we will study this particular case later), freezing the managed system, and other tasks. This function must manage information located in the execute working data (executeWD).

void prepare(){
    //Define domain specific logic
}

Declaration D.31: Execute.prepare() function

isPreparationDone() is in charge of determining whether additional preparation tasks are still pending to be executed. The Execute must perform this check prior to start with execution of adaptation actions. This function must manage information located in the execute working data (executeWD).

bool isPreparationDone(){
    int i, nextActionState;
    bool status = false;
    
    for (i=0; i<TotalActions; i++){
        // Identify the current action to perform
        nextActionState = AdaptationKnowledge[Node_ID].executeWD.action[i].status;
        if (nextActionState == Pending && status == false){
            // Specify the logic to determine whether preparation tasks are still required for the next action
            status = true;
        }
    }
}
D.3 Behavior Specification Patterns

isActionDone() is in charge of determining whether there are pending tasks to be executed regarding the current adaptation action. This function must manage information located in the execute working data (executeWD).

```c
bool isActionDone(){
    int i, nextActionState;
    bool status = false;
    for (i =0; i<TotalActions; i++){
        // Identify the current action to perform
        nextActionState = AdaptationKnowledge[Node_ID].executeWD.action[i].status;
        if (nextActionState == Pending && status == false){
            // Specify the logic to determine whether tasks for the current action in the plan are required
            status = true;
        }
    }
    return status;
}
```

Declaration D.33: Execute.isActionDone() function

isPostExecDone() is in charge of determining whether post-execution tasks are still pending after setting action plan as completed. This function must manage information located in the execute working data (executeWD).

```c
bool isPostExecDone(){
    int i, nextActionState;
    bool status = false;
    for (i =0; i<TotalActions; i++){
        // Identify the current action to perform
        nextActionState = AdaptationKnowledge[Node_ID].executeWD.action[i].status;
        if (nextActionState == Pending && status == false){
            // Specify the logic to determine whether postexecution tasks are still required for the executed action
            status = true;
        }
    }
    return status;
}
```

Declaration D.34: Execute.isPostExecDone() function

invokeNextAction() is in charge of communicating with the related Effector to inform about the adaptation actions that are required into the managed system. Thus, this function must manage data from the AdaptiveKnowledge concerning the Execute and execute working data (executeWD). Below we offer a snippet to
D MAPE-K Formal Templates Documentation

specify the function. Notice that the *Effector* must update the status of the current action (*nextActionState*) once completed to set as *Performed*.

```c
void invokeNextAction()
{
    // Implement the logic to prepare the effector actions to be executed
    int nextActionState, type, param1, status;
    int i;
    bool done = false;

    for (i = 0; i < TotalActions; i++)
    {
        nextActionState =
            AdaptationKnowledge[Node_ID].executeWD.action[i].status;
        if (nextActionState == Pending & done == false)
        {
            AdaptationKnowledge[Node_ID].effectorWD.type =
                AdaptationKnowledge[Node_ID].executeWD.action[i].type;
            AdaptationKnowledge[Node_ID].effectorWD.type =
                AdaptationKnowledge[Node_ID].executeWD.action[i].param1;
            ...
            AdaptationKnowledge[Node_ID].effectorWD.status =
                AdaptationKnowledge[Node_ID].executeWD.action[i].status;
            done = true;
        }
    }
}
```

**Declaration D.35**: *Execute.invokeNextAction()* function

*actionsPending()* is in charge of determining whether planned actions are pending to be transferred into the managed system. This function must manage information located in the execute working data (*executeWD*).

```c
bool actionsPending()
{
    int nextActionState;
    int i;
    bool done = false;

    for (i = 0; i < TotalActions; i++)
    {
        nextActionState =
            AdaptationKnowledge[Node_ID].executeWD.action[i].status;
        if (nextActionState == Pending & done == false)
        {
            done = true;
        }
    }
    if (done) return true;
    else return false;
}
```

**Declaration D.36**: *Execute.actionsPending()* function

*setEffectorFlag()* is in charge of defining the suitable flag to effect the actions into the managed system.

```c
void setEffectorFlag(int effectorAction)
{
    AdaptationKnowledge[Node_ID].flagEf = effectorAction;
}
```

**Declaration D.37**: *Execute.setEffectorFlag()* function
reset() is in charge of resetting the timer counter $t$ prior to enter a state with a time-based invariant. This function is specified for time-triggered approaches, ensuring the correct triggering mechanisms.

```
int reset () {
    return Reset;
}
```

**Declaration D.38:** `Execute.reset()` function

### D.3.7 Refinement: Execute with Search Resource Process

One particular case of `Execute` requires a preparation process in order to identify a resource to be integrated into the managed system, or to be released from it. In this subsection we present a refinement of the Execute behavior pattern presented above in order to cover such self-adaptive systems. The presented refinement (see Fig. D.13 and Fig. D.14) specifies two main phases for the `Execute`, that describe the `Prepare` and `DoAction` phases.

#### D.3.7.1 States

We identify a total of ten main states in this refinement of an `Execute` behavior. Below we present the rationale for each one of them.

- **Waiting** represents states in which the system does not require mitigation actions to be transferred to the managed system. The `Execute` is waiting for leaving on event-triggered (through the `Execute_Add[Node_ID]!` and `Execute_Release[Node_ID]!` signals from a Plan) or under a time-triggered ($t==\text{Period}$).

- **SearchAdd** represents states in which the `Execute` is searching for a resource to be integrated in the managed system.

- **SearchAddInProgress** represents states in which the `Execute` has not managed to identify a candidate resource that can be integrated in the managed system. In such a case, the `Execute` will periodically ($t==\text{Period2}$) look for a candidate resource ($\text{found_res} == \text{findResource()}$) until the search is successful.

- **AddFound** represents states in which the `Execute` has successfully found a resource to be integrates into the managed system.

- **SearchRelease** represents states in which the `Execute` is searching for a resource, used by the managed system, to be released.

- **SearchReleaseInProgress** represents states in which the `Execute` has not managed to identify a candidate resource that can be released from the managed system. In such a case, the `Execute` will periodically ($t==\text{Period2}$) look for a candidate resource ($\text{dispensable_res} == \text{suggestDispensable()}$) until the search is successful.

- **ReleaseFound** represents states in which the `Execute` has successfully found a resource to be released from the managed system.
Figure D.13: Execute with "search resource" behavior pattern with event-triggering approach
Figure D.14: Execute with "search resource" behavior pattern with time-triggering approach
ExecuteAdd represents states in which the Execute initiates adding plan actions designed by the Plan. This state is reached via the Execute_Add[Node_ID] signal (for event-triggered approaches) or the getExecuteFlag() == add condition (for time-triggered approaches). In this state, the Execute checks whether the system is ready to invoke the first action of the plan (isSystemReadyToTakeAction()). If this is the case, the action is invoked via an Effector (invokeNextAction()). Then, the Execute checks whether the plan is completed or not (actionsPending()). This state can be visited multiple times if more actions need to be invoked. If not, the behavior returns to the Waiting state (via PlanComplete).

ExecuteRelease represents states that are analog to the ExecuteAdd state, but for release resource purposes. In this state, the Execute initiates releasing plan actions designed by the Plan. This state is reached via the Execute_Release[Node_ID] signal (for event-triggered approaches) or the getExecuteFlag() == release condition (for time-triggered approaches). In this state, the Execute checks whether the system is ready to invoke the first action of the plan (isSystemReadyToTakeAction()). If this is the case, the action is invoked via an effector (invokeNextAction()). Then, the Execute checks whether the plan is completed or not (actionsPending()). The Execute can visit this state multiple times in case more actions need to be invoked. In case there are not additional actions to be transferred to the managed system (Completed == actionsPending()), the behavior returns to the Waiting state (via PlanComplete).

PlanCompleted represents states in which the plans have been successfully transferred to the managed system with adaptation purposes. This state is a transitional state towards the Waiting state. The PlanCompleted facilitates property specifications regarding the completeness of mitigation actions.

D.3.7.2 Functions and Data Structures

This refinement of the Execute automaton is complemented with a set of data structures and functions that are necessary for the design of the execute logic. The execute logic process is domain specific, however, below we present the rational to be implemented in each of the functions and the data structures that are involved in the execute behavior.

getExecuteFlag() determines the typology of action plans that are required according to a defined plan. This function is used for time-triggering approaches, and mainly reads the values set in the execute flag (flagE). An initial set of options that the function can return are AddRes, ReleaseRes and SatisfiedRes.

```c
int getExecuteFlag(int executeAction) {
    int action = AdaptationKnowledge[Node_ID].flagE;
    AdaptationKnowledge[Node_ID].flagE = complete;
    return action;
}
```

Declaration D.39: Execute.getExecuteFlag() function

findResource() is a preparation function that refines part of the prepare() function for an addition process. This function is in charge of identifying a resource
that possess the required characteristics to be a target for being integrated in the system. The logic in the `findResource()` function may become domain dependent. However, this function is expected to process information from the `ManagedSystemKnowledge` and to return a candidate resource identifier (`foundResource`) for the add process. In case no resources are found, the function must return `NONE` to notify the search failure.

```c
int findResource(){
    int candidate = NONE;
    //TODO: Implement your logic to specify a resource to be included to fulfill the requirements
    return candidate;
}
```

**Declaration D.40: Execute.findResource() function**

`setCandidate()` is a preparation function that refines part of the `prepare()` function for an addition process. This function is in charge of updating the adding action settings in order to specify the resource to be added into the managed system.

```c
void setCandidate(int res){
    AdaptationKnowledge[Node_ID].executeWD.action[0].param1 = res;
    AdaptationKnowledge[Node_ID].effectorWD.param1 = res;
}
```

**Declaration D.41: Execute.setCandidate() function**

`suggestDispensable()` that refines part of the `prepare()` function for a release process. This function is in charge of identifying a candidate resource currently in use in the system, that can be released. The logic in the `suggestDispensable()` function may become domain dependent. However, this function is expected to process information from the `ManagedSystemKnowledge` and to return a candidate resource identified (`dispensable_resource`) to be released from the managed system. In case no resources can be selected for being released, the function must return `NONE` to notify the search failure.

```c
int suggestDispensable(){
    int candidate = NONE;
    //TODO: Implement your logic to identify an existing resource that can be released
    return candidate;
}
```

**Declaration D.42: Execute.suggestDispensable() function**

`setDispensable()` is a preparation function that refines part of the `prepare()` function for a release process. This function is in charge of updating the releasing action settings in order to specify the resource to be released from the managed system.

```c
void setDispensable(){
    AdaptationKnowledge[Node_ID].executeWD.action[0].param1 = res;
    AdaptationKnowledge[Node_ID].effectorWD.param1 = res;
}
```
D MAPE-K Formal Templates Documentation

33

\begin{verbatim}

\textbf{Declaration D.43:} \texttt{Execute.setDispensable()} function

\textbf{isSystemReadyToTakeAction()} refines the function \texttt{isSystemReady()}. This function is in charge of evaluating whether the managed system is ready for transferring the next of the adaptive actions. This function can become domain specific and may require interactions with the Effector. Below we present a sample code for the implementation of the function, where the Execute checks the current state of actions of the Effector.

\begin{verbatim}
const int Pending = -1;
const int Requested = 0;
const int Performed = 1;

bool isSystemReadyToTakeAction() {
    // Check whether the effector is still working
    if (AdaptationKnowledge[Node_ID].effectorWD.status != Performed)
        return false;
    return true;
}
\end{verbatim}

\textbf{Declaration D.44:} \texttt{Execute.isSystemReadyToTakeAction()} function

\textbf{invokeNextAction()} is in charge of communicating with the related Effector to inform about the adaptation actions that are required into the managed system. This function must manage data from the \textit{AdaptiveKnowledge} concerning the execute and effector working data (\texttt{executeWD} and \texttt{effectorWD}). Below we offer a snippet to specify the function. Notice that the Effector must update the status of the current action (\texttt{nextActionState}) once completed to set as \texttt{Performed}.

\begin{verbatim}
void invokeNextAction() {
    // Implement the logic to prepare the effector actions to be executed
    int nextActionState, type, param1, status;
    int i;
    bool done = false;

    for (i = 0; i < TotalActions; i++) {
        nextActionState = AdaptationKnowledge[Node_ID].executeWD.action[i].status;
        if (nextActionState == Pending && done == false) {
            type = AdaptationKnowledge[Node_ID].executeWD.action[i].type;
            AdaptationKnowledge[Node_ID].effectorWD.type = AdaptationKnowledge[Node_ID].executeWD.action[i].type;
            AdaptationKnowledge[Node_ID].effectorWD.param1 = AdaptationKnowledge[Node_ID].executeWD.action[i].param1;
            ... AdaptationKnowledge[Node_ID].effectorWD.status = AdaptationKnowledge[Node_ID].executeWD.action[i].status;
            done = true;
        }
    }
}\end{verbatim}

\textbf{Declaration D.45:} \texttt{Execute.invokeNextAction()} function
actionsPending() is in charge of determining whether planned actions are pending to be transferred into the managed system. This function must manage information located in the execute working data (executeWD).

```c++
bool actionsPending() {
    int nextActionState;
    int i;
    bool done = false;
    for (i = 0; i < TotalActions; i++) {
        nextActionState = AdaptationKnowledge[Node_ID].execute WD.action[i].status;
        if (nextActionState == Pending && done == false) {
            done = true;
        }
    }
    if (!done) return true;
    else return false;
}
```

**Declaration D.46: Execute.actionsPending() function**

**setEffectorFlag()** is in charge of defining the correct flag to effect the actions into the managed system.

```c++
void setEffectorFlag (int effectorAction) {
    AdaptationKnowledge[Node_ID].flagEf = effectorAction;
}
```

**Declaration D.47: Execute.setEffectorFlag() function**

**reset()** is in charge of resetting the timer counter $t$ prior to enter a state with a time-based invariant. This function is specified for time-triggered approaches, ensuring the correct triggering mechanisms.

```c++
int reset () {
    return Reset;
}
```

**Declaration D.48: Execute.reset() function**

### D.3.8 Effector behavior pattern

The *Effector* specifies the actions that an effector component carries out into the managed system in order to accomplish adaptation concerns. This behavior is domain specific, as it must interact with the managed system at hand. The possible actions that an effector behavior can perform into the managed system are to release or add resources, and can contain multiple set of actions to be invoked into the managed system for adaptation purposes. Fig. D.15 shows the *Behavior Specification Patterns* for *Effectors* with event-triggering (top) and time-triggering (bottom) approaches. The behavior is divided in two sections, in order to manage the use of resources in the managed system, mainly to add and release resources from the system. On the left part of the model, we find the effector actions to release resources. On the right part of the model, we find the effector actions to add resources to the managed system.
D.3 Behavior Specification Patterns

D.3.8.1 Instantiation
The Effector applies specific mitigation tasks to the managed system coordinated by the Execute behavior. Multiple Effector behaviors may be instantiated to apply the different actions that Execute behaviors may demand. For instance, multiple Execute can be instantiated for different type of resources to be managed in the managed system.

D.3.8.2 States
We identify a total of four main states in the Effector. Below we present the rationale for each one of the seven main states.

- **Waiting** represents states in which the system does not require mitigation actions to be transferred to the managed system. This is a state where the Effector is on hold. This state can be left based on event-triggered approach (through the Effector_Add[Node_ID]! and Effector_Release[Node_ID]! signals from the Execute) or under a time-triggered approach (t==Period).

- **WaitToAdd** represents states in which the Effector is expected to apply adaptation actions on the managed system in order to integrate a new resource, but the current state of the managed system does not (temporarily) allow the adaptation to take place. In this case, the Effector must wait and periodically (t==Period2) check the managed system state until the adaptation change can be implemented.

- **WaitToRelease** represents states in which the Effector is expected to apply adaptation actions on the managed system in order to release a used resource, but the current state of the managed system does not (temporarily) allow the adaptation to take place. In this case, the Effector must wait and periodically (t==Period2) check the managed system state until the adaptation change can be implemented.

- **AddEffected** describes states in which adding resource actions have been carried out on the managed system. This state requires the application of the actions specified in the addResource() function. Depending on the nature of the managed system, this state may require different adaptation actions.

- **ReleaseEffected** describes states in which releasing resource actions have been carried out on the managed system. This state requires the application of the actions specified in the releaseResource() function. Depending on the nature of the managed system, this state may require different adaptation actions.

- **EffectorCompleted** describes states where the Effector has applied the required adaptation actions on the managed system. This state covers both states for releasing and adding resources actions. This state is defined to facilitate property specification purposes, in order to verify that effect actions are carried out regardless the nature of the effector actions.

D.3.8.3 Functions and Data Structures
The Effector automaton is complemented with a set of data structures and functions that are necessary for the design of the effector logic. Notice that the effector logic process is domain specific, and it may require high levels of refinement for

---

**Figure D.15:** Effector behaviors with event (top) and time (bottom) triggering approaches
D.3.8.1 Instantiation

The Effector applies specific mitigation tasks to the managed system coordinated by the Execute behavior. Multiple Effector behaviors may be instantiated to apply the different actions that Execute behaviors may demand. For instance, multiple Execute can be instantiated for different type of resources to be managed in the managed system.

D.3.8.2 States

We identify a total of four main states in the Effector. Below we present the rationale for each one of the seven main states.

Waiting represents states in which the system does not require mitigation actions to be transferred to the managed system. This is a state where the Effector is on hold. This state can be left based on event-triggered approach (through the Effector_Add[Node_ID]! and Effector_Release[Node_ID]! signals from the Execute) or under a time-triggered approach (t==Period).

WaitToAdd represents states in which the Effector is expected to apply adaptation actions on the managed system in order to integrate a new resource, but the current state of the managed system does not (temporarily) allow the adaptation to take place. In this case, the Effector must wait and periodically (t==Period) check the managed system state until the adaptation change can be implemented.

WaitToRelease represents states in which the Effector is expected to apply adaptation actions on the managed system in order to release a used resource, but the current state of the managed system does not (temporarily) allow the adaptation to take place. In this case, the Effector must wait and periodically (t==Period) check the managed system state until the adaptation change can be implemented.

AddAffected describes states in which adding resource actions have been carried out on the managed system. This state requires the application of the actions specified in the addResource() function. Depending on the nature of the managed system, this state may require different adaptation actions.

ReleaseAffected describes states in which releasing resource actions have been carried out on the managed system. This state requires the application of the actions specified in the releaseResource() function. Depending on the nature of the managed system, this state may require different adaptation actions.

EffectorCompleted describes states where the Effector has applied the required adaptation actions on the managed system. This state covers both states for releasing and adding resources actions. This state is defined to facilitate property specification purposes, in order to verify that effect actions are carried out regardless the nature of the effector actions.

D.3.8.3 Functions and Data Structures

The Effector automaton is complemented with a set of data structures and functions that are necessary for the design of the effector logic. Notice that the effector logic process is domain specific, and it may require high levels of refinement for
managed systems at hand. Below we present the rational to be implemented in each of the functions and the data structures that are (could be) involved in the effector behavior.

**getAction()** is in charge of determining the type of effector actions that are required to be applied into the managed system. Based on information defined in the effector working data (type and status in effectorWD), there are three possible outputs that can be expected from the function: *Nothing* is returned when there are not actions to be effected into the managed system; *addResource* is returned in case a resource must be integrated into the managed system; and, equivalently, *releaseResource* is returned in case a resource must be released. Therefore, this function requires a prior

```c
int getAction(){
    if (AdaptationKnowledge[Node_ID].effectorWD.action.status == Performed){
        return Nothing;
    } else
        return AdaptationKnowledge[Node_ID].effectorWD.action.type;
}
```

**Declaration D.49: Effector.getAction() function**

**isSystemReady()** is in charge of study the current state of the managed system in order to determine whether the current adaptation action can be applied, or the adaptation must wait instead. This function is domain specific, and must be specified based on the logic of the managed system.

```c
int isSystemReady(){
    // Specify the logic to determine whether the system is ready for adaptations to be applied
}
```

**Declaration D.50: Effector.isSystemReady() function**

**addResource()** is in charge of carrying out specific actions to the managed system with the aim of integrating a defined resource into the managed system. This function is domain specific, therefore, the effector logic depends on the casuistry of the managed system domain. The *addResource()* function applies individual actions defined in the effector working data (effectorWD). These actions must be prior defined by the related Execute. Below we present a sample code for the specification of the *addResource()*) function, to illustrate the interactions with the ManagedSystem and with the related Execute.

```c
int resource = AdaptationKnowledge[Node_ID].effectorWD.action.param1;
ManagedSystem[Node_ID].resource[resource] = USED;
// Feedback to the Execute behavior. Communicate Effected action.
AdaptationKnowledge[Node_ID].effectorWD.action.status = Performed;
```

**Declaration D.51: Effector.addResource() function**
**releaseResource()** is in charge of carrying out specific actions into the managed system with the aim of releasing a defined resource from the managed system. This function is domain specific, therefore, the effector logic depends on the casuistry of the managed system domain. The `releaseResource()` function applies individual actions defined in the effector working data (effectorWD). These actions must be priorly defined by the related Execute. Below we present a snippet for the specification of the `releaseResource()` function, to illustrate the interactions with the ManagedSystem and with the related Execute.

```cpp
int resource = AdaptationKnowledge[Node_ID].effectorWD.action.param1;
ManagedSystem[Node_ID].resource[resource] = FREE;
//Feedback to the Execute behavior. Communicate Effected action.
AdaptationKnowledge[Node_ID].effectorWD.action.status = Performed;
```

**Declaration D.52: Effectork.releaseResource() function**

**communicateEffected()** is in charge of providing feedback to the Execute with respect to the completeness of the effector tasks. This function is involved in the correct coordination of actions between the Effector and Execute, through the status variables in the effector working data (effectorWD). Below we show a sample of the code that illustrates the logic of the function.

```cpp
void communicateEffected(){
    AdaptationKnowledge[Node_ID].effectorWD.action.status = Performed;
}
```

**Declaration D.53: Effectork.communicateEffected() function**

**reset()** is in charge of resetting the timer counter t prior to enter a state with a time-based invariant. This function is specified for time-triggered approaches, ensuring the correct triggering mechanisms.

```cpp
int reset (){ 
    return Reset;
}
```

**Declaration D.54: Effectork.reset() function**

### D.4 Property Specification Patterns

The *Behavior Specifications Patterns* are complemented with a set of formal property specifications that support designers with the verification of adaptation behaviors. These are the *Property Specification Patterns*. The properties are specified with a subset of timed computation tree logic (TCTL) in the form of path formulae, as presented in Section D.2.2.2. Path formulae quantify over paths of the state space of the model and can in general be classified into reachability, safety, and liveness properties.

We provide three groups of specification properties. In group 1, we present the *Adaptation Goal Properties*. These group describes properties about the realization of the adaptation goal by the MAPE-K loop. In group 2, we present the...
**Intra-Behavior Properties**, which specify properties about individual MAPE behaviors. Finally, in group 3, we present **Inter-Behavior Properties**, which specify properties about the interaction between multiple behaviors of a MAPE-K loop.

![Diagram of MAPE-K loop]

**Figure D.16**: Groups of specification properties

### D.4.1 Adaptation Goal property pattern

The *Adaptation Goal properties* allow verifying that when the self-adaptive system requires adaptation, eventually the adaptation will be realized by the MAPE-K loop. Different *Adaptation Goal properties* can be defined in order to study the particularities of specific combination of managed and managing system. Below, we present two properties that can exemplify this category.

- **P1.1**: Analyze(X).Unsatisfied --> Analyze(X).Satisfied || Analyze(X).Oversatisfied

  P1.1 defines when an Analyze at node X detects that the managed system is in an unsatisfied state, the MAPE behaviors will eventually adapt the managed system bringing it in the satisfied or oversatisfied state. Property P1.1 is based on the assumption that eventually resources will become available to realize the adaptation goal.

### D.4.2 Intra-Behavior property pattern

**Intra-behavior properties** allow verification of properties of individual MAPE behaviors, independently of the other MAPE behaviors. This properties can be classified depending on their scope of interest into properties for Probe, Monitor, Analyze, Plan, Execute and Effector intra-behavior properties. Below, we present a set of properties that can exemplify this category.

- **P2.1**: Probe(X).Waiting --> Probe(X).Probing
- **P2.2**: A<>forall(ResID:Resource_id) Probe(ResID).Proving
- **P2.3**: A<>forall(ResID:Resource_id) Probe(ResID).Notify
- **P2.4**: Monitor(X).PreProcessing --> Monitor(X).NewChange

234
P2.5: ConcernKnowledge[X].requiredResources >
ManagedSystemKnowledge[X].usedResources
--> Analyze(X).Unsatisfied

P2.6: Plan(X).AddResource --> Plan(X).Satisfied
P2.7: Plan(X).ReleaseResource --> Plan(X).Satisfied

P2.8: Execute(X).ExecuteAdd --> Execute(X).PlanCompleted
P2.9: Execute(X).ExecuteRelease --> Execute(X).PlanCompleted

P2.1 defines that a Probe behavior must eventually leave the Waiting state in order to visit a Probing state. P2.2 and P2.3 are refinements of P2.1, defining that all Probe behaviors must eventually visit the Probing state (P2.2) and the Notify state (P2.3). These properties are required to ensure that changes in the managed system, environment and goal requirements will eventually be sensed by the Probe behaviors. P2.4 defines that a Monitor, in case it is notified by the Probe and starts a PreProcessing phase, it will eventually complete the Pre-Processing and update the Knowledge for the Analyzer through the NewChange state. P2.5 defines that when resources used by the managed system (ConcernKnowledge[X].requiredResources where X identifies an specific managed system) are insufficient according the requirements of the Knowledge (ConcernKnowledge[X].requiredResources), the Analyze behavior will eventually detect this unsatisfied situation. The greater than symbol is an abstract operator to check whether the used resources satisfy the required resources, and needs to be instantiated for the domain at hand. Intra-process property verification becomes particularly interesting for complex behaviors. E.g., it is important to verify that the Plan creates adequate plans on how the managed system should be adapted. P2.6 and P2.7 are two specification properties for a Plan that allow to verify whether adding (P2.6) and releasing a resource (P2.7) from the managed system leads to a Satisfied state. P2.8 and P2.9 specify properties for an Execute and allow to verify whether the plan for adding a resource (P2.8) and releasing a resource (P2.9) from the managed system are properly executed.

D.4.3 Inter-Behavior property pattern

Inter-behavior properties allow verification of properties about the interaction between multiple behaviors of a MAPE-K loop. Properties in this category may have different levels of granularity regarding to the interactions between the MAPE components. The specification of properties in this category should require refinement in order to be align to the behavior specifications refinements. Below, we present three properties that can exemplify this category.

P3.1: Analyze(X).Unsatisfied --> Plan(X).AddResource
P3.2: Plan(X).ReleasePlanReady --> Execute(X).ExecuteRelease
P3.3: Plan(X).AddPlanReady --> Execute(X).ExecuteAdd

P3.1 describes a property to verify the correct collaboration between the Analyze and Plan behaviors. The property states that when Analyze detects an unsatisfactory state of the managed system, eventually the Plan will start planning a resource search and integration. P3.2 and P3.3 describe properties to verify the correct collaboration between Plan and Execute. In particular, P3.2 allows to verify that when Plan has generated a plan to release a resource, this plan is eventually
D MAPE-K Formal Templates Documentation

executed by the Execute. P3.3 allows verification the correct execution of a plan for adding a resource.
Appendix E

List of Publications

Related to the topic of this thesis:


Additional research publications:


Bibliography


[28] P. Brereton et al. “Lessons from applying the systematic literature review process within the software engineering domain”. In: Journal on Systems and Software 80 (4 2007).


Below please find a list of recent publications in the series Linnaeus University Dissertations. For a full list and more information: Lnu.se


Linnaeus University Dissertations

Below please find a list of recent publications in the series Linnaeus University Dissertations. For a full list and more information: Lnu.se


