UNIVERSITÄT
MANNHEIM

A Framework for Engineering Reusable Self-adaptive Systems

Inauguraldissertation

zur Erlangung des akademischen Grades
eines Doktors der Wirtschaftswissenschaften
der Universität Mannheim

vorgelegt von

Christian Walter Krupitzer

aus Heidelberg

Dekan:	Prof. Dr. Dieter Truxius
Tag der Einreichung:	12. Juli 2018
Erstreferent:	Prof. Dr. Christian Becker
Zweitreferent:	Prof. Dr. Samuel Kounev
Tag der Disputation:	14. August 2018
Prüfungsausschuss:	Prof. Dr. Christian Becker (Vorsitzender)
	Prof. Dr. Hartmut Höhle

Abstract

The increasing complexity and size of information systems result in an increasing effort for maintenance. Additionally, miniaturization of devices leads to higher mobility and the need for context-adaptation, especially in new types of systems, such as Cyber-physical Systems or Internet of Things. Self-adaptive Systems (SASs) has the ability to adapt to changes in their environment or the system resources and address the aforementioned challenges. So far, however, development of these systems is frequently tailored towards the requirements of use cases. The research for frameworks and reusable elements — for implementation as well as design processes — is often neglected. Integrating reusable process and implementation artifacts into a framework and offering a tool suite to developers would make development of SASs faster and less error-prone. This thesis presents the *Framework for Engineering Self-adaptive Systems (FESAS)*. It offers a reusable implementation of a reference system, tools for implementation and design as well as a middleware for controlling system deployment.

Due to distribution of systems and an increase of available information, the complexity for adaptation reasoning increases. This can lead to uncertainty at runtime resulting in incompleteness or obsolescence in adaptation goals, models or rules. Therefore, the need for changing the adaptation reasoning arises. As a second contribution, this thesis introduces a new approach for self-improvement of SASs. It complements the SAS with an additional module for meta-adaptation. Unlike existing approaches, the approach is not limited to a specific type of adaptation nor to specific implementation frameworks. The thesis describes the integration of the module for self-improvement with the FESAS Framework.

Following a design science approach, this thesis explains the design of the artifacts based on requirements derived from an analysis of related work. For evaluation, prototypes of the artifacts are implemented in a proof by prototyping approach and discussed regarding their usability, applicability, and performance.

Acknowledgments

This thesis would not have been possible without the help of several people, many of whom might know even be aware that they have contributed to this achievement.

First of all, I would like to thank my advisor Prof. Dr. Christian Becker for his support, mentoring, and his open door. Christian, thank you for the chance to start my academic career in your group. You always supported me and gave me the freedom to follow my own drum. Further, you are responsible for the great atmosphere in the seventh floor. And of course I have to thank you for the "lectures" in food and wine.

I would like to thank Prof. Dr. Samuel Kounev for his willingness to act as the second reviewer as well as Prof. Dr. Hartmut Höhle for his input and joining the board of examiners.

I would like to thank all the people I had the pleasure of working with throughout the years, namely Dr. Patricia Arias-Cabarcos, (student/paper/hiwi) Martin Breitbach, Janick Edinger, Kerstin Goldner, Benedikt Kirpes, Sonja Klingert, Markus Latz, Jens Naber, Dr. Verena Majuntke, Martin Pfannemüller, Dr. Vaskar Raychoudhury, Felix Maximilian Roth, Dominik Schäfer, Prof. Dr. Gregor Schiele, Dr. Sebastian VanSyckel, Anton Wachner, and Dr. Richard Süselbeck. It always felt like being paid for meeting friends instead of working. Especially, I have to thank Sebastian for teaching me science, improving my English skills, and his countless comments on my papers. Thanks to Max for getting the FESAS project on track with me. Furthermore, I have to thank Janick for showing me that sleep is overrated when writing a paper and always challenging papers until the very last second to a deadline. Last, the "special request Martin" gets a dedicated thank for his support in writing papers beyond the normal tasks of a student assistant and his willingness to comment most parts of this thesis.

Further, I would like to thank also all of my co-authors that are not part of our group: Dr. Samy El-Tawab, Alexander Frömmgen, Prof. Dr. Renato Lo Cigno, PD Dr. Malte Lochau, Dr. Michele Segata, Prof. Dr. Andy Schürr, Prof. Dr. Heiner Stuckenschmidt, Timo Sztyler, Dr. Sven Tomforde, and Markus Weckesser. Each of you helped to not only improve a paper, but impart new scientific knowledge to me.

I would like to thank each student and research assistant involved in the project for their contributions, especially, Jannis Bergbrede, Martin Büttner, Guido Drechsel, Julian Herbold, Jean Kaddour, Fabian Kajzar, Tanawat Mark Klaisoongnoen, Michel Klein, Johannes Kräft, Deborah Mateja, Johannes Müller, Julian Otto, Alina Pollkläsener, Johannes Saal, Kai Schoknecht, Daniel Schopp, Florian Schrage, Timo Sturm, Timur Temizer, Aleksandar Tomasovic, Vincent Voß, Nils Wilken, Ruixing Yang, and Ying Zhang.

Last but not least, I would like to thank my family for always being there for me. To my parents, thank you for giving me the opportunity to go my way. To my wife Kathrin, thank you for staying with me in "Baden", your patience while writing papers or this thesis in the evenings, for your unconditional love and support, and for always taking care of me. This accomplishment is also yours. I love you. And I want to thank you for giving birth to Hannes and Helena, who constantly bring so much joy into our life and giving always variety.

Contents

List of Figures

List of Tables

List of Abbreviations

ADL	Architecture Description Language
ALM	Adaptation Logic Manager
AOP	Aspect-Oriented Programming
BASt	German Federal Highway Research Institute
CBD	Component-Based Development
CBP	Component-Based Programming
CBSE	Component-Based Software Engineering
COP	Context-oriented Programming
CPS	Cyber-physical System
CSP	Constraint Satisfaction Problem
DSL	Domain Specific Language
DSPL	Dynamic Software Product Lines
ECA	Event-Condition-Action
EMF	Eclipse Modeling Framework
FESAS	Framework for Engineering Self-adaptive Systems
GMF	Graphical Modeling Framework
IoT	Internet of Things
IT	Information Technology
JMS	Java Message Service
MAS	Multi-agent System

PCS	Platoon Coordination System
PDE	Plug-in Development Environment
PIM	Platform Independent Model
PSM	Platform Specific Model
Pub/Sub	Publish/Subscribe
QoS	Quality of Service
RE	Requirements Engineering
SAS	Self-adaptive System
SBSE	Search-based Software Engineering
SE	Software Engineering
SLoC	Source Lines of Code
SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
SOS	Self-organized System
SVC	Smart Vacuum Cleaner
TARL	Topology Adaptation Rule Language
V2I	Vehicle-to-Infrastructure
VM	Virtual Machine
WEKA	Waikato Environment for Knowledge Analysis

List of Listings

Glossary

Adaptation Logic Manager (ALM)

The ALM is an external layer on top of a SAS for addressing the need for self-improvement. It integrates a MAPE-K cycle for reasoning. Further, it communicates with the adaptation logic of the SAS through clearly defined interfaces (cf. Section 6.2 for the system model and Section 7.4 for the prototype implementation).

FESAS Adaptation Logic Template

It offers a model for the adaptation logic. The functionality for reasoning on adaptation follows the MAPE-K model. It is separated from complementary modules that handle the context data and the connection to the managed resources (cf. Section 6.1.1).

FESAS Component Template

Model for a specific MAPE component. It separates generic functionality, such as communication between MAPE components, and specific code for the functional logics of the MAPE components to improve the reusability of code (cf. Section 6.1.1).

FESAS Framework

The FESAS Framework combines the FESAS Adaptation Logic Template, the FESAS Component Template, the FESAS Middleware, the FESAS Repository, and the FESAS Workflow for development of SASs. It is complemented by the FESAS IDE for convenient usage of the FESAS Framework and the ALM for self-improvement.

FESAS IDE

The FESAS IDE extends the Eclipse IDE with two plug-ins: the *FESAS Development Tool* and the *FESAS Design Tool* (cf. Section 7.1). The system developer uses the *FESAS Development Tool* for developing the functional logics' code, which is then stored in the FESAS Repository. Using the *FESAS Design Tool*, the system designer configures the components of an adaptation logic and specifies, which code the FESAS Middleware loads at runtime. Both plug-ins incorporate the FESAS Workflow.

FESAS Middleware

The FESAS Middleware components – JSON Config Parser, Proxy ALM, Middleware Starter, and Local Repository – interact with the FESAS Repository and the ALM (in case of self-improvement) during the deployment of a SAS and for self-improvement (cf. Section 6.1.4 for the system model and Section 7.3 for the prototype implementation).

FESAS Repository

Code repository which stores the code of the functional logic elements with metadata. This repository is used during deployment and self-improvement (cf. Section 6.1.3 for the system model and Section 7.2 for the prototype implementation).

FESAS Workflow

Captures the FESAS development process. It is divided into two parts: designing the adaptation logic and developing the functional logic elements. The FESAS Middleware controls the deployment and integrates the design models and code for the functional logics (cf. Section 6.1.2).

Self-adaptation

The self-adaptiveness property aggregates the functionality of the self-* properties to offer autonomous reaction to changes in the system and the environment (cf. Section 2.1).

Self-adaptive System (SAS)

A self-adaptive system (SAS) is able to adjust its behavior in response to its perception of the environment and the system itself. The "self" prefix indicates that the SAS decides autonomously (i.e., without or with minimal interference) how to adapt or organize to accommodate changes in its context and environment (cf. Section 2.2).

Self-improvement

Self-improvement of the adaptation logic is the adjustment of the adaptation logic to handle former unknown circumstances or changes in the environment or the managed resources (cf. Section 2.1.4).

1. Introduction

Today, people are surrounded by smart and connected devices. *Gartner Inc.* estimated that 8.4 billion devices are connected in the IoT worldwide in 2017, reaching 20.4 billion by 2020 [144]. The growing number of mobile and embedded devices in combination with the omnipresence of (wireless) network connections results in an increased distribution of Information Technology (IT). This increases the complexity of these systems, but also enables new types of systems, such as Cyber-physical Systems (CPSs) that integrate computation, networking, and physical processes [200]. This facilitates new applications, such as autonomous driving, ambient assisted living / smart home, or Industry 4.0. However, it requires integration of all available, highly specialized, and heterogeneous devices, ranging from embedded sensor nodes to servers in the cloud. Further, the inclusion of data streams with sensor data and web data leads to an increasing complexity in system development. Additionally, as these system are mobile, changing environmental conditions increase the complexity even further.

Self-adaptive Systems (SASs) adjust parameters or adapt components to reflect changes in their operating environment or in the system [229, 278]. They are seen as one solution to the aforementioned complexity issues as they can adapt (i) to new environmental conditions and (ii) to requirements that are not known at design time. These systems separate (i) resources that offer the system functionality to users / backend systems and (ii) the adaptation logic that controls these resources [11, 198, 229]. While SASs can reduce the complexity in the aforementioned system domains, developing and configuring SASs is a very difficult, error-prone, and time-consuming task [76]. As identified in [96, p. 2] "*we need [..] systematic development, deployment, management and evolution*" of SASs. Various works [76, 96, 229] describe specific design and implementation challenges for SASs. This thesis aims at reducing the complexity in the development of SASs and presents a development approach that focuses on reusability, continuous improvement, and integration of development activities.

1.1. Problem Definition

The developed SASs are often use case specific and optimized to their application areas or devices. While this might result in an optimized SAS, this reduces the reusability of artifacts. Various elements of the adaptation logic can be reused, such as distribution structures (e.g., [401]), communication mechanisms (e.g., Pub/Sub systems), structures for handling knowledge (e.g., distributed databases), or components of reference architectures. Literature suggests that reusable development processes and components in the adaptation logic can reduce the complexity in the development of SASs [76, 229]. Additionally, agile methods or rapid prototyping support shorter development cycles to avoid expensive changes late in the project [181]. These methods require well-defined interfaces for modules and fine granular exchange of system's code. This granularity enables simplified exchange and reuse of algorithms for reasoning on adaptation in other systems. However, as we have shown in [228], literature does not address sufficiently the reusability of code on a fine granular level in SASs [228]. This granularity would enable to store code in a repository and reuse it in a well-defined architecture for the adaptation logic. The architecture facilitates to focus on the implementation of specific adaptation reasoning algorithms and can abstract from common issues as communication in the adaptation logic.

Established concepts to reason about adaptations in SASs use models, rules, goals, or utility functions [234]. Uncertainty at runtime can lead to incompleteness or obsolescence of goals, rules, or models as well as non-optimized utility functions. Accordingly, in [229], we identified the challenge of integrating a mechanism for self-improvement, i.e., adapting the adaptation logic. Some authors already offer self-improvement, e.g., [9, 115, 194, 274, 298, 363]. We showed the shortcomings of these approaches in [226, 227]. They mainly focus on either structural or parametric self-improvement as well as either on reactive or proactive reasoning for self-improvement. The approaches react on different triggers: changes in the context, managed resources, or user preferences, i.e., change of goals. Monitoring of triggers and execution of self-improvement is often specific for an application. An open issue is to separate generic elements and application-specific parts that need customization as well as offering self-improvement as a self-contained module that can easily be integrated with frameworks for developing SASs. Additionally, for offering flexibility to developers, the approaches should integrate development support for reactive and proactive self-improvement as well as not focus on only structural or parametric self-improvement. None of the available approaches addresses all issues.

As shown in [224], approaches for developing SASs offer different types of support. This includes reference implementations, tools, or rather abstract definitions of methodologies. However, often an approach targets only one development activity. None of the approaches covers all activities of the development of SASs and also integrates self-improvement. Further, the approaches restrict developers to specific settings or do not address reusability. To lower complexity in developing SASs, it is beneficial to complement reusable components for the adaptation logic with a generic development process. To abstract from specifics of the process, tools should encapsulate the development activities. As mentioned, the integration of a library of reusable code can further speed up the development. Last, self-improvement at runtime should be integrated.

1.2. Research Questions

Following the problem definition, the objective of this thesis is:

*The **integrated support** of the development of a **reusable** and **improvable** adaptation logic for SASs throughout the whole lifecycle.*

Marked in bold letters are the main elements for this thesis: (i) supporting the developer throughout the whole lifecycle of a SAS with (ii) reusable artifacts – including tools, processes as well as reusable components for SASs – and (iii) self-improving the system, i.e., adaptation of the adaption logic. Accordingly, the thesis will answer the following three research questions:

***RQ*1 Reusability**: How to make the adaptation logic more reusable?

***RQ*2 Self-improvement**: How to adapt the adaptation logic at runtime?

***RQ*3 Integrated development**: How to support the development of SASs with tools and processes throughout the complete lifecycle?

1.3. Contributions

This thesis presents a generic framework for developing reusable SASs integrating self-improvement. Its main contributions are as follows.

First, a thorough analysis of the state of the art is presented. Based on several publications [220,224,226,227,229], the analysis is tripartite and analyzes subsets of the available approaches focusing on (i) adaptive systems in general, (ii) development approaches for SASs, and (iii) self-improvement of SASs. Throughout this thesis, the results of the analysis act as knowledge base for the definition of requirements (cf. Chapter 5) and the design of the main artifacts. However, it can be used by other researchers to identify further research gaps in SASs research.

Second, the design of the first main artifact is the FESAS Framework that targets the development of reusable SASs. If integrates a reference architecture with reusable components, a well-defined process enriched with tools as well as deployment support. Further, it offers several elements to support the integration of a module for self-improvement though adapting the adaptation logic at runtime. Accordingly, in contrast to related approaches, the FESAS Framework offers end-to-end development support throughout the lifecycle of a SAS.

Third, the second main artifact is the so called Adaptation Logic Manager (ALM) for self-improvement. It aims at simplifying the development by offering a reusable, flexible, and customizable approach which extends the flexibility offered by other approaches in literature that are often optimized for a specific setting. The ALM is self-contained and can be integrated in frameworks for developing SASs. Accordingly, we present it separated from the FESAS Framework in this thesis. However, it is perfectly integrated with the FESAS Framework as the FESAS Framework offers all necessary elements for integration.

Fourth, this thesis describes an integrated prototype implementation that combines the tools and elements of the FESAS Framework with the ALM. Specifically, this thesis presents the implementation of the FESAS Framework, including the FESAS IDE, the FESAS Middleware, the FESAS Repository, and the ALM. The implementation is the foundation for a *proof by prototyping* evaluation.

Last, this thesis evaluates the prototype. Therefore, the FESAS Framework and the ALM were applied for implementation of several SASs. In these case studies, we analyze the usability of the tools, the performance of the ALM, as well as the degree of reusability that the FESAS Framework enables.

1.4. Structure

The structure of the thesis reflects these contributions. Next, Chapter 2 introduces the concepts *self-* properties* and *self-adaptive systems*, presents control structures for SASs and sorts SASs into the research landscape. Chapter 3 explains the research methodology for this thesis based on the *Design Science Research Methodology Process Model* according to Peffers *et al.* [289]. As knowledge base for the requirements specification and the design, Chapter 4 analyzes related approaches and identifies their shortcomings. Based on the analysis of related work, Chapter 5 derives the requirements for the FESAS Framework and the ALM. Chapter 6 presents the design of (i) the FESAS Framework, (ii) the ALM as well as (iii) the integration of both in an end-to-end development approach. Following the design, Chapter 7 describes the implementation of an integrated prototype for both design artifacts. However, both artifacts – FESAS Framework and ALM – can be used independently from each other. Subsequently, Chapter 8 presents the evaluation of the prototypes. Chapter 9 integrates the evaluation results in a cross-case discussion of the suitability of the design artifacts in regard of the research questions and presents future work. Last, Chapter 10 concludes the thesis with a summary of the results.

1.5. Formal Conventions

In order to structure the work semantically different font styles are incorporated. Important terms or names are formatted in an italic font style. The term *Self-adaptive System* is an example for this convention. Code or any other technical textual content is displayed in the following manner: `Class`. Regarding the styles for citations, four different types are present. The [Ref] citation is the normal citation style for indirect citations. An extended [Ref,page] citation with a page number marks direct citations or indirect citations of monographs. The citation "e.g., [Ref1,Ref2,...]" is used for naming a subset of references while "c.f. [Ref1(,...)]" specifies one or several references for additional information.

2. Background

The previous chapter motivated the present thesis and specified its research questions as well as its contributions. This chapter introduces the theoretical background by defining important terms and presenting related concepts. At first, Section 2.1 presents an overview on the *self-* properties* that establish self-management of (self-)adaptive systems. Based on that, Section 2.2 discusses different definitions for the term *Self-adaptive System* (SAS) and presents the commonly accepted system model which divides the managed resources – that perform the system's functionality – from the adaptation logic which controls the adaptation of the managed resources. Following, Section 2.3 compares different control structures for implementing the adaptation logic of a SAS. Last, Section 2.4 distinguishes SASs from similar concepts in the research landscape.

2.1. Self-* Properties

In the literature, *self-* properties* are seen as fundamental for self-management of software systems [198, 234, 320]. According to Salehie and Tahvildari [320], these properties can be ordered hierarchically into three levels: (i) primitive level, (ii) major level, and (iii) general level. On the primitive level, *self-awareness* and *context-awareness* are the basic functionality to retrieve information about the system resources as well as the surrounding environment (cf. Section 2.1.1). The Self-CHOP properties of the major level use the awareness to offer autonomic system functionality (cf. Section 2.1.2). On the general level, systems that fulfill the self-adaptiveness property aggregate the functionality of the major level and offer autonomous reaction to changes in the system and the environment (cf. Section 2.1.3). Figure 2.1 shows the hierarchy of the self-* properties. Additionally to these three levels, recent research (e.g., [112, 226, 315, 361]) identified the need to not only adjust the system resources but also the control mechanism for adaptation. Section 2.1.4 describes the corresponding *self-improvement* functionality.

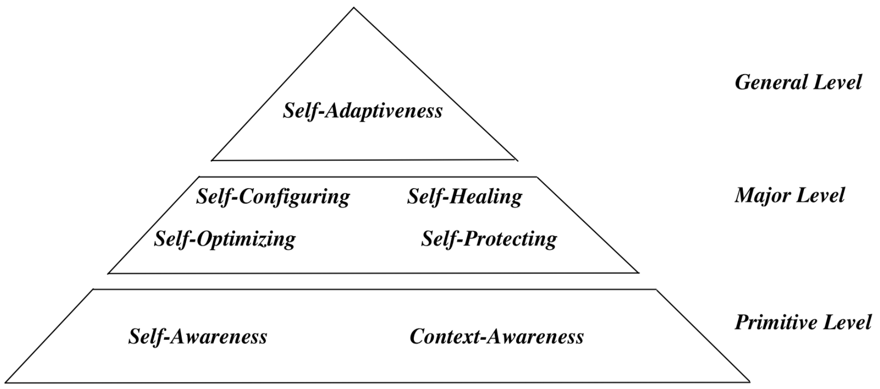


Figure 2.1.: Hierarchy of the self-* properties according to [320]. Based on the primitive functionalities of self-awareness and context-awareness, the Self-CHOP properties enable autonomic behavior. Self-adaptation subsumes and controls the Self-CHOP functionality.

The set of the here presented self-* properties focuses on the relevant ones for this thesis. Additional properties may be found in the literature, e.g., [41, 234].

2.1.1. Primitive Level: Self-awareness and Context-awareness

For reasoning on adaptation, the system has to be aware of itself as well as its environment. The primitive level describes this basic functionality of any adaptive system. These functionalities are self-awareness and context-awareness.

Whereas some researches define *self-awareness* as the ability of the system to represent itself, i.e., modeling its software and hardware resources as well as current system state and behavior (e.g., [183]), the outcome of a Dagstuhl seminar on self-aware computing includes the reasoning on adaptation into the definition of self-awareness [216]. In this thesis, we refer to the first perspective as self-awareness whereas the second includes the reasoning which is seen as part of the adaptation decision. Hence, self-awareness describes the ability of a system, to be aware of itself, i.e., to be able to monitor its resources, state, and behavior.

The second aspect of the primitive level refers to the awareness of the system's context. *Context-awareness* describes the system's awareness of its operational environment, the so called context [325]. Again, different definitions of context exist. From a human-machine interaction perspective, Dey formulated a broad definition of context:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.* [101, p. 5]

Contrary, researchers in the domain of SASs define context-awareness more specific as the operational environment of the system (e.g., [285], [278], [76], or [56]). As this definition complements this thesis' definition of self-awareness, in this thesis, context-awareness refers to the system's environment. For achieving context-awareness, the system has to (i) use sensors to collect information about its environment and (ii) reason about the information [229].

The distinction of self- and context-awareness offers further advantages. For self- and context-awareness, the system has to observe different elements. Whereas for self-awareness most often virtual sensor's have to observe the internal system state, for context-awareness often a combination of virtual sensors – e.g., for network bandwidth – and physical sensors – e.g., for conditions of the physical environment, such as light or location – have to gain the necessary information. This conceptually differs for the implementation, hence, the division of self- and context-awareness. Whereas all adaptive systems should react to changes in the system and the environment, i.e., be aware of itself and its environment, not all of them are context-adaptive, i.e., change the environment.

2.1.2. Major Level: Self-CHOP Properties

The major level integrates four properties: self-configuration, self-healing, self-protection, and self-optimization [198]. In this thesis, the name self-CHOP properties is used (cf. [234]). Other works refer to them as self-* or self-management properties (e.g., [198]). *Self-configuration* implies that the system is able to configure itself instead of requiring an error-prone manual installation and configuration [234]. *Self-healing* contains detecting and patching issues without human support. This is linked to self-diagnosing [311] and self-repairing [93]. *Self-protection* refers to automatic defense against malicious attacks and actions [320]. *Self-optimization* is mainly important during the operation phase of a system. Software is permanently monitoring its own performance in order to adjust tuning parameters to changes in the environment. Some researchers refer to this property as self-tuning or self-adjusting [346]. As the Self-CHOP properties rely on information of the system resources and the surrounding environment, they are build upon self-awareness and context-awareness.

2.1.3. General Level: Self-adaptation

Different taxonomies for self-adaptation or adaptation have been developed over the years. Rohr *et al.* describe a classification schema for self-adaptation research [313] among the dimensions *origin*, *activation*, *system layer*, *controller distribution*, and *operation*. Salehie and Tahvildari present an overview on the landscape of self-adaptive software and related research challenges, including their own taxonomy for self-adaptation [320] based on *the object to adapt, realization issues, temporal characteristics*, and *interaction concerns*. Handte *et al.* [167] classify the adaptation support for pervasive applications into *time*, *level*, *control*, and *technique*. Macías-Escrivá *et al.* [244] cluster in their survey *approaches*, *research challenges*, and *applications* for SASs, however, they do not define self-adaptation. Bashari, Bagheri, and Du [30] classify adaptation in *goal*, *cause*, and *mechanism*. Additionally, several works discuss aspects of self-adaptation. Zhang and Cheng distinguish different types of adaptation – *one-point adaptation, guided adaptation*, and *overlap adaptation* – according to the functional interaction of the old and new system configuration [413]. McKinley *et al.* highlight the difference regarding *parameter* vs. *compositional* adaptation [254]. In [278], the authors discuss the spectrum of adaptation from *static* activities to *dynamic* ones. Three Dagstuhl seminars in 2008, 2010, and 2013 focused on research issues regarding the engineering of SASs [76, 94, 96] in *modeling*, *development processes*, *decentralization*, or *verification*. All these works provide important insights into the field of SASs. However, none of them gives an integrated view, incorporating different existing works and aspects focusing on self-adaptation rather than the system or its implementation, which is often domain-specific. In [229], we state such a uniform taxonomy for self-adaptation. In the following, this section presents our taxonomy on self-adaptation (see Figure 2.2).

The first dimension of the taxonomy, *time*, describes when an adaptation is executed in relation to an event which triggers the adaption. The facets are proactive and reactive adaptation. Even though a proactive adaptation is de-

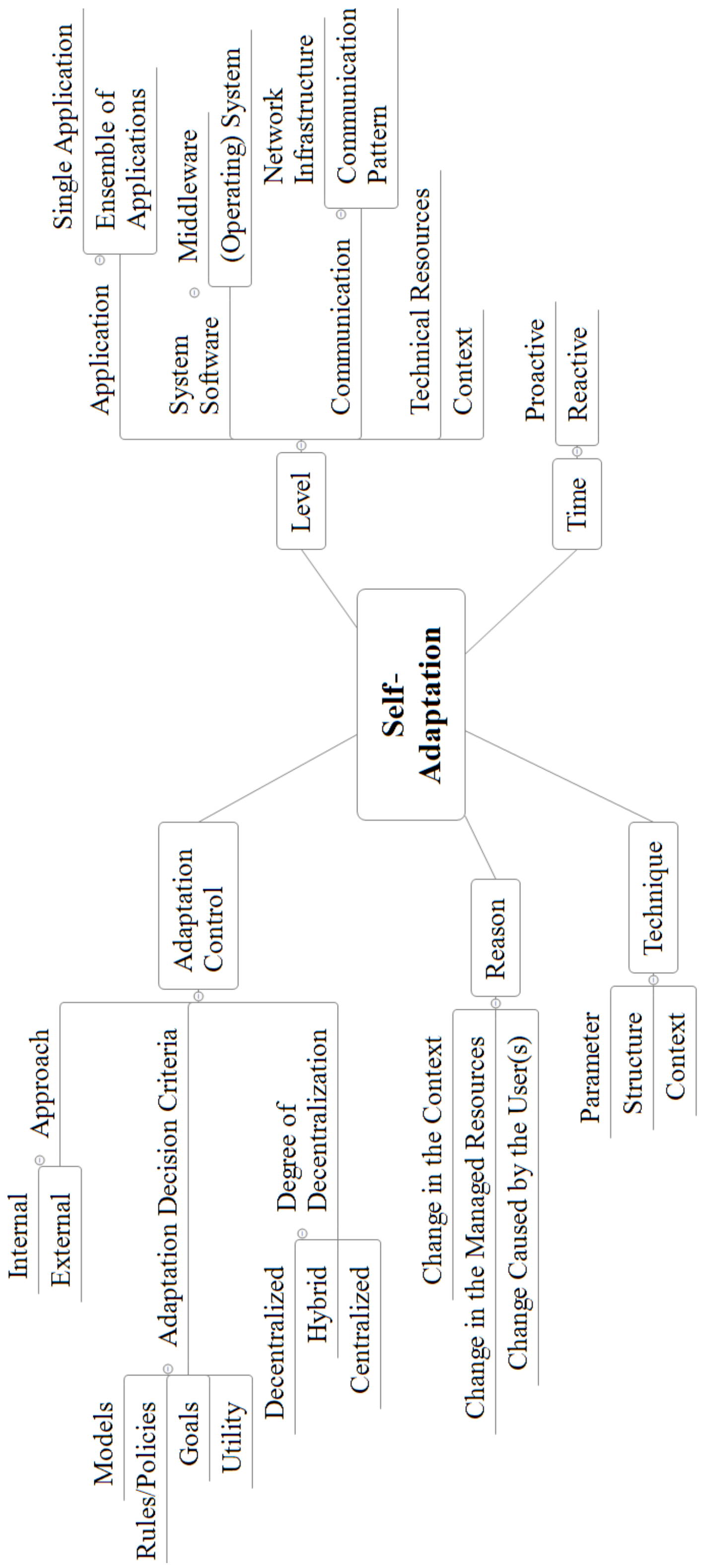


Figure 2.2.: The taxonomy of self-adaptation [229] describes self-adaptation with the dimensions *time*, *reason*, *level*, *technique*, and *adaptation control*.

sirable from the users' point of view, reactive approaches are predominant as proactive adaptations are more difficult to develop [167]. Both techniques can be applied simultaneously: reactive adaptation might act as backup if proactive adaptation fails.

The system has to analyze the potential *reasons* for adaptation, hence, it requires to observe these triggers. Linked to self- & context-awareness, the taxonomy clusters the reasons into:

- environment (e.g., the state of an environment variable changes),
- managed resources (e.g., hardware or software fault), or
- user (e.g., a change in the composition of the user group or the user's preference).

The dimension *level* describes where to adapt. This can be the application, system software, communication, technical resources, or context. The user or other backend systems interact with a single application or an ensemble of distributed applications. Hence, for distributed applications, communication is relevant. Regarding communication, two perspectives are relevant w.r.t. adaptation: a switch of the logical communication patterns – e.g., from a point-to-point communication to a publish-subscribe approach – or a technical adaptation of the network connection, e.g., an adaptation of a mobile device's Internet connection from using 3G/4G to Wifi. System software which can be the operating system or middleware abstract from hardware. Technical resources subsume all types of hardware. All of these hardware and software components are subsumed as managed resources. Last, the context, i.e., the system environment, can be adapted.

[254] describes two *techniques* for adaptation: Parameter adaptation is bound to the adjustment of parameters, structural adaptation triggers changes of the algorithms or system components. In [229], we add context adaptation.

Whereas the previous four dimensions describe properties of self-adaptation, the last dimension refers to how to enable the self-adaptation. First, while internal approaches integrate the control for adaptation with the resources that should be adapted, external ones split them in separated modules [128]. Second, different criteria for the adaptation decision are present in literature: models, rules and policies, goals, and utility functions [234]. These approaches might be combined. Third, the degree of decentralization for the adaptation control can vary between totally decentralized systems, in which each subsystem has a full adaptation control functionality that controls a specific part of the system, and totally centralized ones in which only one subsystem controls adaptation of the whole system [401]. Hybrid approaches are possible as well.

2.1.4. Self-improvement

As mentioned above, the control mechanism for adaptation relies on different metrics for the adaptation reasoning, namely: (i) models, (ii) rules/policies, (iii) goals, or (iv) utility functions. However, they have some shortcomings. Due to uncertainty in SASs at run-time (cf. [307] for a discussion of factors for uncertainty), there is a gap between the design time and the runtime. This might result in outdated or incomplete metrics for the adaptation decision. Some researchers try to tackle this gap by shifting design activities to the runtime and integrating learning of system metrics at runtime (e.g., [361]). However, some uncertainty still exists as it is not guaranteed that the runtime design procedures can handle all situations. Additionally, the configuration space for adaptation decisions is rather large. For example, an evaluation of Lightstone [238] examined that typical middleware products provide between 384 and 1200 configuration/registry parameters. This results in between 10^{115} and 10^{361} system configurations. Obviously, a rule set with adaptation rules can not cover all configurations. On the other hand, models may provide freedom and support learning the specific configuration at runtime. However, they introduce additional uncertainty due to abstraction provided by models. All these issues lead to the requirement of an additional self-* property: the *self-improvement* property. As there was no common definition present in literature, we define self-improvement in [226] as:

> *Self-improvement of the adaptation [control mechanism] is the adjustment of the adaptation logic to handle former unknown circumstances or changes in the environment or the managed resources.* [226]

Similar to self-improvement, other authors use the terms *meta-adaptation* [149,161,182,291], *system evolution* [9,274,291,353] and *meta-self-awareness* [216] to describe self-improvement. Hillmann and Warren define meta-adaptation as "*adaptation in the adaption process itself*" [182, p. 297]. System evolution describes an adaptation of the adaptation control [291]. Meta-self-awareness describes "*a system's awareness of its own self-awareness*" [216, p. 8], in which self-awareness is defined similar to self-adaptation (cf. [216]).

In accordance to these definitions, in our understanding, a system can only self-improve if the adaptation control itself changes. Otherwise, the adaptation control can neither handle unknown situations nor improve the performance of adaptations. In contrast, self-optimization changes the managed resources but not the adaptation control. The same is true for hierarchical self-optimizing approaches (e.g., [193] or [154]) as the hierarchy offers decision-making on different levels but does not change the adaptation control mechanism in a substantial way. The increasing importance of self-improvement is reflected in the research community by arising workshops, such as the *International Workshop on Self-Improving System Integration*. This thesis contributes to this research with an inclusion of a generic reusable approach to self-improvement at runtime.

2.2. Self-adaptive Systems

The term *Self-adaptive System* encompasses, in its broadest definition, different types of systems which are able to adapt themselves, such as the adjustment of the human eye to the conditions in the environment. A narrower definition just refers to *Self-adaptive Software (Systems)*. When using the term *Self-adaptive System* in this work, the second, more specific meaning of *Self-adaptive Software System* is addressed. However, often these terms are used interchangeably [76,229,278]. The literature provides several definitions of SASs. The following gives a comparison of definitions in order to show the evolution and variety of SASs. Additionally, it presents the system model of such systems.

One of the first definitions of SASs is presented by Laddaga in a DARPA Broad Agency Announcement on SASs in 1997 and became later the first commonly agreed definition:

> *Self Adaptive Software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible.* [233, p. 17]

Two years later, Oreizy *et al.* define SASs in a similar way:

> *Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation.* [278, p. 55]

In contrast to Laddaga, Oreizy *et al.* include user input explicitly as change reason. In 2009, Salehie *et al.* summarize the core of SASs to be as follows:

> *The key point in self-adaptive software is that its life-cycle should not be stopped after its development and initial set up. This cycle should be continued in an appropriate form after installation in order to evaluate the system and respond to changes at all time. Such a closed-loop deals with different changes in user requirements, configuration, security, and a number of other issues.* [320, p. 4]

This way, the authors point out the importance of the capability of a SAS to make and implement decisions at runtime without an interruption of the system. Hence, the capability to improve the system at runtime in regard to unforeseen events becomes an important characteristic. The participants of the first Dagstuhl seminar on *Software Engineering for Self-adaptive Systems* agreed on the following, quite generic definition:

> *[Self-adaptive systems] are able to adjust their behaviour in response to their perception of the environment and the system itself.* [76, p. 1]

Additionally, the participants of the Dagstuhl seminar clarified the term *self*:

> *The "self" prefix indicates that the systems decide autonomously (i.e., without or with minimal interference) how to adapt or organize to accommodate changes in their contexts and environments.* [56, p. 49]

In contrast to this non-specific view, a recent definition from a Dagstuhl seminar on verification in SASs focuses on the triggers for adaptation:

> *Self-adaptive software systems adapt to changes in the environment, in the system itself, in their requirements, or in their business objectives.* [328, p. 1]

The collection of definitions is not exhaustive but shows that the perception of SAS in literature varies in the details of what triggers adaptation. First, definitions as the one from Oreizy *et al.* [278] focus on changes in the environment. This is for the understanding of SASs in this thesis not enough, as changes in the system itself – leading to self-healing – are not included. Hence, to distinguish from context-aware systems, this thesis includes changes in the system. Second, some authors as [328] detail the elements that might be changed. However, this might be too narrow. Further, different understandings of these elements might limit the applicability of the definitions, e.g., for the definition of [328] it is not clear, whether the user is responsible for the change in business objectives or it is part of the environment. In contrast, the third category of definitions is rather generic regarding the adaptation reasons. Additionally, the third category shifts from self-adaptive software systems to self-adaptive systems, which highlights the fact that these systems not only adapt software but also hardware resources. Hence, this thesis follows the definition of the first Dagstuhl seminar:

> *[Self-adaptive systems] are able to adjust their behaviour in response to their perception of the environment and the system itself. The "self" prefix indicates that the systems decide autonomously (i.e., without or with minimal interference) how to adapt or organize to accommodate changes in their contexts and environments.* ([76, p. 1]; [56, p. 49])

From an architectural point of view, a SAS is composed of two parts: a managing system as well as a managed system [76, 96, 328]. For both elements, different terms are present in literature. The managed system is often denoted as managed resources (e.g., [229]), managed elements (e.g., [198]), adaptable subsystem (e.g., [327]), or system under observation and control (e.g., [361]). In this thesis, we use the term *managed resources.* The managed resources are a set of resources $MR = \{mr_1, ..., mr_n\}$ with mr_i as any kind of software and hardware, e.g., servers, laptops, smart phones, robots, or unmanned vehicles. In literature, the terms control mechanism (e.g., [327]), observer/controller (e.g., [361]), adaptation logic (e.g., [229]), autonomic manager (e.g., [198]), or adaptation control (e.g., [167]) are used interchangeably for the managing system. This thesis uses the term *adaptation logic.* The adaptation logic $AL = \{al_1, ..., al_n\}$ is a set of software modules al_i that observes the environment and the managed resources, analyzes the need for adaptation, plans such adaptations as well as

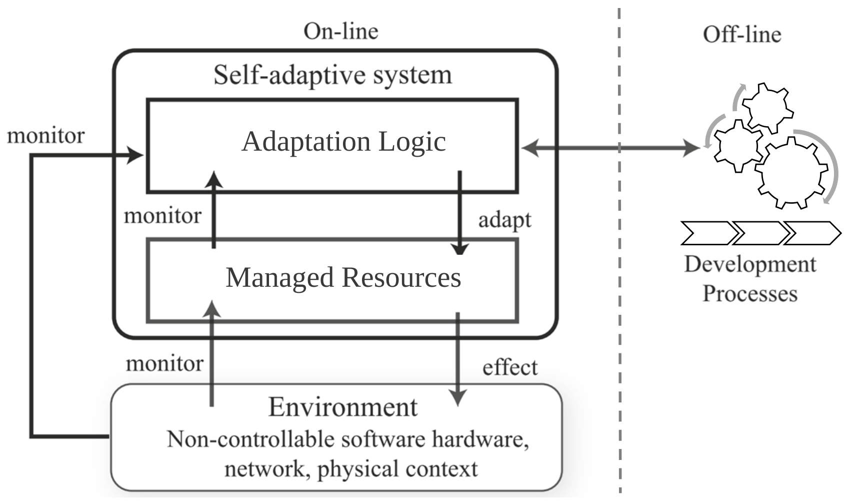


Figure 2.3.: System model of a SAS from an architectural point of view (adapted version: in contrast to [328], the adaptation effects the environment).

controls the execution of the adaptation. Hence, the common SAS is defined as a tuple $SAS = (AL, MR)$ with the adaptation logic AL and the managed resources MR [229]. Figure 2.3 shows the architectural model of a SAS.

Our survey [229] showed that most approaches do not sufficiently include context. Whereas most approaches monitor the context, explicit adaptation of context is often not included and the environment remains uncontrollable for the adaptation logic [11]. This can lead to undesired adaptation results especially in system domains with mobile systems. Therefore, this thesis uses the concept of context-altering SAS as proposed in [229]. The context-altering SAS does not just (accidentally) affect the environment, but it deliberately adapts it through modeling context and context-altering capabilities as a construct to enables reasoning about it[1]. This extends the SAS to a triple $SAS = (AL, MR, Ctx)$ with the adaptation logic AL, the managed resources MR, and the context Ctx. Different context variables – e.g., temperature, noise, location, or available devices – define the physical context [329]. Additionally, users, social interactions, or their task define the user context [329]. They are influenced via actuators of the managed resources. Therefore, the context is modeled as set $Ctx = \{ctx_1, ..., ctx_n\}$ where each ctx_i symbolizes a context variable, e.g., the temperature.

[1]In the following, this thesis omits the extension *context-altering* and just uses SAS.

The adaptation control characteristic of the taxonomy for self-adaptation [229] describes general implementation issues, abstracted from the implementation of the adaptation logic for a specific use case. The adaptation logic can be intertwined with the rest of the application or separated. For analyzing and planning of the adaptation, the adaptation logic can use rules/policies, goals, models, or utility functions. Another issue is the degree of decentralization of the logic resulting in a distribution of the adaptation control functionality on different sub-system parts. Further, the task of the adaptation logic is to implement functionality that addresses the other four dimensions of the taxonomy on self-adaptation [229] (cf. Section 2.1.3) as these are relevant for the adaptation decision. Next, Section 2.3 compares control structures for the adaptation logic.

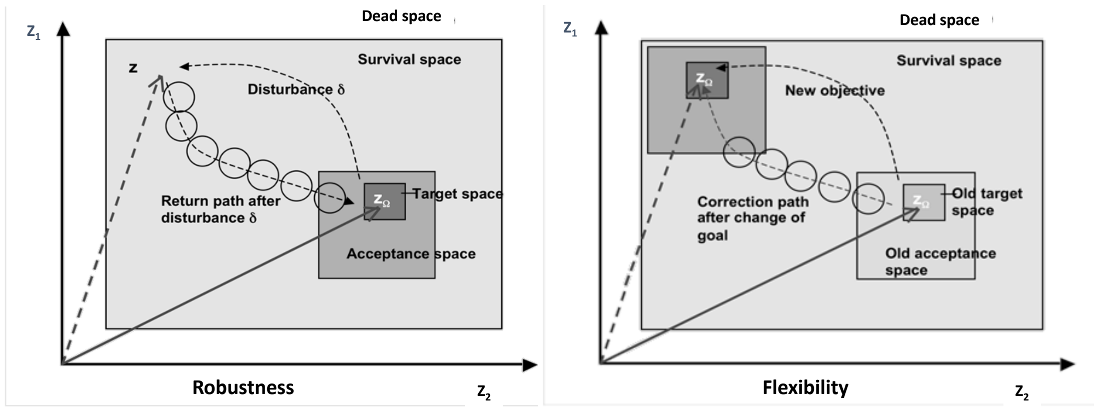


Figure 2.4.: Model of an adaptive system according to [327]. The left side shows a robust reconfiguration, the right side shows a flexible one.

2.3. Control Structures for Adaptation Control

The goal of the adaptation logic is to fulfill the adaptation targets. According to [327], five dimensions describe the adaptation logic's functionality. First, the adaptation logic has to determine the current state of the system and its environment. Second, it has to evaluate the current system performance w.r.t. some evaluation criteria. Third, if the system's performance is not in the target space or acceptance space, it has to be adapted by the adaptation logic. Fourth, the adaptation logic has to tolerate disturbances of the system performance as long as the system is in a survival space, i.e., it can achieve a system state that is acceptable through adaptation. Last, the adaptation logic has to define a reconfiguration of the system for implementing changes in the resources. These reconfigurations lead to a reconfiguration path of different intermediate settings with the objective to either adjust the system to the given objectives, called *robustness*, or the adjustment to new objectives, hence, new target and acceptance spaces, called *flexibility* [327]. Figure 2.4 presents this model of adaptive systems.

In case of a robot, the adaptation Target could be to move to a certain target. For the reconfiguration, the adaptation logic determines adaptation actions. These actions have an influence on both the environment and system resources. The adaptation logic detects these changes in a subsequent analysis. Using the example again, it could be that the robot moved a few steps, but appeared to be in front of an obstacle so the movement speed drops to zero. This way, a continuous feedback loop is created that is reinforced by positive feedback, while negative feedback triggers a change [56]. This feedback loop is a core principle of the adaptation logic [56, 76, 198].

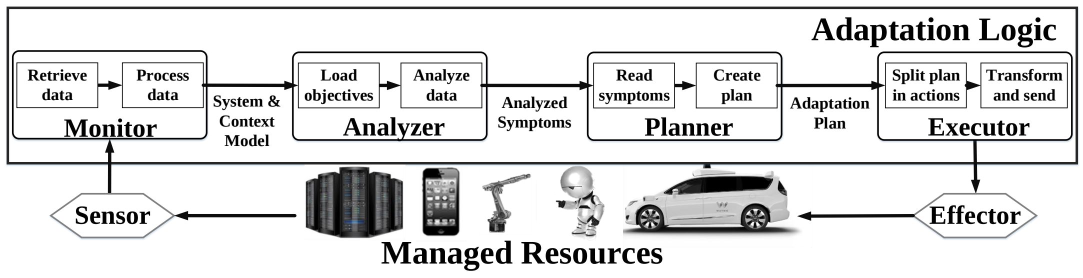


Figure 2.5.: MAPE Cycle according to Kephart and Chess [198]. Own visualization based on [221]. The knowledge base is omitted here.

Figure 2.5 shows the MAPE cycle by Kephart and Chess [198], a specific feedback loop structure from the Autonomic Computing domain. The MAPE cycle is named by its four main steps: monitor, analyze, plan, and execute. The *monitor* function collects data of the system and the environment and pre-processes it, e.g., through filtering, reliability analysis, or categorization. Afterwards, the *analyzer* decides if adaptation is needed by comparing the monitored data and system objectives. The *planner* computes the adaptation plan based on the monitored information and interpretations of the analyzer. If more than one possible adaptation is found, the planner has to decide which one matches the given system objectives best. Finally, the *executor* allocates the planning instructions and sends them to the corresponding managed resources. This way, the adaptation is triggered in the managed resources. Afterwards, the process starts over again. Sensor and effectors are interfaces to gain data from managed resources or send instructions, respectively.

The MAPE cycle is the de facto standard for control loops of SASs [76, 229, 384]. Other authors propose similar feedback control for SASs, such as, the sense-plan-act control [218], the autonomic control loop [105], the models@run.time reference architecture [21], the requirements reflection architecture [39], the observer/controller architecture [362], or the MIAC/MRAC controllers [56]. Patikirikorala *et al.* [288] provide an overview on such structures. However, all implement a feedback loop structure similar to the MAPE cycle. Some structures subsume analyzing and planning as reasoning [234]. Additionally, feed-forward loops can enable proactive adaptation. However, their complexity hinders implementation. Further, current machine learning frameworks eliminate the need of proficient mathematical knowledge for implementing prediction of future system states. In combination with the increasing computational power as well as omnipresent cloud resources, proactive adaptation based on predictive analysis can be implemented on top of the well-known feedback loops. This is a suitable alternative for implementing proactive adaptation in feed-forward loops.

The taxonomy on self-adaptation defines the required functionality an adaptation logic has to offer. Obviously, the MAPE components have to integrate this functionality. In [229], we mapped the dimensions time, reason, level, and technique of the taxonomy to the MAPE functionality. The *time* dimension influences the decision of analyzing algorithms as proactive recognition of the need for adaptation has other requirements – especially the need for predictions – as reactive detection of changes. Monitoring should be continuous no matter whether the adaptation is proactive or reactive. The *reason* dimension influences monitoring, analyzing, and planning, as it describes the reasons for adaptation and, therefore, the aspects that should be monitored, where analyzing has to determine changes as well as the issues, that must be addressed with the adaptation plans. The *level* for adaptation is obviously important for planning and executing as these activities must be aware of the elements that should be adapted. Monitoring has to determine the elements for the levels that are present in the managed resources. The *technique* dimension influences the planning and executing, as planning describes which adaptation techniques to use on which elements and executing controls the application of the techniques. The fifth dimension *adaptation control* describes the structure of the adaptation logic and is not related to any specific MAPE functionality as this dimension is concerned with the implementation details of the aforementioned four dimensions. Table 2.1 presents the mapping of the MAPE functionality to the dimensions of our taxonomy.

Table 2.1.: Mapping of the MAPE functionality to the dimensions of the taxonomy from [229] (cf. Section 2.1.3).

	Time	**Reason**	**Level**	**Technique**
Monitoring	Continuous	What to monitor	Identification of the levels	—
Analyzing	Algorithms vary for reactive and proactive adaptation	Where to analyze	—	—
Planning	—	What should planning influence	Adaptation plans address these levels	Plans for performing the techniques
Executing	—	—	Execution of the change on the levels	Execution of the techniques on different elements

2.4. Research Landscape

SASs are related to other research disciplines and areas. On the one hand, SAS research integrates concepts from other research domains. On the other hand, some research disciplines are similar to SAS. This section sorts the SAS research into the research landscape and introduces related concepts.

First of all, there are different terms that are used interchangeably to describe the same concept: Dynamically Adaptive Systems (e.g., [414]), Autonomic Systems (e.g. [320]), Self-managing Systems (e.g., [320]), Self-adaptive Systems (e.g., [76]), or Self-adaptive Software Systems (e.g., [279]). This thesis constantly uses the term *Self-adaptive Systems*. All of them follow the system model of Section 2.2. Further, there are other approaches that are similar to SASs.

The mostly related concept is Autonomic Computing [198, 234]. In the Autonomic Computing domain, researchers integrate principles from biology, mainly from the autonomous nervous system [234], to equip systems with autonomic capabilities. The former presented MAPE control loop arose in the Autonomic Computing area. Researchers relate Autonomic Computing and SASs differently. Whereas Salehie and Tahvildari [320] consider SASs as a subcategory of Autonomic Computing, McCann and Huebscher [191] use the terms interchangeably. We argue that SASs are the overarching concept [229] as both include self-adaptation, but Autonomic Computing is rooted in Cloud Computing.

In contrast to the definition of self-awareness provided in Section 2.1.1, the authors of [216] add capabilities to (i) reason on the knowledge of self-awareness and (ii) acting accordingly. This makes their definition of self-aware computing systems identical to this thesis view on SASs. However, in accordance with other works (e.g., [183], [234], and [229]), this thesis clearly distinguishes self-awareness – capturing knowledge on itself – and self-adaptation, i.e., acting on self-awareness and context-awareness. Similarly, the term context-awareness – sometimes referred to as situation-awareness [61] – might be defined as in Section 2.1.1 or in an extended version with the reasoning process on context-awareness [325].

For the first *International Conference on Self-Adaptive and Self-Organizing Systems*, the program chairs defined SASs as top-down systems with central control [24], whereas Self-organized Systems (SOSs) are dedicated units that organize themselves without a central instance [24], i.e., they work bottom-up. However, reviewing the current literature of SASs shows that SASs offer both, centralized and decentralized system control [229]. Hence, the division from [24] does neither fit for defining SASs nor for distinguishing SASs and SOSs anymore. The "dedicated units" in SOSs might be implemented as SASs. Accordingly, the proper relation seems to be that a SOS can be composed of SASs.

Pervasive/Ubiquitous Computing [388] aims at the seamless integration of IT and everyday devices to support humans by smart IT. These systems are often context-aware and adaptive. However, they target solutions in the IoT domain rather than generic systems.

Organic Computing is associated with systems that use bio-inspired concepts to implement organic behavior [268]. Similar to SAS, Organic Computing systems try to achieve the self-* properties (cf. Section 2.1). In contrast, they focus on (i) the integration of principles from nature-inspired computing, (ii) emergence of systems for shifting design activities to runtime, and (iii) the human-in-the-loop as first class entity rather than an element to avoid.

Additionally to these closely related research domains, the domain of SASs relies on approaches from many other domains for different tasks, such as reasoning, modeling, implementation of the adaptation logic, or adaptation enactment. These include control theory, artificial intelligence, reflective computing, models@run.time, ad-hoc networks, dependable computing, embedded systems, Multi-agent Systems (MASs), Service-Oriented Architectures (SOAs), Aspect-Oriented Programming (AOP), Component-Based Development (CBD), distributed systems, fault-tolerant computing, nature-inspired computing, or robotics [76, 234, 320].

This thesis sees SASs as general concept of systems that can adapt themselves and, through that, also the environment for handling changes in the system or its environment. They are not grounded on a specific application domain (e.g., as Autonomic Computing) nor on a specific system model (e.g., as MASs). Hence, the results of this thesis are generically applicable to many of the presented different research streams.

3. Research Methodology

Information Systems research offers several facets. The objective of this thesis is to develop a new framework for engineering SASs focusing on reusability of code. This fits the *design science* paradigm, which "*seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts*" [180, p. 75].

Design science roots in the work of Simon [337]. It is centered around the idea to create and evaluate one or more initial design alternatives that fit the problem of interest. Iteratively, the alternatives and revised designs are evaluated until the best solution to the problem is identified. Design science is a research paradigm [31] which is embodied in different methodologies (e.g., [180,289]). This work follows the methodology of Peffers *et al.* [289] as it covers the traditional phases of Software Engineering, hence, it is compatible with software development activities. Peffers *et al.* defined six steps [289]: (i) problem identification and motivation, (ii) definition of the objectives for a solution, (iii) design and development, (iv) demonstration, (v) evaluation, and (vi) communication (cf. Figure 3.1). In the following, this section explains the different steps in detail and maps these steps to the procedure used within this thesis.

Step (i) identifies the problem and motivates its relevance. For the identification of the problem, sufficient knowledge of the problem is necessary [289]. In this thesis, this knowledge is represented by an analysis of related work. Accordingly, Chapter 4 identifies the problem by analyzing the current state of the art in engineering adaptive systems, development support for SASs, and self-improvement for identifying research gaps. Often, for such overviews, the *systematic literature review* technique is used to get a complete systematic overview on the field and identify all relevant literature [208]. As the mentioned topics are very broad and the characteristics of the publications are highly diversified, instead of a systematic literature, we did a *systematic mapping* [292] to identify the structure of the research field and analyze the most relevant literature.

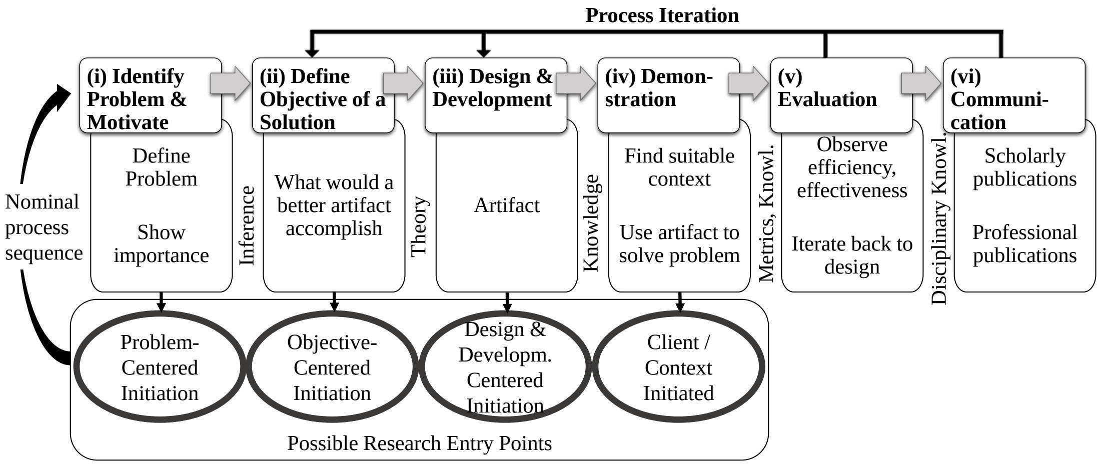

Process Iteration
(i) Identify Problem & Motivate
(ii) Define Objective of a Solution
(iii) Design & Development
(iv) Demon-stration
(v) Evaluation
(vi) Communi-cation
Define Problem
Show importance
Inference
What would a better artifact accomplish
Theory
Artifact
Knowledge
Find suitable context
Use artifact to solve problem
Metrics, Knowl.
Observe efficiency, effectiveness
Iterate back to design
Disciplinary Knowl.
Scholarly publications
Professional publications
Nominal process sequence
Problem-Centered Initiation
Objective-Centered Initiation
Design & Developm. Centered Initiation
Client / Context Initiated
Possible Research Entry Points

Figure 3.1.: Design Science Research Methodology Process Model according to Peffers *et al.* [289] (own visualization based on [289]).

Based on the problem identification, step (ii) defines the objectives. Peffers *et al.* divide quantitative and qualitative objectives [289]. Quantitative objectives define measurements in which the new artifact outperforms current solutions. Qualitative ones describe how the new artifact provides solutions to problems that have not been addressed so far. For both categories of objectives, the problem identification acts as source of knowledge. In this work, the requirements analysis in Chapter 5 represents the identification of qualitative objectives.

Step (iii) is the most important step as it involves creating the artifact, i.e., the solution to the problem. The solution can be models, methods, constructs, or other instantiations. More general, Peffers *et al.* define the artifact as "*any designed object in which a research contribution is embedded in the design*" [289, p. 55]. The artifact created in this thesis is the FESAS Framework including the Adaptation Logic Manager. Accordingly, Chapter 6 describes the design of both and their components.

Showing the feasibility of the created artifact is split in two steps. First, step (iv) demonstrates the applicability of the artifact to solve one or more of the issues defined in the problem identification (step (i)). This can be done by using the artifact, e.g., in experiments, case studies, simulations, or proofs of concept [289]. Following a proof by prototyping approach, Chapter 7 describes the implementation of a prototype of the FESAS Framework following the design from Chapter 6. Second, step (v) covers the analysis of the performance of the artifact for fulfilling the defined objectives. This thesis provides a mixture of evaluation methods: functional testing, static and dynamic analyses, controlled experiment, field study, and informed arguments. Chapter 8 presents details on the evaluation. Following, Chapter 9 discusses the evaluation and the fulfillment of the identified underlying problem, i.e., the requirements defined in Chapter 5.

Step (vi) includes the communication of the solution, e.g., in research publications or to management (in business settings). Mapped to this thesis, on the one hand, the thesis itself communicates the solution. Further, parts of the presented work have been published in various scientific papers (cf. Figure 3.2).

The methodology of Peffers *et al.* [289] enables the flexibility to start at almost any step. As this thesis starts at step (i), it follows a problem-centered initiation approach (cf. Figure 3.1). Further, it targets an iterative process with jumps back to the design (step (iii)). This thesis presents the different steps in sequential order. However, the process of working out the content was not sequential as newer publications extend the contributions of older ones. Figure 3.2 presents the adapted version of the process for this thesis as well as the mapping of steps to chapters. Additionally, the figure provides an overview on the iterations and their corresponding communication in form of publications.

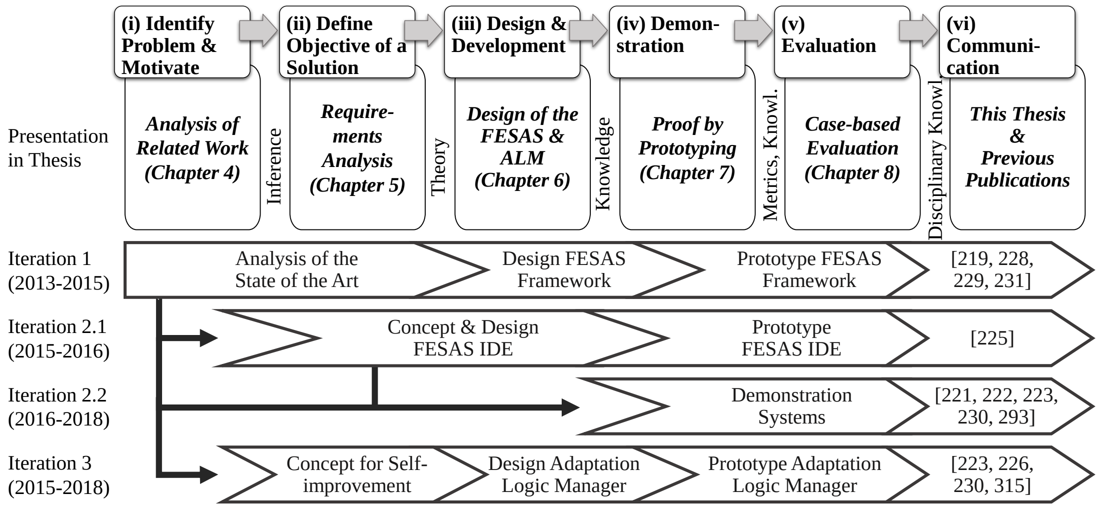

(i) Identify Problem & Motivate
(ii) Define Objective of a Solution
(iii) Design & Development
(iv) Demon-stration
(v) Evaluation
(vi) Communi-cation
Presentation in Thesis
Analysis of Related Work (Chapter 4)
Inference
Require-ments Analysis (Chapter 5)
Theory
Design of the FESAS & ALM (Chapter 6)
Knowledge
Proof by Prototyping (Chapter 7)
Metrics, Knowl.
Case-based Evaluation (Chapter 8)
Disciplinary Knowl.
This Thesis & Previous Publications
Iteration 1 (2013-2015)
Analysis of the State of the Art
Design FESAS Framework
Prototype FESAS Framework
[219, 228, 229, 231]
Iteration 2.1 (2015-2016)
Concept & Design FESAS IDE
Prototype FESAS IDE
[225]
Iteration 2.2 (2016-2018)
Demonstration Systems
[221, 222, 223, 230, 293]
Iteration 3 (2015-2018)
Concept for Self-improvement
Design Adaptation Logic Manager
Prototype Adaptation Logic Manager
[223, 226, 230, 315]

Figure 3.2.: Research methodology based on an adapted version of the *Design Science Research Methodology Process Model* of Peffers *et al.* [289].

4. Related Work

This chapter presents related work in three areas: (i) engineering adaptive systems, (ii) development approaches for SASs, as well as (iii) approaches for self-improvement. First, Section 4.1 introduces concepts for engineering SASs and concepts from related research areas as the ones presented in Section 2.4. The section is based on the survey of [229] as well as the technical report of [220] which provides an updated version[1]. Second, Section 4.2 compares different development approaches, focusing on frameworks, guidelines, tools, design processes, and methodologies. This is a more narrow overview on development approaches which focuses on the specific requirements for developing SASs. The section presents the results of the analysis of such approaches in [224][2]. Third, Section 4.3 summarizes the comparison of systems that offer self-improvement. Its foundations are the survey of [226] as well as the updated technical report of [227][3]. Last, Section 4.4 demarcates this thesis from related work by analyzing shortcomings of the presented related work and motivating the research gap for this thesis.

4.1. Engineering Methods for Self-adaptive Systems

As described in Section 2.4, the research of SASs is related to different other domains. Hence, it makes sense to integrate these related disciplines when analyzing related works. Therefore, in [220] and [229], we analyze engineering approaches for research disciplines that are related to SASs for showing the diversity of approaches and the type of development support (cf. research question *RQ1*) that are present in literature. In the following, this section presents a summary of [220] and [229]. Appendix A.1 presents the captured data of the most relevant

[1] [220] and [229] are joint works with M. Breitbach, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker.

[2] [224] is joint work with M. Pfannemüller, V. Voss, and C. Becker.

[3] [226] and [227] are joint works with M. Pfannemüller, F. M. Roth, and C. Becker.

approaches. For further details of the analysis, the interested reader is referred to [220] and [229].

4.1.1. Model-based Approaches

Models are one of the four decision criteria for adaptation [229, 234]. Runtime models or models@run.time support monitoring – e.g., by representing the system and its environment – and reasoning, e.g., by comparing a model capturing the actual system state with a desired system state. According to [191] and [400], three types of models are used for monitoring and reasoning in a SAS: (i) system models, (ii) adaptation decision models, and (iii) environment models. Three types of system models can be found in literature. Architectural models represent the system's architecture (e.g., [128, 140, 279]). Feature models capture the functionality of software (e.g., [4, 32, 72, 112, 262, 263]). Behavioral models describe the behavior of the system and the possible transitions or its performance (e.g., [26, 152, 153, 213]). Adaptation decision models subsume goal models, rules/policies, utility functions, or its requirements. If the system does not fulfill its goals, adaptation is required. These models support reasoning and the creation of adaptation plans (e.g., [75, 155, 218, 253, 290, 383]). Last, environment models capture the system's context (e.g., [43] or [329]).

Different approaches for SASs use model-driven techniques, e.g., Dynamic Software Product Lines (DSPL) (e.g., [4, 28, 30, 38, 72, 85, 157, 164, 262, 293, 294, 321], combinations with model-driven development (e.g., [128, 145, 154, 163]), metamodel-based automatic reasoning (e.g., [237, 378–380, 382]), verification of system states (e.g., [127, 330, 414]), or models@run.time for SAS construction (e.g., [35, 46, 115, 129, 261, 381]). However, models only support monitoring and reasoning and need to be accompanied by mechanisms for adapting the managed resources, such as Aspect-Oriented Programming (AOP) (e.g., [127, 263]), Component-Based Development (CBD) (e.g., [36, 157]), architecture-based approaches (e.g., [35, 140, 278]), or service-oriented approaches (e.g., [163, 255]).

4.1.2. Architecture-based Approaches

A SAS must be aware of its structure (cf. Section 2.1.1). Architecture-based approaches use software architectures for representing the system structure and reasoning about adaptation [83, 105, 140]. Some approaches use agent-related techniques for implementing architectural change (e.g., [393, 403]) or dedicated architectural components for controlling the adaptation (e.g., [140, 147, 218, 278, 279]). Architecture-based approaches are accompanied by factors for controlling the adaptation, such as strategies (e.g., [140]), policies (e.g., [146, 147, 193, 198, 403]), goals (e.g., [174, 218, 351]), task characteristics (e.g., [295, 344]), or constraints (e.g., [140]). Architecture-based adaptation is complemented with architectural patterns (e.g., [23, 100, 139, 305, 401]), resource prediction for improving self-adaptation (e.g., [82]), or dynamic architectural styles (e.g., [280]). All architecture-based approaches implement MAPE components or components with comparable functionality. Architectural models for monitoring the resources (e.g., [35–37, 62, 79–81, 128, 140, 247, 278, 279, 335]) or Architecture Description Languages (ADLs) for describing architectural models (e.g., [7, 62, 78, 79, 81, 114, 148, 243, 245, 279, 390, 391]) are used in combination with metrics – such as policies, goals, strategies, or constraints – for reasoning about adaptation. All analyzed approaches are external, i.e., they have a dedicated adaptation logic.

4.1.3. Reflection-based Approaches

Reflection is the ability of software to examine and possibly modify its structure (structural reflection) or behavior (behavioral reflection) at runtime [49, 248]. Reflection is divided into two activities: introspection refers to the observation of an application's own behavior, intercession is the reaction on introspection's results [254], i.e., structural, parameter, or context adaptation. Reflection can be used on different levels. Architectural reflection (or structural reflection) enables to reflect on the software architecture of an application, i.e., its components, interconnections, or data types (e.g., [69, 107, 142, 165, 254, 264, 309, 359, 359]). Behavioral reflection refers to reflection on the behavior of the software, e.g., algorithms for computation or communication mechanisms [254, 355, 389]. Reflective middleware (e.g., [13, 48, 86, 211, 212]) can support runtime reconfiguration of a component-based system. Additionally, Sawyer *et al.* propose the use of reflection techniques for introspection of requirements at runtime [323].

4.1.4. Programming Techniques

Different programming paradigms support the execution of parameter and structural adaptation. In Component-Based Development (CBD), also known as Component-Based Software Engineering (CBSE) and Component-Based Programming (CBP), software components encapsulate functionality that can independently from each other be developed, deployed, and composed [352]. Different authors propose adaptation on a component level (e.g., [6, 33, 36, 37, 47, 107, 187, 189, 195, 209, 218, 239, 254, 275, 335, 351]). In Aspect-Oriented Programming (AOP), software is divided in distinct concerns. The goal is to re-use generic functionality in different classes (cross-cutting concerns). Therefore, logical concerns are separated from the specific implementation [203]. McKinley *et al.* propose AOP for enabling separation of concerns, which leads to simplified compositional adaptation [254]. Different authors integrate techniques from AOP in their SASs (e.g., [73, 162, 188, 262, 263]). Other authors propose generative programming for SASs [254, 276, 402]. In generative programming, software is built with the help of high-level descriptions that are mapped to generic classes, templates, aspects, and components [88]. Adaptive programming techniques can support the development of SASs [3, 158, 339] through learning of new behavior at runtime. In Context-oriented Programming (COP), the context is incorporated as a first-class construct in programming languages (e.g., [22, 184, 197, 322]. COP enables reasoning on the context and to appropriately adapt. The presented programming techniques mainly support planning and executing compositional, reactive adaptation.

4.1.5. Control Theory-based Approaches

Section 2.3 highlights the importance of control structures for the adaptation logic [56]. Control structure engineering for SASs focuses on feedback loops which results in reactive adaptation. It poses different challenges: developing reference architectures for control loops, creating a catalog of control loop structures, middleware support for control loop integration, verification & validation techniques to test and evaluate control loops' behavior, and integration of the user [76]. To address these issues, knowledge from control theory can be used for the development of SASs [124]. Researchers propose learning approaches in order to dynamically adapt the control law (e.g., [210]), modeling of feedback loops as first class design elements (e.g., [175, 232, 267]), or using control theory for automatic design of controllers for reasoning (e.g., [104,123]). The reader is referred to [288] for further information on using control theory for designing SASs.

4.1.6. Service-oriented Approaches

Services are encapsulated and autonomous units of software that fulfill a specific task [255]. In Service-Oriented Computing (SOC) such services are used to "*support the development of rapid, low-cost, interoperable, evolvable, and massively distributed applications*" [284, p. 64]. The key for SOC is the Service-Oriented Architecture (SOA), which enables finding, using, and connecting services. The service approach can be mapped to SASs. The naive approach for building a service-oriented SAS is to model the managed resources' functionality as services. The adaptation logic decides, which services should run and in which order. Therefore, dynamic exchange of services enables structural adaptation [103]. Different frameworks for SASs use service-oriented techniques as described (e.g., [60,102,128,145,156,163,240,255]). Other approaches combine AOP and SOC for building SASs (e.g., [73, 188]), take advantage of SOAs as technical base for implementing the variability of DSPL (e.g., [28]), integrate agent-based approaches and SOAs (e.g., [138]), use component models which are instantiated with service implementations at runtime (e.g., [71, 195]), apply requirements engineering techniques for building service-oriented SASs (e.g., [303]), adopt the Active Components concept for engineering decentralized control (e.g., [296]), or focus on Quality of Service (QoS) aspects within service-oriented self-adaptation (e.g., [65, 178]). Often, models are used for reasoning, which services need to be adjusted/changed (e.g., [163, 195, 255]). The re-composition or exchange of services concerns the planning and executing activities and needs to be accompanied by monitoring and analyzing procedures.

4.1.7. Agent-based Approaches

A software agent is a piece of software that solves a specific problem autonomously and cooperatively [110]. A Multi-agent System (MAS) is a system of agents that share common goals and, therefore, communicate and cooperate. Several authors discussed aspects in the construction of self-organizing / self-adaptive MASs, such as dynamics and decentralized control in a MAS (e.g., [97,393,403]), emergence and self-organization for MASs (e.g., [98]), a methodology for engineering self-organizing MASs (e.g., [40,99]), as well as architectures (e.g., [250,358]) and design patterns for adaptable MASs (e.g., [87,100,139]). Other authors integrate a MAS and SOA for building SASs (e.g., [138]) or use CBD for designing autonomous agents as components (e.g., [26]). The variety of agent-based approaches offers (in theory) both temporal aspects of self-adaptation. Adaptation happens mainly on the application level, but other levels are feasible, too. Most approaches rely on a decentralized adaptation logic as a MAS should not be controlled by a central unit. Self-adaptive MASs focus on the planning functionality.

4.1.8. Nature-inspired Approaches

Systems in the nature are composed of independent but interacting components [272]. Each component adapts individually. The resulting system behavior is different from the behavior of the individual components (cf. emergence) [56, 98]. Nature-inspired systems can be categorized into four key metaphors: biological, physical, chemical, and social. In the following, we present examples for nature-inspired mechanisms in SASs. Biological mechanisms, such as flocking [249,333,412], foraging [50,106,249], nest building [249], molding [249], local inhibition [272, 334, 357], lateral inhibition [356], chemotaxis [68, 236, 272], embryogenesis [249], morphogen gradient [272], local monitoring [272], quorum sensing [249, 256, 272], consensus [68, 272], firefly synchronization [392], stigmergy [68, 236, 249, 365], web weaving [52, 249], brood sorting [249], and the human immune system [68, 90] or human autonomous nervous system [198, 249] have been applied in SOSs or SASs. Physical approaches mainly focus on the metaphor of potential fields pioneered by Kathib who employed them for obstacle avoidance in path planning in [201], e.g., used for SASs in [68, 250, 310, 338, 395]. Chemical approaches copy behavior observed in chemical reactions and find their application in SASs as a mean for programming service composition (e.g., [27]) or coordination modeling (e.g., [68, 376]). Social approaches concentrate on market mechanisms and norms (e.g., [319, 366, 373]). Most approaches focus on planning. [120, 244, 411] provide overviews on nature-inspired computing.

4.1.9. Formal and Verification-based Approaches

Dynamic behavior through self-CHOP properties aggravates to prove the correctness of SASs [160, 271]. However, a SAS requires to ensure behavioral and structural guarantees [96], especially in safety-critical domains, such as traffic light systems [298]. In contrast to regular software, SASs further require verification at runtime to match requirements and adaptation decisions [59]. For these tasks, different simulation-based verification frameworks (e.g., [135, 194, 205–207, 251, 396, 400]), (formal) design methods (e.g., [150, 159, 160, 213, 271, 297, 340]), language specifications (e.g. [92, 275, 371]), model checking techniques (e.g., [18, 85, 122, 125, 257, 415]), Domain Specific Languages (DSLs) for describing self-adaptation (e.g., [8, 17, 78, 348]), or stochastic approaches (e.g. [58, 130]) exist. These approaches mainly address the analyzing and planning part of the MAPE loop. A comprehensive overview on existing techniques for assurance of SASs and corresponding challenges is given in [328]. Further information on formal methods in SASs can be found in [397].

4.1.10. Machine Learning-based Approaches

Learning in a SAS is tightly coupled to self-optimization. A SAS continuously optimizes structure, parameters, or algorithms of the managed resources in order to become more efficient with regard to performance or cost [198]. Learning in a SAS focuses on structural optimization (e.g., [108, 112, 113, 126, 281, 286, 298, 358]). Several approaches use reinforcement learning for structural optimization (e.g., [63, 108, 204, 282, 349, 416]). Other works address parameter optimization (e.g., [25, 53, 354]). Algorithmic optimization aims at optimizing the algorithms of the managed resources (e.g., [196, 278]). The field of Search-based Software Engineering (SBSE) [171] applies search-based metaheuristic techniques to software engineering – e.g., genetic programming – for examining large search spaces of candidate solutions to find a (near) optimal solution to problems in designing software [173]. Contrary to the traditional approach, dynamic SBSE applies the principles of SBSE at runtime to determine the most suitable system configuration during planning of self-adaptation (e.g., [14, 169, 170, 172, 255, 304, 308, 385, 386, 417, 418]). Learning approaches mainly support the planning component. Adequate procedures for monitoring, analyzing, and executing have to be added. Further, learning can be used for meta-adaptation of the adaptation logic, e.g., through learning new rules for adaptation (e.g., [2,134,362]).

4.1.11. Requirements-oriented Approaches

For SASs, a specialized form of Requirements Engineering (RE) is needed as the dynamic nature of a SAS potentially results in uncertainty at runtime [323], due to a lack of information at design time [76]. Literature provides more information on causes for uncertainty at runtime (cf. [307,405]). Different authors propose to include runtime capabilities for RE, e.g., modeling requirements at runtime (e.g., [29,168,259]) or reasoning through requirements reflection at runtime (e.g., [39,265,287,336]). Additionally, different authors present requirement languages tailored to SASs (e.g., [29,76,89,155,302,404,409]). However, most of them do not explicitly support uncertainty. Runtime capabilities for managing requirements are important for reasoning on adaptation. RE approaches usually address the whole MAPE cycle. As requirements represent one type of goals the SAS should fulfill, RE approaches are mainly goal-based.

4.1.12. Further Approaches

There are further approaches, which cannot be categorized into one of the categories presented above. In task-based adaptation, the system determines suitable adaptation policies based on the users' task characteristics [295,344]. Some approaches use middleware-centric adaptation for achieving structural adaptation (e.g., [165,262,317]). Additionally, some authors propose development processes (e.g., [11,238,331,361,414]) and modeling dimensions (e.g., [12,55]) for the engineering of SASs. Further, different authors propose the use of design patterns for SASs (e.g., [23,87,100,120,132,133,139,270,305,341,401]).

4.2. Development Approaches for Self-adaptive Systems

The previous section presented a rather exhaustive overview on approaches for engineering (self-)adaptive systems that covers a broad range from rather abstract concepts – as programming paradigms or mathematical theories for verification of systems properties – to specific approaches for reasoning on adaptation or architectural models for the adaptation logic. However, it did not focus on the type of development support – e.g., tools, guidelines, or frameworks – offered by the approaches. This section presents the results of [224], where we analyzed and classified development approaches specific for SASs into frameworks, guidelines, tools, design concepts, and methodologies. This section analyzes the research gap in current state of the art approaches for developing SASs focusing on the research questions *RQ1* and *RQ2*. In the following, this section first presents our descriptions of the five categories. After, it discusses the results of the analysis presented in [224]. Parts of this section are taken from [224][4].

In general, a *framework* is an abstraction providing generic functionality that can be extended by user-written code. In the context of SASs, approaches based on frameworks often provide a combination of a reference architecture, tools, middleware, development process definitions, or component libraries. *Guidelines* support designers and developers by offering a detailed sequence of working steps that have to be performed. Typically, they do not offer tools, libraries, or reference architectures, but they can be accompanied accordingly. The provision of design and development *tools* can make the software engineering process more efficient as they automatize parts of development processes and lower error-proneness. Some approaches offer only one tool for a specific task. By contrast, others offer whole tool suites supporting several development activities. *Design concepts* are focused on design time activities and, therefore, do not directly target implementation activities. These include design principles and patterns, but do not offer libraries or specific implementation proposals. However, they can integrate tools, e.g., for modeling. In general, a software development *methodology* is a strategy or procedure to deal with a certain problem. It can be limited to a specific task or cover the whole development process. Besides, it can include process definitions, modeling languages, or analysis techniques. However, it is not as extensive as a framework and is not bound to a step-by-step guideline.

In [224], we worked out a taxonomy for comparing development support of approaches for developing SASs. The taxonomy has 18 dimensions related to: the users (here: developers and designers), the context of use, the type of sup-

[4] [224] is joint work with M. Pfannemüller, V. Voss, and C. Becker.

port, the support for integrating adaptation, and proof of concept. Appendix A.2 describes the dimensions in detail. Using the taxonomy, we analyzed 26 development approaches for SASs. Additionally, we categorized these approaches into the five categories mentioned above. Table 4.1 shows an aggregated overview on the approaches. Furthermore, Appendix A.3 provides the full data of the comparison from [224]. In the following, this section discusses the results of the comparison. As the various categories of development approaches are fundamentally different, approaches of the same category are compared with each other. This guarantees that the comparison of the development approaches provides meaningful findings. Due to the variety of tools but also the fact that many tools are not publicly available, we omitted them for this comparison.

The analysis identified architecture-based, model-based, service-oriented, agent-based, and component-based frameworks. Most offer support at runtime, some also at design time. All frameworks consider reusability on the adaptation infrastructure level. However, they differ regarding reusable processes and components as well as middleware. Furthermore, none of the approaches focuses on reusing existing code on an intra-component level. The support of adaptation mechanisms differs. Adaptation is achieved through models, adaptation services, coordination mechanisms, or middlewares. All frameworks offer compositional adaptation, confirming their component-based structure. Some frameworks claim to provide comprehensive tool support, e.g., *Rainbow* [140] or *MUSIC* [163]. The comparison of the approaches indicates that present frameworks do not only focus on runtime support but also include design time activities into their support. But support for self-improvement is often not integrated and, hence, they often do not cover the whole lifecycle.

Our overview [224] compared three guidelines. The *Software Engineering Guideline* [331] is agent-based and achieves adaptation through the construction and execution of the *Organic Design Pattern*. The *Development Approach and Automated Process* [5] is architecture-based and achieves adaptation through different modules. However, both provide support at design- and runtime, as [331] includes a design pattern and [5] provides a reference architecture. Last, the *SE processes for SAS* [11] describe a rather generic applicable process that does not specify the adaptation support. None of the guidelines offer libraries for supporting the development process.

Table 4.1.: Overview on the development approaches analyzed in [224]. For this thesis, we added the approaches EUREMA [377, 380] and DESCARTES [190, 215]. Arch. = Architecture

Name of approach	**Year**	**Type of approach**	**Type of support**
Rainbow [77, 140]	2004	Arch.-based	Framework
Model-Driven Approach [66]	2008	Model-based	Framework
Meta-Self [102]	2008	Service-oriented	Framework
SodekoVS [347]	2009	Agent-based	Framework
MUSIC [163]	2012	Model-based, Service-oriented	Framework
Architectural Framework for Self-Configuration & Self-Improvement [362]	2011	Arch.-based	Framework
FUSION [112]	2010	Model-based	Framework
SASSY [255]	2011	Service-oriented	Framework
Zanshin [336]	2012	Control-based	Framework
StarMX [20]	2009	Arch.-based	Framework
MOSES [65]	2012	Service-oriented	Framework
Software Mobility Framework [247]	2010	Arch.-based	Framework
GRAF [10]	2012	Model-based	Framework
DESCARTES [190, 215]	2016/17	Model-based	Framework
EUREMA [377, 380]	2018	Model-based	Framework
Software Engineering Guideline [331]	2010	Agent-based	Guideline
Development Approach and Automatic Process [5]	2015	Arch.-based	Guideline
SE Processes for SAS [11]	2013	not defined	Guideline
Modeling Dimension [13]	2009	Design concept	Concept
Design Space [55]	2013	Design-driven	Concept
High Quality Specification [242]	2013	Model-based	Methodology
Behavioral corridors [271]	2010	Verification-based	Methodology
General Methodology for Designing SOSs [151]	2007	Design concept	Methodology
FORMS [398]	2010	Model-based	Methodology
DYNAMICO [375]	2013	Control-based	Methodology

Both analyzed design concepts provide reusable design principles for designing SASs. The designer involvement is similar as they have to explore and apply design dimensions or answer design questions. Both approaches do neither use libraries, processes, nor reference architectures, but are extendable. In general, both design concepts are very similar and offer abstract support for developers.

Methodologies tend to be focused around a procedure dealing with a specific problem. For instance, Gershenson's *General Methodology* [151] gives a standardized view on the development process of SASs. Two approaches, *FORMS* [398] and *DYNAMICO* [375], provide reference models, but do not provide an implementation of them. Opposite, the other methodologies share a high level of abstraction as they focus on giving generic guidance on the development process. Hence, their specific implementation support is rather abstract and limited.

4.3. Approaches to Self-improvement

In [226], we present a comparison of approaches for self-improvement. The technical report of [227] extends [226] with recent approaches. Parts of both works[5] are integrated in this section. In the following, this section discusses the approaches for self-improvement to show the relevance of research question *RQ2*.

Approaches for self-improvement target different system domains, such as model-driven evolutionary systems, DSPLs, machine learning-based approaches, and requirements-aware systems. Such approaches reconfigure the adaptation logic using reasoning procedures based on goal models [194, 218, 274, 350, 353] or runtime models [121, 161, 377], formal/verification methods [194, 273], architecture-based approaches [274], machine learning algorithms [9, 109, 115, 116, 298, 363], requirements-awareness [287, 291], and rules/strategies [149, 342]. In [226] and [227], we compared 18 approaches for self-improvement using the taxonomy for self-adaptation from [229] (presented in Section 2.1.3). Table 4.2 shows the results of the comparison with an additional approach [377]. In the following, this section summarizes the state of the art for self-improving systems. For detailed information about the approaches, the reader is referred to [226] and [227].

[5] [226] and [227] are joint works with M. Pfannemüller, F. M. Roth, and C. Becker.

Table 4.2.: Approaches for self-improvement classified in [226, 227] with one additional approach [377] using the taxonomy of [229]. The level is always the application (here: the adaptation logic). System administrators are classified as users. Ctx = Context; Param = Parameter Adaptation; Struct = Structure Adaptation; React = Reactive Adaptation; Pro = Proactive Adaptation; MR = managed resources.

Approach	**Time**	**Reason**	**Technique**	**Adaptation Control**		
				Approach	Decision Criteria	(De)centralization
3LA [218]	React	Ctx/MR/User	not specified	External	Goal	Centralized
ActivFORMS [194]	React	Ctx/MR/User	Param	External	Goal	Centralized
NoMPRoL [350]	React	Ctx	Param	External	Model/Rules	Centralized
DCL [274]	not specified	User	Struct	External	not specified	Centralized
PLASMA [353]	React	MR/User	Param	External	Model/Goal	Centralized
FUSION [116]	React	Ctx/MR	Param	External	Goal/Utility	Centralized
KAMI [115]	React/ Pro	Ctx	Param	External	Model	Centralized
OTC [298]	Pro	Ctx	Param	External	Utility	Centralized
OTC DPSS [363]	React/ Pro	Ctx	Param/ Struct	External	Utility	Decentralized
DSPLs [9]	React/ Pro	Ctx/User	Param	External	Model/Utility	Centralized
Reqs@RT [287]	React	MR/User	Param	Internal	Rules	Centralized
RAMUN [109]	React	MR	Param	External	Model/Goal	Centralized
Adapt. Layer [342]	React	Ctx/MR	Param/ Struct	External	Rules	Centralized
Update Ctlr. [273]	React	Ctx/MR/User	Param/ Struct	External	Model	not specified
M-A Strategies [149]	React	Ctx/MR	depends on strategy	External	depends on strategy	Centralized
M-A Layer [291]	React	Ctx/MR/User	not specified	External	Model/Rules	not specified
Models@RT [121]	React	Ctx/MR	Param	Internal	Model/Rules	Centralized
Transformer [161]	React	Ctx	Structural	External	Model	Centralized
EUREMA [380]	React	Ctx/MR	Structural	Internal	Model	Centralized

Most of the approaches integrate reactive self-improvement, only three combine reactive and proactive behavior. A single approach works purely proactive. In many cases, a reactive self-improvement can be sufficient as usually the adaptation logic should find an appropriate adaptation for the managed resources. However, proactive self-improvement is necessary to avoid situations that the adaptation logic cannot handle. Such situation would lead to delays in the system and malfunction in the worst case. There is a high diversity regarding the adaptation reason indicating that the reason might be use case specific. The majority of the methods offers parametric self-improvement. Only three approaches provide both types of self-improvement. Additionally, none of the approaches with structural self-improvement has proactive behavior. This might be beneficial to better fit changes in the managed resources or the context. The fact that almost every method works with an external approach corresponds to the findings of [128]. There, the authors claim that an external approach offers better maintainability as well as extensibility. Only one approach works decentralized. All remaining approaches offer a centralized approach to self-improvement as a centralized setting facilitates a global view on the SAS. However, a central approach leads to a higher processing time as the amount of data for analysis is higher than for decentralized, local decision making. Weyns *et al.* describe different time scales for response times of hierarchical layers in a SAS [401]. There might occur situations in which the adaptation logic is unable to adapt its managed resource adequately. In this case, self-improvement should provide an adaptation as fast as possible. However, usually self-improvement executes runtime intensive optimization modules in order to find the best plan to improve the adaptation logic. Hence, the advantage of global decision-making for self-improvement outweighs the prolonged response time. The decision criteria are mixed for reactive self-improvement. This reveals that the used decision criteria might be use case specific.

In [226] and [227], we present further approaches that do not focus on but could handle specific aspects for self-improvement. For the comparison above, these approaches were excluded as they are not integrated into an automatic approach for self-improvement at runtime. In the following, we summarize them.

Other authors propose design methodologies or patterns for interaction of adaptation loops (e.g., [18, 92, 300]). In [283], the authors describe a technique for synthesizing changes of different versions of controllers (comparable to an adaptation logic). *DYNAMICO* adds an additional layer for monitoring and reasoning on adaptation objectives [375]. Hölzl *et al.* present a software development lifecycle that relies on the connection of three loops [186]. For self-improvement, the adaptation loop feeds the design process with feedback, hence, developers can update the system. However, these approaches mainly target design time or only offer concepts. Through machine learning, it is possible to improve the adaptation logic, e.g., through learning new rules or updating goals. For further information about these approaches, the interested reader is referred to the overviews presented in [116] or [229]. However, most of these approaches are highly use case dependent [116] or cannot cope with new context situations [229].

One has to mention, that it might be possible to achieve self-improvement by considering the adaptation logic itself as managed resource and adding an additional component for self-improvement. This way, common approaches for building SASs (e.g., Rainbow [140] or Archstudio [278]), or some of the approaches presented in Section 4.1 could be used for self-improvement. However, current research projects have not addressed this so far.

4.4. Demarcation of this Thesis from Related Work

This section discusses the former presented related work. It focuses on the main aspects of the research questions: the degree of reusability, self-improvement, and the integrated support for development of SASs.

As presented in Section 4.1, many approaches might be used for developing SASs. Architecture-based approaches have in common that specific components or layers control and execute adaptation. Often, approaches are based on models for controlling adaptation. For the composition of artifacts, service-based techniques are beneficial. Here, services can be composed in different chronological orders and service implementations are exchangeable for handling changing (environmental) requirements. Therefore, service architectures are commonly used for simplifying the compositional adaptation. Beside others, Organic Computing [268] or Autonomic Computing [198] are derived from principles of adaptation found in biology, physics, or chemistry that are transferred to IT. However, most systems based on the above mentioned approaches are lacking the ability of generalization as they are tailored to specific use cases. Design decisions related to the specific needs of the use cases complicate reusability of the resulting artifacts. The reusability is additionally limited as reusable and well-defined processes and components for design and implementation are often missing [96] or offer a non-optimal level of abstraction. Furthermore, the authors mostly focus on one of the categories (architecture-based, model-based, or service-based) instead of combining them in order to benefit from their advantages – e.g., the abstraction of service orientation and ease of model-driven design – while minimizing the respective drawbacks – e.g., the fixation on specific use case requirements for improving reusability (cf. research question *RQ1*). Further approaches integrate control theory or formal methods for runtime verification. These methods often provide high-level concepts and neither reference implementations nor prototypes.

The Dagstuhl seminar in 2009 identified a lack of frameworks tailored towards designing and engineering SASs [76]. This changed in the recent years as presented in [224]. Beside of frameworks that combine different artifacts for supporting the implementation of SASs, process definitions, reference architectures, modeling approaches, guidelines, methodologies, or tools that are customized to the specific requirements of developing SASs are presented in the recent years. However, most of the above mentioned shortcomings are true for most of the development approaches presented in Section 4.2. Especially, there is a lack regarding generic process elements for SAS design and development in combination with reusable implementations of artifacts such as component libraries or reference architectures [96]. Furthermore, an implementation library of reusable components like control structure elements would support the development of the adaptation logic for a SAS [96]. If available, these libraries require the use of specific programming concepts or programming languages. Integrating the process definitions and a library of components into a framework enhanced with tools would simplify the development of SASs and results in faster and less error-prone development (cf. research question *RQ1* and *RQ3*). From the analyzed development approaches, the *MUSIC* framework [163], *EUREMA* [377,380], and *DESCARTES* [190, 215] are the most complete approaches as they combine a model-driven approach, development process, tools, and a reference system. However, they are related to a specific modeling approach. All approaches except of

EUREMA [377,380] and *FUSION* [112] do not offer end-to-end support addressing all phases: design, deployment, and self-improvement at runtime.

Section 4.3 presents approaches for self-improvement. The spectrum ranges from examples for parametric self-improvement – such as the integration of a rule learner that is able to learn rules at runtime after a context change (e.g., [9,298]) – to approaches that focus on the structural self-improvement of the feedback loop at runtime (e.g., [274, 363]). Our analysis of approaches for self-improvement [226, 227] (cf. Section 4.3) shows different shortcomings of existing approaches. They mainly focus on either structural or parametric self-improvement of the adaptation logic as well as either on reactive or proactive reasoning. The approaches react on different triggers: changes in the context, managed resources, or the user, i.e., change of goals. However, monitoring of triggers and execution of self-improvement is often specific for an application. Therefore, it might be beneficial to separate generic elements and application-specific parts that require customization. This enables reuse of the approach for self-improvement. Additionally, for offering flexibility to developers, the approaches should integrate development support for reactive and proactive self-improvement and not focus on only structural or parametric self-improvement. As shown in [226] and [227] (cf. Section 4.3), none of the available approaches addresses all issues.

The FESAS Framework presented in this thesis addresses these shortcomings. It complements a process definition for developing SASs with development tools that abstract from common tasks, such as implementation of communication between the MAPE components in the adaptation logic. Further, it separates reusable components for building the adaptation logic and specific algorithms within the MAPE component (cf. research question *RQ1*). Additionally, it controls the deployment of the system. At runtime, the FESAS Framework offers an approach to self-improvement (cf. research question *RQ2*). Hence, the FESAS Framework offers end-to-end support from design time to deployment and runtime (cf. research question *RQ3*). The following chapter describes the concept of the FESAS Framework in more detail and derives requirements, the framework components have to fulfill. The identified shortcomings of existing related work influence these requirements.

5. Requirements Analysis

Based on the analysis of related work as knowledge base, this chapter derives requirements for the artifacts considering the research questions. First, Section 5.1 presents a high-level concept for the FESAS Framework. Parts of this section are based on [219] and [231][1]. Following, this chapter derives the requirements for the FESAS Framework based on the concept. These requirements target (i) the support offered by the framework at design time and for system deployment (cf. Section 5.2) as well as (ii) the self-improvement functionality at runtime (cf. Section 5.3). These requirements influence the system design presented in Chapter 6 as well as the implementation presented in Chapter 7.

5.1. Concept for a Framework for Developing Reusable Self-adaptive Systems

Deriving from research question *RQ1*, the FESAS Framework should offer generic support for engineering reusable SASs. Therefore, it integrates generic applicable development processes and reusable components. The term "generic" denotes the applicability in different system domains. This is complemented with support for self-improvement based on meta-adaptation [219,228,231] to meet research question *RQ2*. The combination of these functionalities in an end-to-end process that offers support (i) at design time but also (ii) for self-improvement at runtime targets research question *RQ3*. Figure 5.1 shows the conceptual design of the framework. The framework provides tools for designing and developing a SAS, a reference architecture for the adaptation logic, and a library with reusable components for building the adaptation logic. The design is based on model-driven engineering. A design model captures requirements, goals, constraints, and an initial architecture connecting the adaptation logic and managed resources. The

[1] [231] is joint work with S. VanSyckel and C. Becker.

FESAS Framework transforms the design model into a runtime system model. The system model offers an initialization of the system based on a reference architecture. The reference architecture is designed in a general way in order to apply to many different application domains and consists of a communication middleware and containers for the MAPE elements. These containers integrate two elements: (i) a communication module and (ii) the business logic. The communication module enables data exchange in the adaptation logic. The business logic is the functionality of a MAPE component, e.g., for observing the managed resources or planning adaptation. Whereas the containers and communication modules are generic, the business logic might be implemented specifically for a SAS or one of the available business logics of the library is used. At runtime, the FESAS Framework equips containers for the MAPE components of the adaptation logic with business logic functionality and distributes them on sub systems according to the system model. Further, the FESAS Framework connects adaptation logic and managed resources as specified in the system model.

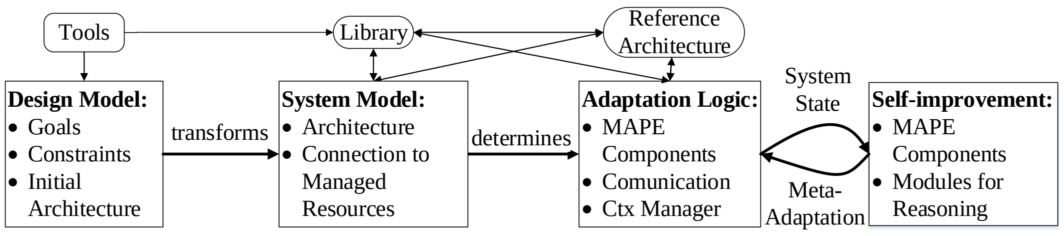


Figure 5.1.: The FESAS Framework supports designers and developers at the design time and during deployment of SASs with the focus on reusability and integrates self-improvement at runtime.

As pointed out by [161] and [278], within changing environments or for mobile systems it is not sufficient if only the adaptation logic adapts the managed resources. Also the adaptation logic behavior itself should evolve to reflect changes as the used adaptation reasoning approach has to reflect substantial environmental changes. Therefore, once the system is running, newly detected requirements and constraints for the system are used for refinement of the system model, which can trigger the reorganization of adaptation logic components, i.e., self-improvement. The FESAS Framework offers corresponding modules for that purpose as well as integrates development support for these tasks in its processes and tools.

Finally, development tools enables capturing information for the system model and supporting developers in implementing the business logics. As reusability of code is one of the main design considerations, this needs to be reflected in the tools for reusing code and system designs.

5.2. Requirements for the Development Framework

Based on the analysis of related work (cf. Section 4) and the research questions *RQ1* and *RQ3*, we identified the following requirements for the FESAS Framework to support developers in building reusable SASs:

$\boldsymbol{R_{Dev}}$**1: External control** The adaptation logic and the managed resources should be separated for higher maintainability and reusability.

$\boldsymbol{R_{Dev}}$**2: Reference architecture** The adaptation logic requires to be built on a reference architecture that supplements (i) a well-known standard control structure with (ii) modules that offer supportive functionality.

$\boldsymbol{R_{Dev}}$**3: Flexible adaptation control** A framework for developing SASs should (i) support different adaptation control patterns and (ii) offer deployment mechanisms for initializing the MAPE structure according to the patterns.

$\boldsymbol{R_{Dev}}$**4: Context management** Integrating a context manager that maps specific retrieved context data to generic context models enables reuse of context reasoning algorithms.

$\boldsymbol{R_{Dev}}$**5: Generic adaptation support** The framework should support (i-a) parametric and structural self-adaptation as well as context adaptation, (i-b) reactive and proactive adaptation, and (i-c) all adaptation decision criteria, i.e., rules/policies, goals, models, and utility functions. Therefore, (ii) encapsulation of the adaptation mechanism is required.

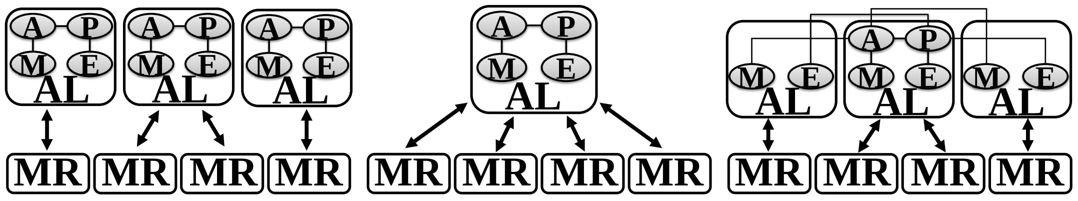
$\boldsymbol{R_{Dev}}$**6: Connection to managed resources** The adaptation logic requires to be connected to the managed resources. Developers should be supported with (i) communication modules and (ii) configuration procedures.

$\boldsymbol{R_{Dev}}$**7: Integrated Development** A development process with development tools should integrate a reference architecture and reusable implementation artifacts with self-improvement at runtime.

In the following, this section explains these requirements and their rationales. Related to research question *RQ1* the requirements for the FESAS Framework are centered around the reusability of code for the adaptation logic. For building the adaptation logic, there are two different approaches present in literature (cf. Section 2.2): The managed resources and the adaptation logic can be (i) integrated (*internal approach*) or (ii) separated (*external approach*). As we aim at a reusable adaptation logic, the adaptation logic should be separated from the managed resources for improving maintainability [128, 229], hence, enabling reusability. Otherwise, the adaptation logic would be intertwined with the managed resources, which results in optimized but highly customized and less reusable systems. Accordingly, the FESAS Framework requires to support an external approach for the adaptation logic (requirement $R_{Dev}1$).

As shown in the analysis of related work (cf. Section 4), all approaches have in common that the adaptation logic is composed of: (i) control structure components and (ii) supportive components. The supportive components implement generic functionality, e.g., for handling data or communication. In many approaches in literature, this functionality is intertwined with the control structure components or not present at all. If present, it is often highly specific for an approach. This limits its reusability. Therefore, for supporting the developers, on the one hand, it is important that the adaptation logic follows a well-known control structure (requirement $R_{Dev}2.i$). On the other hand, the functionality of supportive components requires to be abstracted and modularized for simplified reusability (requirement $R_{Dev}2.ii$). This includes clearly defined interfaces and processes – which might be codified in tools – for using these components. Both requirements lead to the definition of a reference architecture for the adaptation logic that supplements a well-known control structure with modules that integrate common supportive functionality (requirement $R_{Dev}2$). To support this, it is necessary that designers can use tools that are able to abstract from the low-level requirements of specific system implementation by offering a high-level abstraction and integrating the generic supportive functionality. Specific requirements might be implemented separately, but – once implemented – will be integrated into the SAS autonomously by the framework.

In [401], the authors propose patterns for decentralized control of SASs. Decentralized control means that at least one MAPE function is distributed on several MAPE components. For a fully decentralized control pattern, each MAPE functionality would be present in all sub-systems of the adaptation logic (cf. Figure 5.2a). In contrast, Figure 5.2b shows a fully central adaptation logic having one MAPE component for each MAPE function. A hybrid pattern combines centralized and decentralized MAPE elements. As an example, Figure 5.2c shows the master slave pattern from [401]. This hybrid pattern integrates centralized analyzing and planning, however, monitoring and execution are distributed.



(a) Full decentralized adaptation logic. (b) Centralized adaptation logic. (c) Hybrid master slave pattern [401].

Figure 5.2.: Comparison of adaptation logic structures (AL = adaptation logic, MR = managed resources, M,A,P,E = MAPE components).

The locus of control is independent from the distribution of the MAPE components to hardware. A decentralized adaptation control is not necessarily distributed. It might be possible, that the decentralization results from a split of responsibilities, however, still all MAPE components are running on the same system. As an example, for a smart home system that controls the heater it makes sense to have several monitors: one for monitoring the room temperature, one for monitoring the heater as well as one for monitoring whether the window is opened or closed. Contrary, this system would have one planner as the setting for the heater might be adjusted to the room temperature but also to the state of the window: if the window is opened in winter, it does not make sense to turn on the heater; rather, the planner informs the inhabitant about the open window. However, all of these MAPE components are part of the same central smart home control system. So comparable to network design, the adaptation control can be divided into a logical overlay – the locus of adaptation control which might be centralized, decentralized, or hybrid – and a physical underlay, which describes the mapping of the control pattern to MAPE components and their distribution on the hardware. A framework for developing SASs requires to offer (i) design support for different adaptation control patterns (requirement $R_{Dev}3.i$)

and (ii) deployment support for deploying the MAPE components at runtime according to the defined patterns (requirement $R_{Dev}3.ii$).

The survey of [229] showed that SASs often do not sufficiently include the context (cf. Section 2.2). Whereas almost every approach monitors the context, explicit alteration of context is often not addressed and the environment remains uncontrollable for the adaptation logic [11]. Even if context adaptation is not intended, context-awareness is essential because of the necessity to respond to environmental changes with self-adaptation. Additionally, an abstraction from specific context information improves reusability. Extending the adaptation logic with a context manager that (i) offers a knowledge base for context information and (ii) maps retrieved context data to generic context models enables reuse of context reasoning algorithms (requirement $R_{Dev}4$).

Related work [76,229,254] identified three basic mechanisms for self-adaptation: (i) parameters, (ii) the system's structure, or (iii) the context. Further, literature distinguishes [229] reactive adaptation and proactive adaptation. Last, different adaptation decision criteria – rules/policies, goals, models, and utility functions – can be found in literature [229, 234]. The framework should be flexible enough to support all of them (requirement $R_{Dev}5.i$). However, in combination with the reference architecture (cf. requirement $R_{Dev}2$), [161] describes a new challenge: the composition of the modules. Therefore, it is important to encapsulate the adaptation mechanisms (requirement $R_{Dev}5.ii$) and make them accessible through clearly defined interfaces. This supports the integration of existing approaches for adaptation reasoning and, hence, reusability (cf. research question *RQ1*).

An external adaptation logic requires a connection to the managed resources. Different communication technologies are available, such as socket-based 1:1 connection or using a communication middleware. As the connection relies on generic technologies, it can be offered to developers in communication modules (requirement $R_{Dev}6.i$). Additionally, support for the configuration of the connection between managed resources and adaptation logic is necessary (requirement $R_{Dev}6.ii$). The design of the reference architecture (cf. requirement $R_{Dev}2$) also requires to appropriately consider the connection to managed resources.

Related to research question *RQ3*, the approach requires to support the whole lifecycle of the SAS. Accordingly, the FESAS Framework should integrate implementation support in form of a reference architecture (cf. requirement $R_{Dev}2$) and reusable implementation artifacts (cf. research question *RQ1*) with runtime support for self-improvement (cf. research question *RQ2*) through a well-defined development process and development tools (requirement $R_{Dev}7$).

5.3. Requirements for Self-improvement at Runtime

As demanded by research question *RQ2* and *RQ3*, the FESAS Framework should be complemented by an approach for self-improvement, i.e., meta-adaptation of the adaptation logic at runtime. On the one hand, this approach should be integrated with the FESAS Framework. On the other hand, to not limiting its reusability, it should be self-contained so that it can be integrated with other frameworks for developing SASs. As the self-improvement logic is a dedicated reusable element, this thesis presents the approach for self-improvement separated from the FESAS Framework, however, the reader should always keep in mind that both are an integrated system to support the whole lifecycle of SASs. Based on the shortcomings we identified in [226] and [227] (cf. Section 4.3), we define the following requirements for an approach that enables self-improvement within SASs:

$R_{SI}1$: **Meta-adaptation mechanisms** As reasoning for meta-adaptation seems to be application-specific (cf. [226, 227]), the approach (i) should support modules for parametric and structural self-improvement and (ii) modules must be exchangeable.

$R_{SI}2$: **External control** Implementation may not be intertwined with the adaptation logic and, therefore, supports different types of adaptation logics.

$R_{SI}3$: **Monitoring** Generic monitoring mechanisms should support developers.

$R_{SI}4$: **Proactive reasoning** The approach should offer generic prediction procedures which can be easily integrated into specific reasoning modules.

$R_{SI}5$: **Flexible control** The approach requires to support centralized, decentralized, and hybrid control structures for self-improvement.

$R_{SI}6$: **Integration** To offer a generic and reusable approach for integrating the self-improvement control, it should be self-contained.

In the following, we describe these requirements in detail. Self-improvement through meta-adaptation of the adaptation logic can have various characteristics. In general, adaptation mechanisms are divided in (i) parameter adaptation, (ii) context adaptation, or (iii) structural adaptation (including the exchange of algorithms) [229, 254]. However, as we focus on adapting the adaptation logic, we neglect context adaptation here. The choice of the meta-adaptation technique depends on specific objectives, the meta-adaptation should address. As we showed in Section 4.3, the reason for self-improvement is application-specific. In accordance, Gui *et al.* identified that most adaptation frameworks do not provide clearly defined modules for adaptation reasoning [161]. Further, the implementation of meta-adaptation mechanisms may be application-specific, too. Requirement $R_{SI}1.i$ claims to support different self-improvement mechanisms. Therefore, the system has also to support exchangeability of self-improvement modules for specific applications (requirement $R_{SI}1.ii$).

The control for self-improvement can be (i) intertwined with the system – here: the adaptation logic – or (ii) implemented in external elements. [342, 401] describe trade-offs of an additional layer for self-improvement: increased execution time, complexity, risk of failure, and better control. In a layered approach for self-improvement, this means the reaction time for self-improvement is higher in contrast to the one for normal self-adaptation of the adaptation logic. The adaptation logic can still react to problems in the managed resources and adapt them, however, these adaptations might be non-optimal. Further, the changes performed by the self-improvement layer are more substantial. Consequently, the increased reaction time for self-improvement resulting from the layered approach can be neglected. The complexity might be increased with an additional layer as further interactions are necessary. This is definitely a shortcoming which also introduces a higher risk of failure. The alternative would be an internal approach: The integration of the self-improvement functionality into the (distributed) adaptation logic of a SAS. In [128], the authors showed that an external approach offers a better maintainability. Further, [342] states that an external approach improves implementation control. Last, the additional layer offers general applicability supporting requirement $R_{SI}1$ for exchangeability of elements that demands this flexibility. Therefore, requirement $R_{SI}2$ claims an external approach for the control instance for self-improvement.

The self-improvement approach has to provide mechanisms to monitor meta-adaptation triggers (requirement $R_{SI}3$), such as failures in the resources or context changes [229]. As we discussed in [226], those triggers for self-improvement are the context of the system, the managed resources, or the user(s), i.e., changing goals. For example, concept drift – i.e., a substantial change over time of the statistical properties of the relevant variables – can make the model in the adaptation logic obsolete and, in this case, analysis for triggers delivers faulty results. Additionally, triggers can change over time. Hence, proactive analysis of data – which can recognize concept drift – must be supported (requirement $R_{SI}4$). Further, proactive reasoning enables proactive meta-adaptation and can reduce delays as meta-adaptation can happen faster than reactive meta-adaptation [229]. However, the support of reactive meta-adaptation as back-up mechanism if proactive meta-adaptation fails – e.g., a situation was wrongly predicted – is still necessary.

Control within an adaptation logic can be decentralized, hybrid, or centralized [401]. A decentralized approach benefits from higher robustness but introduces coordination overhead. A centralized approach reduces coordination overhead at the cost of having a potential performance bottleneck and single point of failure. Our analysis in [226] showed that most approaches for self-improvement are centralized. Centralization simplifies to have a global view on the system, which is necessary for self-improvement. However, interaction patterns such as the ones presented by Weyns *et al.* [401] can be applied to the adaptation control for self-improvement. Decentralized settings may perform faster meta-adaptation decisions in large systems. Consequently, approaches for self-improvement should support centralized and decentralized coordination patterns (requirement $R_{SI}5$).

The approaches analyzed in Chapter 4 offer either development support for SASs or for self-improvement. Except *FUSION* [112] and *EUREMA* [377, 380], none of them offer an integrated approach. We claim that this is necessary for accessing the full potential. Accordingly, the ALM should be integrated with the development approach for the adaptation logic to offer a continuous workflow for developers. Further, this increases reusability as approaches or algorithms for self-adaptation might be reused for self-improvement. Supporting research question *RQ3*, requirement $R_{SI}6$ demands an end-to-end development support for integrating the development of the adaptation logic with the logic for self-improvement.

6. A Framework for Engineering Reusable and Self-improving Self-adaptive Systems

Based on the concept and the requirements defined in Chapter 5, this chapter presents the design of the FESAS Framework. It covers the whole lifecycle of a SAS. First, during development, the framework supports the developers and designers of SASs with a process that defines the workflow of the development. This workflow is accompanied by a template that supports the reusability of the adaptation logic components as well as artifacts that control the deployment of the adaptation logic. Additionally, tools can incorporate the concepts of the design and simplify the development process. Section 6.1 presents the workflow, the architecture template as well as the artifacts that control the deployment. Second, the framework should support self-improvement at runtime. Section 6.2 introduces the concept of the *Adaptation Logic Manager* (ALM) that adds an additional layer on top of the SAS for self-improvement through meta-adaptation of the adaptation logic. As the ALM is self-contained and not restricted to the FESAS Framework, it is presented separately.

6.1. System Model for the FESAS Framework

The concept presented in Section 5.1 is the underlying concept for the FESAS Framework. According to research question *RQ1*, the main design rationale is the striving for reusability of modules for the adaptation logic of SASs. Hence, the FESAS Framework requires to offer a reference architecture for the adaptation logic. This reference architecture is based on the MAPE model (cf. Section 2.3). The focus lies on the division of specific, customized code and reusable modules, e.g., for communication, knowledge distribution, or standard adaptation modules, such as rule-based planning approaches. This enables the integration and combination of existing approaches and frameworks, such as the ones presented in Section 4.1 and Section 4.2. Further, the FESAS Framework is not restricted to one specific adaptation decision criterion and can integrate modules for analyzing and planning based on rules/policies, goals, models, or utility [229, 234] (cf. requirement $R_{Dev}5$). Section 6.1.1 presents the *FESAS Adaptation Logic Template* which is designed with reusability in mind and acts as underlying reference architecture for the adaptation logic (cf. requirement $R_{Dev}2$).

One important aspect for reusability beside of the division of customized and reusable code modules is the definition of a process for (i) developing new elements, (ii) integrating existing elements, and (iii) storing these elements as well as using them at runtime for system deployment. This process should be complemented by tools and artifacts for improving the process usability (cf. research question *RQ3* and requirement $R_{Dev}7$). Both, tools and artifacts, should be independent from specific programming languages. Hence, it is important to define a reusable workflow independent from specific implementation technologies. Section 6.1.2 defines such a process workflow.

Last, for the deployment of the adaptation logic – i.e., the distribution and integration of the reusable modules as well as plugging in the customized code – specific artifacts are required. The FESAS Framework integrates the use of a repository for code and adaptation logic modules. Again, the repository should be independent from specific technologies, i.e., a reusable design is necessary. Section 6.1.3 presents the design of the FESAS Repository as well as its interaction with the SAS. Finally, the FESAS Middleware brings everything together: it controls the deployment of the adaptation logic, i.e., according to the process workflow, it sets up the adaptation logic modules and adds code from the FESAS Repository to the components of the reference architecture as defined in the design model of the SAS. Section 6.1.4 describes the activities the FESAS Framework handles during deployment of the adaptation logic when starting a SAS as well as its artifacts.

6.1.1. FESAS Adaptation Logic Template

For building the adaptation logic, there are different approaches present in literature. The managed resources and the adaptation logic can be integrated following the *internal approach* or separated according to the *external approach*.

As we aim at a reusable adaptation logic, the adaptation logic should be separated from the managed resources as this improves reusability (cf. requirement $R_{Dev}1$). All approaches have in common that the adaptation logic is composed of [229, 403]: (i) control structure components including components for the connection to the managed resources and, optionally, (ii) supporting components (cf. requirements $R_{Dev}2.i$ and $R_{Dev}2.ii$), such as the context manager specified by requirement $R_{Dev}4$. Figure 6.1 shows the elements of the *FESAS Adaptation Logic Template*. Integrating this reference architecture presented in [228] into the FESAS Framework satisfies requirement $R_{Dev}2$. In the following, this section introduces the FESAS Adaptation Logic Template. Parts of this section are taken from [228][1].

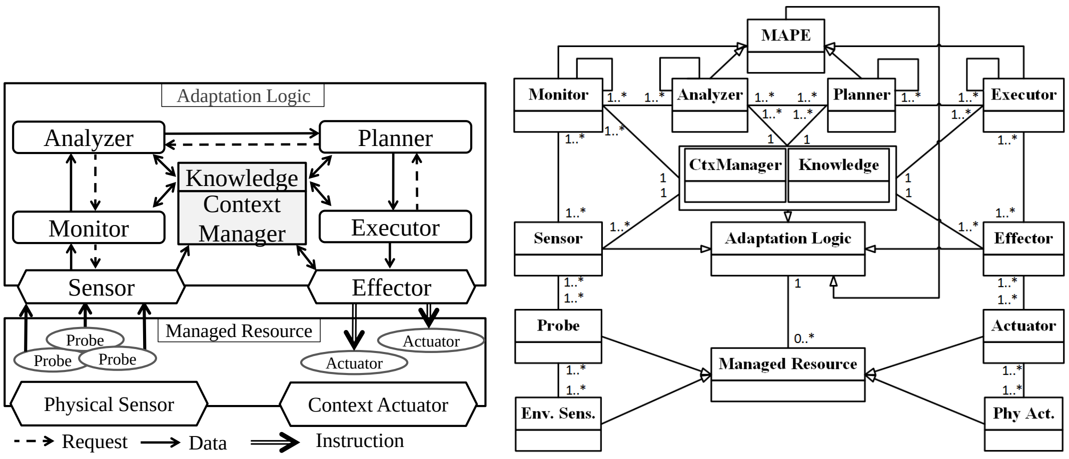


Figure 6.1.: The FESAS Adaptation Logic Template extends the MAPE cycle with a context manager for abstraction from specific information (cf. [228]). Left: concept, right: component model.

Within the SASs research community, the MAPE-K model [198] is the common approach for an adaptation logic control structure [76,229,320,384]. Accordingly, the FESAS Adaptation Logic Template integrates this model for adaptation control. The *knowledge* component acts as a central repository for all kinds of data used in the adaptation logic and supports the MAPE functionality. Often, a SAS is a system-of-systems and the adaptation logic is distributed in these cases. Therefore, the adaptation logic template offers flexibility for decentralization and distribution of its components (cf. requirement $R_{Dev}3.i$). This requirement is addressed in the FESAS Adaptation Logic Template as the components for the

[1] [228] is joint work with F. M. Roth, S. VanSyckel, and C. Becker.

MAPE functionality are optional for a specific subsystem, but must be present in the global view of the SAS. Furthermore, one subsystem can have multiple MAPE loops, e.g., for supporting different self-CHOP properties [198]. Following the MAPE-K model [198], the template provides sensors and effectors for interaction with the managed resources. *Sensors* read information from so called probes. These probes store information about (i) the resources themselves (self-awareness), (ii) the environment (context-awareness) – collected by the managed resources using *physical sensors* (e.g., GPS, infra-red sensor, or camera) –, and (iii) information about the users. An *effector* uses specified interfaces, so called *actuators*, to enact the adaptation actions. These actuators trigger changes of the managed resources or adapt the environment using *context actuators* of the managed resources.

Additionally to the MAPE functionality, the FESAS Framework integrates the *context manager* [228] to improve the context-awareness and support context adaptation (cf. requirement $R_{Dev}4$). The context manager (i) receives context information which is gathered through physical sensors of the managed resources, (ii) aggregates the data of sensors, and (iii) maintains a context model. Furthermore, the context manager integrates an architectural model of the managed resources. This way, the MAPE functionality can be decoupled from the specific data retrieved from the managed resources. Therefore, the MAPE components can operate on a higher level of abstraction. This improves the reusability of the MAPE components and algorithms within these components as there is no need to customize them for each use case.

The FESAS Adaptation Logic Template is a rather high level concept. Driven by the idea of further improving reusability, in [228], we presented the *FESAS Component Template* for a MAPE component. In the following, the section presents this template. Based on the *Template Method* pattern [137], the MAPE components are composed of an exchangeable logic – e.g., for an analyzer, this would be an algorithm for analyzing the monitored data – and logics for communication and data handling, i.e., serialization and access to a knowledge repository. The communication logic offers methods for receiving and sending data to other components as well as requesting data from other components. Figure 6.2 shows the template for a MAPE component.

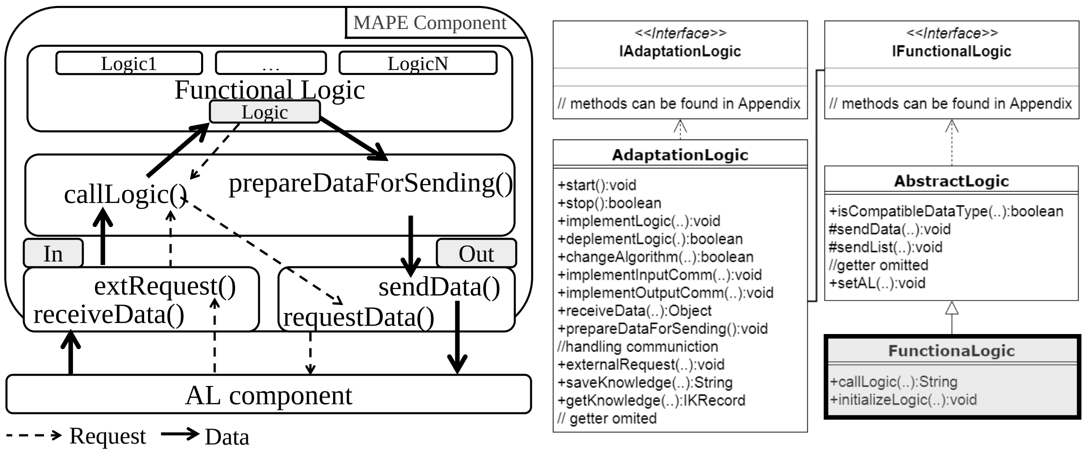


Figure 6.2.: The FESAS Component Template (cf. [228]) separates generic functionality, such as communication between MAPE components, and specific code for the algorithms of the MAPE functionality. This improves the reusability of code. Most of the methods are implemented by components of the FESAS Framework. Developers can focus on implementing the functional logic (bold marked box). Appendices B.1 and B.2 provide the descriptions of the interfaces.

The separation of the functionalities in reusable communication sub-components and customizable functional logics supports reusability. Further, encapsulating a specific algorithm within a functional logic element enables a separation from reusable methods for using the communication module or controlling the workflow. In object-oriented programming languages, inheritance supports this structure. Functional logic elements inherit most methods from an abstract logic. Only the `callLogic()` method for its function has to be implemented or the developer has to specify which logic should be loaded from a repository. Additionally, one MAPE component can integrate various functional logics, e.g., a monitoring component with different algorithms for different sensor types. The component template enables reusability w.r.t. different aspects. First, the component itself is reusable. It offers a skeleton of methods for calling the functional logic, communication, and knowledge handling. Furthermore, exchanging the functionality is simplified as the interfaces stay stable. Additionally, it is possible to use the same component skeleton for all adaptation logic components. Only the functional logic must be customized to its purpose, which satisfies requirement $R_{Dev}5.ii$. Second, reusability is offered as the interfaces provide generic mechanisms for communication and knowledge handling.

6.1.2. FESAS Development Process

The previous section introduced the FESAS Framework's system model. This section is based on [225][2] and presents how developers can use the FESAS Framework for building SASs. Having a defined process for SASs development combined with a reference architecture addresses the lack of (i) reusable processes and specific implementation guidelines as well as (ii) implementation components identified in [96] and [11]. This addresses research question *RQ3* and requirement $R_{Dev}7$. The process integrates two roles: the *system developer* who writes code and the *system designer* who defines the configuration of a SAS. In the following, this section describes the workflow for developing a SAS and shows how the FESAS Framework supports developers of SASs (cf. Figure 6.3).

At design time, system developers implement the functional logics for the MAPE components of SASs using the *FESAS Development Tool*. Further, the developers have to define metadata for these elements. Appendix B.3 shows this metadata. This metadata is used during deployment by the FESAS Middleware for identifying functional logics that fit the system design. The code including the metadata is stored in the FESAS Repository. System designers specify the system's design for a specific application using the model-based *FESAS Design Tool*. The FESAS Design Tool translates the created design models to system models (cf. Appendix B.4 for the syntax of these models). Section 7.1 describes the implementation of the prototypes for these tools. Both tools abstract from learning the syntax for the system models and the metadata of functional logics for reducing error-proneness as well as simplify the use of the FESAS Adaptation Logic Template and the FESAS Component Template.

At runtime, the FESAS Middleware configures the adaptation logic using the information in the system design model as well as the descriptions of the implemented components that the FESAS Repository stores. Based on the selected algorithms in the functional logics, parameter and/or structural adaptation (including algorithmic adaptation) can be initiated by the adaptation logic (cf. requirement $R_{Dev}5.i$). It is possible for system designers to use the FESAS Repository as a kind of app store just by defining the metadata of the functional logics. This is enabled by the abstraction of the context manager from specific managed

[2] [225] is joint work with F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schürr.

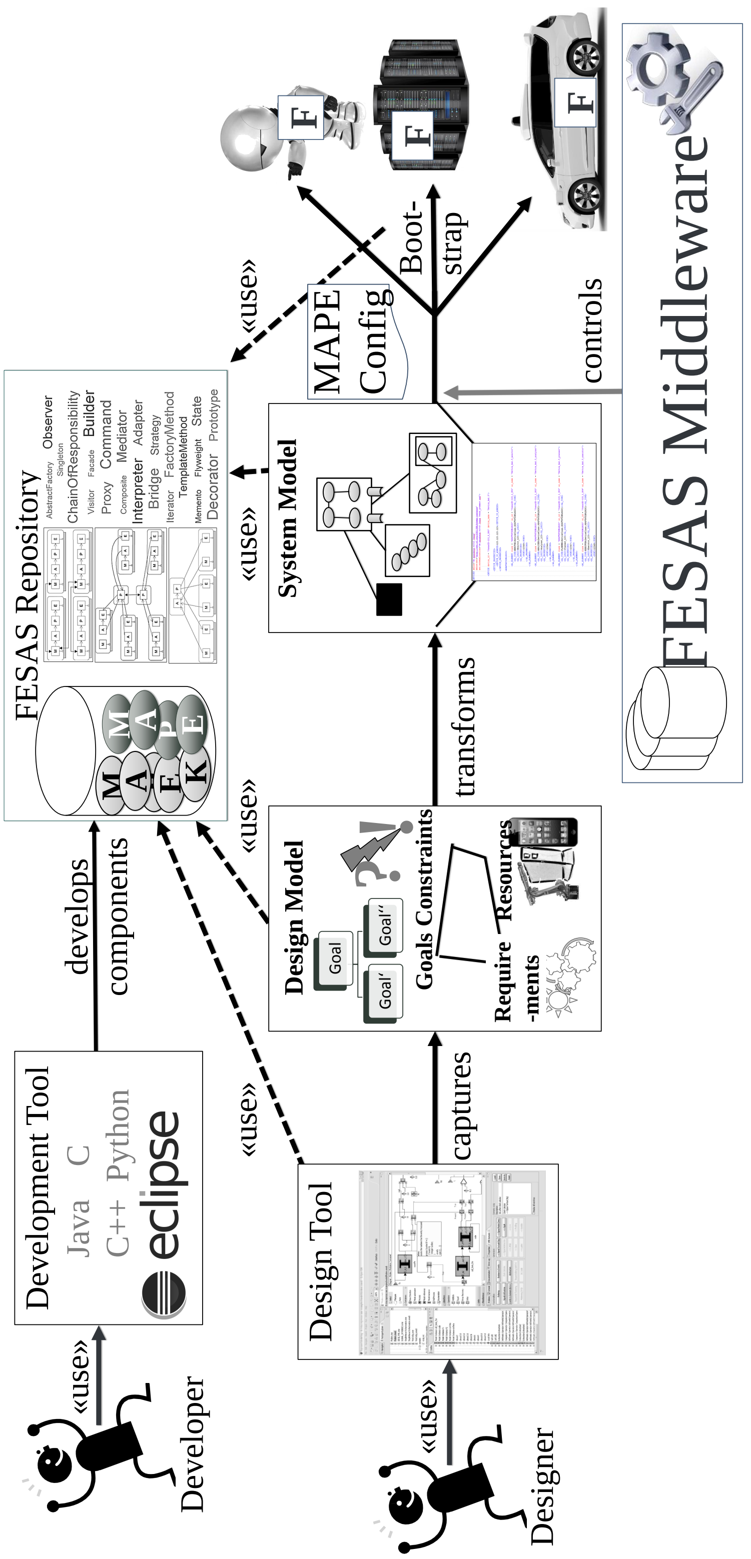
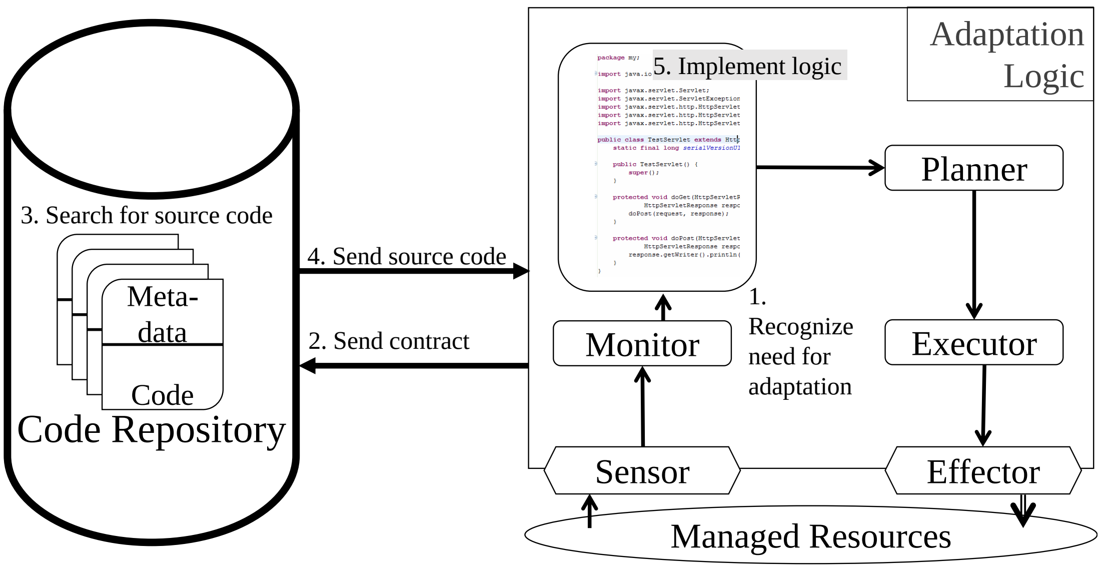


Figure 6.3.: Workflow for developing SASs with the FESAS Framework (cf. [225]). An 'F' indicates that a device runs the FESAS Middleware.

resources. On the other hand, if customized functionality is required, a system developer can write code according to definitions of the system designer. The FESAS Framework offers reference implementations of the FESAS Adaptation Logic Template for a specific system infrastructure, i.e., a specific programming language and/or communication middleware (cf. Section 7.3). When starting a SAS, the FESAS Framework uses the FESAS Repository to initialize the adaptation logic with the functional and communication logics as specified in the system model. Once the initialization is finished, the adaptation logic controls the SAS.

6.1.3. FESAS Repository

The formerly presented FESAS Adaptation Logic Template as well as the FESAS Component Template offers reusability of code and exchangeability of functional logics that implement customized MAPE functionality. For supporting developers in the exchange of functional logics (cf. requirement $R_{Dev}5.ii$), the FESAS Framework integrates a mechanism for dynamic code reloading from the FESAS Repository. First, This section presents the concept. Next, it introduces the process for this mechanism as well as the design of the FESAS Repository.


```
package my;

import java.io

import javax.servlet.Servlet;
import javax.servlet.ServletException
import javax.servlet.http.HttpServlet
import javax.servlet.http.HttpServlet
import javax.servlet.http.HttpServlet

public class TestServlet extends Http
    static final long serialVersionUI

    public TestServlet() {
        super();
    }

    protected void doGet(HttpServletR
            HttpServletResponse respo
        doPost(request, response);
    }

    protected void doPost(HttpServlet
            HttpServletResponse respo
        response.getWriter().println(
    }
}
```

Figure 6.4.: Concept for dynamic code reloading of the functional logic for MAPE components out of a code repository.

Figure 6.4 shows the mechanism for loading a new functional logic element. When the system discovers the need for change inside the adaptation logic (cf. step (1) in Figure 6.4), it initiates a reloading process by sending a request containing a contract to the code repository (cf. step (2)). Loading a logic can be necessary for several reasons, such as starting the initialization of the adaptation logic or meta-adaptation for self-improvement. The repository stores the functional logic that consists of source code and metadata information. It processes the request by mapping the metadata from the contract to the stored information, for finding the corresponding source code (cf. step (3)). Once found, the source code of the appropriate functional logic element is sent back to the SAS (cf. steps (4)). There, the code of the functional logic is integrated into the corresponding MAPE component (cf. step (5)).

The mechanism is divided into two tiers. The *Local Repository* serves as an intermediate repository within the adaptation logic that loads new code for functional logics from the *Remote Repository*. As depicted in Figure 6.5, every MAPE component is connected to the Local Repository.

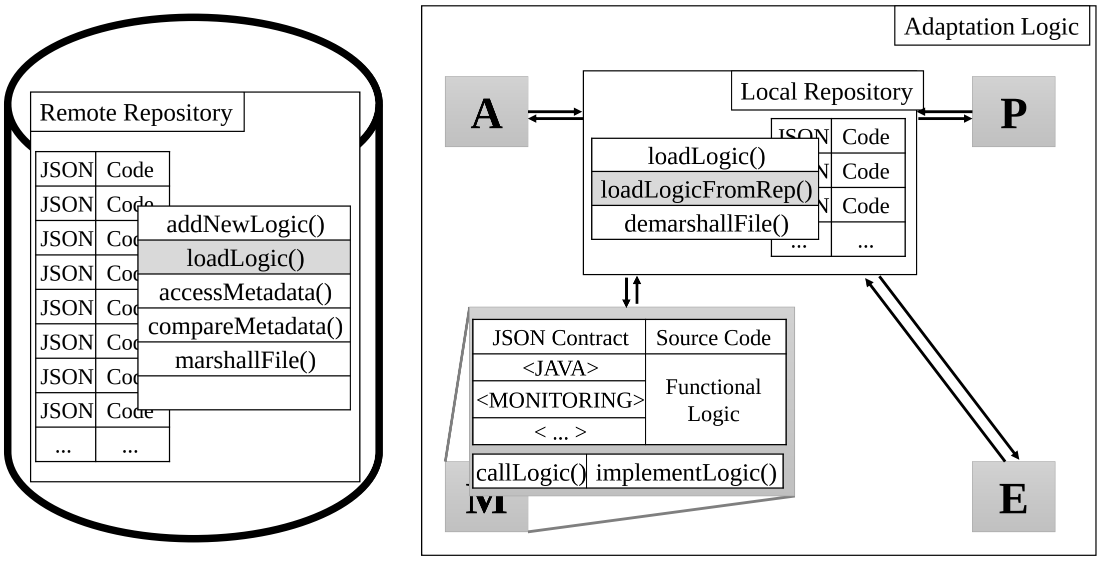


Figure 6.5.: System model for the dynamic code reloading mechanism. The figure omits the SAS Setup Service which enables loading of a functional logic from the Local Repository.

When one MAPE component triggers loading code (cf. Figure 6.6), it specifies a contract. This JSON-based contract formally describes the properties of the requested functional logic. It provides information about general attributes of the required functional logic element as well as specific functional properties, a description, and possible dependencies to other files. Appendix B.5 shows this metadata. Due to the dynamism of the functional logics, the Local Repository always first sends a request for analysis of the contract to the Remote Repository. If an appropriate element is available, the Remote Repository sends a metadata description back. The Local Repository compares the metadata with the logic elements that are stored locally. If the same logic element has been requested recently, it is still available locally and can be used. If the logic is requested for the first time or an updated version is available, the Local Repository forwards the request to the Remote Repository. The Remote Repository marshals and ships the corresponding archive file to the Local Repository. The Local Repository demarshals the file, stores it, and returns the code to the MAPE component.

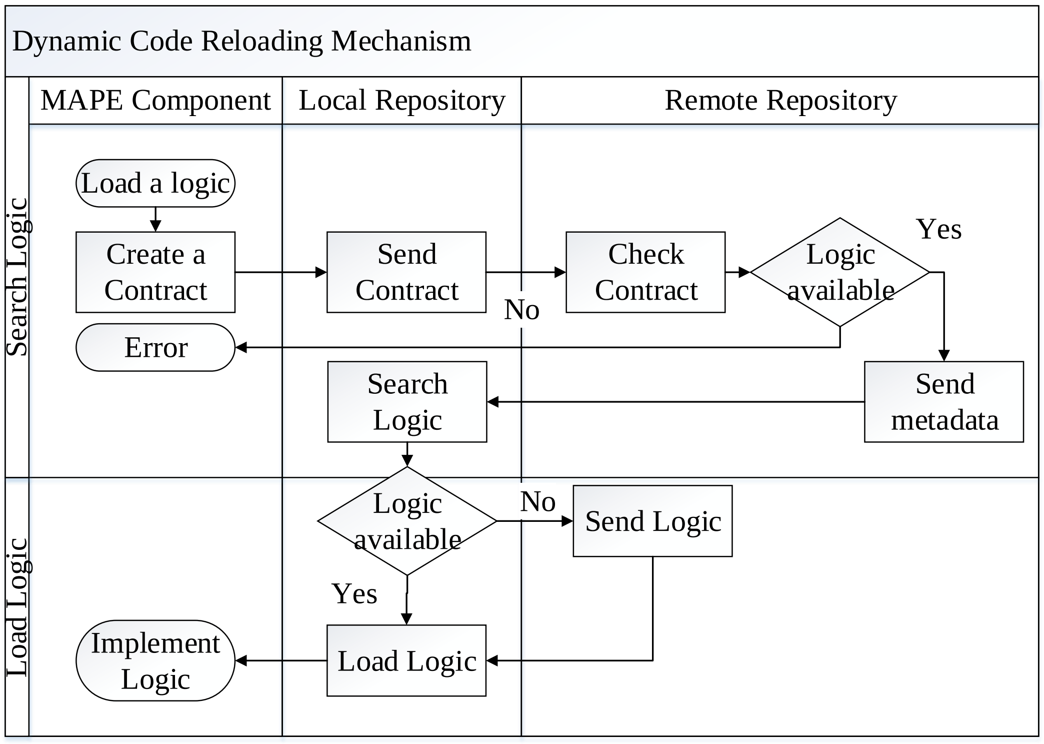


Figure 6.6.: The activity diagram shows the process for dynamic code reloading using the FESAS Repository.

Appendix B.6 shows the interface of the Remote Repository. It offers methods for loading a functional logic, storing/updating a new functional logic as well as removing a functional logic. These methods have to be implemented in case an individual repository should be used. Further, the SAS needs to implement the interface for the Local Repository and offers a mechanism for integrating the loaded code into the MAPE components. As part of this thesis, both is done for the prototype implementation of the FESAS Framework in Java (cf. Section 7.3).

6.1.4. FESAS Middleware

The deployment of the system follows the process described in Section 6.1.2. As this process involves different types of activities, the FESAS Middleware integrates various components for enabling a separation of concerns. At start, the FESAS Middleware components on each subsystem read files that specify the information of the system model, start the corresponding MAPE components, adjust settings of the components, and load the specified functional logic elements. Furthermore, these configuration files determine the communication structure between MAPE components. Appendix B.4 shows the syntax and examples of these model files. In the following, this section presents the components of the FESAS Middleware and how it supports deployment of a SAS.

Figure 6.7 shows the components of the FESAS Middleware. The starting method of the SAS (e.g., the `main()` method in Java) triggers the start of the FESAS Middleware – i.e., the component `Middleware Starter` is started – and hands over the configuration files as parameters for the starting process. Appendix B.7 presents these configuration files in detail. The middleware starter uses the configuration files with the start parameters to configure the SAS and controls the deployment process. Next, it configures the JSON Parser, loads the specified JSON files containing the system model for the SAS, parses them, and loads the adaptation logic components with the specified functional logic elements using the FESAS Repository (cf. Section 6.1.3). The Local Repository is the connection of the FESAS Middleware to the FESAS Repository. It acts as a local storage of the code of functional logic elements.

The SAS needs to implement the FESAS Adaptation Logic Template as well as to offer the SAS Setup Service specifically implemented for the used platform,

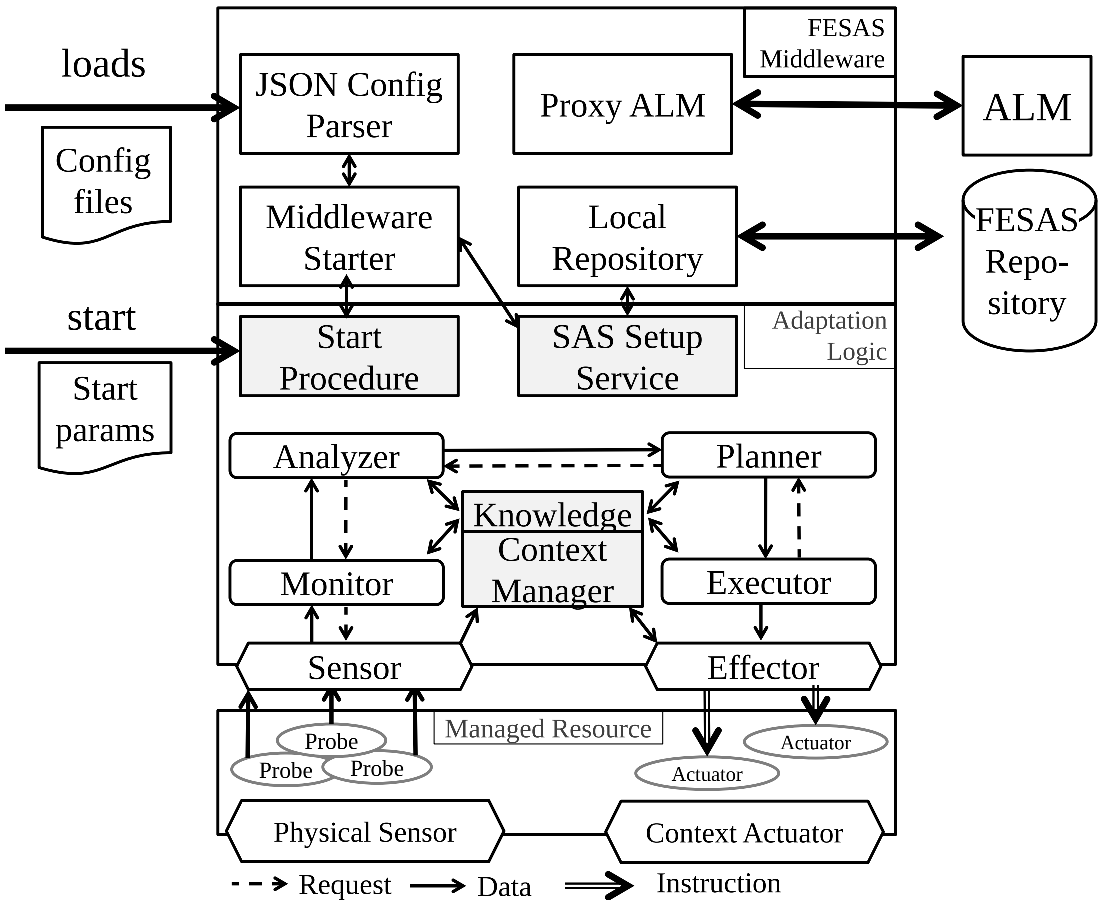


Figure 6.7.: The FESAS Middleware components – JSON Config Parser, Proxy ALM, Middleware Starter, and Local Repository – interact with the FESAS Repository and the ALM (in case of self-improvement). Developers additionally have to implement the SAS Setup Service for the target platform of the SAS.

i.e., the used programming language or communication middleware. Therefore an interface defines methods for (i) creating adaptation logic components, (ii) initializing communication channels between adaptation logic components, (iii) initializing communication with the managed resources, as well as (iv) starting the system after the completion of the initialization (cf. Appendix B.8). The FESAS Middleware uses this service to integrate the code of the functional logics – which is loaded into the Local Repository – in the skeletons of the MAPE components, i.e., for initializing the adaptation logic with its specific MAPE functionality.

Afterwards, the FESAS Middleware controls the initialization of the communication between the MAPE components and between adaptation logic and managed resources. Therefore, the specified communication modules are loaded into the MAPE components and are configured according to the system model using the methods of the SAS Setup Service (addressing requirement $R_{Dev}3ii$). Additionally, the FESAS Middleware establishes the connection of adaptation logic and managed resources. Therefore, it loads the functional logics for sensors and effectors of the adaptation logic and configures them (addressing requirement $R_{Dev}6.ii$). This configuration might range from specifying IP addresses and ports for 1:1 push-based socket connections, opening ports for pull-based socket connections, or setting up more advanced approaches, e.g., a web server, bus system, or publish/subscribe middleware. It is possible to integrate approaches from related work, e.g., such as the Probe/Gauge Reporting Bus from the Rainbow Framework [77, 141], the Touchpoint concept from the IBM Autonomic Computing Reference Model [193], or the ProbeMeister system [74]. For offering the highest possible configurability and flexibility, again, the sensors and effectors support the definition of own functional logic elements and the FESAS Frameworks supports the integration of own communication modules (addressing requirement $R_{Dev}6.i$). Developers just have to ensure that the managed resources are compatible with the chosen communication approach, e.g., if a sensor shall use a pull-based socket communication, the managed resources must open a socket according to the information of the system model.

Additionally, the start of the middleware invokes the start of the Proxy ALM if the self-improvement functionality is enabled in the configuration files. The Proxy ALM connects to the Adaptation Logic Manager (ALM) which controls the self-improvement of the SAS at runtime. Through this connection, the ALM receives information about the SAS (i.e., its adaptation logic and managed resources) and can trigger adaptation of the adaptation logic. Therefore, the Proxy ALM is informed by the MAPE elements in case of structural or parameter changes, such as the integration or substitution of functional logic elements. Additionally, the Proxy ALM periodically checks the adaptation logic for changes. On request of the ALM, the Proxy ALM sends the current system state and received data from the managed resources. The next section presents the design of the ALM and the Proxy ALM.

6.2. System Model for the Adaptation Logic Manager

In [281], the authors discuss a trade-off in planning self-adaptation. On the one hand, the planning process has to be fast, hence, sub-optimal planning techniques – e.g., rules or simplified model-based approaches – are prominent as they offer a fast response by keeping the complexity low. However, these do not provide an optimized solution. On the other hand, the fast sub-optimal planning in combination with uncertainty result to non-optimal or even wrong adaptation decisions. As a consequence, authors propose to use optimal planning approaches, such as dynamic SBSE [169]. However, for these approaches the planning time as well as the required computational resources increase extensively. Meta-adaptation might offer the solution: simple but fast planning approaches can be used in the adaptation logic whereas self-improvement can improve the planning process over time without delaying the planning. It is an external layer on top of a SAS which addresses the need for self-improvement [182] as identified in, e.g., [161, 278] (cf. Section 2.1.4).

This section presents the design of the ALM and the functionality of its components. [315][3] presents a first concept of an approach that offers self-improvement. This first draft was further conceptualized in [223]. The works published in [223][4], and [230][5] both used the design presented in this section to complement SASs with the functionality of the ALM.

6.2.1. Design of the Adaptation Logic Manager

The ALM adds an adaptation logic on top of an existing SAS. Hence, it sees the adaptation logic of the SAS itself as managed resources and might switch it through meta-adaptation, i.e., adaptation of the MAPE components through:

Structural meta-adaptation: Switch of the interaction pattern between MAPE components or exchange of MAPE components; in the FESAS Framework, the latter includes a change of the functional logic.

[3] [315] is joint work with F. M. Roth and C. Becker.

[4] [223] is joint work with J. Otto, F. M. Roth, A. Frömmgen, and C. Becker.

[5] [230] is joint work with T. Sztyler, J. Edinger, M. Breitbach, C. Becker, H. Stuckenschmidt.

Algorithmic meta-adaptation: This specific form of structural adaptation triggers the exchange of the internal structure of a MAPE component; in the FESAS Framework, this changes an encapsulated algorithm within a functional logic.

Parametric meta-adaptation: Adjustment of one or several parameters of the functional logic of a MAPE component including adding new rules.

In the following, this section explains the design of the ALM. Further, it describes the connection of the ALM to a SAS.

As an external approach – which splits the managed resources and the adaptation logic – benefits from higher maintainability and reduced dependability, we applied an external approach to the ALM and added a layer on top of the SAS (requirement $R_{SI}2$). Hence, the resulting system is composed of three layers. First, the *Managed Subsystem Layer* contains the managed resources that offer their functionality to users or backend systems. Second, the *Self-Adaptation Layer* contains the adaptation logic. Both layers build up the SAS. Third, on top, the *Self-Improvement Layer* contains the ALM. Comparable to the adaptation logic itself, the ALM uses the MAPE model as feedback loop structure as this is well established in the domain of SASs [56]. This MAPE loop controls the adaptation logic as managed resource. Figure 6.8 visualizes the system model of the ALM. Due to simplicity reasons, the figure shows a centralized approach. However, the concept of the ALM is not restricted to a central approach but supports decentralized or hybrid interaction patterns for the ALM (requirement $R_{SI}5$).

The ALM interacts with the *Proxy ALM* which has to be integrated into the adaptation logic. This component captures the structure of the adaptation logic and collects the configuration of the MAPE components, such as the used rules. Further, it collects the data of the managed resources that is captured by the sensors of the adaptation logic. The Proxy ALM sends all captured data to the ALM. Vice versa, the Proxy ALM receives instructions from the ALM. Then, it uses the SAS Setup Service for changing (i) the structure of the adaptation logic or (ii) parameters of the MAPE components, e.g., adding new rules. We defined a protocol between ALM and adaptation logic as well as interfaces the Proxy ALM and the adaptation logic have to support for reconfiguration at runtime. This

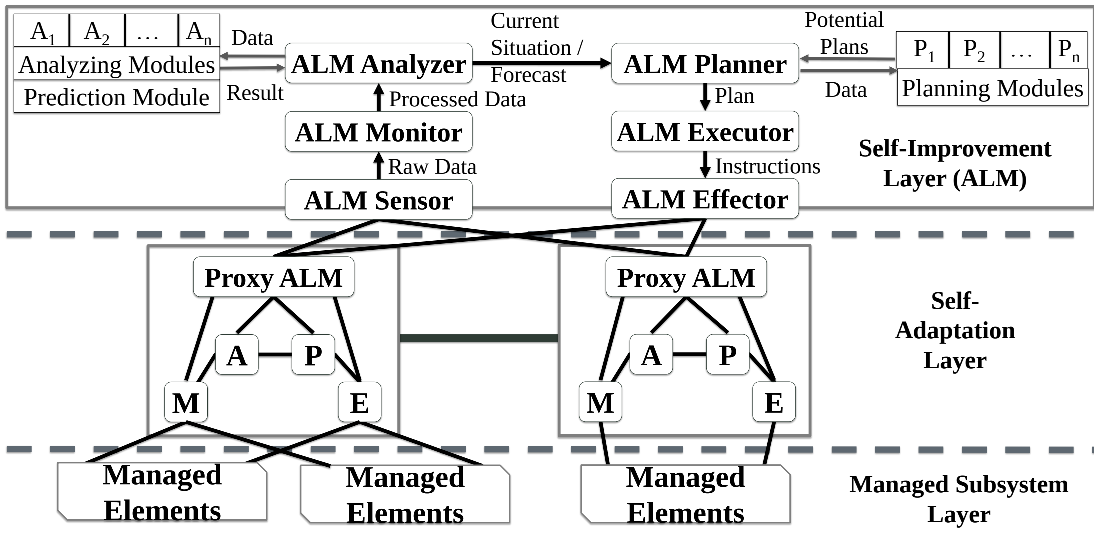


Figure 6.8.: System model of the integration of the ALM into a SAS. Due to simplicity reasons, the figure shows a centralized ALM, however, a decentralized variant is possible. Sensors, effectors, and supportive elements of the adaptation logic are omitted. The adaptation logic is distributed on two systems, the managed resources on three systems.

protocol[6] supports operations to (i) activate or deactivate MAPE components, (ii) change rules in a MAPE component, as well as (iii) change connections between MAPE elements, i.e., the distribution of the adaptation logic. Correspondingly, the adaptation logic must implement interfaces that support these operations. Consequently, the ALM is not restricted to the FESAS Framework and could be integrated into SASs implemented with different frameworks (cf. Section 4.2). The only requirements are the provision of a SAS Setup Service for meta-adaptation of the adaptation logic and a corresponding Proxy ALM implementation following the interface presented in Listing 6.1. This clear separation of ALM and adaptation logic combined with the well-defined interaction protocol supports requirement $R_{SI}6$. In the following, this section introduces the functionality of the MAPE components of the ALM.

6.2.2. Components of the Adaptation Logic Manager

As shown in Figure 6.8, the main components of the ALM are: (i) ALM Monitor, (ii) ALM Analyzer, (iii) ALM Planner, and (iv) ALM Executor. The

[6]Appendix C.1 presents a detailed description of the protocol.

```java
interface IProxyALM {

  public ALMData getSystemState();

  public ALMData getConnections();

  public void updateSystemStatus(AdaptationLogicElement status);

  public void updateConnection(Connection newConn,boolean delete);

  public ALMData changeComponents(ALMData request);

  public ALMData changeCommunication(ALMData request);

  public ALMData changeLogic(ALMData request);

  public ALMData changeRulebase(ALMData request);

  public ALMData getSensorData();

  public void updateSensorData(String data);

  public ALMData changeAlgorithm(ALMData request);
}
```

Listing 6.1: Interface of the Proxy ALM with methods for structural and parametric self-improvement. The interface is written in Java.

ALM Analyzer is supported by a data prediction module. Both, the ALM Analyzer and the ALM Planner can use different improvement modules for analyzing and planning self-improvement. The connection between ALM and Proxy ALM is managed by the ALM Sensor and the ALM Effector. The remainder of this section presents the functionality of the different elements.

The *ALM Sensor* collects from the Proxy ALMs (i) the structure of the adaptation logic, (ii) information about the MAPE components as well as (iii) data collected from the managed resources. The data is handed to the *ALM Monitor* which pre-processes it for the reasoning process (requirement $R_{SI}3$). Further, the *ALM Monitor* stores the data captured from managed resources to a data base and the information on the adaptation logic composition to a graph structure.

Both is easily accessible through well-defined interfaces by developers for analyzing and planning of self-improvement. Additionally, the monitor triggers the analysis of the data in the ALM Analyzer.

The *ALM Analyzer* uses the data from the managed resources to reason about their performances and characteristics. The process supports proactive and reactive reasoning (requirement $R_{SI}4$). For fulfilling the requirement for exchangeability (requirement $R_{SI}1.ii$), developers can define which analyzing modules the ALM Analyzer should use. Each of the modules implements algorithms for reactive or proactive analysis and decides if meta-adaptation is necessary. These modules require minor customization for the specific application only or own modules can be integrated. Further, the ALM Analyzer integrates prediction. Prediction enables to forecast future states and classify current states of the managed resources based on machine learning procedures for forecasting, classification, and clustering. Developers can use the prediction service within own developed analyzing modules. The ALM Analyzer collects the results from the analyzing module(s) and then sends the information to the ALM Planner.

The *ALM Planner* integrates different improvement modules for planning. One improvement module can integrate different types of meta-adaptation. The concept of planning modules supports different types of meta-adaptation. This fulfills requirement $R_{SI}1.i$. In order to enable developers the implementation of planning modules for specific applications, generic interfaces for the three types of self-improvement are integrated within the ALM Planner. These types can be extended with individual ones. As the ALM Planner has to decide at runtime between various modules for planning of self-improvement, the ALM supports different executions strategies following the Strategy Design Pattern [137]:

- `PREDEFINED`: The ALM Planner calls the specified planning modules in a predefined order and the first module that is able to handle the analyzed situation will be used for planning.
- `RUNTIME`: All planning modules run concurrently and the ALM Planner uses the first plan that is returned.
- `UTILITY`: The ALM Planner calls all approaches and collects utility values – specifying the benefit of self-improvement using the corresponding planning approach – and uses the plan with the highest utility.

The ALM Planner collects the results of the improvement modules depending on the specified execution strategy. Then, it triggers the adaptation process by transferring the relevant plan to the *ALM Executor* which translates the plan to specific instructions for change in the adaptation logic. Last, the *ALM Effector* sends the instructions to the corresponding Proxy ALM which uses the SAS Setup Service for adapting the MAPE components. Therefore, similar for monitoring the adaptation logic, the SAS must implement the ALM Protocol (cf. Appendix C.1). Additionally, the adaptation logic must integrate a component that implements the interface for the Proxy ALM (cf. Listing 6.1).

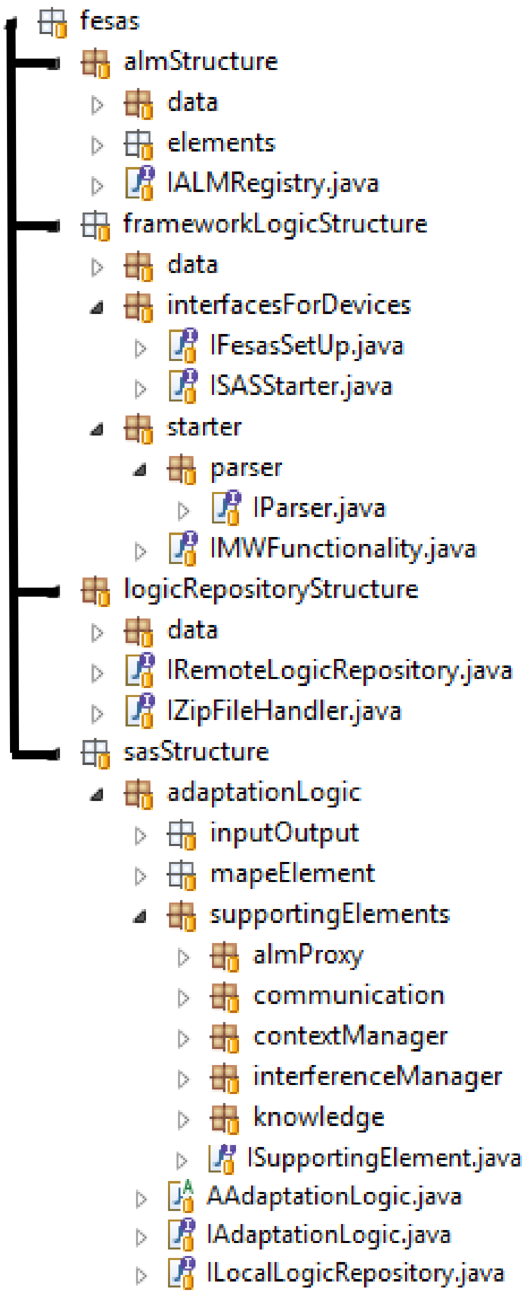
The layered approach is comparable to the hierarchical decentralization pattern described in [401]. As the authors claim in [401], the reaction time increases in higher layers. However, as described in Section 5.3, this additional execution time is not an issue as self-improvement is a back-up for self-adaptation. Additionally, integrating proactive self-improvement reduces the delays that results due to different time scopes.

7. Prototype Implementation

The previous chapter described the concept of the FESAS Framework. The framework spans the whole lifecycle of a SAS, from design time to runtime including self-improvement at runtime through meta-adaptation of the adaptation logic. For proving the feasibility and usefulness of the framework, we applied a proof by prototyping approach (cf. Chapter 3). Therefore, prototype implementations addressing the different elements of the concept have been implemented[1]. All are grounded on a set of interfaces defined in Java based on the design of the FESAS Framework from Chapter 6. The Maven module *fesas-libs-structure* provides interfaces for all components of the FESAS Framework. It is structured into four packages: `almStructure`, `frameworkLogicStructure`, `logicRepositoryStructure`, and `sasStructure` (cf. Figure 7.1). Each of them contains interfaces for the components that are part of the corresponding elements of the FESAS Framework. The package `almStructure` offers the interfaces for the elements of the ALM and a package with the metadata, e.g., for registration at the ALM or for the ALM protocol for communication between adaptation logic and ALM. The package `frameworkLogicStructure` is divided into packages for the data, interfaces for the components that the adaptation logic has to offer for using the FESAS Framework – e.g., for the SAS Setup Service –, and for starting the FESAS Middleware. The package `logicRepositoryStructure` contains the relevant interfaces for the functionality of the FESAS Repository. Last, the package `sasStructure` subsumes all interfaces for implementing the FESAS Adaptation Logic template and the FESAS Middleware. Figure 7.1 shows the structure of the Maven module.

This chapter presents the prototypes ordered by the phase of the software lifecycle they support. First, Section 7.1 describes the FESAS IDE. The FESAS IDE offers tools that encapsulate the development tasks of the FESAS workflow as pre-

[1]These are public available on the FESAS project website: `https://fesas.bwl.uni-mannheim.de/`



```
fesas
    almStructure
        data
        elements
        IALMRegistry.java
    frameworkLogicStructure
        data
        interfacesForDevices
            IFesasSetUp.java
            ISASStarter.java
        starter
            parser
                IParser.java
            IMWFunctionality.java
    logicRepositoryStructure
        data
        IRemoteLogicRepository.java
        IZipFileHandler.java
    sasStructure
        adaptationLogic
            inputOutput
            mapeElement
            supportingElements
                almProxy
                communication
                contextManager
                interferenceManager
                knowledge
                ISupportingElement.java
            AAdaptationLogic.java
            IAdaptationLogic.java
            ILocalLogicRepository.java
```

Figure 7.1.: The *fesas-libs-structure* Maven module that contains the Java interfaces and packages for the components of the FESAS Framework. It is divided into four packages: `almStructure`, `frameworkLogicStructure`, `logicRepositoryStructure`, and `sasStructure`.

sented in Section 6.1.2 and supports developers and designers of SASs in using the FESAS Adaptation Logic Template. Second, the FESAS Repository is a central element for the deployment of SASs using the FESAS Framework. Section 7.2 explains the implementation of the FESAS Repository and the connection between adaptation logic and FESAS Repository. Third, Section 7.3 presents the implementation of the FESAS Middleware. The FESAS Middleware provides different elements that support the deployment of SASs. Further, it is complemented by a Publish/Subscribe (Pub/Sub) system for communication between the MAPE elements. Fourth, the FESAS Framework integrates the ALM for self-improvement of the adaptation logic at runtime. Section 7.4 describes the implementation of a prototype of the ALM. Additionally, the FESAS Framework offers different reference systems that might be used as foundation for implementing a SAS. Their implementations are presented in Section 7.5.

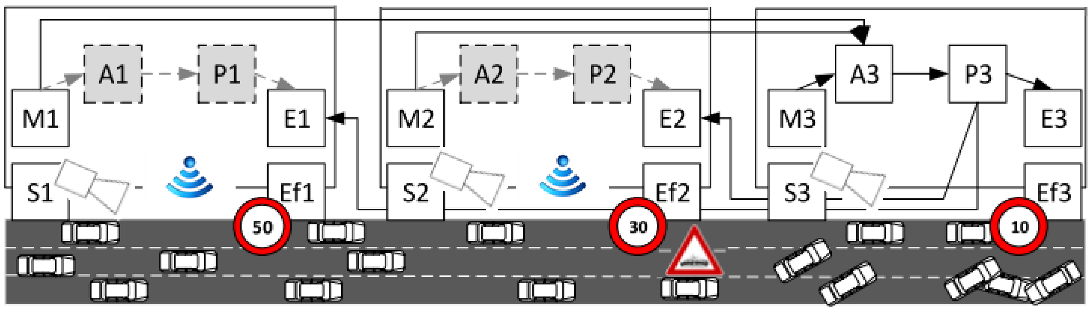


Figure 7.2.: This figure shows the SmartHighway system after a structural self-improvement was triggered by the self-improvement layer in response to an accident. The dashed elements in the adaptation logic indicate its structure before the adaptation.

Besides the implementation of the prototype, this section also describes how to use the prototype to implement a SAS using the following example scenario. The *SmartHighway* traffic management system controls digital traffic signs installed on a highway and self-driving vehicles[2] via Vehicle-to-Infrastructure (V2I) communication. Traffic cameras measure the amount of vehicles and the system calculates the current traffic flow. High density with heterogeneous velocities can lead to traffic jams [364]. Such situations are analyzed for reacting accordingly by adjusting speed limits to homogenize the traffic flow. Additional modules en-

[2]Each self-driving vehicle itself is a SAS. However, in our research we do not target to build self-driving cars and see them as black box systems.

able congestion avoidance through re-routing and releasing shoulders. Figure 7.2 shows the SmartHighway system that represents the adaptation logic. Each section of the highway is managed by a dedicated adaptation logic to optimally adapt the traffic flow. However, as traffic patterns might change abruptly due to congestion or accidents, this decentralized setting might contradict the adaptation goal. A switch of the communication pattern in the adaptation logic, e.g., from a fully decentralized setting to a coordinated one (as presented in [401]) would enable region-wide planning with single components for analysis and planning. Figure 7.2 illustrates structural self-improvement as a response to an accident.

7.1. Implementation of the FESAS IDE

The FESAS Middleware supports the development of reusable SASs. However, a study with Master students (cf. Section 8.2.2) showed that implementing the functional logics and writing the configuration files manually is error-prone as well as time-consuming since the developer has to learn the specific syntax for the configuration files and the development workflow. The *FESAS IDE* [225] addresses these issues. In the following, this section presents the FESAS IDE. Parts of this section are based on [225][3].

The FESAS IDE extends the Eclipse IDE with two plug-ins: the *FESAS Development Tool* and the *FESAS Design Tool*. We choose the Eclipse IDE as it is open-source and commonly used. Both tools are implemented using the Plug-in Development Environment (PDE), a customized version of Eclipse for plug-in development. Eclipse itself is implemented using OSGi[4] and Java. OSGi enables a modularization of Java code. With PDE, full access to all modules of Eclipse is available. This enables, e.g., changing the menu bar of Eclipse or extending a development perspective as well as using the functionality of perspectives, e.g., the syntax highlighting offered by the Java perspective. Therefore, developers have to specify extension points that hook into the Eclipse system and change the program flow.

The system developer uses the *FESAS Development Tool* for developing the functional logics' code, which is then stored in the FESAS Repository. Using the

[3] [225] is joint work with F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schürr.
[4]OSGi's website: `https://www.osgi.org/`

FESAS Design Tool, the system designer configures the components of an adaptation logic and specifies, which code the FESAS Middleware loads at runtime. Mapped to the SmartHighway scenario, the developer implements code – e.g., for analyzing the current traffic conditions – and the designer defines the structure of the adaptation logic as well as the information exchange between sections. Through this separation of concerns, designers can build a SAS almost without coding effort by using the FESAS Repository as "app store" for finding suitable MAPE functionality. Both plug-ins are integrated by extending the menu bar of Eclipse. Next, this section introduces the *FESAS Development Tool* and the *FESAS Design Tool*. Further, it describes how the system designer and system developer can use the plug-ins to speed up the development of a SAS as well as the implementation of the tools.

7.1.1. FESAS Development Tool

The *FESAS Development Tool* supports developing the functional logics of MAPE components in Java. Developers can use an existing Eclipse installation and solely have to add the *FESAS Development Tool* plug-in. The process for using the *FESAS Development Tool* is divided into five steps: (i) creating a FESAS project, (ii) initializing a logic element and its metadata file, (iii) writing source code and testing the functionality, (iv) preparing the elements for the repository, and (v) committing the prepared elements to the FESAS Repository. Figure 7.3 shows the different steps and the corresponding dialogs in Eclipse. Next, this section provides general implementation details for the *FESAS Development Tool*. Afterwards, it illustrates the implementation of the five steps for development.

As the plug-in is integrated as menu bar into the Eclipse IDE, developers can use all known features, such as syntax highlighting, on-the-fly compilation, and simplified integration of libraries. All functionality of the plug-in is controlled via the menu bar or context menus. Therefore, extension points are specified for each entry of the *FESAS Development Tool* menu group. Each of these extension points is linked to a class that extends the `org.eclipse.jface.dialogs.Dialog` class for building a dialog in Eclipse. For implementation of a dialog, the method `createDialogArea(..)` is overwritten for specifying the look-a-like of the dialog.

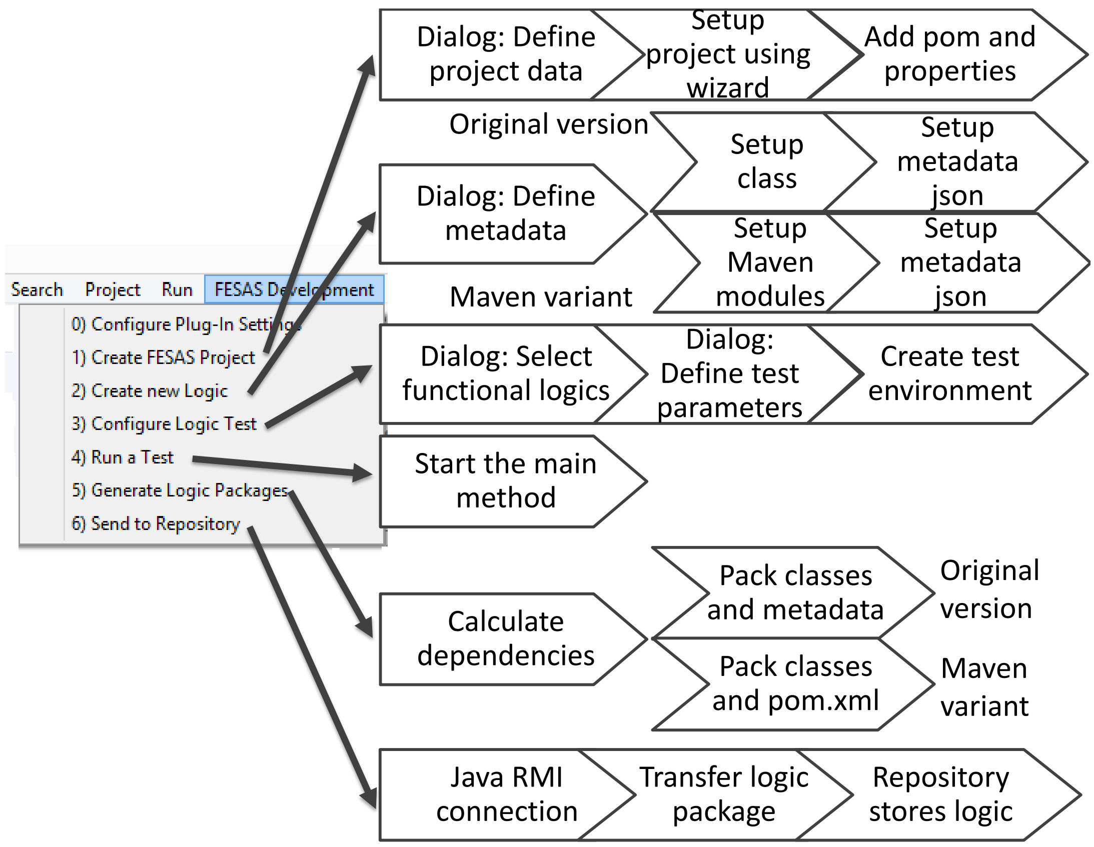


Figure 7.3.: Different steps of the workflow with the FESAS Development Tool mapped to the menu entries. The first step for configuration of the tool is omitted.

Additionally, at least the method `okPressed()` is implemented to handle the action when the developer clicked the dialog's `OK` button. These handlers might rely on additional functionality, e.g., to set up a functional logic or the test environment. Additional handlers are implemented for updating of the dialog triggered by user input or handling an abortion. The plug-in relies on Eclipse functionality and uses the GUI, the project wizard as well as the Java developer tools. Further, it extends the Java perspective with a view for the metadata of functional logics and integrates custom elements. In the following, this section describes the different activities of using the plug-in as well as their implementation.

As prerequisite, a developer has to create a FESAS project, which is a customized Maven project. The plug-in configures this project, i.e., it defines the dependencies to the FESAS Middleware Maven library in the Maven `pom.xml` of the project, sets the properties, and creates all relevant folders as well as the package structure. The process is integrated into the *Eclipse Project Wizard* for creating a new project, i.e., it is identical to the creation of any other project in Eclipse, however, it is triggered by the menu bar of the plug-in. Next, the developer has to generate a logic element, e.g., for the SmartHighway system, this could be a planning logic that calculates a speed limit depending on the current traffic situation. The developer specifies the metadata of the functional logic which is used by the repository to identify the purpose of the functional logic.

```java
public class CLASSNAME extends AbstractLogic implements LOGIC_INTERFACE {

  public CLASSNAME() {
    super();
    SUPPORTED_INFORMATION_TYPES
    this.informationType = OWN_INFORMATION_TYPE;
    type = LogicType.LOGIC_TYPE;
    shortName = "CLASSNAME";
  }

  public void initializeLogic(HashMap<String, String> properties)
  { [...] // code removed }

  public String callLogic(IKnowledgeRecord data) {
    if (data instanceof KnowledgeRecord) {
      if (data.getData() instanceof OBJECT) { [...] // code removed }
      return "Not the expected data type! It is: " + data.getData().
          getClass().getSimpleName();
    }
    return "Not a KnowledgeRecord! It is: " + data.getClass().
        getSimpleName();
  }
  [...] // code removed
}
```

Listing 7.1: Template for a functional logic (omitting comments with instruction for developers). All terms in capital letters (except of OBJECT) are wildcards, which are substituted with the information captured in the creation dialog. The method `callLogic(..)` runs when the functional logic is called.

The tool initializes a Java source code file as well as a JSON file that contains the metadata. For both files, templates exist. These templates contain wildcards for the information captured by the plug-in through a dialog. Listing 7.1 shows such a template for the functional logic. The template is a Java class that contains different wildcards written in capital letters. These wildcards are substituted by the metadata, e.g., the wildcard `CLASSNAME` is substituted by the name of the functional logic, the wildcard `SUPPORTED_INFORMATION_TYPES` by an array with all supported information types as specified in the creation dialog. The *FESAS View* extends the Eclipse GUI and offers an overview on the metadata of a selected functional logic source code file.

Further, developers can start a system for testing. Using dialogs in Eclipse, the developer configures the relevant subset of MAPE components as well as defines sample data for simulating input of managed resources. Therefore, developers have to specify the components they would like to test and add the input data. As an example for the SmartHighway system, the analyzer and planner can be specified as functional logics that should be tested and the output of the monitor in form of traffic flows can be coded as sample data. The *FESAS Development Tool* uses pre-defined Java classes as templates and generates the test environment. The test system provides the debugging support known from Eclipse as it can be executed as Java program. This allows detecting errors early in the development process without the need to set up the full SAS.

After finishing the implementation and testing phase, the *FESAS Development Tool* packs the code into archive files. We implemented two different prototypes which can be used depending on the type of FESAS Repository implementation. For the first prototype, the archive file contains the metadata as JSON file and the functional logic as well as files containing referenced code, so called *dependencies*, as Java byte code. The plug-in calculates the dependencies using the ASM ClassReader library[5]. Therefore, the class file of the functional logic is loaded and the plug-in searches recursively in the byte code for any related classes that are not part of the Java library or FESAS libraries through analyzing the imports, related classes, as well as referenced libraries. The plug-in adds all identified dependencies to the archive file.

[5]ASM ClassReader website: `http://asm.ow2.org/asm33/javadoc/user/org/objectweb/asm/ClassReader.html`

So far, this section has described the prototype presented in [225]. We built a second prototype which relies on the Maven structure for analyzing the dependencies of a functional logic. First, this prototype builds one Maven module for the functional logic, one for dependencies classes that is used for all functional logics, and one parent module that combines a functional logic module with its dependencies. Second, this prototype reads the `pom` files from the parent development project and, through that, from the functional logic and related projects with dependencies. Third, it generates an aggregated `pom` file that integrates all dependencies specified in Maven. This enables the repository of the FESAS Middleware at deployment to load all Maven dependencies directly from a Maven repository, hence, the resulting archive file is smaller. User-defined classes are integrated into the archive file similar to the former described version of the plug-in, however, they have to be specified in the project of the functional logic or the dedicated dependency project (which is set up by the *FESAS Development Tool*).

Using the plug-in, a developer sends the functional logics to the FESAS Repository using a connection via Java RMI. Developers can specify the connection to the FESAS Repository in a configuration file. A SAS that runs the FESAS Middleware loads the code from the repository as described in Section 7.3.1. With the help of the *FESAS Design Tool*, which is described in the next section, the system designer writes the configuration files that specify which logic should be loaded at system deployment.

7.1.2. FESAS Design Tool

Besides implementing the functional logics of the MAPE components, the SAS has to be designed. The *FESAS Design Tool* supports the system designers who specify which (sub)systems host which parts of the managed resources and/or the adaptation logic. Additionally, the system designer specifies the functional logic that the components load. Further, the system designer uses the *FESAS Design Tool* to describe the connections between the components, such as which sensor retrieves data from which managed resources or from which monitor an analyzer receives information. This allows the implementation of distribution patterns, such as the ones from [401].

The *FESAS Design Tool* is an Eclipse plug-in that supports model-driven development for designing the system. It provides a graphical editor in which the system designer specifies the configuration of the adaptation logic. Further, the plug-in is complemented with a generator for JSON files representing the information specified in the editor. These files are used in the system deployment process to configure the SAS. The *FESAS Design Tool* is based on the *Eclipse Modeling Framework (EMF)*[6], the *Graphical Modeling Framework (GMF)*[7], and the *Acceleo*[8] code generator. Figure 7.4 depicts the creation of configuration files using the *FESAS Design Tool*. Next, this section presents the implementation of the editor using EMF and GMF and the JSON file generation with Acceleo.

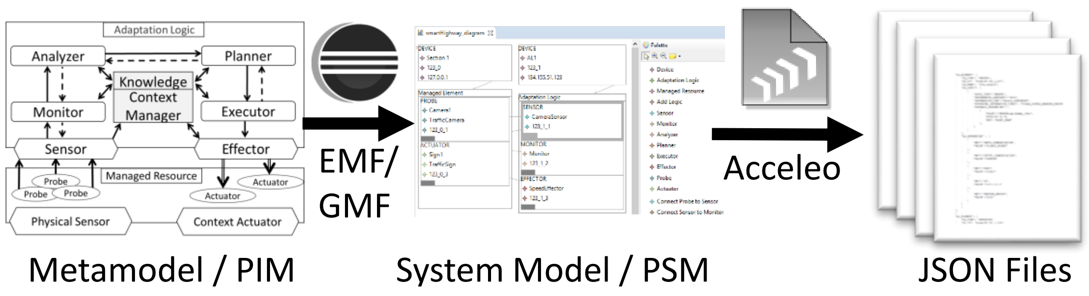


Figure 7.4.: Process of JSON configuration file generation with EMF / GMF and Acceleo.

EMF and GMF are Eclipse plug-ins. As other model-driven development approaches, EMF specifies a Platform Independent Model (PIM) and a Platform Specific Model (PSM). The PIM represents the metamodel and is implemented as an Ecore model with EMF. It is based on the FESAS Adaptation Logic Template presented in Section 6.1.1. Elements of the metamodel represent the MAPE components with their properties, including the contracts for functional logics that have to be loaded at runtime. The meta elements of the PIM are the elements that describe the SAS, i.e., the MAPE components. The EMF generator model generates code based on the PIM and, hence, act as Decorator [137]. The PIM's elements are the base for the implementation of PSMs. A PSM specifies the components of a specific adaptation logic, its configuration, and the connections between the components. For each SAS, a PSM is generated using the Ecore model. Having the EMF files only, a developer would have to write the

[6] EMF's website: `https://eclipse.org/modeling/emf/`
[7] GMF's website: `http://www.eclipse.org/modeling/gmp/`
[8] Acceleo's website: `http://www.acceleo.org/pages/welcome/en`

PSMs manually by writing XML files complying to the EMF syntax. For a more convenient use of the plug-in, we implemented a GMF-based graphical editor.

GMF supports the development of graphical editors for all types of models and purposes. The GMF runtime engine combines different components: Graphical Definition, Tooling Definition, Mapping Definition, and GenModel. The GMF Graphical Definition describes the different elements, the editor should be able to draw (e.g., a box for representing a monitor component). Further, it describes their lookalike on the diagram canvas. The GMF Tooling Definition defines the tools, e.g., for each component that the user should create on the diagram canvas, a tool has to be specified. The tools are shown in the so called palette. The GMF Mapping Definition maps the tools and their graphical definition with the EMF Ecore model. This enables an automatic creation of an element in the PSM, once an element is added to the diagram canvas. Further, the system designer can only use the elements of the palette. The structural correctness of the model is guaranteed as EMF/GMF permits to use the elements of the palette only as specified in the metamodel, e.g., it prohibits to connect a monitor directly to a planner component. The GMF GenModel combines all these information in a definition of the editor and generates an Eclipse plug-in that represents the editor's functionality. GMF offers functionalities to generate all of the required components out of an Ecore model. Accordingly, for implementation of the *FESAS Design Tool*, we first needed to define the PIM as Ecore model. With GMF we than automatically generated all components for the editor and adjusted them slightly, e.g., delete elements that should not be drawable or change the colors and symbols for elements in the editor. The logic of the editor and storing of the content of the diagram canvas to the model file is offered by GMF out of the box.

System designers can drag and drop elements from the palette into the canvas (e.g., an analyzer component), specify the functional logic of these elements, and add connections (e.g., to a planner or another analyzer). Connections between components are represented by arrows in the canvas and by elements in the palette. Further, arrows between `device` and `managed element` as well as `adaptation logic` groups indicate which physical device deploys which adaptation logic elements and which parts of the managed resources. Figure 7.5 shows the diagram canvas with a subset of the SmartHighway system. The system follows the fully decentralized pattern as each section is managed by a dedicated

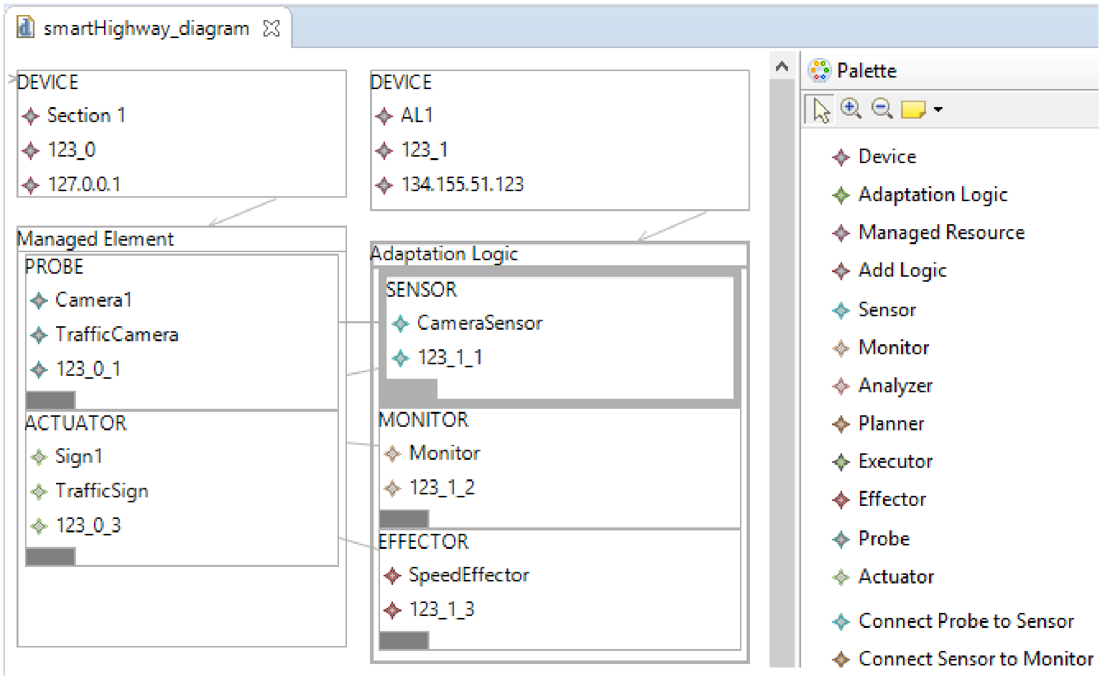


Figure 7.5.: Diagram canvas with an exemplified SAS. The left side shows the editor. There, a subset of the SmartHighway system showing elements of one section is specified. The right side shows the palette with the available elements for system design.

part of the adaptation logic with full MAPE functionality. Further, the designer defines the connection between managed resources – e.g., traffic cameras and traffic signs – and the adaptation logic using the plug-in.

For further convenience, it is possible to choose a distribution pattern. The plug-in offers a fully centralized pattern, a fully decentralized pattern – i.e., the adaptation logic is separated into several subsystems that all have full MAPE functionality, however, no connection between each other –, and the decentralization patterns from [401]. This functionality is a custom implementation and does not extend the EMF/GMF functionalities. As a first experiment for the vision of having a self-* IDE, we implemented a mechanism that can choose a suitable pattern depending on system properties, such as scalability, responsiveness, and local vs. global optimization. We added this mechanism as additional functionality of the *FESAS Design Tool*. However, as this is in preliminary stage, that function is excluded from further investigation in this thesis.

```
[template public generateComm(aDiagram :DiagramElement)]
[file (('config_Comm.json'), false, 'UTF-8')]
{
  "COMM_ADAPTATIONLOGIC": ['['/]
  [for (anAdaptationLogic:AdaptationLogicElement | aDiagram.
      diagramAdaptationLogicConn)]
  [for (sensor: SensorElement | anAdaptationLogic.sensorALConn)]
    [for (sen_mon: SensorMonitorComm | sensor.mon_sen)]
      [for (aLogic: ALLogicElement | sensor.AL_LOGIC)]
    {
      "COMM_ELEMENT": {
        "COMM_TYPE": "[sen_mon.COMM_TYPE/]",
        "COMM_RECEIVER": "[sen_mon.RECEIVER.AL_ID/]",
        "COMM_SENDER": "[sen_mon.SENDER.AL_ID/]",
        "COMM_INFO_TYPE": "[sen_mon.COMM_INFO_TYPE/]",
        "COMM_INFO_CATEGORY": "[sen_mon.COMM_INFO_CATEGORY/]"
      }
    },
      [/for]
      [/for]
    [/for]
    ... // code removed
    }
[/file]

[/template]
```

Listing 7.2: Excerpt of an Acceleo template showing the template that is filled with the data collected in the editor for the connections between the MAPE components.

Once the system is designed in the editor, the next step is the creation of configuration files. For that, the plug-in uses the functionality of the open source code generator *Acceleo*. Acceleo enables the transformation of EMF models to various formats, such as HTML, XML, or JSON. Acceleo uses the EMF-based PSM and templates of the JSON files as input for the configuration file generation. The templates are composed of static JSON data and wildcards. During file generation, the Acceleo generator replaces the wildcards with the values of the PSM represented by the model specified in the editor. Listing 7.2 shows an excerpt of such an Acceleo template. For each device, Acceleo creates one JSON file with the information about the components as well as one file with the information regarding connections. The FESAS Middleware uses these files during the deployment of a SAS as specified in Section 6.1.2. Acceleo offers reference implementations in Java based on EMF models. We extended and customized these implementations.

```
{
  "DEVICE_PROPERTIES": [
    {
        "TYPE": "device_id",
        "VALUE": "fesasID-123_0_000"
    },... // code removed],

  "AL_ADAPTATIONLOGIC": [{
    "AL_TYPE": "MONITOR",
      "AL_ID": "fesasID-123_1_001",
      "AL_NAME": "Device_km1_monitor",
      "AL_LOGIC": [
         {
          "SUPPORTED_INFORMATION_TYPES":[
              "Context_CAMERA_DATA"],
        ... // code removed],
      "AL_PROPERTIES" :[... // code removed
  ],

  "MANAGEDRESOURCES": [{
    "MR_RESOURCE": {... // code removed}},
  ]
}
```

Listing 7.3: Excerpt of a generated JSON file for showing its structure with some data from the SmartHighway system.

Listing 7.3 shows the structure of a resulting configuration JSON file. As a lightweight and language-independent representation, JSON suits this use case. The file for a device has a JSON element for the device as root element. This element contains the device's properties as JSON attributes and JSON elements for the adaptation logic, managed resources, and the communication. Each JSON element represents an array with the adaptation logic components and/or the probes and actuators of the managed resources. The communication links between MAPE elements are specified in a dedicated file. The adaptation logic components have contracts for the logic they should load. The combination of editor and code generator is optimized for the FESAS Framework, however, not restricted to it. The separation of the editor and configuration file generation enables to use the plug-in for other languages by adjusting the Acceleo templates.

7.2. Implementation of the FESAS Repository

As described in Section 6.1.3, the FESAS Repository stores the functional logics for MAPE components. Throughout the development of the FESAS Framework, we experimented with different implementations of the FESAS Repository. The implementations also influenced the generation of functional logic elements. However, all use the same principle: the separation of code and metadata. Furthermore, the metadata syntax stayed stable and the mechanisms for storing the metadata and code in the FESAS Repository also were reused in the different variants. In this section, we describe the three variants of the FESAS Repository.

As presented in Section 4.1, aspect-oriented approaches are promising for the separation of code that should be adapted and cross-functional concerns that are not influenced by adaptation. Hence, it seems legit to use this approach also in the adaptation logic to separate cross-functional concerns as data handling and aspects: application-specific concerns, i.e., the implementation of the MAPE functionality. This is in accordance with the nature of the FESAS Adaptation Logic Template and Component Template. We implemented a prototype using the *TRAP-J* software tool [318] which extends Java with AOP capabilities. Furthermore, in contrast to usual AOP-based approaches where the "weaving" process – i.e., the integration of aspects – happens at system deployment, TRAP-J supports weaving at runtime. Using this in the adaptation logic supports self-improvement. Packaging during development just required to bundle the aspect and all dependencies as well as adding the metadata. Furthermore, the mechanism for loading code in the adaptation logic is offered by TRAP-J. We implemented a fully-fledged prototype. However, the prototype comes with the cost of learning a new syntax as TRAP-J introduces additional programming concepts and extends the Java syntax as well as the need of additional tools. For the sake of usability, we decided to implement an alternative.

The second variant (called *original variant* in the following) does not change the developers workflow [225]. It does not require developers to learn a new tool. For loading code into the adaptation logic, we implemented a dynamic code reloading mechanism based on the *Java class loader*. However, the approach did not proof its applicability for development in large systems as sometime dependencies are not identified and the calculation time increases with the project size. The FESAS IDE offers this mechanism for small-scale projects.

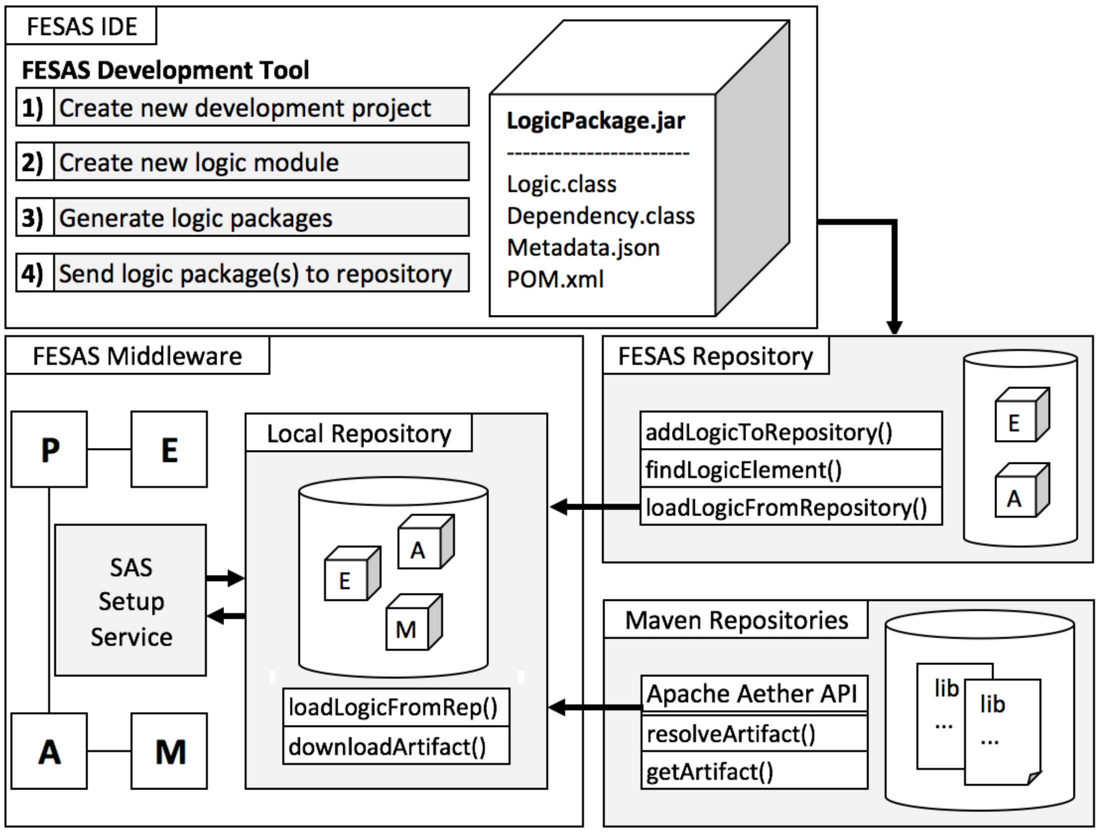


Figure 7.6.: System model for the FESAS Repository with all involved elements.

As Maven enables the loading of dependencies in the form of jar packages and its common usage in Java development projects, we decided to extend the existing mechanism with support for Maven (called *Maven variant* in the following). The obvious advantage is that Maven specifies all integrated external dependencies in a `pom` file. This reduces the search for dependencies to only searching newly developed classes. We decided to re-design the FESAS IDE, so that all dependent classes that are newly implemented by developers are stored in the same project and are automatically packed together with the functional logic classes. Figure 7.6

presents the Maven-based approach. In the following, this section describes how the FESAS Repository works. Any differences between the last two mentioned variants are described.

Using the FESAS IDE, the developer can send a functional logic package to the FESAS Repository. This package contains the metadata, the code of the functional logic, the code of custom implemented dependent classes, and the `pom` file in case of the Maven variant or all other dependent classes for the original variant, respectively. The FESAS Repository is implemented in Java. The main class is `RemoteLogicJarRepository` which implements the interface `IRemoteLogicRepository` (cf. Appendix B.6).

Figure 7.7 shows the sequence diagram for the interaction between FESAS IDE, FESAS Repository, and FESAS Middleware. In the following, this section explains the most important methods of the FESAS Repository. The first method is called `addLogicToRepository()` and is responsible for receiving new logic packages from the FESAS IDE. The method is called by the *FESAS Development Tool* via `Java RMI` when sending the packaged jar files. When called, the method first stores the incoming logic package into a temporary folder. It then extracts the JSON files and reads the metadata information, creating a `LogicElementMetadata` object and stores the package. For the Maven variant, also the `pom` file is stored. Analysis of the `pom` file happens at the FESAS Middleware. The file path of the stored package is added to a nested hash map. The first layer of the hash map groups the logic elements by logic type. The logic metadata object is then used as a key to the logic file path on the second layer. This results in the following nested hash map scheme:
`HashMap<LogicType; HashMap<LogicElementMetadata; String»`

The second major method of the repository is `findLogicElement()` and resolves a JSON contract for finding the most suitable logic package. The method first filters all available logic packages by the logic type and programming language specified in the contract. A utility function is used to measure the similarity of contract and logic metadata. The method returns to the FESAS Middleware the JSON metadata that describes the logic with the highest utility value.

The FESAS Middleware calls the method `loadLogicFromRepository()` if the logic named by the method `findLogicElement()` is not yet stored locally in

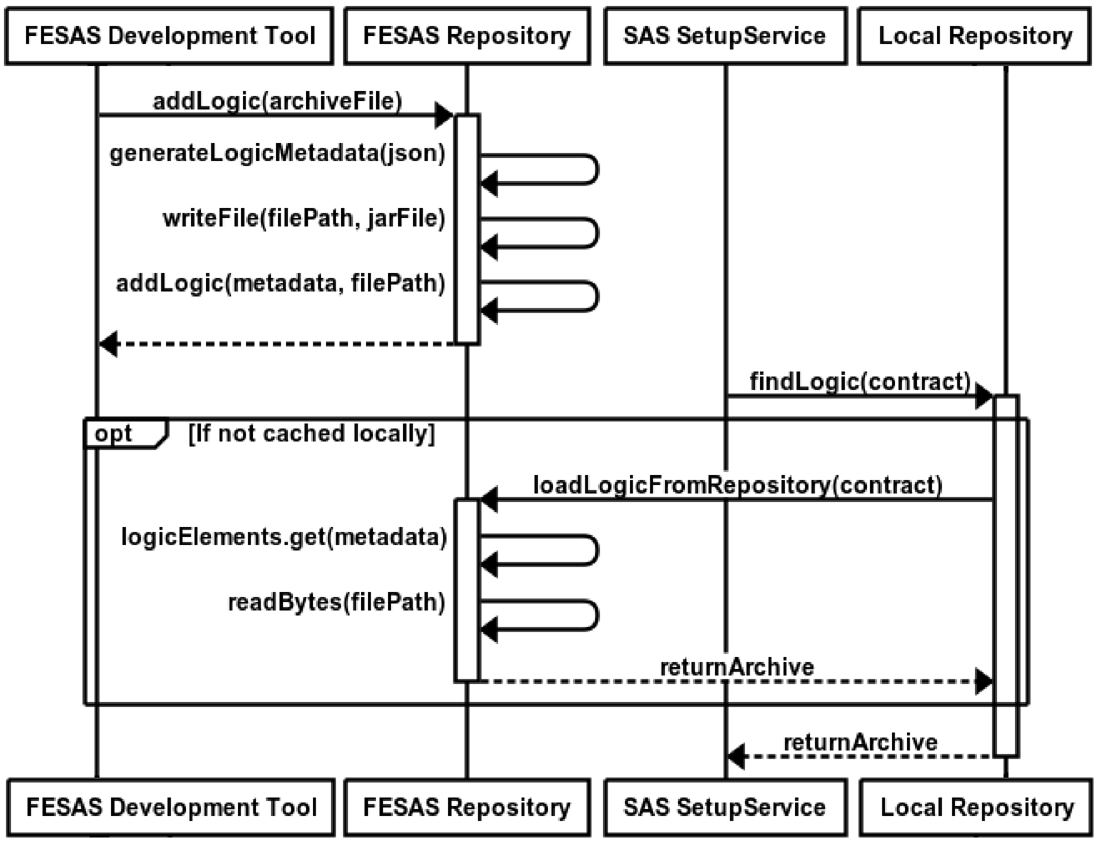


Figure 7.7.: Sequence Diagram for the interaction of FESAS Development Tool, FESAS Repository, and SAS (created using websequencediagrams.com).

the adaptation logic. The method call includes the JSON metadata which was determined by the method `findLogicElement()`. It performs a look-up on the nested hash map in order to get the file path of the requested logic package. The logic package is then returned to the FESAS Middleware. The following section explains the handling of the package in the adaptation logic.

7.3. Implementation of the FESAS Middleware

Based on the design of the FESAS Framework presented in Section 6.1, the FESAS Middleware controls the system deployment, connects managed resources and adaptation logic as well as adaptation logic and ALM, manages the context, and offers modules for communication between the MAPE elements. This section introduces the implementation of (i) the prototype of the FESAS Middleware and (ii) a publish/subscribe communication system for communication between MAPE elements. The prototype is implemented in Java as it (i) offers support for many different operating systems as well as (ii) can be used for developing mobile apps for Android devices. Subsequently, Section 7.3.1 present the implementation of the components of the FESAS Middleware which control the deployment of the adaptation logic and self-improvement. Afterwards, Section 7.3.2 describes a reusable Pub/Sub approach which is used for communication between MAPE components of the adaptation logic.

7.3.1. FESAS Middleware Components

The FESAS Middleware consists of different components. It is responsible for the deployment of the adaptation logic at the start of the system. In this first prototype, the following components are implemented in Java: the Middleware Starter, the Local Repository, the Proxy ALM, the JSON Parser, and the SAS Setup Service. The Java-based implementation of the middleware is available as Maven library and can easily be integrated into an adaptation logic that follows the system model from Section 6.1.

The Middleware Starter is triggered when starting the system. It is implemented as Singleton and controls the start process by loading the Proxy ALM, the Local Repository, the JSON Parser, and the SAS Setup Service. Figure 7.8 shows a sequence diagram of the start-up procedure and the interplay of the components. Next, this section describes the components in detail.

If self-improvement is enabled, the Proxy ALM is started at first. The Proxy ALM is implemented as a Singleton. Several elements of the system interact with it. First, all MAPE elements send updates of their state if they add or remove a functional logic to the Proxy ALM. Second, the sensor or monitor has to transmit the data received from managed resources to the Proxy ALM. This has to be triggered by developers through calling the `updateSensorData` method. The Proxy ALM caches these data and sends it to the ALM on request. Third, the Proxy ALM is responsible for the registration at the ALM. Therefore, three options are implemented: (i) Java RMI, (ii) WebSockets, or (iii) local

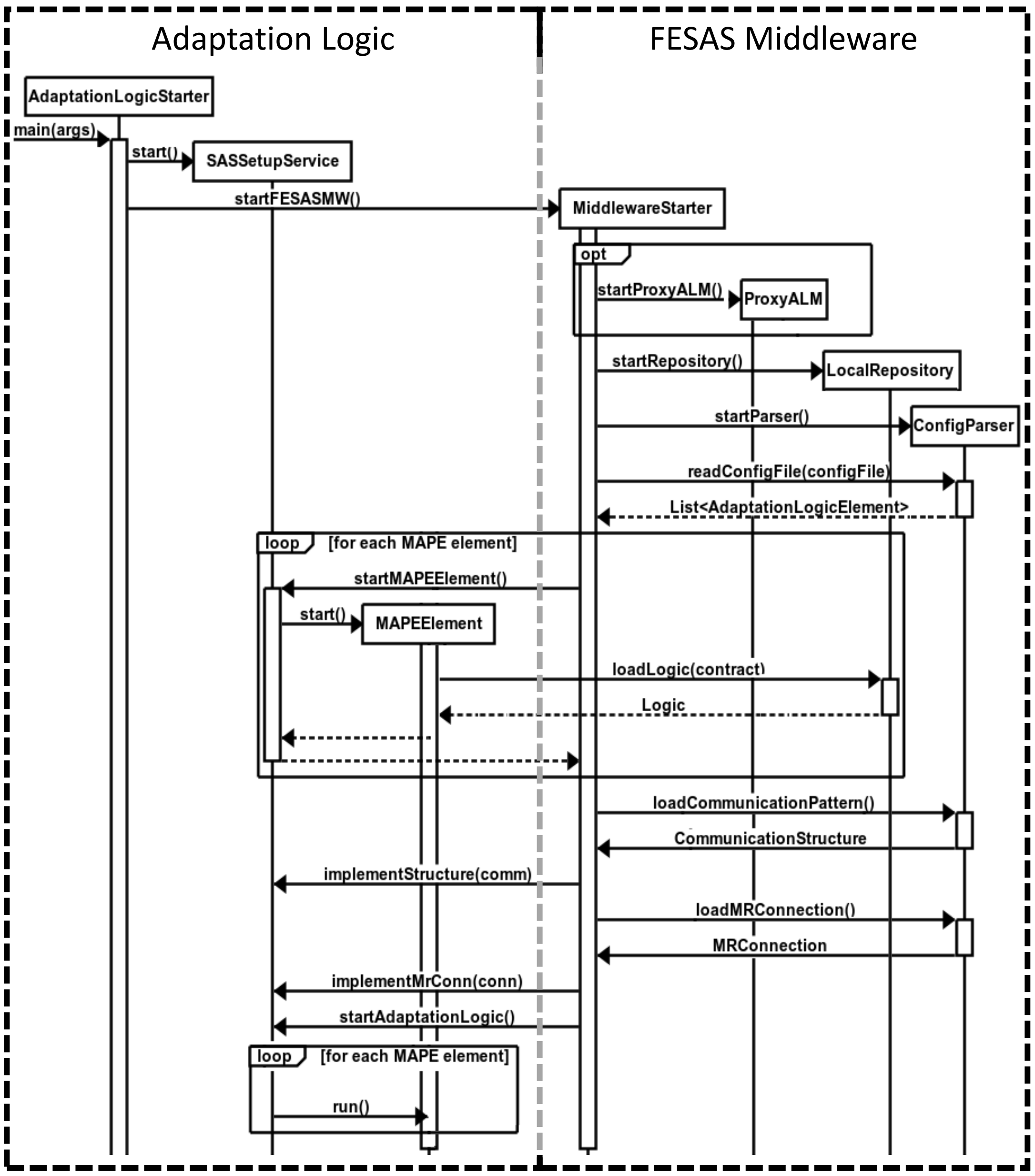


Figure 7.8.: The sequence diagram shows the interaction of the FESAS Middleware components with the MAPE components during the start of the system. Interactions with the ALM and the FESAS Repository as well as sub-modules of the components are omitted (created using websequencediagrams.com).

method calls. Whereas the first two options are used for SASs which separate the ALM from the adaptation logic, the third option is used for the single module ALM. The registration is done by the `RegistrationHandler` component of the

Proxy ALM. It sends registration requests to the ALM. Last, the Proxy ALM contains the ALMProxyRequestHandler which processes all requests from the ALM and transmits them to the Proxy ALM which uses the SAS Setup Service to adapt the adaptation logic, e.g., add/remove a functional logic. After the initial registration, Proxy ALM and ALM communicate using sockets. All requests between Proxy ALM and ALM use the JSON format according to the protocol presented in Appendix C.1. The *Gson*[9] library supports marshalling and demarshalling of the data from Java objects to JSON and vice versa.

Additionally, the Local Repository is started and sets up a connection to the FESAS Repository. Depending on whether a dedicated, remote repository on a server or a local, integrated variant of the FESAS Repository is used, the connection is either established via *Java RMI* or local method calls. Besides a mechanism for transferring a Java class including demarshalling of the class and implemented dependencies from binary data to a Java class object, the repository integrates a mechanism for dynamic class loading based on the standard Java class loader. Further, the Local Repository caches loaded classes in a HashMap in case they are requested again. The original variant of the FESAS Repository from [225] uses this mechanism to load all files from the archive. For the Maven variant described in Section 7.2, the Maven `pom` file specifies dependencies that are available as Maven archives. Therefore, the Local Repository uses the `MavenXpp3Reader` class from the *Apache Aether*[10] library to resolve the dependencies defined in the `pom` file and load the classes accordingly. Hence, the middleware supports both approaches of loading dependent classes: from a archive file as well as from Maven repositories. Finally, the Local Repository triggers loading the functional logic into the Java Virtual Machine, returns an object of it, and the MAPE component that started the loading process integrates this object. Figure 7.9 shows a sequence diagram with the dynamic class loading process in the adaptation logic from the extended Maven variant.

[9]Gson library on Github: `https://github.com/google/gson`
[10]Apache Aether website: `http://maven.apache.org/aether.html`

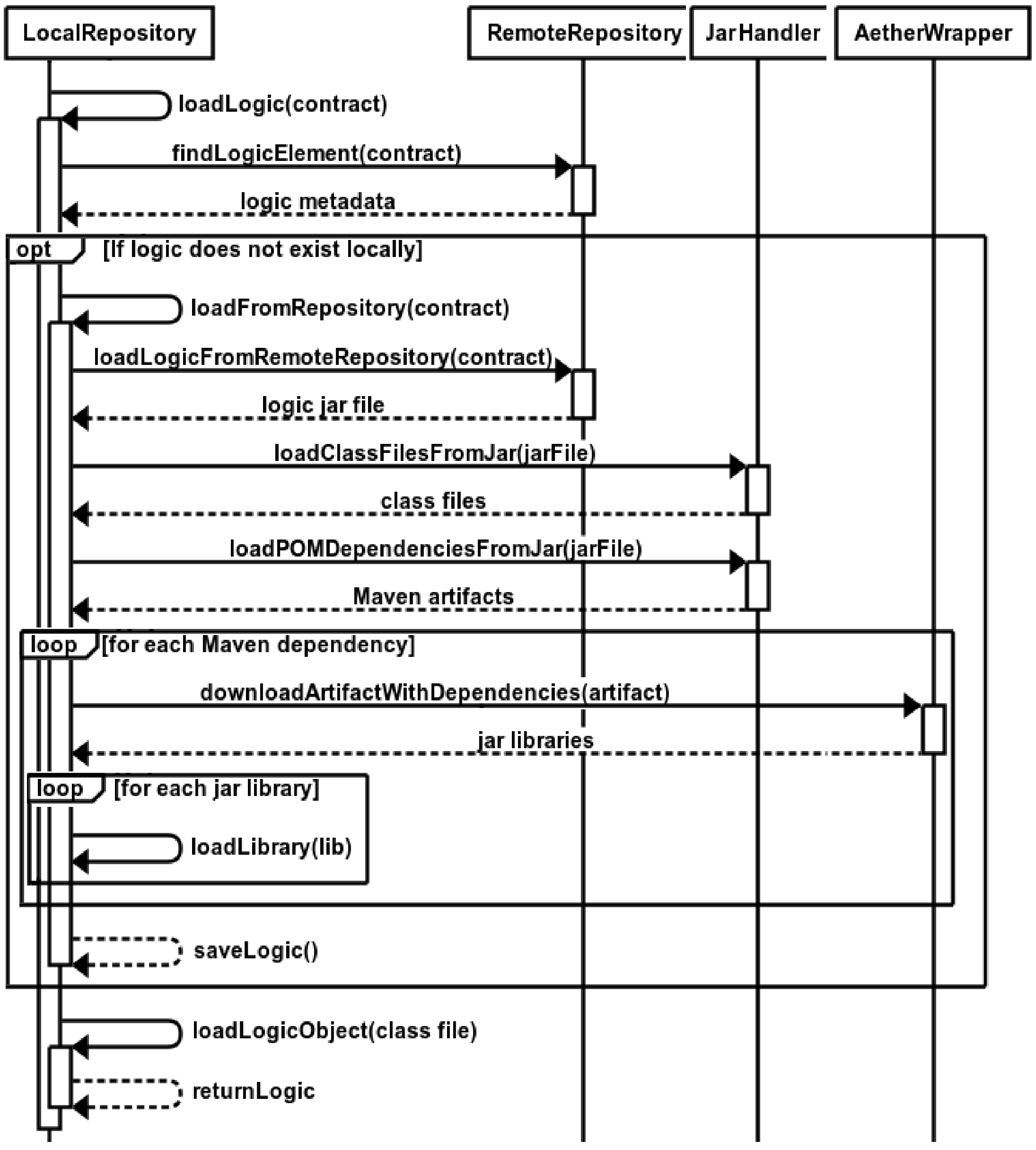

LocalRepository
RemoteRepository
JarHandler
AetherWrapper
loadLogic(contract)
findLogicElement(contract)
logic metadata
opt
[If logic does not exist locally]
loadFromRepository(contract)
loadLogicFromRemoteRepository(contract)
logic jar file
loadClassFilesFromJar(jarFile)
class files
loadPOMDependenciesFromJar(jarFile)
Maven artifacts
loop
[for each Maven dependency]
downloadArtifactWithDependencies(artifact)
jar libraries
loop
[for each jar library]
loadLibrary(lib)
saveLogic()
loadLogicObject(class file)
returnLogic

Figure 7.9.: The sequence diagram shows the process of dynamic class loading in the adaptation logic. This process involves the FESAS Repository, the Local Repository, and additional modules for handling the jar files in the FESAS Repository (*JarFileHandler*) as well as dynamic loading of jar files from the Maven repositories using *Apache Aether* (created using websequencediagrams.com).

Crucial for the deployment process is the analysis of the system model stored in JSON configuration files. These files are processed by the JSON Parser. It also integrates the *Gson*[9] library to read JSON data. After parsing the JSON configuration files, the JSON Parser returns a list with all MAPE elements and their configurations to the Middleware Starter. The Middleware Starter than initializes the adaptation logic using the SAS Setup Service.

The SAS Setup Service loads MAPE elements and configures the adaptation logic. The prototype's implementation of the SAS Setup Service implements the interface described in Section 6.1.4 (cf. Appendix B.8 for the interface definition). It receives the information for a MAPE element from the Middleware Starter and uses the reference system's implementation of MAPE elements based on the FESAS Adaptation Logic Template (cf. Section 6.1.1) to start each MAPE element. Additionally, using the Local Repository, it loads the functional logic and configures the MAPE element. After starting all MAPE elements, the SAS Setup Service sets up the communication between the MAPE elements. At runtime, the SAS Setup Service enables self-improvement through exchange of functional logics or change of the communication pattern between MAPE elements.

During deployment, the middleware configures the sensors for sensing information from managed resources. The prototype of the FESAS Middleware offers several communication modules for connecting managed resources and adaptation logic. Developers just need to specify the preferred communication approach and configure it using the FESAS IDE, i.e., no coding is necessary.

The first communication approach enables a push-based communication: The sensor can open a server socket and awaits data from the managed resources. Second, for pull-based communication, the sensor can start a client socket which is configured with IP addresses and ports of managed resources. The sensor uses this information and the socket to regularly request new data from managed resources. This approach can be used for effectors, too. Third, the adaptation logic might be implemented as web server. For the implementation of the web server, the *Socket.IO Java Server*[11] is used. This library enables to run a lightweight, stand-alone Java-based web server and supports the *WebSocket* protocol. Sensors use this functionality to open a WebSocket. After receiving a request from the managed resources, the connection is stored and the effector can use it for sending

[11]Socket.IO library on Github: `https://github.com/scalecube/socketio`

instructions back to the managed resources. These three approaches can be used out-of-the-box and are integrated into the reference systems (cf. Section 7.5).

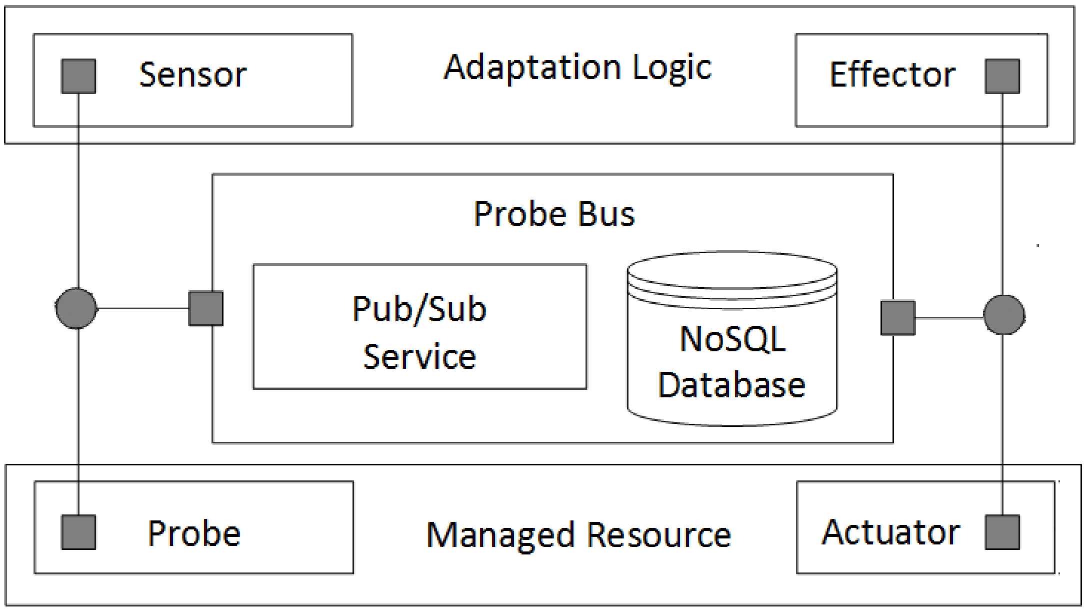


Figure 7.10.: Architecture of the probe bus: The interaction modules can be provided as library to developers. The prototype is implemented using the *ActiveMQ* JMS broker as Pub/Sub Service and *MongoDB* as NoSQL database.

Additionally, we experimented with the implementation of a probe bus (cf. the Probe/Gauge Reporting Bus from the Rainbow Framework [77,141]). Figure 7.10 shows the architecture of the probe bus. The implementation is based on the Java Message Service (JMS) API. Implementations of a JMS require a message server, the so called JMS broker. As the underlying technology for the JMS broker, the *ActiveMQ*[12] server is used. Data is stored in the NoSQL database *MongoDB*[13]. The system follows a topic-based approach as known from Pub/Sub systems (cf. [118]). Therefore, topics are defined based on the context feature space presented by Schmidt, Beigl, and Gellersen in [329]. Probes of the managed resources can send data to the JMS broker. The sensors of the adaptation logic can subscribe for specific topics. Vice versa, the effectors push data to the JMS broker, actuators of the managed resources can register for this data. Contrary to the former three communication approaches, this approach supports

[12]Apache ActiveMQ website: `http://activemq.apache.org/`
[13]MongoDB website: `https://www.mongodb.com/`

n:m communication, i.e., settings with a distributed adaptation logic where the information from the managed resources is shared between the instances of the adaptation logic. However, as the bus system requires a message server as well as a database, this approach is only suitable for larger systems and systems with at least one fully-fledged device that can host the JMS broker. The system is not fully integrated in the FESAS IDE. Therefore, it requires manual integration effort from the developers.

As described in Section 6.1.4, the context manager is responsible for storing the information captured by the sensors as well as offering an abstraction from the managed resources. The former presented JMS broker-based approach for connection of adaptation logic and managed resources could be used as context manager. It satisfies all functional requirements: (i) stores context data in a database, (ii) offers access to this data for all MAPE components, and (iii) provides a clear interface between the effectors and actuators for controlling the adaptation of the managed resources. However, as the approach requires a message server it is only suitable for systems that integrate fully-fledged devices. An alternative implementation is provided in the *PROACTIVE* framework [369]. It integrates the context broker [368, 370] for context management. The context broker offers a Pub/Sub approach for context management. We also built a prototype using the PROACTIVE context broker. However, for both approaches, a clear definition of the context data and the interfaces to the actuators are necessary. Such a definition can rely on context models (cf. [43, 329]). So far, for the FESAS Framework, the context model is not standardized. Accordingly, no reference implementation for the context manager is integrated.

For the SmartHighway system the components of the FESAS Middleware are used without any modification. The connection to the highway devices is established via sockets. The FESAS Middleware configures the sockets based on information defined in configuration files through the designer by using the *FESAS Design Tool*. The context information is managed implicitly by sending the relevant data from one MAPE component to another. This is possible as historical context data is not relevant in the SmartHighway scenario. Mapping of context information to sections and vice versa of instructions to sections is done using the information specified in the system model. There, a sensor and an effector is specified for each section.

7.3.2. A Reusable Publish/Subscribe Approach for Communication in the Adaptation Logic

In a SAS, there are four types of communication: within the adaptation logic, between adaptation logic and managed resources, within the managed resources, as well as between the managed resources and their environment, such as legacy systems. The former section presented the connection between adaptation logic and managed resources. In contrast, this section presents a reusable approach for communication within the adaptation logic. Parts of this section are taken from [228][14]. The connection between different managed resources and managed resources and the environment is out of scope of this work.

As the adaptation logic can be decentralized, the communication shall not be coupled to specific elements, but instead to the information itself (requires *space decoupling*). Furthermore, the different subsystems of the adaptation logic may have various MAPE loops. Their activities are not synchronized (requires *synchronization decoupling*) and have varying execution times due to different hardware, algorithms, and purposes (requires *time synchronization*). A Publish/Subscribe (Pub/Sub) approach offers decoupling from space, synchronization, and time, and is not restricted to a specific implementation approach or language [118]. Further, Pub/Sub approaches decouple sender and receiver. If a component wants to receive some specific data, it can register itself at the Pub/Sub service and becomes a *subscriber*. The registration bases on topics (predefined keywords), content (e.g., event properties), or event types and can be event-specific or related to a pattern of events [118]. When a component wants to make data available, it sends the data to the Pub/Sub service and becomes a *publisher*. A Pub/Sub service can be a central component or distributed.

The Pub/Sub service implements the interface specified in Listing 7.4. The interface offers the typical methods for a Pub/Sub system as proposed in literature, e.g., [118]. These are methods for subscribing, deleting existing subscriptions, publishing data, and notifying the availability of new data. For supporting the distributed nature of the Pub/Sub system, the interface offers additionally methods for subscription at remote elements of the Pub/Sub system. In the following, this section describes the implementation of the Pub/Sub system.

[14] [228] is joint work with F. M. Roth, S. VanSyckel, and C. Becker.

```
{
public interface IPubSub {
  public void subscribe(eventCategory, publisher, subscriberID);
  public void subscribe(eventCategory, subscriberID);
  public void subscribe(eventType, publisher, subscriberID);
  public void subscribe(eventType, subscriberID);
  public void subscribeExternal(eventCategory, subscriber);
  public void subscribeExternal(eventType, subscriber);
  public void unsubscribe(eventCategory, publisher,
    subscriberID);
  public void unsubscribe(eventCategory, subscriberID);
  public void unsubscribe(eventType, publisher, subscriberID);
  public void unsubscribe(eventType, subscriberID);
  public void unsubscribeExternal(eventCategory, subscriber);
  public void unsubscribeExternal(eventType, subscriber);
  public void publishEvent(event);
  public void notify(event, publishingSystem);
}
```

Listing 7.4: Interface for the Pub/Sub system. The interface is written in the Java syntax and offers the typical methods for Pub/Sub systems.

In our approach, components register based on *information categories* and *information types*. Categories relate to MAPE component types according to a topic-based Pub/Sub approach [118]. Additionally, use-case dependent information types can be used. Furthermore, our approach divides between registration of events of a specific system (compared to event-specific registration) vs. registration for all systems (event patterns). Besides other properties, the event has a `knowledge ID` which enables subscribers to load data that is related to the event. Subscribers may first receive the event and request relevant data with this `knowledge ID`. The process of registration and publication of events is shown in Figure 7.11. In the following, we discuss subscription and publication of events.

The degree of domain and deployment knowledge influences the design of the Pub/Sub system (cf. Table 7.1). On the one hand, having more deployment knowledge enables a fine-granular registration which avoids message overhead as

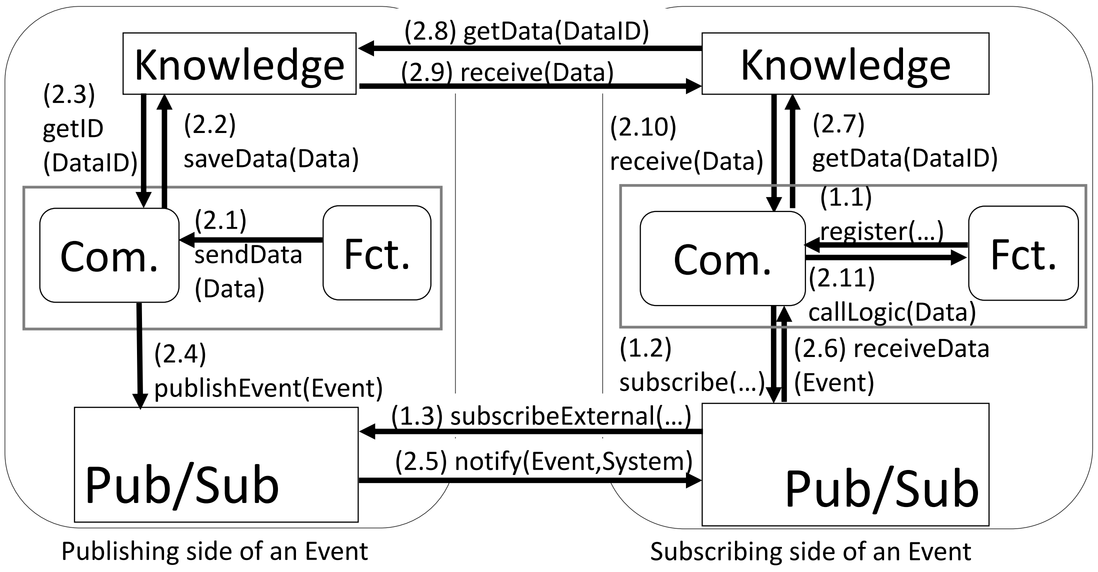


Figure 7.11.: The reusable Publish/Subscribe system enables communication between MAPE elements. The split of event data and knowledge, i.e., the data itself, reduces communication workload. Com. = Communication Logic, Fct. = Functional Logic.

the relevance of received events is increased but is more static. On the other hand, integrating more domain knowledge reduces message overhead, too, but leads to more customization and reduces reusability. In the following, this section describes the Pub/Sub system.

		Domain knowledge	
		Yes	**No**
Deployment knowledge	**Yes**	Registration for specific event types of specific instance	Registration for general event categories of specific types
	No	Registration for specific event types	Registration for general event categories

Table 7.1.: Trade-off between deployment knowledge and domain knowledge for the topic registration at the Pub/Sub service when choosing between registration for event types vs. event categories and between specific systems or all systems.

Subscribing for events: The functional logic of a component specifies which data it requires (cf. step *(1.1)* in Figure 7.11). The communication logic uses the appropriate `subscribe()` method *(1.2)*, depending on whether it uses the information type or category and whether the scope of the registration is one specific subsystem or the entire system. The Pub/Sub service adds the subscriber to its list for the specified events. If the subscription is for a local component, the subscription is completed at this stage. For a remote component, the local Pub/Sub service subscribes for relevant events at the remote Pub/Sub service(s) of the corresponding subsystem(s) or all systems *(1.3)*. Unsubscribing has the same workflow like subscription, hence, it is omitted here.

Publishing an event: If a functional logic calls the `sendData()` procedure *(2.1)*, the communication logic first stores the data *(2.2/2.3)*, and, then calls the `publishEvent()` procedure *(2.4)*. The Pub/Sub service notifies all other Pub/Sub services that have subscribed for the event *(2.5)* and transmits the event. After receiving an event, the Pub/Sub service hands the information to the subscribed components *(2.6)* which use the communication logic for reading the data. If the event is relevant for a component, the communication logic loads the event's data *(2.7 - 2.10)* and starts the `callLogic()` procedure with the data *(2.11)*.

The FESAS Middleware incorporates two implementations of the Pub/Sub system: one is based on local methods calls for a centralized adaptation logic, the other uses the BASE middleware [34] for communication across sub-systems of the adaptation logic. In literature, different alternatives to the Pub/Sub approach can be found. Closest related to the field, Sykes, Magee, and Kramer presented *FlashMob*, which offers a gossip-based protocol for supporting the information exchange for decentralized decision making in SASs. Additionally, [118] discusses different alternative communication paradigms.

7.4. Implementation of the Adaptation Logic Manager

Based on the design presented in Section 6.2, prototypes for the ALM have been implemented. In [223], we present a first version of the ALM prototype. This version adds a layer on top of the adaptation logic according to the design presented in Section 6.2. Additionally, we implemented three modules for planning structural and parametric self-improvement and evaluated them in the SmartHighway scenario. This first prototype and the modules were further abstracted from the use cases, hence, in the current version, they are reusable. Based on this reusable version, we present a single module version of the ALM in [230].

This modularized ALM targets smaller SASs, where in contrast to the layered version, the ALM does not need to coordinate different parts of the adaptation logic running on various systems. We used this version in combination with a customized planner module in a fall detection system to adapt the fall detection algorithm according to the position of the fall detection device. However, as both variants of the ALM – the layered ALM and the single module ALM – base on the same implementation, the single module ALM is not described in detail.

This chapter describes the implementation of the ALM prototype as well as the three modules for planning structural and parametric self-improvement. First, Section 7.4.1 presents the implementation of the ALM according to the design presented in Section 6.2. Second, Section 7.4.2 explains the implementation of the three modules for planning structural and parametric self-improvement. A detailed discussion of the ALM's performance in the SmartHighway scenario follows in the evaluation part of this thesis (cf. Section 8.3.2). Parts of this section are based on [223][15].

7.4.1. Adaptation Logic Manager: Reference System

The implementation of the ALM follows the design presented in Section 6.2. Hence, the ALM is implemented as an additional layer on top of the adaptation logic and integrates a MAPE-K feedback loop structure [198]. The prototype of the ALM is implemented using Java 8. In the following, this section presents the implementation of the ALM's MAPE components.

Figure 7.12 presents the registration process of a SAS. The ALM has an `RegistrationHandler` component that continuously listens for registration requests from a Proxy ALM `RegistrationHandler`. After receiving a request for registration, the information is stored in the `ALMRegistry`. The connection can be established with WebSockets, Java RMI, or method calls.

The ALM Sensor loads the registered Proxy ALMs from the `ALMRegistry` when it senses information. The ALM collects the following data from the registered Proxy ALMs: (i) the structure of the adaptation logic, (ii) the properties of the MAPE components as well as (iii) the data that the adaptation logic collected

[15] [223] is joint work with J. Otto, F. M. Roth, A. Frömmgen, and C. Becker.

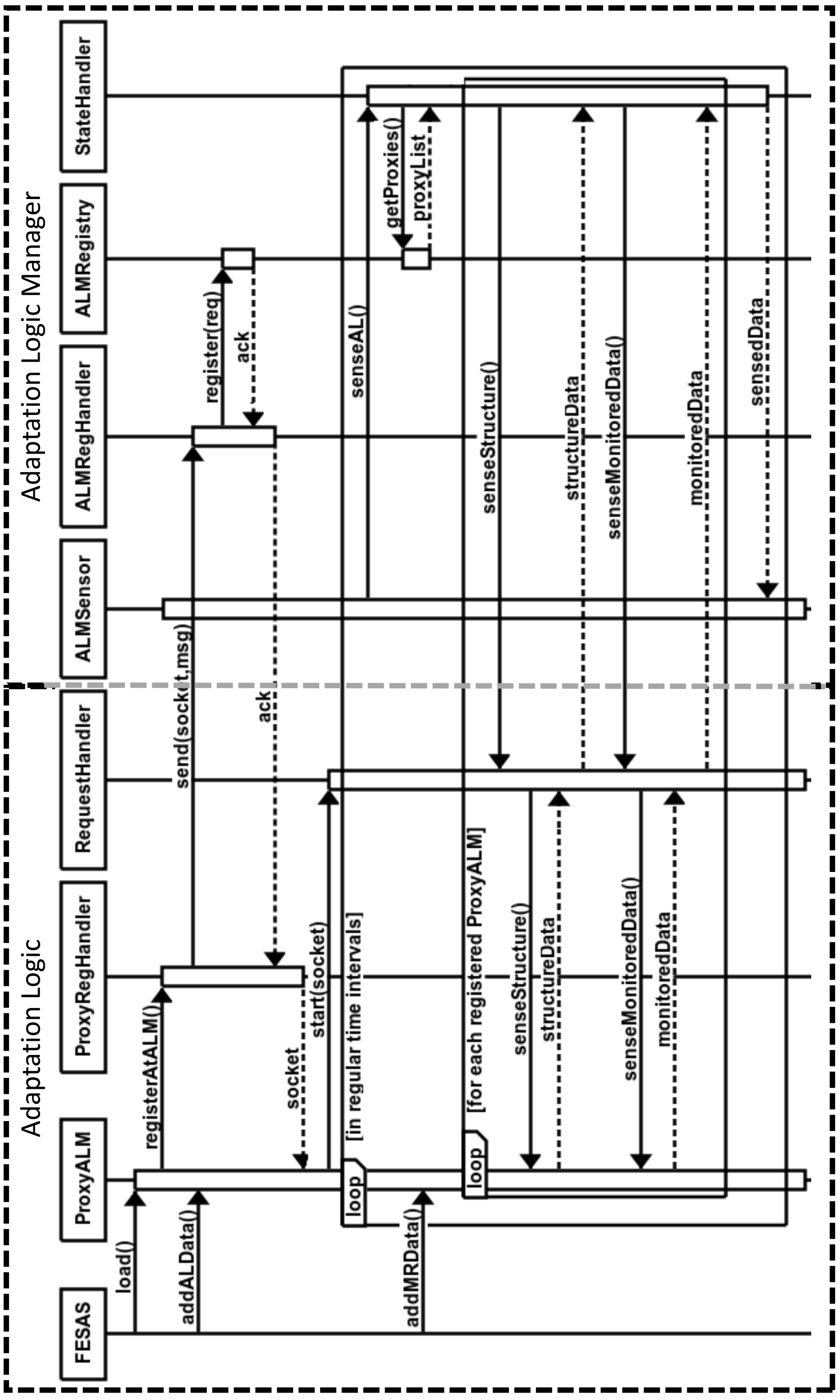


Figure 7.12.: The diagram shows the interaction between ALM and Proxy ALM for the registration of a subsystem of the adaptation logic as well as a request for sensing information. The connection between the components of the adaptation logic and Proxy ALM is abstracted, only the start of the Proxy ALM and two updates of information are shown (created using websequencediagrams.com).

```
{
    "domainNodes": [{
        "fesasID": "streetSection1",
        "name": "Street section",
        "type": "DOMAIN",
        "properties": [{
            "key": "roadCondition",
            "value": "good"
        },... // code removed
        }]
    },... // code removed
    }],
    "domainRelations": [{
        "from": "streetSection1",
        "to": "fesasID-123_1_008",
        "type": "directed",
        "labels": ["ManagedBy"],
        "properties": [{
            "key": "utilization",
            "value": "0.9"
        }]
    }... // code removed
}
```

Listing 7.5: Example JSON description of the domain knowledge for the SmartHighway scenario from [223]. The ALM appends this information to the `FESASGraph`.

from the managed resources. For each type of data, the ALM has one ALM Sensor that periodically connects to the registered Proxy ALMs. The ALM Monitor preprocesses the data from the adaptation logic and stores the data collected from managed resources into a *MongoDB*[16] NoSQL database. Later, the data can be used for prediction, analysis, and planning self-improvement. Through specified service interfaces, developers can access the data. Further, the ALM Monitor builds a graph of the adaptation logic's structure – the so called `FESASGraph` – that can be used for reasoning, e.g., for structural meta-adaptation of the adaptation logic. The *GraphStream*[17] library was selected for the implementation of the `FESASGraph`. Using the Observer pattern [137], it is possible to register observers

[16]MongoDB website: `https://www.mongodb.com/`
[17]GraphStream website: `http://graphstream-project.org/`

that are informed when the graph changes. This can trigger an update of data in registered analyzing or planning modules. Using this, developers can access data about the adaptation logic's structure and use this data for reasoning. Consequently, developers do not have to implement the monitoring of the adaptation logic and the managed resources.

As analysis may require the integration of domain-related information into the `FESASGraph`, we implemented the `DomainKnowledgeAppender` following the Decorator pattern [137]. It reads information from a JSON file which specifies additional elements and connections and appends it to the graph. Therefore, only the path to the file has to be defined. Furthermore, this approach supports the separation of development and design tasks as a designer without implementation experience can specify the file. Listing 7.5 shows an example of such a JSON file for the SmartHighway scenario which adds information about the street conditions.

The ALM Analyzer integrates a prediction module that offers different operations for analyzing the data collected from the managed resources based on the *Waikato Environment for Knowledge Analysis* (WEKA) [185] machine learning framework. The prediction module is a generic component. Through WEKA, a large set of regression learners can be used in the prediction module, such as Bayesian Networks, Multilayer Perception/Artificial Neural Networks, or Support Vector Machines for Regression. Developers use the prediction through minor configuration of specified parameters, such as the amount of considered data items, specifying the machine learning algorithm, or the amount of forecast steps. For the SmartHighway system [223], we implemented time-series forecasts based on Support Vector Machines for Regression in the data prediction module for prediction of traffic situations. Whereas the prediction can be used out of the box, minor customization for interpreting the results is necessary. Our approach supports the developer through encapsulation of the generic elements, clearly defined interfaces to the customization parts, and documentation on how to use the prediction module. Additionally, implementing specific interfaces, developers can implement analyzing modules that use the prediction module and provide a use case specific analysis of the data. The ALM Analyzer informs the ALM Planner about the analysis results.

The ALM Planner decides which improvement modules should run. Figure 7.13 shows the interfaces. Each of the planning modules implements at least one of the interfaces for parameter, structure, or algorithm meta-adaptation. The interface `IRuntimeOptimization` enables the support of multiple types of meta-adaptation. This interface defines functions that all meta-adaptation techniques must provide: besides the method `getName()`, the methods `getExpectedUtiliy()` and `getExecutionRank()` are required for the implementation of the different execution strategies: predefined order, shortest execution time, and utility-based (cf. Section 6.2). Apart from that, the interface extends the Java interface `Callable` to support multi-threading for parallel processing of ALM Planner modules. The ALM Planner starts the modules and waits for the results based on the defined execution strategy.

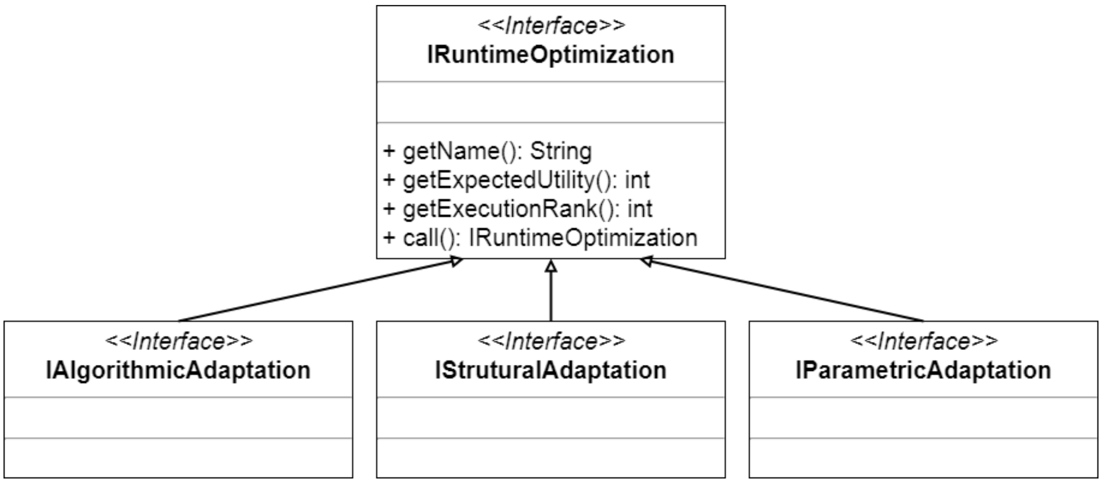


Figure 7.13.: Interface hierarchy of runtime optimizations. For each type of meta-adaptation – algorithmic, structural, and parametric – one interface exist. All extend the generic `IRuntimeOptimization` interface. Attributes and specific methods of the extended interfaces are omitted.

In [223], we focused on structural self-improvement mapped to a graph structure of the adaptation logic. Therefore, every specific implementation of structural self-improvement has to be able to: (i) generate new and identify obsolete connections, (ii) return new and obsolete connections, and (iii) serialize new and obsolete connections. The ALM offers implementations of the five patterns from [401] in form of abstract classes that implement the interface `IStructuralAdaptation` in order to provide as much reusable code as possible.

As an example, for adapting a *Regional Planning* pattern or switching from another pattern to it, every structural self-improvement implementing this pattern has to identify a regional planner, connected analyzers, connected executors, and obsolete planners. Moreover, dispensable planners have to be identified. Listing 7.6 contains the code of the method for generating new connections. The methods automatically generates the list of new connections based on the information, which nodes are the regional planners, their new connected analyzers, and executors. The connections from analyzers to the new regional planner are generated in lines 4-10 while connections from the regional planner to executors are generated in lines 12-18 of Listing 7.6. Similarly, the method `generateObsoleteConnections()` processes obsolete connections but is not discussed in detail.

```
1  public void generateNewConnections() {
2    Iterator<Node> iterator;
3
4    iterator = connectedAnalyzers.iterator();
5    while (iterator.hasNext()) {
6      Node node = (Node) iterator.next();
7      newConnections.put(node, regionalPlanner);
8      newConnectionsSerializable.put(node.getAttribute("fesasID").
           toString(),
9          regionalPlanner.getAttribute("fesasID").toString());
10   }
11
12   iterator = connectedExecutors.iterator();
13   while (iterator.hasNext()) {
14     Node node = (Node) iterator.next();
15     newConnections.put(regionalPlanner, node);
16     newConnectionsSerializable.put(regionalPlanner.getAttribute("
           fesasID").toString(),
17         node.getAttribute("fesasID").toString());
18   }
19 }
```

Listing 7.6: Method for generating new connections in the class `AbstractRegionalPlannerPattern`.

In case the ALM Planner has found an adaptation plan for self-improving the adaptation logic, it sends the plan to the ALM Executor. For the sake of demonstrating the complete approach, in the following a structural self-improvement is presented. First, the ALM Executor checks the type of meta-adaptation specified in the plan in order to call different subsequent operations depending on the type of meta-adaptation. In case of structural meta-adaptations, the method `executeStructuralAdaptation()` is called. It expects an input parameter of the type `StructuralAdaptationData` (cf. Listing 7.7). This method is responsible for creating a JSON string that includes the instructions which are sent to the Proxy ALM for the implementation of self-improvement. In case of structural meta-adaptations it consists of which connections are to add or remove in order to create the target structure of the system. Lines 4-28 of Listing 7.7 contain exemplary code for the creation of instructions for adding new connections. Furthermore, the changes are also implemented in the `FESASGraph` to keep a current state of the systems structure within the ALM. Using socket-based connections, the ALM Effector sends these instructions to the corresponding Proxy ALMs. A Proxy ALM uses the SAS Setup Service (cf. Section 6.1.4) for triggering the changes in the adaptation logic, such as switching the coordination pattern or changing parameters of MAPE components.

All of the described components are integrated into the reference system (cf. Section 7.5). The FESAS Framework directly supports the distribution of the MAPE components of the ALM, fulfilling requirement $R_{SI}5$. Figure 7.14 shows an overview on all elements of the ALM prototype. The main focus on the implementation is the connection between the adaptation logic and the ALM as well as the provision of data for reasoning on self-improvement. Therefore, all the gray elements from Figure 7.14 can be used without any adjustments. These elements trigger the sensing of information from the adaptation logic subsystems about their structure, components, as well as the captured data from managed resources. The sensed information is stored in the `FESASGraph` and the MongoDB database. Clear interfaces enables developers of self-improvement reasoning modules to access this data for analyzing and planning self-improvement. The module for prediction on the data about the managed resources can be easily configured and the results are integrable into the developed analyzing modules. Only the storing of data collected about the managed resources and the reasoning process require customization. In the following, this section describes the provided support for the necessary customizations.

```java
private void executeStructuralAdaptation(StructuralAdaptationData structuralAdaptation) {

  [...] // code removed
  newConnections = structuralAdaptation.getNewConnections();
  obsoleteConnections = structuralAdaptation.getObsoleteConnections();
  Iterator<Entry<String, Collection<String>>> iterator = newConnections.entrySet().iterator();
  Iterator<String> innerIterator;
  [...] // code removed

  while (iterator.hasNext()) {
    Entry<String, Collection<String>> newConnection = iterator.next();
    from = FesasGraph.getInstance().getNode(newConnection.getKey().toString());
    targets = (List<String>) newConnection.getValue();

    innerIterator = targets.iterator();
    while (innerIterator.hasNext()) {
      String target = (String) innerIterator.next();
      to = FesasGraph.getInstance().getNode(target);

      String system = from.getId().substring(16, 17);
      String almProxyX = to.getId().substring(8, 11) + "0000";

      [...] // code removed
      instructions.add(wire(almProxyX, connectionType, to.getAttribute("fesasID").toString(), from.getAttribute("fesasID").toString(), InformationType.Analyzing_TRAFFIC, InformationCategory.ANALYZER));

      [...] // code removed
      FesasGraph.getInstance().addEdge(from.getId() + "-TO-" + to.getId(), from, to);
    }
  }
  [...] // code removed
}
```

Listing 7.7: An example of the method for generic execution of structural meta-adaptations in the ALM Executor.

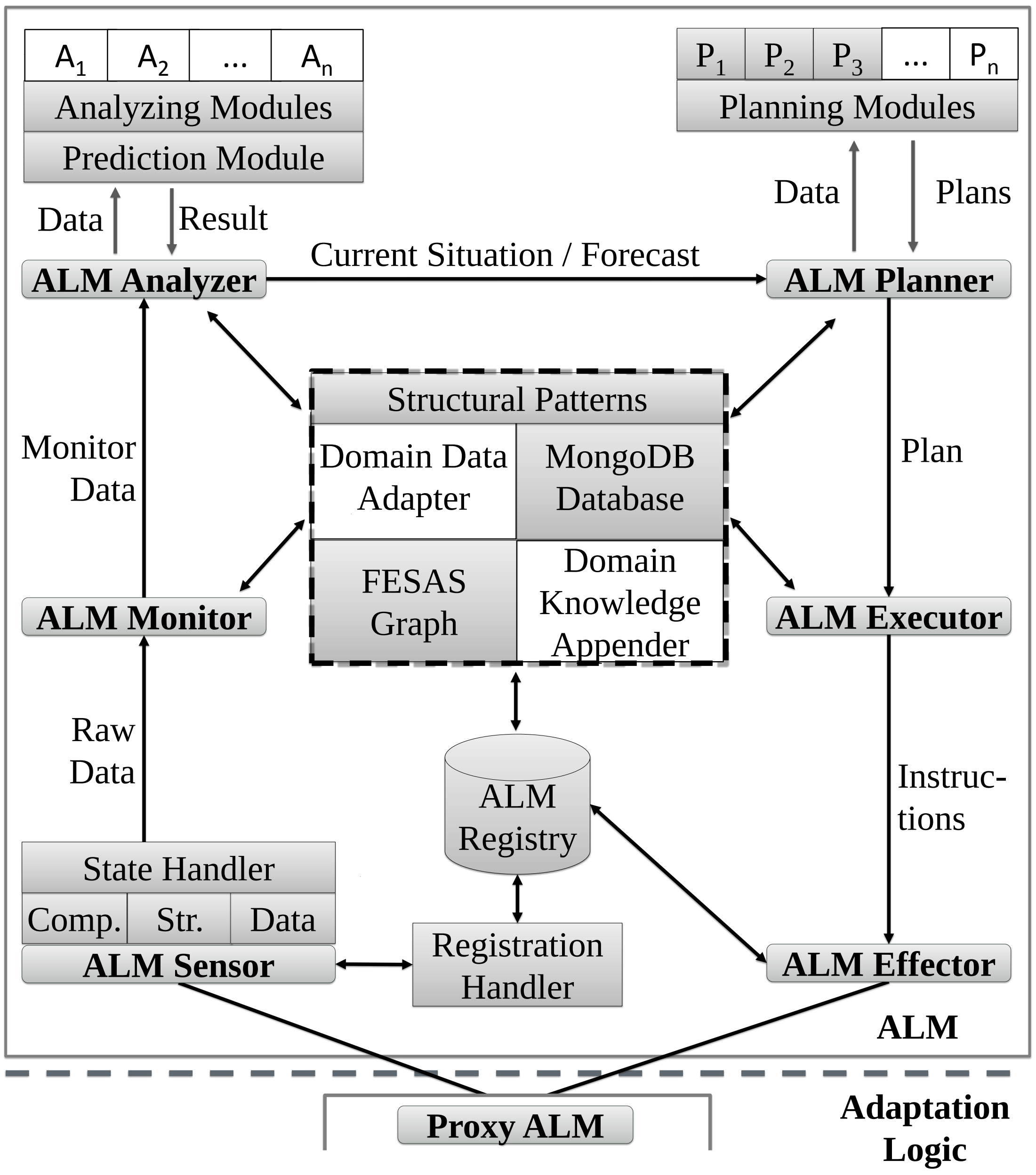


Figure 7.14.: Overview on the implemented components of the ALM. The ALM prototype presented in this section provides all gray elements. White elements need to be implemented or adjusted by the developers. The three modules for the planning process presented in Section 7.4.2 are available and generically usable. The modules of the SmartHighway system for analyzing require adjustments. The complex in the middle of the ALM – `FESASGraph`, `DomainKnowledgeAppender`, `DataAdapter`, `MongoDBDatabase`, and `StructuralPatterns` – act as knowledge repository.

The captured data of the managed resources depends on the use case. Therefore, a flexible solution for storing data is necessary. The non-SQL MongoDB database enables this flexibility. In theory, developers can store the data as captured. For a more structured approach, the `IDataAdapter` interface can be extended (cf. Appendix C.2). A reference implementation of such an extension shows the necessary steps for developers: definition of a pattern for parsing data and mapping the parsed information to JSON. This minor customization enables the storing of any kind of captured information adjusted to the use case.

The data of the adaptation logic is stored automatically in the graph-based structure of the `FESASGraph`. Developers do neither need to take care of the storing nor the capturing processes if they use the FESAS Framework. Further, they can access this data through specified methods. If necessary, the data can be enriched with use case specific information of the managed resources. Therefore, a JSON file for the `DomainKnowledgeAppender` has to be specified as described above. For future work, we plan to extract the information automatically with the FESAS Design Tool.

As analyzing and planning of self-improvement is often application-specific [226], developers might need to adjust existing modules or implement new ones and plug them into the ALM Analyzer or the ALM Planner, respectively. Therefore, the prototype implementation of the ALM offers a wide support for the developers. For the analyzing procedure, developers can use the prediction module. Further, the access to the automatically captured data of the adaptation logic and the managed resources is possible. The combination of different analyzing modules support the integration of a backup mechanism if proactive reasoning fails. The same is true for planning of self-improvement: Through the strategies, the ALM Planner can manage different planning modules simultaneously. The necessary adjustments on existing planning modules are documented. Further, various patterns for planning of structural self-improvement exist. The following section presents three reusable planning modules for self-improvement [223].

Execution of self-improvement is simplified through the integration of the FESAS Framework as it implements the interfaces for meta-adaption. However, the ALM can be integrated with other frameworks for developing SASs that implement the ALM protocol (cf. Appendix C.1) for communication between the Proxy ALM and the ALM.

7.4.2. Adaptation Logic Manager: Planning Modules

In the following, this section describes the implementation of three planning modules used for self-improvement in the SmartHighway system [223]. These modules have been further standardized. They offer (i) dynamic parametric self-improvement through rule learning, (ii) structural self-improvement based on a fixed rule set, and (iii) dynamic model-based structural self-improvement. For all three approaches, this section highlights the generic reusable parts and the parts that require customization. Alternatively, it is possible to use related work, e.g., the strategies for meta-adaptation presented in [149], the mechanism for generating adaptation paths to achieve the desired target system configuration from [304], or one of the other approaches analyzed in Section 4.3.

7.4.2.1. Parametric Self-improvement

The first module offers continuous meta-adaptation of the adaptation logic's Event-Condition-Action (ECA) rule set. Such a rule consists of events specified by a set of conditions and a set of corresponding actions for adapting the managed resources. These actions are applied if the conditions are met by the managed resources or the environment variables. Having found a matching rule, the planner returns the corresponding action(s).

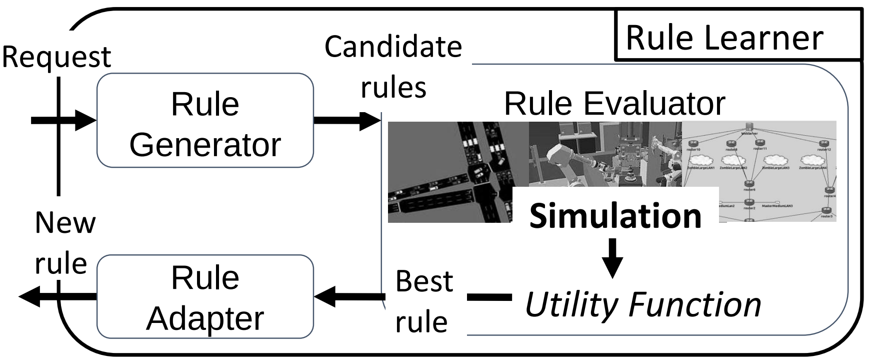


Figure 7.15.: The module for learning rules generates and evaluates new rules. If the rule learner finds an optimized rule, the module for parametric self-improvement adapts the rule set in the adaptation logic.

The ALM is able to change the adaptation logic's rule set, i.e., adding, removing, or updating rules. Figure 7.15 presents the module for rule learning. The learning module integrates a simulation of the managed resources. The simulation is constantly started with new parameter combinations that represent the properties of managed resources and rules for adaptation. The module uses the simulations to retrieve the reaction of the managed resources to the adaptation specified by the rules. As a next step, a utility function is applied to the collected data of the simulation runs. The function assigns a utility value to each simulation run, i.e., to each combination of condition and action of a rule candidate. Then, the module analyzes which rule is the best for a specific situation by comparing the utility values of the runs.

```java
public ALMData changeRulebase(ALMData request) {

  if (request.getPropertyValue("action").equals(ALMProperty.
     ADD_RULE)) {
    addRule(request.getPropertyValue("element"), request.
       getPropertyValue("rule"));
  }else if (request.getPropertyValue("action").equals(ALMProperty.
     REMOVE_RULE)) {
    removeRule(request.getPropertyValue("element"), request.
       getPropertyValue("rule"));
  }else {
    return new ALMData(ALMMsgType.ERR, ALMCommand.LOG);
  }

  return new ALMData(ALMMsgType.RESP, ALMCommand.RUL);
}

private void addRule(String ALObject, String rule) {
  [...] // code removed
}

private void removeRule(String ALObject, String rule) {
  [...] // code removed
}
```

Listing 7.8: Implementation of the method for rule meta-adaptation in the Proxy ALM.

The ALM Planner triggers an update of the rules in the adaptation logic. The FESAS Framework integrates a mechanism for exchanging rules. Listing 7.8

provides the implementation of this mechanism in the Proxy ALM. The ALM uses this mechanism for modifying the rule set. It is possible to use other frameworks for the adaptation logic if they offer a rule change mechanism.

The simulator and the utility function are application-specific. Through modularization, it is possible to exchange the simulator and the utility function. Consequently, the mechanisms for learning can be reused by integrating another simulator and defining the relevant parameters for learning and the utility function. This allows to customize the learning module for a specific application.

In [223], we presented a rule learner for the SmartHighway scenario. This learning module calculates the speed limit for a specified traffic flow using the SUMO traffic simulation[18]. As learning parameter, different speed limits for a specific traffic flow are applied. The module uses the following utility function for finding the best rule for a traffic situation:

$$U_{Highway} := \alpha - \left(\beta * \sum wait + \sum lostTime\right) \tag{7.4.2.1}$$

$\sum wait$ is the accumulated number of simulation steps where the speed of a vehicle was below 0.1 m/s, $\sum lostTime$ is the accumulated number of seconds that each vehicle lost due to driving below its target speed. These values are logged by the SUMO simulation for every vehicle. Simulation runs with different weighting factor β lead to a weighting factor of 7 for a balanced utility function. However, for having a correlation between a large result and a high utility, the function subtracts the sum of the two parts from a constant factor α. Simulation runs showed that the sum does not exceed 1,000 for the track used in the simulation, therefore, we chose 1,000 for the factor α. It may vary for other tracks. The module then uses the result with the highest utility and constructs a new rule consisting of the average flow per lane as the condition and the calculated speed limit as the action. Listing 7.9 shows an example for such a rule. This rule states that if the amount of vehicles exceeds 600 vehicles/hour the speed limit is set to 100 km/h (33.33 m/s). Section 8.3.2 presents the evaluation of the module.

[18]SUMO's website: `http://www.dlr.de/ts/sumo/en/`

```xml
<rules>
  <condition>
    <entry>
      <key>flow</key>
      <value>600.0</value>
    </entry>
  </condition>
  <action>
    <entry>
      <key>speedLimit</key>
      <value>33.333</value>
    </entry>
  </action>
  <attributes />
</rules>
```

Listing 7.9: An example for a rule in the SmartHighway scenario from [223].

7.4.2.2. Structural Self-improvement

The structure of the adaptation logic and the managed resources can be modeled as a graph, i.e., changes in the adaptation logic and managed resources are reflected as changes in the graph. We implemented two graph-based modules for structural self-improvement. Both modules integrate a graph-based representation of the SAS structure with rule-based and model-based planning for structural self-improvement. They specify when to switch the coordination pattern of the adaptation logic as shown in Figure 7.16. There, the approaches have triggered a switch from a fully decentralized non-coordinated setting to a hybrid *Master/Slave* pattern (cf. [401]). Next, this section presents a graph-based structural self-improvement approach using a static rule set and the *Topology Adaptation Rule Language* (TARL) [345]. Afterwards, this section introduces a variant using a dynamic rule set implemented with the *Neo4j* graph database.

The TARL Module

TARL [345] offers a general topology adaptation model. A TARL rule consists of a condition specification and an execution specification. The condition contains (potentially multiple) graph patterns and matches those on the input graph. The corresponding execution statement specifies the changes to the input graph. If a condition matches, the corresponding execution statement is triggered. Contrary to the dynamic rule learner, for structural meta-adaptation of the adaptation logic, the current state of the adaptation logic (e.g., its distribution) is relevant. Therefore, TARL registers as an observer for the `FESASGraph` to receive updates when the adaptation logic changes. TARL compares the graph with the patterns that are defined in the TARL rules. The patterns integrate the current or predicted traffic situation. If TARL finds corresponding patterns, it returns the associated execution parts of the matching rules. The execution statements contain adaptation actions. The functionality of the module is generically usable and, hence, can be used in different applications. However, developers have to adjust the module by specifying patterns and writing corresponding TARL rules. A listing showing the implementation can be found in Appendix C.3. Appendix C.4 provides an example of a TARL rule from the SmartHighway system [223].

The Neo4j Module

We implemented a second module for structural improvement using the Neo4j[19] graph database (called Neo4j module in the following). When the ALM Planner triggers the Neo4j module, the module generates queries for the Neo4j graph database using pre-defined but customizable rule templates. The Neo4j module registers as observer at the `FESASGraph`, hence, it gets informed about the current structure of adaptation logic and uses the information of the `FESASGraph`, the data captured from the managed resources as well as the predicted data of the ALM analyzer to build up the query. These queries change the graph of the adaptation logic structure stored in the Neo4j graph database. The Neo4j module

[19]Neo4j website: `https://neo4j.com/`

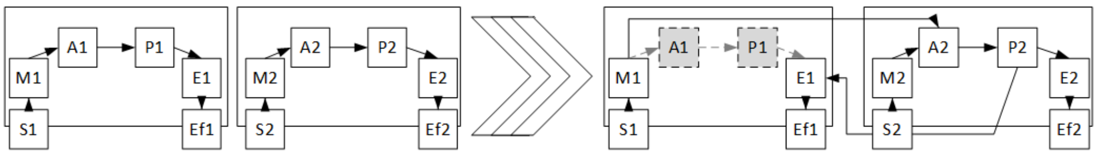


Figure 7.16.: The figure shows structural self-improvement of a distributed adaptation logic. The distribution pattern switches from a fully decentralized pattern to a hybrid *Master/Slave* pattern.

then compares the current status of the adaptation logic stored in the `FESASGraph` with the version in the database after performing the queries. Differences trigger self-improvement for adjusting the adaptation logic structure.

The methods for querying the database as well as the comparison of the graph structures are generically usable. Developers have to adjust the queries. Listing 7.10 shows the query for switching the current pattern to a *Regional Planning* pattern in the SmartHighway scenario. At first, a highway section which includes the attribute `jam=true` has to be found. This is realized by matching a node `s` labeled as `:DOMAIN` for which `where s.jam=true` applies. In case a match exists, the corresponding node is stored in the variable `s`. Afterwards, the regional planner has to be determined. The most posterior section with a jam on the highway becomes the regional planner. Once the query is defined, *Neo4j* tries to find a match on the current graph. In case a situation is present in which a jam section exists and the other parts of the query also apply, the result has to be processed. Our system uses this module for structural meta-adaptation of the adaptation logic. However, the modules can integrate other types of meta-adaptation or combinations thereof. Section 8.3.2 presents the evaluation of the presented modules. A listing showing the implementation can be found in Appendix C.5.

```
try (Transaction tx = graphDb.beginTx()) {
  result = graphDb.execute("match (x)-[*1..3]->(c:CONTEXTMANAGER)
      -[*2]->(t:DOMAIN)-[*1..6]->(s:DOMAIN)<-[*4]-(p:PLANNER)
      where s.jam=\"true\" and ((x:ANALYZER)OR(x:EXECUTOR))
      return distinct p,x");
```

Listing 7.10: Neo4j-based implementation of a structural self-improvement showing the creation of the query for the Neo4j database.

7.5. Implementation of Reference Systems

Beside of the middleware components and the FESAS Repository, an implementation of the MAPE components of the FESAS Adaptation Logic Template (cf. Section 6.1.1) for the adaptation logic is necessary for using the FESAS Framework. Different reference implementations of a SAS are available. The systems are implemented in Java and offer implementations of the SAS

Setup Service as well as components for the MAPE elements based on the FESAS Component Template (cf. Section 6.1.1). Further, these reference systems integrate the FESAS Middleware and the Pub/Sub system as well as support the implementation of centralized and distributed adaptation logics. Additionally, they are used as foundation for the ALM. In the following, this section presents the reference systems and reusable functional logics for MAPE elements.

7.5.1. Reference Systems

Two variants are available for implementing a SAS with a central adaptation logic. The first reference system can be used for a central, non-distributed adaptation logic. Interactions between MAPE components are based on Java method calls. The second reference system integrates an http server. It can be used for implementing client-server based SASs which use WebSockets for the connection between managed resources and adaptation logic, e.g., for connecting Android devices as managed resources. Both reference systems can also be used as base for implementing the layered version of the ALM.

A third reference system targets the implementation of distributed adaptation logics. This reference system uses the *BASE middleware* [34] for data transmission via the network. BASE has been tailored towards the requirements of pervasive environments with many heterogeneous devices. Accordingly, the MAPE elements are implemented as BASE services. Further, the Pub/Sub system is adjusted to integrate the BASE communication mechanisms.

All of these reference systems are complemented by so called wrappers. Figure 7.17 shows its structure. Similar to the composition pattern [137], they integrate adaptation logic and middleware components, trigger the start of the reference system, and store the configuration files including the system models and specific configuration properties. Additionally, in case of using an internal FESAS Repository, these wrappers also store the functional logics for the SAS. Wrappers and reference systems can be used for implementing an adaptation logic or the ALM. For the *SmartHighway* scenario, the adaptation logic uses the wrapper for the decentralized referenced system as communication across systems is necessary after a structural self-improvement. Further, the ALM for the system uses the wrapper with the central reference system.

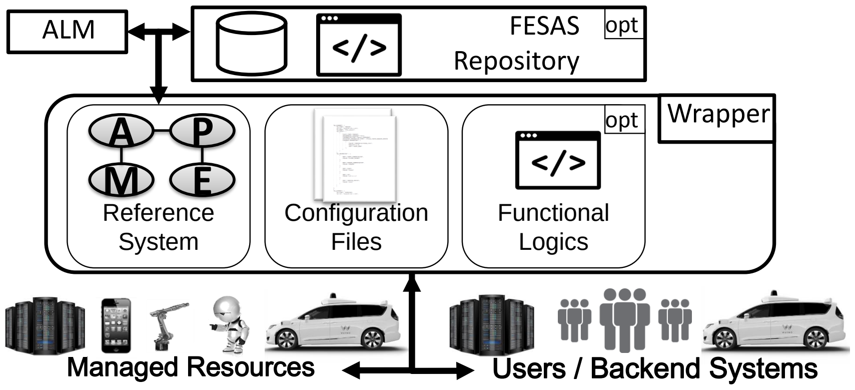
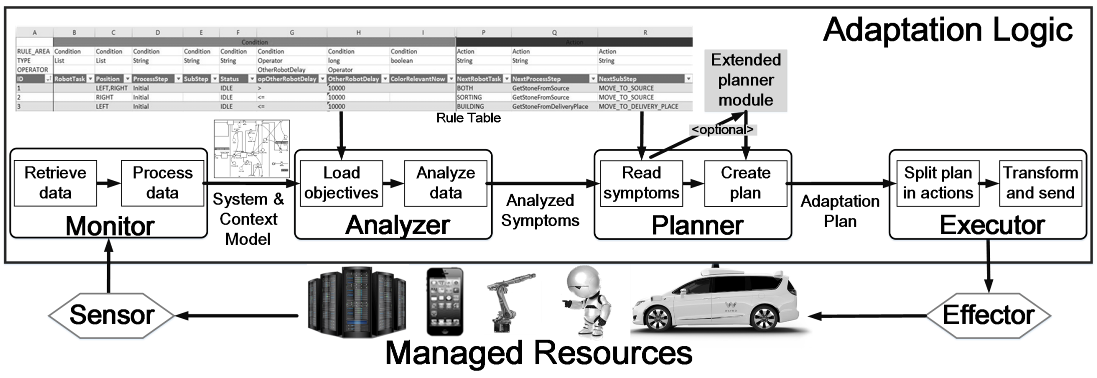


Figure 7.17.: The FESAS Wrapper combines the reference system contained of the FESAS Middleware and the MAPE components with the configuration files. Further, in case of a self-contained version, code for the functional logics is integrated. Otherwise, the system loads the code from the FESAS Repository.

7.5.2. Adaptation Logic Modules

The reference systems contain an implementation of the MAPE components, however, not of functional logic elements. Through the modularity of the FESAS Adaptation Logic Template it is possible to integrate different functional logics that can be found in literature, such as the self-managing framework for resources discovery / monitoring from [246], *Plato* [308] for identifying target system states for supporting analyzing, or *Hermes* [304] for generating plans for adapting managed resources. Additionally, we implemented different modules that can be reused. In the following, this section presents some modules for the adaptation logic. Former, Section 7.4.2 described modules for the ALM.

For the SmartHighway system, we implemented a rule-based planner. This planner can be reused by defining rules in an XML file. [222] complements a rule-based planner with a reusable analyzer. For improving usability compared to [223], developers can define rules in a spreadsheet without the need to learn a syntax, such as XML. These rules combine the expected state for the analyzer as well as the corresponding actions for planning. This approach offers functional logics for monitoring, analyzing, planning, and execution. Only minor adjustments for monitoring and executing are necessary. Figure 7.18 shows the process for reasoning using the spreadsheet rule-based approach. For extending the planning process, developers can integrate a specific extended planner module.



A	B	C	D	E	F	G	H	I	P	Q	R
	Condition								Action		
RULE_AREA	Condition	Condition	Condition	Condition	Condition	Condition	Condition	Condition	Action	Action	Action
TYPE	List	List	String	String	String	Operator	long	boolean	String	String	String
OPERATOR						OtherRobotDelay	Operator				
ID	RobotTask	Position	ProcessStep	SubStep	Status	opOtherRobotDelay	OtherRobotDelay	ColorRelevantNow	NextRobotTask	NextProcessStep	NextSubStep
1		LEFT,RIGHT	Initial		IDLE	>	10000		BOTH	GetStoneFromSource	MOVE_TO_SOURCE
2		RIGHT	Initial		IDLE	<=	10000		SORTING	GetStoneFromSource	MOVE_TO_SOURCE
3		LEFT	Initial		IDLE	<=	10000		BUILDING	GetStoneFromDeliveryPlace	MOVE_TO_DELIVERY_PLACE

Figure 7.18.: Process for rule-based reasoning from [222] with spreadsheet-based rule definition mapped to the elements of the adaptation logic. Note: The knowledge base is omitted.

Rule-based analyzing and planning is efficient and easy to use. However, it is limited in its applicability as large systems either require a large rule set for covering all possible system states or some degree of freedom, which might lead to situations for which rules do not exist. To address these shortcomings, we implemented a DSPL approach for analyzing and planning [293]. Figure 7.19 shows an overview of the architecture of the DSPL approach. In contrast to usual DSPL approaches, our approach integrates a context feature model additional to the system feature model. Modeled as Constraint Satisfaction Problem (CSP), both models are linked. Hence, changes in the context of the system are mapped to new configurations in the system feature model. Additionally, costs of adaptations and priorities support the adaptation decision, e.g., priorities are used to solve conflicts. The implementation can be reused with minor customizations in the MAPE functionalities – mainly the monitoring and executing functions – and a definition of the models for the system features and the context relations.

In [316], we present a nature-inspired approach to conflict resolution in adaptation planning. There, we target smart peer groups, i.e., devices that interact in a shared environment. Consequently, as these devices can adapt their context, context adaptation by one device might influence the performance of others. Our module uses the flocking [249, 333, 412] principle for mediating individual configuration parameters of several devices. The module can be used for planning in SASs, however, it is not integrated in the FESAS Framework as functional logic.

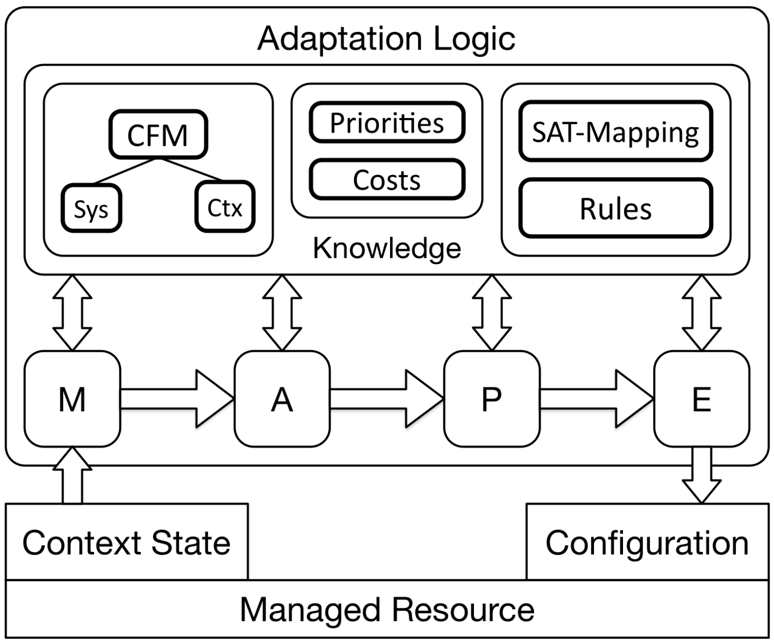


Figure 7.19.: Extended MAPE-K cycle with a DSPL context feature model and additional information for adaptation planning from [293]. Sys = System Features Model, Ctx = Context Model, SAT Mapping = Satisfiability Model Solver.

Additionally, the FESAS Framework offers several functional logics that are in preliminary states and have not been published yet. In the following, those are described on an abstract level. For the *SmartHighway* system, different utility-based approaches to adjust the speed limit have been implemented in Bachelor and Master theses as an alternative for the ECA rule-based approach. Additionally, a Bachelor student built an aggregating analyzer that controls several sub modules each analyzing a different parameter of the system and aggregates the result. In two Bachelor theses, students used the FESAS Framework and added modules that use machine learning to handle freedom in planning. A self-learning analyzer offers an automatic adaptation of the prediction algorithm for analyzing. Besides, we implemented a planner that optimized pre-configured plans based on reinforcement learning.

This chapter provided an overview on the prototypes of the FESAS Framework and the ALM. This prototype is a proof of concept for the design presented in Chapter 6. For proving its applicability, the prototype was evaluated in several case studies. The following chapter presents these evaluations.

8. Evaluation

The *Design Science Research Methodology Process Model* of Peffers *et al.* [289], used in this thesis (cf. Chapter 3), integrates two central elements: the design of the artifact and its evaluation. Chapter 6 presented the design of the FESAS Framework. This chapter presents the second central element: the evaluation of the FESAS Framework and the ALM based on the proof of concept prototype implementation presented in Chapter 7.

Table 8.1.: Design evaluation methods according to Hevner *et al.* [180].

Observational	Case Study: Study artifact in depth in business environment
	Field Study: Monitor use of artifact in multiple projects
Analytical	Static Analysis: Examine structure of artifact for static qualities (e.g., complexity)
	Architecture Analysis: Study fit of artifact into technical IS architecture
	Optimization: Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior
	Dynamic Analysis: Study artifact in use for dynamic qualities (e.g., performance)
Experimental	Controlled Experiment: Study artifact in controlled environment for qualities (e.g., usability)
	Simulation: Execute artifact with artificial data
Testing	Functional (Black Box) Testing: Execute artifact interfaces to discover failures and identify defects
	Structural (White Box) Testing: Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation
Descriptive	Informed Argument: Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility
	Scenarios: Construct detailed scenarios around the artifact to demonstrate its utility

The evaluation integrates various methods based on the description of design evaluation methods from Hevner *et al.* [180] (cf. Table 8.1). For further information on the design evaluation methods, the interested reader is referred to [180]. In the following, this section describes how the methods are applied for the evaluation of the FESAS Framework. Afterwards, Chapter 9 combines the results of the evaluations in a cross-case discussion.

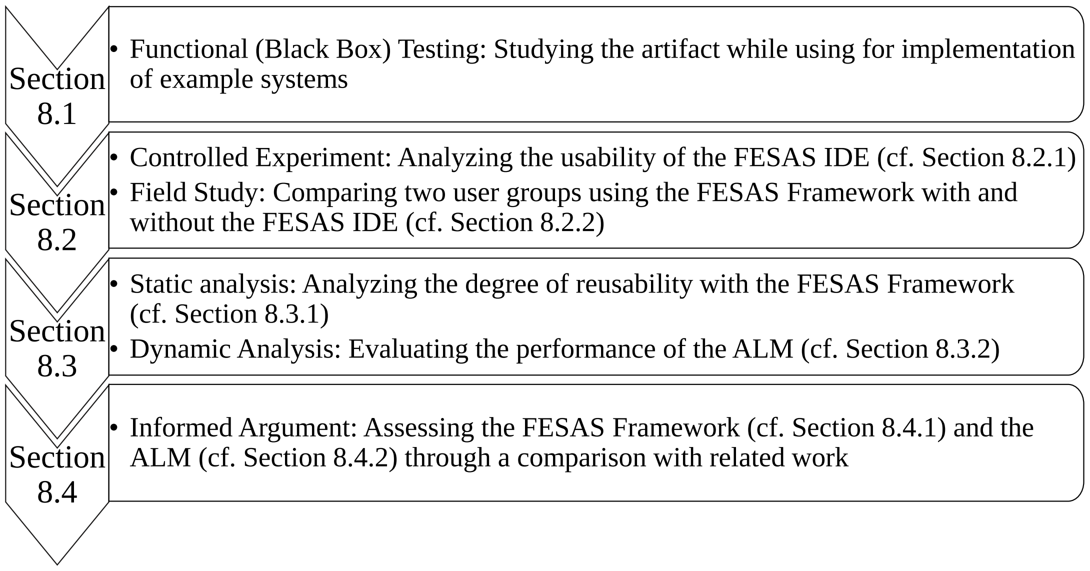

Section 8.1
• Functional (Black Box) Testing: Studying the artifact while using for implementation of example systems
Section 8.2
• Controlled Experiment: Analyzing the usability of the FESAS IDE (cf. Section 8.2.1)
• Field Study: Comparing two user groups using the FESAS Framework with and without the FESAS IDE (cf. Section 8.2.2)
Section 8.3
• Static analysis: Analyzing the degree of reusability with the FESAS Framework (cf. Section 8.3.1)
• Dynamic Analysis: Evaluating the performance of the ALM (cf. Section 8.3.2)
Section 8.4
• Informed Argument: Assessing the FESAS Framework (cf. Section 8.4.1) and the ALM (cf. Section 8.4.2) through a comparison with related work

Figure 8.1.: This thesis combines different design evaluation methods from [180] for evaluating the FESAS Framework, the FESAS IDE, and the ALM for developing SASs.

The evaluation of the FESAS Framework addresses all five categories of evaluation methods from [180]. Figure 8.1 subsumes the used evaluation methods. Asadollahi et al. [20] proposed several quality attributes for frameworks for developing SASs: *flexibility*, *extendibility*, *usability*, *reusability*, *performance*, and *scalability*. In the following, we analyze these attributes except of scalability for the FESAS Framework and the ALM.

First, Bachelor, Master and PhD students used the FESAS Framework to implement SASs within student projects, theses, or conference publications. These works proved the generic applicability as well as the functional correctness of the FESAS Framework and represent *functional testing*. Section 8.1 presents the application domains of these systems.

Second, Section 8.2 presents the results of two experiments on using the FESAS IDE. The experiments focus on the quality attributes flexibility in development, usability of the FESAS IDE, extendibility of the existing elements, and the perceived performance for development. The first part of the section shows the results for evaluating the usability of the FESAS IDE. This study was first published in [225]. The second part of the section presents the results of an experiment with Master students. One group was allowed to use the FESAS IDE for implementation of SASs, a second group had to implement SASs using the FESAS Framework without FESAS IDE support. This experiment shows the benefits of the FESAS IDE but also of the FESAS Framework in general. According to Hevner *et al.* [180], both evaluations represent *controlled experiments* and *field studies* from the category of observational evaluation methods.

Third, Section 8.3 explains the results of two analytical evaluations focusing on reusability of code, flexibility at runtime, and performance at runtime of the ALM. Based on a *static analysis* of systems implemented with the FESAS Framework components and the FESAS IDE, we analyzed the degree of reusability contributed by the FESAS Framework. These static analyses were published in [225] and [228]. Additionally, in [223] we performed a *dynamic analysis* of the prototype implementation of the ALM and the modules presented in Section 7.4.

Last, Section 8.4 evaluates the FESAS Framework and the ALM in contrast to the related work presented in the Sections 4.2 & 4.3 in a qualitative categorization using the taxonomies from [229] and [224]. This represents an *informed argument* (cf. [180]).

Similar to other PhD theses in the area of SASs [91,242], we use a protocol for the evaluations presented in Section 8.2-8.4 to support a structured analysis. Based on [408], Luckey [241, p. 180] propose the following structure for an evaluation which is used in this thesis:

- **Evaluation Design.**
  - Evaluation Questions.
  - Evaluation Propositions.
  - Units of Analysis.
  - Linking Data to Propositions.
  - Interpretation Criteria.
- **Evaluation Execution.** *(opt)* Description of the evaluation process and created artifacts.
- **Evaluation Results & Discussion.** Description and interpretation of the results.

8.1. Case Study based Testing Evaluation

This evaluation includes testing of the artifacts' functionalities and focus on *whether* the artifact performs it functions correctly rather than *how* it performs. Bachelor / Master students used the FESAS Framework, FESAS IDE, and the ALM to implement SASs and SOSs in theses and student projects. Additionally, PhD students used the FESAS Framework for implementing SASs within conference publications (cf. [221, 223, 228, 230, 293]). This conforms an unstructured *functional testing* (cf. [180]). As students are less experienced than senior developers, this also shows the ease-of-use of the FESAS Framework. The system domains are in accordance with SASs research (cf. the use cases mentioned in [77, 199, 331]). As the focus lies on the functionality of the FESAS Framework rather on its performance, this testing evaluation does not address specific requirements from Chapter 5 explicitly. This unstructured evaluation does not follow the aforementioned protocol but presents each case separately.

In the following, this section presents the implemented SASs in the domains: (i) smart traffic management, (ii) platooning coordination, (iii) adaptive cloud management, (iv) Industry 4.0, (v) smart home infrastructure, (vi) smart vacuum cleaner, (vii) fall detection, (viii) adaptive tunnel lighting, and (ix) code offloading. All domains are associated with CPSs, autonomous robotics, and cloud computing. All implementations used the FESAS Framework, including the reference system and FESAS IDE for building the adaptation logic of the SASs and implementing a connection to managed resources. Some of the implementations resulted in the reusable functional logic modules presented in Section 7.5.2. Further, all these systems are part of the settings in which the data for the human-oriented experimental evaluation (cf. Section 8.2) and the technology-oriented analytical evaluation (cf. Section 8.3) were collected. Parts of this section are taken from [221][1], [222][2], [223][3], [225][4], [228][5], [230][6], and [293][7].

[1] [221] is joint work with M. Breitbach, J. Saal, C. Becker, M. Segata, and R. LoCigno.

[2] [222] is joint work with G. Drechsel, D. Mateja, A. Pollkläsener, F. Schrage, T. Sturm, A. Tomasovic, and C. Becker.

[3] [223] is joint work with J. Otto, F. M. Roth, A. Frömmgen, and C. Becker.

[4] [225] is joint work with F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schürr.

[5] [228] is joint work with F. M. Roth, S. VanSyckel, and C. Becker.

[6] [230] is joint work with T. Sztyler, J. Edinger, M. Breitbach, C. Becker, H. Stuckenschmidt.

[7] [293] is joint work with M. Pfannemüller, M. Weckesser, and C. Becker.

8.1.1. Smart Traffic Management

Different works proposed use cases for SASs centered around traffic management, e.g., routing [407], adaptive control of traffic lights [151, 298, 363], self-healing traffic cameras [13, 18, 384, 399], or self-organized traffic management [299, 343]. In accordance, students built the SmartHighway traffic management system which is presented in the introduction of Chapter 7.

The adaptation logic of the SmartHighway traffic management system is implemented using the FESAS Framework. The different modules for adaptation of the traffic flow – speed limits, re-routing, and releasing shoulders – use the same MAPE components and communication logics, only their functional logics differ.

SUMO and TraCI simulate the managed resources. The *SUMO*[8] traffic simulator simulates human-driven and self-driving vehicles as well as the infrastructure, i.e., V2I communication, traffic signs, and traffic cameras. *TraCI*[9], an addition to SUMO, allows to change parameters during the simulation and, therefore, to control (i) traffic signs, releasing shoulders, or re-directions and (ii) reactions of self-driving vehicles to V2I communication. The adaptation logic's sensors and effectors communicate with TraCI via sockets. TraCI adapts the simulation parameters accordingly, e.g., by setting a speed limit.

The implementation of the system is published in [225, 228]. Additionally, in [223], we present an extension of the SmartHighway system with the ALM that offers structural self-improvement and parametric self-improvement through learning of adaptation rules (cf. Section 7.4.2).

8.1.2. Infrastructure-aided Platooning Coordination

Platooning is driving in convoys of semi-automated vehicles with a distance of only a few meters between them [312]. Vehicles need to drive autonomously or at least control the longitudinal distance. In the *iCOD* project[10], we implement a system for self-organized infrastructure-aided cooperative driving through coordination of platoons. The Platoon Coordination System (PCS) coordinates the

[8]SUMO's website: `http://www.dlr.de/ts/sumo/en/`
[9]TraCI's website: `http://sumo.dlr.de/wiki/TraCI`
[10]iCOD project website: `http://icod.bwl.uni-mannheim.de/`

formation of platoons. It receives information from drivers, e.g., their destination, searches a suitable platoon, and navigates the vehicle to the platoon. After reaching the platoon, a vehicle uses vehicle-to-vehicle communication and sensors (e.g., distance sensors) for controlling the joining process. Once in a platoon, the vehicle autonomously keeps the lane and the distance to preceding vehicles. Further, if a platoon meets another platoon, the PCS decides whether they should merge or overtake. The PCS receives updates from vehicles about their positions constantly and, hence, can determine when a vehicle should leave a platoon or a platoon should be dissolved. Figure 8.2 shows the platooning process.

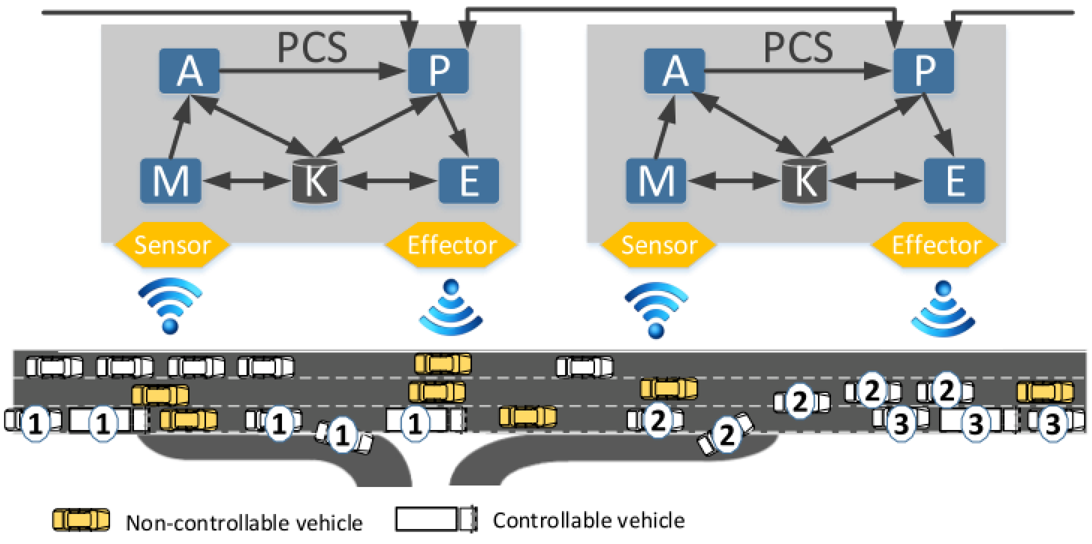


Figure 8.2.: The platooning process on a highway controlled by the PCS. A vehicle leaves `platoon (1)`, `platoon (2)` overtakes `platoon (3)` and, additionally, a vehicle joins `platoon (2)`.

Bachelor / Master students used the FESAS Framework to build the PCS which controls a demonstrator for coordinating self-driving Mindstorms robots [221][11]. Additionally, a team project with Master students integrated the PCS and the platooning simulator *PLEXE* [332] into a testbed for infrastructure-aided platooning coordination for evaluating various coordination strategies. There were able to reuse sub-modules of the analyzing and planning algorithms used for platooning coordination in the demonstrator.

[11]A video showing the platooning demonstrator can be found at: `https://www.youtube.com/watch?v=Nnrbq-4Dn24`

8.1.3. Adaptive Cloud Management

Within the Autonomic Computing domain, the focus is on self-adaptive management of data centers and cloud resources, e.g., [16, 140, 199, 214]. We also build such a system for the evaluation of the FESAS Framework [228] and the FESAS IDE [225]. The underlying use case is a cloud scenario comparable to the *Amazon EC2* cloud service. Each data center has one adaptation logic for self-management to (i) start new servers in case of high utilization, (ii) migrate virtual machines (VMs) from one server to another for workload balancing or in case of server defects, and (iii) shut down servers in case of reduced workload.

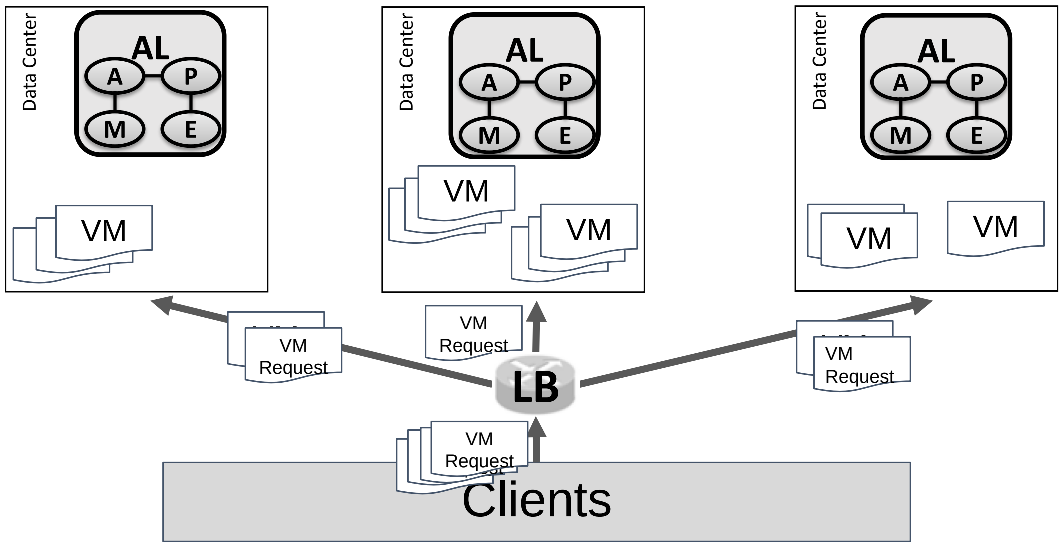
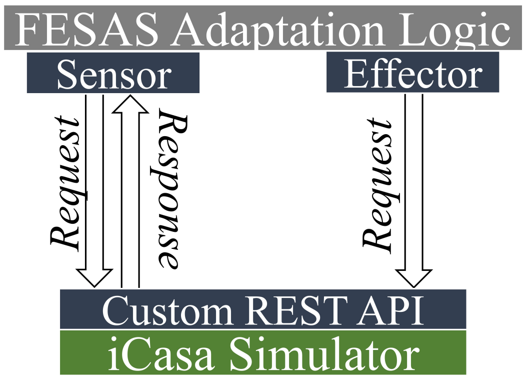


Figure 8.3.: The distributed adaptation logic controls several data centers and their servers as well as VMs. Client requests are simulated. LB = Load balancer, AL = Adaptation Logic.

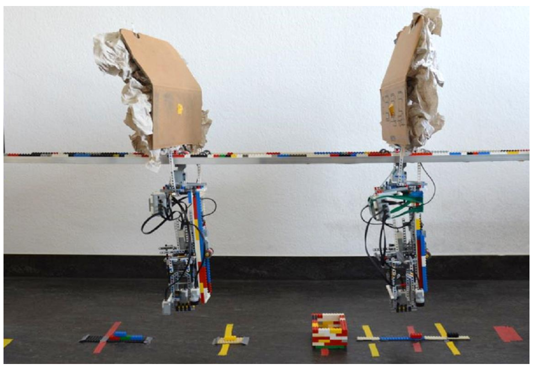
Figure 8.3 shows the system. The managed resources are composed of servers and VMs; both are simulated. Different data centers host servers. Each data center has one adaptation logic performing the aforementioned adaptations. Communication between a data center and an entity that simulates requests from clients for starting a VM is established with the BASE middleware [34].

8.1.4. Industry 4.0

Industry 4.0 applications integrate CPSs, IoT, and Cloud Computing for seamless interactions of humans and machines in the *smart factory* [179]. Several authors apply SASs in the field of Industry 4.0 mainly for building intelligent production cells (e.g., [160, 331, 347, 360]). We implemented three example SASs in the area of Industry 4.0: (i) the iCasa Verde system, (ii) the smart warehouse control, and (iii) the collaborative industry robots. All have been developed in student projects with Master students using the FESAS Framework and the rule-based reasoning module from [222]. Moreover, for the iCasa Verde and the smart warehouse only the rule table was adjusted.



(a) Architecture of the iCasa Verde system based on [222].



(b) The collaborative industry robots based on [222].

The iCasa Verde system incorporates the smart home simulation *iCasa*[12], which enables developers to create and configure smart home devices, e.g., lights, humidity, or presence sensors. In the context of iCasa Verde, iCasa simulates a self-adaptive greenhouse: Various sensors and actuators pursue the goal of creating an optimal environment for distinct plants in different zones. A custom REST API for iCasa supports accessing data measured by the sensors and adapting devices through the effector (cf. Figure 8.4a). As a second use case, students implemented a smart warehouse based on the iCasa Verde system. The system controls different warehouse zones, e.g., adjusts the temperature differently.

As third system, students implemented collaborative industry robots[13]. These robots (cf. Figure 8.4b) construct a two dimensional assembly of LEGO bricks through (i) sorting the bricks according to their color as well as (ii) assembling the product. Each robot is responsible for either sorting or building. If one robot fails, the other one performs both tasks.

[12]iCasa web site: `http://adeleresearchgroup.github.io/iCasa/snapshot/index.html`
[13]A video of the robots can be found at: `https://youtu.be/XH9tFyKVOU8`

8.1.5. Smart Home

Smart home technology enables the intelligent use of everyday objects equipped with computing resources and connected via a network. An adaptation logic receives measurements of the environment – such as temperature, light intensity, or movements in the house – and reacts through actuators, e.g., turn on the heater, close the shutter, or perform an emergency call. Several authors propose the use of SASs within the context of smart homes (e.g., [54,72,192,244,404]).

Using the FESAS Framework, Master students implemented adaptation logics for two smart home systems in six weeks student projects. In both systems, the environment of the house is simulated in Java. Both systems adjust the temperature through a heater and air condition, if the difference between the current temperature and the desired target temperature exceeds a threshold. Further, the system published in [225] regularly checks the number of people in a room and can turn off the light, if the room is empty. Figure 8.5 sketches the system model of the two systems.

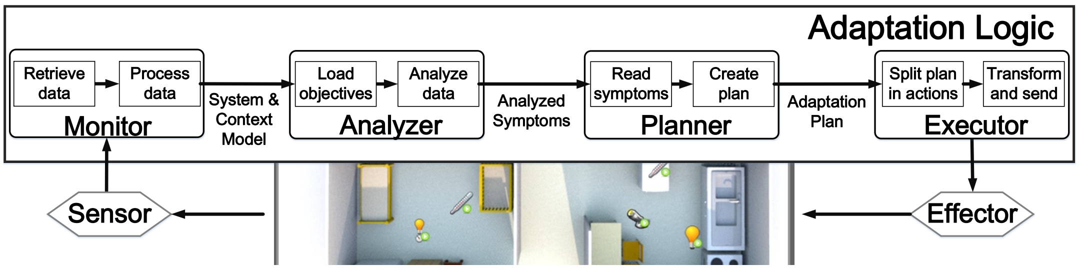
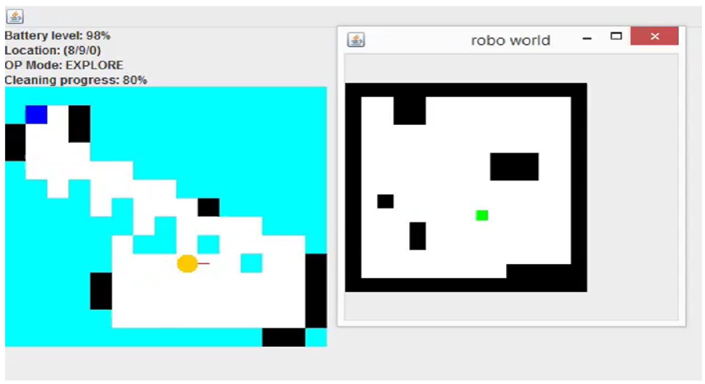


Figure 8.5.: System model of the smart home systems. The adaptation logic captures the data measured with sensors and controls the environment through actuators.

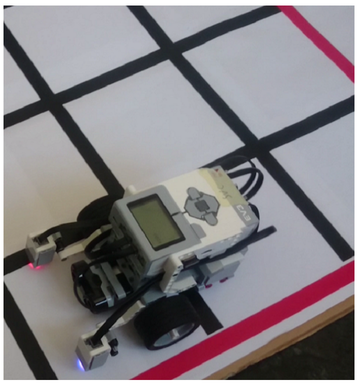
8.1.6. Smart Vacuum Cleaner

In accordance with literature (e.g., [131]), two Master students implemented in a six weeks study project a simulation of a Smart Vacuum Cleaner (SVC). The robot is simulated in an environment that can be configured (size and amount of cells to clean). The logic of the robot is implemented with the FESAS Framework. This system was one of the use cases of [225]. In a Bachelor's thesis, the simulation was extended (cf. Figure 8.6a) to identify the best algorithm for different room setups [221]. Additionally, the student tested the system with a Mindstorms robot[14] (cf. Figure 8.6b). Both are published in [221]. Using the FESAS Framework, an adaptation logic adds intelligence to the robot by acting as smart path finding system. It analyzes the sensor data of the robot – e.g., to identify obstacles – and adjusts the robot's next actions, e.g., calculates a new path. For the simulation and the robot demonstrator, the same adaptation logic was used. Minor adjustments were only necessary for the communication between managed resources and adaptation logic.



Battery level: 98%
Location: (8/9/0)
OP Mode: EXPLORE
Cleaning progress: 80%

(a) The SVC simulation. The left part shows the knowledge of the robot, the right part the simulated room.



(b) The SVC robot. Lines abstract the navigation.

Figure 8.6.: The SVC simulation and the SVC robot demonstrator.

8.1.7. Self-improving Fall Detection

In literature, different systems for fall detection are present. These approaches can be clustered in (i) wearables, (ii) ambient-based, and (iii) vision-based [266]. Wearables include different types of devices with sensors that are attached to a human's body. Ambient-based approaches integrate different sensors into the environment for detecting falls using vibration, video, and audio signals. Vision-based fall detection systems combine various algorithms to detect falls in video streams. In [230], we present a wearables-based self-improving fall detection system based on the FESAS Framework in combination with the ALM. One outcome of the paper was the self-improvement module mentioned in Section 7.4.

[14]The focus was on the driving and navigation functions and omitted cleaning.

Algorithms for wearable fall detection systems integrate a specific sequence of actions [266,410]: (i) data acquisition, (ii) data pre-processing, (iii) fall detection, and (iv) fall alert. We mapped these activities to the MAPE cycle: The MAPE components capture the data (action (i) and (ii)), analyze whether a fall has happened (action (iii)), and plan the respective reaction (action (iv)), e.g., informing caregivers in case of a fall. Additionally, the self-improvement module in our fall detection system detects a position change of the wearable device and adapts the fall detection algorithm in the analyzer accordingly. Figure 8.7 presents the system design.

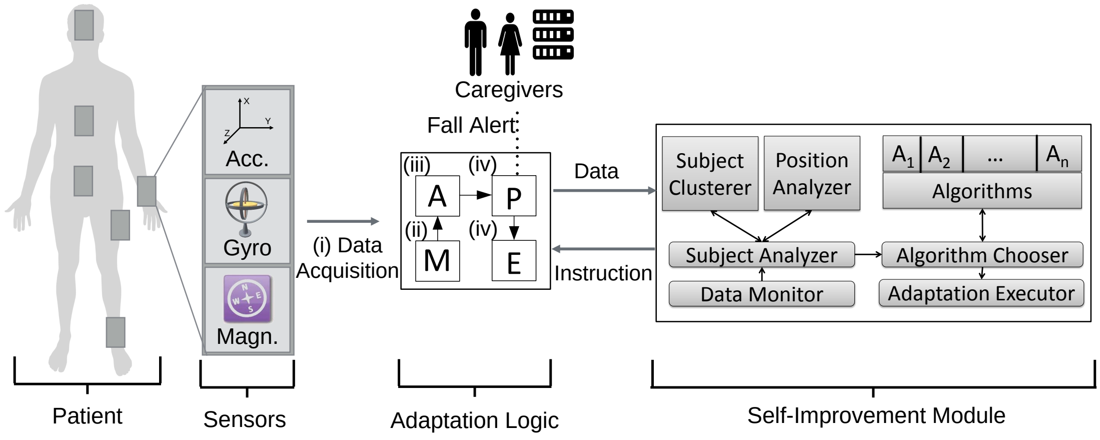


Figure 8.7.: Design of the self-improving fall detection system (taken from [230]). The adaptation logic's MAPE elements enable fall detection. The self-improvement module detects the current position of the device and adapts the fall detection algorithm in the analyzer accordingly.

8.1.8. Adaptive Tunnel Lighting

In [225], we presented an adaptive tunnel lighting system. Having a high contrast of the light within a tunnel compared to the outside brightness can dazzle drivers. Therefore, lights in tunnels need to be adapted according to outside light circumstances. The brightness of the environment can change multiple times per day through the change between day and night light or through weather conditions such as rain, snow, or sunshine. Using the FESAS Framework, two Master students implemented an adaptive lighting system of a tunnel similar to [70] in a six weeks student project. The lighting of the tunnel as well as the light sensors as managed resources are simulated on a web server implemented in JavaEE. A light-weight context model simulates the environment of the web server. The connection between the tunnel as managed resources and the adaptation logic is established via HTTP. The adaptation logic controls the light intensity based on the environmental light conditions. Additionally, a graphical user interface shows the tunnel lighting. This is controlled via JSON and HTTP requests. Figure 8.8 presents the system and its components.

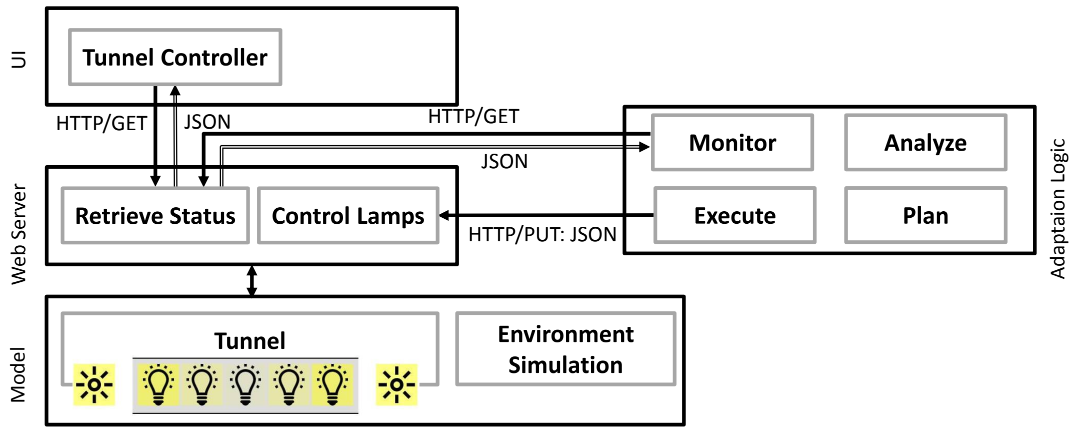


Figure 8.8.: The components of the adaptive tunnel lighting system and the interaction. The system is composed of a web server simulating the tunnel, the adaptation logic as well as the user interface. Interaction is enabled through HTTP requests.

8.1.9. Code Offloading

Using the DSPL approach for adaptation planning from [293], we implemented a SAS for managing the Tasklet system. The Tasklet system provides a middleware-based infrastructure for distributed computing on heterogeneous devices [324]. Therefore, three entities are available: resource providers, resource consumers, and resource brokers. Resources providers offer a Tasklet virtual machine for running code. Each resource provider registers at a broker. Brokers form a peer-to-peer overlay network. Resource consumers send requests for resource providers to a broker. A resource consumer may specify different Quality of Computation goals for a Tasklet, e.g., reliability, speed, or security. For more details on the Tasklet system, the interested reader is referred to [324]. Figure 8.9 shows an overview of an example overlay network topology.

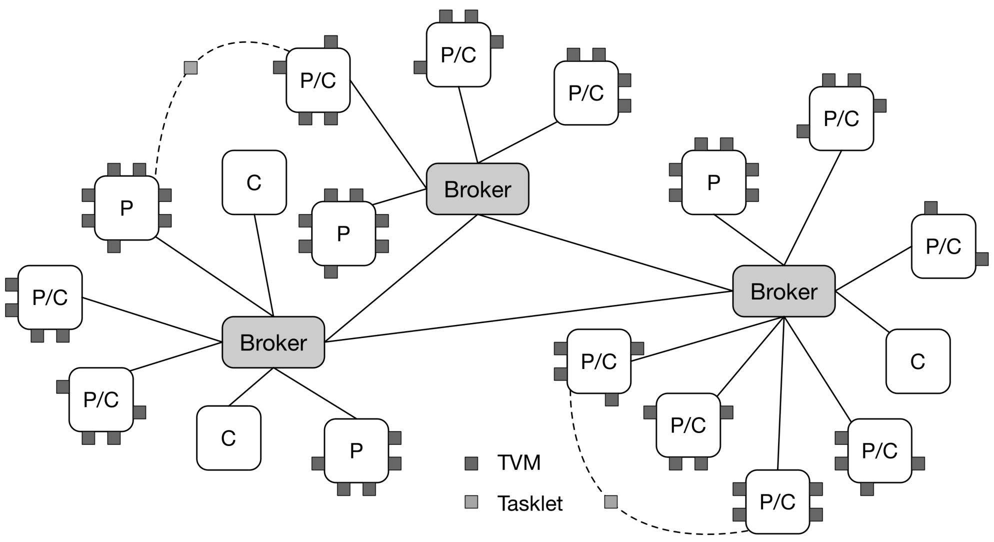


Figure 8.9.: Schema of an example Tasklet network topology with resource consumers, providers, and brokers based on [324]. P = Tasklet Producer, C = Tasklet Consumer. The Broker manager is omitted.

In [293], we simulated the Tasklet system using the simulation presented in [111]. Additionally, we extended the simulation by a Broker Manager. It stores a list with all brokers. The Broker Manager monitors all brokers and adapts their behavior by changing their configurations according to the instructions of the adaptation logic implemented using the FESAS Framework.

8.1.10. Summary

This section introduced several SASs implemented using the FESAS IDE and the FESAS Framework, some in combination with the ALM. Often, students could reuse different elements, e.g., functional logics, system design, or the whole adaptation logic. Further, as mostly less experienced Bachelor/Master students implemented the SASs in an often short time frame, this shows the ease of use of development with the FESAS Framework. Additionally, the implementation of these system with the FESAS Framework supported the functional testing of its elements. In the following evaluations, we target the usability, applicability, enabled reusability as well as the performance of the FESAS IDE, the FESAS Framework in general and the ALM. For these evaluations, we analyzed the example SASs and interviews with the developers.

8.2. Human-oriented Experimental and Observational Evaluation

This section presents the evaluation of the FESAS IDE and the FESAS Framework in general. For both evaluations, we asked students to use the FESAS Framework and the FESAS IDE to implement SASs. The first evaluation was originally published in [225]. For this thesis, we tripled the amount of students participating in the experiment to receive more reliable results. The purpose is to show the usability of the FESAS IDE. For the second evaluation, we performed an observational evaluation in the form of a field study. For this experiment, Master students that attended a lecture on SOSs and SASs in two different years were asked to implement SASs. The first group had to implement their systems using the FESAS Framework without the FESAS IDE. The second group used the FESAS IDE. This evaluation measures the suitability of the FESAS IDE to support the use of the FESAS Framework. Next, Section 8.2.1 presents the usability evaluation. After, Section 8.2.2 shows the results of the applicability evaluation.

8.2.1. Controlled Experiment: Usability of the FESAS IDE

We asked Bachelor, Master, and PhD students that used the FESAS IDE for developing SASs to evaluate the usability of the FESAS Development Tool. As the implementation of the SASs and the evaluation of the FESAS Development Tool happened in a controlled environment, this represents an experimental evaluation (cf. [180]). Parts of this section are taken from [225][15].

Evaluation Design

For the evaluation, we neglected the FESAS Design Tool. The students mostly implemented SASs with a central adaptation logic and reused existing configuration files for the system models. Students had to fill out questionnaires after the use of the FESAS IDE. The evaluation is structured according to the protocol defined in the introduction of Chapter 8.

Evaluation Questions: This evaluation is driven by the question of the suitability of the FESAS IDE – here: the FESAS Development Tool – to support the development of SASs. Hence, we derive the following evaluation questions:

[15] [225] is joint work with F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schürr.

$\boldsymbol{EQ_{EE}}$**1** Does the FESAS IDE hide process details of the FESAS Framework?

$\boldsymbol{EQ_{EE}}$**2** Has the FESAS IDE high usability?

$\boldsymbol{EQ_{EE}}$**3** Does the FESAS IDE accelerate the development of SASs?

Evaluation Proposition: According to the evaluation questions, we propose:

$\boldsymbol{PR_{EE}}$**1** The FESAS IDE does not require developers to learn the FESAS Framework workflow.

$\boldsymbol{PR_{EE}}$**2** The FESAS IDE is easy to learn and easy to use.

$\boldsymbol{PR_{EE}}$**3** The FESAS IDE supports the development of reusable SASs.

$\boldsymbol{PR_{EE}}$**4** The FESAS IDE supports the testing of SASs.

$\boldsymbol{PR_{EE}}$**5** The FESAS IDE accelerates the development of SASs.

Units of Analysis: Bachelor, Master, and PhD students used the FESAS IDE for implementing SASs. Afterwards, they filled out a questionnaire (cf. Appendix D.1). We derived the following question items from the *ISO 9241-11 Guidance on Usability* standard and the definition of usability in the *ISO/IEC 9126-1 Software Product Quality Model* standard:

$\boldsymbol{QI_{EE}}$**1** The tool has a short training period.

$\boldsymbol{QI_{EE}}$**2** The tool facilitates using FESAS.

$\boldsymbol{QI_{EE}}$**3** The tool is easy to use.

$\boldsymbol{QI_{EE}}$**4** The tool supports reusability of code.

$\boldsymbol{QI_{EE}}$**5** The tool supports simplified exchange of MAPE algorithms.

$\boldsymbol{QI_{EE}}$**6** The tool eliminates the implementation of general issues.

$\boldsymbol{QI_{EE}}$**7** The tool supports testing in the development phase.

$\boldsymbol{QI_{EE}}$**8** The tool is well integrated into the FESAS development process.

$\boldsymbol{QI_{EE}}$**9** The tool accelerates development with FESAS.

$\boldsymbol{QI_{EE}}$**10** The tool accelerates development in general.

Linking Data to Propositions: The 10 question items and the propositions can be linked with each other:

- $PR_{EE}1$ is linked to $QI_{EE}2$, $QI_{EE}8$, and $QI_{EE}9$.
- $PR_{EE}2$ is linked to $QI_{EE}1$ and $QI_{EE}3$.
- $PR_{EE}3$ is linked to $QI_{EE}4$ and $QI_{EE}5$.
- $PR_{EE}4$ is linked to $QI_{EE}7$.
- $PR_{EE}5$ is linked to $QI_{EE}6$ and $QI_{EE}10$.

Interpretation Criteria: All participants of the experiment could rate their consensus with the question items on a 5 point Likert scale ranging from `Strongly disagree` to `Strongly agree`. Additionally, it was possible to not answer. A majority of `(strongly) agreed` answers indicates support for a question item.

Evaluation Execution

We accompanied Bachelor, Master, and PhD students that used the FESAS IDE for implementing SASs in all aforementioned domains except the fall detection system. In total, we asked 2 Bachelor, 13 Master, and 3 PhD Students. The 13 Master students built different SASs in six weeks student projects. The 2 Bachelor students used the FESAS IDE within their theses. The PhD students used the FESAS IDE for implementation of SASs for conference publications. We acknowledge, that this is a rather small set of participants. Hence, repetition on a broader size is future work. The evaluation was done two times. The first group of participants results from the publication in [225]. The second group of Master students attended a lecture about SOSs and SASs in the fall term 2017.

Evaluation Results & Discussion

The tool seems to be well integrated in the FESAS workflow ($QI_{EE}8$), abstracts from FESAS specific activities ($QI_{EE}2$), and, therefore, accelerates the process of development with the FESAS Framework ($QI_{EE}9$). This supports proposition $PR_{EE}1$. Regarding proposition $PR_{EE}2$, the results indicate that the *FESAS Development Tool* needs only a short training period ($QI_{EE}1$) and is easy to use ($QI_{EE}3$). Further, it supports reusability of code ($QI_{EE}4$) and simplifies the exchange of algorithms in the MAPE components ($QI_{EE}5$). This promotes proposition $PR_{EE}3$. Additionally, participants perceived an acceleration of the development in general ($QI_{EE}10$) and through eliminating the implementation of general issues (e.g., communication) in specific ($QI_{EE}6$), backing proposition $PR_{EE}5$. However, the developers of the example systems identify that the integration of the testing process could be improved ($QI_{EE}7$). This disproves proposition $QI_{EE}4$. Table 8.2 shows the aggregated results of the interviews.

Table 8.2.: Answers of students using the FESAS Development Tool to implement a SAS (n=18 students).

	(Strongly) disagree	**Neutral**	**Agree**	**Strongly agree**	**No answer**
$\boldsymbol{QI_{EE}1}$ The tool has a short training period.	11.1%	16.7%	38.9%	27.8%	5.6%
$\boldsymbol{QI_{EE}2}$ The tool facilitates using FESAS.	0.0%	5.6%	33.3%	55.6%	5.6%
$\boldsymbol{QI_{EE}3}$ The tool is easy to use.	0.0%	22.2%	55.6%	16.7%	5.6%
$\boldsymbol{QI_{EE}4}$ The tool supports reusability of code.	0.0%	0.0%	61.1%	27.8%	11.1%
$\boldsymbol{QI_{EE}5}$ The tool supports simplified exchange of MAPE algorithms.	0.0%	0.0%	33.3%	66.7%	0.0%
$\boldsymbol{QI_{EE}6}$ The tool eliminates the implementation of general issues.	0.0%	11.1%	22.2%	55.6%	11.1%
$\boldsymbol{QI_{EE}7}$ The tool supports testing in the development phase.	0.0%	22.2%	38.9%	16.7%	22.2%
$\boldsymbol{QI_{EE}8}$ The tool is well integrated into the FESAS development process.	0.0%	16.7%	44.4%	27.8%	11.1%
$\boldsymbol{QI_{EE}9}$ The tool accelerates development with FESAS.	0.0%	5.6%	44.4%	44.4%	5.6%
$\boldsymbol{QI_{EE}10}$ The tool accelerates development in general.	0.0%	27.8%	27.8%	44.4%	0.0%

8.2.2. Field Study: Applicability of the FESAS IDE

Additionally, we asked students to implement different SASs using the FESAS Framework in a field study, an observational evaluation method (cf. [180]). In the following, the section presents the evaluation results.

Evaluation Design

One group of students only used the FESAS Framework components, the other group used the FESAS IDE. Having the two group of students using different tool sets, enables us to compare the use of the FESAS Middleware manually in contrast to using the FESAS IDE. The evaluation is structured according to the protocol defined in the introduction of this chapter.

Evaluation Questions: The main evaluation question is:

EQ_{FS}1 Does the FESAS IDE improve the applicability of the FESAS Framework for implementing SASs?

Evaluation Proposition: According to the evaluation question, the propositions focus on the differences of the application of the FESAS Middleware in contrast to the use of the FESAS IDE:

PR_{FS}1 The FESAS IDE simplifies the initialization and configuration of the FESAS development environment.

PR_{FS}2 The FESAS IDE simplifies the development of SASs compared to manually using the FESAS Framework components and corresponding workflow.

PR_{FS}3 The perceived benefits of using the FESAS Framework for developing SASs are increased when using the FESAS IDE.

Units of Analysis: Two groups of Master students attending a lecture on SOSs and SASs implemented SASs using the FESAS Framework and FESAS IDE, respectively. These students were interviewed afterwards. Their answers are captured in a questionnaire (cf. Appendix D.2). As most of the students reused existing JSON configuration files, we target the FESAS Development Tool with this evaluation. In accordance with the *ISO 9241-11 Guidance on Usability* standard and the definition of usability in the *ISO/IEC 9126-1 Software Product Quality Model* standard, we define the following question items:

QI_{FS}1 It is easy to configure the FESAS[16] tool set.

QI_{FS}2 I had problems in installing the FESAS tool set.

QI_{FS}3 I was able to directly start programming with the FESAS tool set.

QI_{FS}4 FESAS was helpful for implementing a self-adaptive system.

QI_{FS}5 I had problems in configuring my self-adaptive systems (writing configuration files).

[16]The term *FESAS* describes either the FESAS Framework or the FESAS IDE depending on the group.

$\boldsymbol{QI_{FS}}$**6** I had problems in writing code for the functional logic elements for the adaptation logic.

$\boldsymbol{QI_{FS}}$**7** I had problems in adding the functional logic elements to the repository.

$\boldsymbol{QI_{FS}}$**8** I had problems in implementing the sensors/effectors.

$\boldsymbol{QI_{FS}}$**9** I had problems in connecting managed resources and adaptation logic.

$\boldsymbol{QI_{FS}}$**10** Overall, FESAS helped me in the implementation.

$\boldsymbol{QI_{FS}}$**11** Overall, it was easy to use FESAS.

$\boldsymbol{QI_{FS}}$**12** Overall, I think FESAS speed up the development.

$\boldsymbol{QI_{FS}}$**13** Overall, I would recommend FESAS for implementing a SAS.

Linking Data to Propositions: The 13 question items and the proposition can be linked:

- $PR_{FS}1$ is linked to $QI_{FS}1$ to $QI_{FS}3$.
- $PR_{FS}2$ is linked to $QI_{FS}4$ to $QI_{FS}9$.
- $PR_{FS}3$ is linked to $QI_{FS}10$ to $QI_{FS}13$.

Interpretation Criteria: Identical to the former evaluation of the usability of the FESAS IDE, participants rate their consensus with question items using a 5 point Likert scale. The scale ranges from 1 (`Strongly disagree`) to 5 (`Strongly agree`) with the possibility to answer with "No answer possible".

Evaluation Execution

The first group of 11 students implemented their SASs in the fall term 2015 using only the components of the FESAS Middleware and the reference systems. In the fall term 2017, a second group of 12 students implemented their SASs using the FESAS IDE. Again, we neglected the FESAS Design Tool as the students reused existing configuration files for the system models. As the sizes of the group are almost identical, the results are comparable.

All students had six weeks to implement their SASs in teams of 2-3 students. The course instructors were available for support. The students studied Business with a specialization in IT or Business Informatics. They had mostly small to medium experience in the implementation of software in general and no experience in the implementation of adaptive systems. The SASs cover almost

Table 8.3.: Results of the comparison ($QI_{FS}1$ to $QI_{FS}7$) of the user group without the FESAS IDE (n=11 students) and the one using the FESAS Development Tool to implement a SAS (n=12 students). Answers for a question item may not sum up to 100% as students could leave out a question.

	Group fall term 2015 without FESAS IDE					**Group fall term 2017 with FESAS IDE**				
	Strongly disagree	**Dis-agree**	**Neu-tral**	**Agree**	**Strongly agree**	**Strongly disagree**	**Dis-agree**	**Neu-tral**	**Agree**	**Strongly agree**
$\boldsymbol{QI_{FS}1}$ It is easy to configure the FESAS tool set.	0.0%	27.3%	27.3%	45.5%	0.0%	0.0%	25.0%	0.0%	66.7%	8.3%
$\boldsymbol{QI_{FS}2}$ I had problems in installing the FESAS tool set.	18.2%	27.3%	45.5%	9.1%	0.0%	16.7%	25.0%	33.3%	16.7%	8.3%
$\boldsymbol{QI_{FS}3}$ I was able to directly start programming with the FESAS tool set.	9.1%	72.7%	0.0%	18.2%	0.0%	0.0%	33.3%	16.7%	25.0%	16.7%
$\boldsymbol{QI_{FS}4}$ FESAS was helpful for implementing a self-adaptive system.	0.0%	18.2%	18.2%	54.5%	9.1%	0.0%	0.0%	8.3%	41.7%	50.0%
$\boldsymbol{QI_{FS}5}$ I had problems in configuring my self-adaptive systems (writing configuration files).	0.0%	9.1%	9.1%	36.4%	0.0%	25.0%	25.0%	25.0%	0.0%	0.0%
$\boldsymbol{QI_{FS}6}$ I had problems in writing code for the functional logic elements for the adaptation logic.	9.1%	27.3%	9.1%	36.4%	9.1%	50.0%	33.3%	0.0%	0.0%	0.0%
$\boldsymbol{QI_{FS}7}$ I had problems in adding the functional logic elements to the repository.	9.1%	18.2%	18.2%	18.2%	9.1%	16.7%	16.7%	8.3%	8.3%	0.0%

Table 8.4.: Results of the comparison ($QI_{FS}8$ to $QI_{FS}13$) of the user group without the FESAS IDE (n=11 students) and the one using the FESAS Development Tool to implement a SAS (n=12 students). Answers for a question item may not sum up to 100% as students could leave out a question.

	Group fall term 2015 without FESAS IDE					**Group fall term 2017 with FESAS IDE**				
	Strongly disagree	**Dis-agree**	**Neu-tral**	**Agree**	**Strongly agree**	**Strongly disagree**	**Dis-agree**	**Neu-tral**	**Agree**	**Strongly agree**
$\boldsymbol{QI_{FS}8}$ I had problems in implementing the sensors/effectors.	18.2%	18.2%	0.0%	9.1%	18.2%	33.3%	16.7%	16.7%	0.0%	0.0%
$\boldsymbol{QI_{FS}9}$ I had problems in connecting managed resources and the adaptation logic.	9.1%	36.4%	0.0%	9.1%	18.2%	16.7%	41.7%	16.7%	0.0%	8.3%
$\boldsymbol{QI_{FS}10}$ Overall, FESAS helped me in the implementation.	0.0%	18.2%	9.1%	63.6%	9.1%	0.0%	8.3%	8.3%	50.0%	33.3%
$\boldsymbol{QI_{FS}11}$ Overall, it was easy to use FESAS.	9.1%	45.5%	18.2%	27.3%	0.0%	0.0%	8.3%	16.7%	41.7%	25.0%
$\boldsymbol{QI_{FS}12}$ Overall, I think FESAS speed up the development.	9.1%	9.1%	27.3%	45.5%	9.1%	0.0%	0.0%	16.7%	50.0%	33.3%
$\boldsymbol{QI_{FS}13}$ Overall, I would recommend FESAS for implementing a self-adaptive system.	0.0%	18.2%	27.3%	54.5%	0.0%	0.0%	0.0%	16.7%	50.0%	25.0%

all domains presented in Section 8.1 with the exception of the domains of fall detection and adaptive authentication.

Evaluation Results & Discussion

While the first group without the FESAS IDE had a larger, manually setup effort, the second group was guided by the FESAS IDE and the setup procedures of the Eclipse IDE. Accordingly, the results of the evaluation are not surprising: the second group perceived an easier configuration ($QI_{FS}1$) and installation ($QI_{FS}2$) of the FESAS development environment and was more often able to start directly ($QI_{FS}3$). This supports the proposition $PR_{FS}1$.

Using the FESAS Framework without the FESAS IDE forces developers to learn the syntax for the metadata of the code for the functional logics. This results in different issues, visible in the captured data of the evaluation. Students from the group that used the FESAS IDE agreed more often that FESAS was helpful in the implementation ($QI_{FS}4$). They also encountered less issues in (i) configuring the SASs ($QI_{FS}5$), (ii) implementing ($QI_{FS}6$) and handling ($QI_{FS}7$) code of the functional logics, and (iii) connecting managed resources and the adaptation logic ($QI_{FS}8$ & $QI_{FS}9$). This promotes proposition $PR_{FS}2$.

Last, proposition $PR_{FS}3$ targets the perceived benefits of the FESAS Framework. The students that used the FESAS IDE agreed more often that the FESAS Framework (i) supports them in the development ($QI_{FS}10$), (ii) is easy to use ($QI_{FS}11$), (iii) accelerates the development ($QI_{FS}12$), and (iv) that they would recommend FESAS ($QI_{FS}13$). The Tables 8.3 & 8.4 show the results.

8.3. Technology-oriented Analytical Evaluation

This section provides an analytical evaluation (cf. [180]) of the artifacts, i.e., the FESAS Framework and the ALM. The section is twofold. First, Section 8.3.1 provides a static analysis for evaluating the degree of reusability enabled by the FESAS Framework and the FESAS IDE. The focus is on the support of the FESAS Framework for developers rather than the performance of the implemented SAS. Hence, comparisons of systems without adaptation logic to self-adaptive variants are omitted as this would evaluate the MAPE functionality rather than the development support offered by the FESAS Framework. Second, Section 8.3.2 evaluates the performance of the ALM in a dynamic analysis. Here, the focus is on the runtime performance and the additional benefits introduced through self-improvement.

8.3.1. Static analysis: Reusability through the FESAS Framework

The main objective of the FESAS Framework is the reusability of code. To measure this, we performed a static analysis (cf. [180]) of the FESAS Framework. We measured the degree of reusability of systems that have been implemented using the FESAS Framework (cf. [228]) and the FESAS IDE (cf. [225]). First, this section presents the evaluation design. Second, it describes the execution of the reusability measurements on the system level. Last, this section shortly discusses the results. Parts of this section are taken from [225][17] and [228][18].

Evaluation Design

This evaluation analyzes the degree of reusability that the FESAS Framework offers. This includes not only the code that is offered by the FESAS Framework – e.g., the implementation of the FESAS Adaptation Logic Template – but also the code generated with the FESAS IDE, e.g., parts of a functional logic. Additionally, the reusability of the design elements, i.e., the system model and the configuration files, is analyzed. It follows the protocol presented in the introduction of Chapter 8.

Evaluation Questions: The static analysis has the evaluation questions:

$\boldsymbol{EQ_{SA}1}$ How much code of a SAS implemented using the FESAS Framework can be reused?

$\boldsymbol{EQ_{SA}2}$ Are parts of the code generated for the functional logic reusable?

$\boldsymbol{EQ_{SA}3}$ Are parts of the system models reusable?

Evaluation Proposition: The proposition is that the SASs implemented using the FESAS Framework are reusable to a large extent. Hence, the FESAS Framework should meet the following criteria:

[17] [225] is joint work with F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schürr.
[18] [228] is joint work with F. M. Roth, S. VanSyckel, and C. Becker.

***PR_{SA}*1** Except the functional logic code, most parts of SASs implemented using the FESAS Framework are reusable.

***PR_{SA}*2** The MAPE components generated with the FESAS Framework are reusable with the exception of clearly defined parts.

***PR_{SA}*3** The system models are reusable with minor customizations, only.

Units of Analysis: First, this section analyzes the reusability on the component level, i.e., the reusability of code within one MAPE component. Second, reusability measurements on the system level are presented. Third, the reusability within the design of the systems is studied. We apply these three measurements to various implementations of SASs in different system domains.

Linking Data to Propositions: The resulting data is the degree of reusability offered by using the FESAS Framework on a component level (cf. proposition $PR_{SA}1$), the system level (cf. proposition $PR_{SA}2$), and within the system design (cf. proposition $PR_{SA}3$) analyzed in different use cases.

Interpretation Criteria: In accordance with literature (cf. [77, 377]), we used the Eclipse plug-in *CodePro AnalytiX* to measure the *Source Lines of Code* (SLoC) that is provided by the FESAS Framework and generated by the FESAS IDE in contrast to the code developers have to implement. This is linked to the propositions $PR_{SA}1$ and $PR_{SA}2$. For proposition $PR_{SA}3$, we manually compared the overlap in the used JSON configuration files for system models.

Evaluation Execution

In [228], we analyzed the reusability on the component level – i.e., the level of reusability within a MAPE component – enabled through the FESAS Framework. For this analysis, we focus systems for the SmartHighway (cf. Section 8.1.1) and the adaptive cloud (cf. Section 8.1.3) scenarios. For each scenario, we implemented two system variants with fully decentralized control and using the *Regional Planning* control pattern.

Additionally, we analyzed the reusability on a system level in [228] and [225]. As the FESAS Framework was extended from [228] to [225] and [225] presented five example systems instead of only two, this section presents the results of the analysis of reusability from [225]. The relevant systems were implemented in six weeks student projects accompanying a Master's lecture. The students implemented SASs in the domains: (i) SmartHighway, (ii) adaptive cloud, (iii) smart home, (iv) smart vacuum system, and (v) adaptive tunnel lighting.

We further analyzed the reusability within the system design, i.e., the reusability of the system model in form of the JSON configuration files (cf. Section 6.1.4). This was done by a qualitative analysis of the used configuration files.

Evaluation Results & Discussion

As an example for the degree of reusability on the component level, we focus the `Analyzer` of the highway use case. It is composed of: (i) an `AdaptationLogic` superclass, (ii) the `CommunicationLogic`, (iii) a reusable `Analyzer` implementation, and (iv) the functional logic, which is composed of an `AbstractLogic`, and a subclass with a customized `callLogic()` implementation. All implement corresponding interfaces. The implementation has in total 538 SLoC. The functional logic itself has 70 SLoC (13% of the component's code). The analyzer component with the communication logic has 468 SLoC (87%). Hence, only a small amount of code is customized to the use case, supporting proposition $PR_{SA}1$.

Table 8.5 shows the results of the evaluation of reusability on a system level. As all systems can use the FESAS reference system without customization, all implementations have 7,608 SLoC for the adaptation logic components. Only the configuration files and the functional logics vary. As most use cases have a centralized adaptation logic, their configuration files are almost identical (except the logic contracts' properties). We divided the SLoC of the functional logics in lines that are offered by the FESAS Framework or generated by the FESAS IDE and SLoC written by the developers. For the example cases, the developers implemented between 101 and 216 SLoC in all functional logic elements whereas between 117 and 142 additional lines are generated. Moreover, all developers had to implement functional logics and their dependencies (between 0 and 283 SLoC for the example cases). As a result, the degree of generated code is between 93.93% and 97.59%. This supports the criteria for proposition $PR_{SA}2$.

In [228], both systems rely on the same configuration files for system deployment. Both, the distribution of the MAPE functionality as well as the communication structure can be reused for the two settings of fully decentralized control and the *Regional Planning* pattern. Four of the five example systems rely on a centralized pattern. Only the SmartHighway system integrates a different

Table 8.5.: SLoC that the FESAS Framework provides or the FESAS IDE generates versus SLoC implemented by the developers (percent values indicate share of systems' SLoC). AL = adaptation logic.

	Cloud	**Highway**	**Tunnel**	**Home**	**Cleaner**
AL components	7,608	7,608	7,608	7,608	7,608
Functional logics	243	320	253	333	316
Implemented	101	193	129	216	191
Generated	142	127	124	117	125
Dependencies	238	246	80	283	-X-
Total SLoC	8,089	8,174	7,941	8,224	7,924
Generated code	95.81%	94.63%	97.37%	93.93%	97.59%

pattern. For reusing the patterns, only the information type, the name of the functional logic that should be used, and the amount of sensors and effectors vary. Therefore, it is possible to describe decentralization patterns with the configuration files and reuse them, which addresses proposition $PR_{SA}3$. The FESAS IDE enables additional support for the reusability as it enables the definition of patterns out of the box.

We acknowledge that the presented example cases are rather small and simplified. However, all systems offer full functionality. As a metric, we measured the SLoC. We acknowledge, that SLoC is not the perfect measurement variable as code can be written differently. However, it can be used as an indicator for showing the amount of code that is offered by the FESAS Framework and generated by the FESAS IDE versus the amount the developer has to write. Further, this evaluation approach is in line with SASs research (e.g., [77]). Still all propositions are supported and, hence, the evaluation questions are answered successfully.

8.3.2. Dynamic Analysis: Evaluation of the Adaptation Logic Manager

In [223], we evaluated the performance of the ALM and the three modules for the parametric and structural self-improvement from Section 7.4.2 in the SmartHighway use case system (cf. Section 8.1.1). The evaluation represents a dynamic analysis (cf. [180]) of the ALM. Parts of this section are taken from [223][19]. Appendices E.1, E.2, and E.3 provide charts with the detailed evaluation results.

[19] [223] is joint work with J. Otto, F. M. Roth, A. Frömmgen, and C. Becker.

Evaluation Design

Following the protocol presented in the introduction of this chapter, this section presents the evaluation results of the feasibility of our ALM. We performed various simulations of a German highway in different settings. First, we analyzed the performance of the adaptation logic having subsystems controlling each section of the highway independently. As there is no self-improvement involved, this is the baseline measurement. Second, we analyzed the rule learning component. Third, we compared the two modules for structural adaptation. Due to the different time horizons of the runtime optimization modules – continuous rule learning vs. spontaneous structural adaptation – we separated the tests.

Evaluation Questions: In the evaluation, we want to analyze the utility of self-improvement offered by the ALM. We omit algorithmic self-improvement in the use case and addressed the following evaluation questions:

$\boldsymbol{EQ_{DA}1}$ Does parametric self-improvement enhance the performance?

$\boldsymbol{EQ_{DA}2}$ Does structural self-improvement enhance the performance?

Evaluation Proposition: The evaluation questions can be mapped to the following propositions:

$\boldsymbol{PR_{DA}1}$ Parametric self-improvement enhances the system performance.

$\boldsymbol{PR_{DA}2}$ Structural self-improvement enhances the system performance.

$\boldsymbol{PR_{DA}3}$ Self-improvement in general enhances the system performance.

Units of Analysis: To study the ALM and the three ALM modules presented in Section 7.4.2, we applied self-improving capabilities to the SmartHighway system and observe a part of the German highway A8 between the interchange Stuttgart and the junction Stuttgart-Degerloch near Stuttgart airport. Figure 8.10 presents the simulated area of the highway. We used real traffic datasets from the German Federal Highway Research Institute (BASt)[20] and [326]. During the simulation, the volume of traffic is adjusted hourly to reflect a typical working day. The simulated period varies between 5 hours (from 6am to 11am) for the evaluation of self-improvement through structural adaptation and up to a whole day starting at midnight for the rule learner. Our simulated highway includes

[20]BASt website: `https://www.bast.de/BASt_2017/DE/Home/home_node.html`

a track of around 10 kilometers. We divided it into 4 sections. For structural adaptation, we simulated an accident as well as a daily road work.

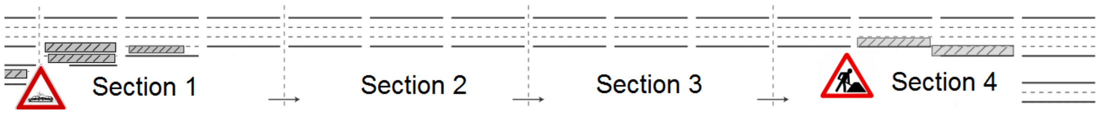


Figure 8.10.: An overview of the highway that is controlled by the SmartHighway system and the ALM. Because of the construction site in section 4 and an accident in section 1, the gray marked areas are closed in the second setting for evaluation of structural self-improvement.

We ran the system on different desktop PCs and laptops with Windows. The adaptation logic and the ALM run with Java version 8. TraCI used Python 2.7. For the simulation, a slightly modified version of SUMO is used which is based on the source code of Version 0.23.0. The modifications are necessary in order to collect the raw data of the traffic cameras.

Linking Data to Propositions: The different propositions are studied in the SmartHighway use case. Hence, they are directly related to performance data of the use case system and the specific implementation of the ALM modules.

Interpretation Criteria: As performance measurement for evaluating a simulation run, we used the average waiting time of vehicles during a simulation run for all vehicles of a specific time interval. The waiting time is the time where the speed of a vehicle was below 0.1 m/s. Hence, the lower the waiting time is, the better it is as vehicles spend less time in traffic jams. The number is logged by the SUMO traffic simulation. We calculated the average waiting time for five minute intervals based on all vehicles that started during that interval in a simulation run. To get more reliable values that are not influenced by random deviations, the simulation runs are repeated 50 times. Finally, we calculate the average of the waiting time for an interval over all runs. To make the settings comparable, the integral of the values is calculated using the Romberg method [314], which is a numerical procedure for integral estimation.

Evaluation Execution

As baseline measurements of the adaptation logic's performance, we first ran two series of 50 simulations each without an adaptation logic as well as with an adaptation logic using a fixed rule set for both tracks. The self-improvement layer is inactive. Hence, there are no rules added during the simulation and structural self-improvement is not active. The adaptation logic is able to set three different speed limits. The speed limit is set to unrestricted if the status of the highway is free. In case of congested traffic, a limit of 120 km/h is set and finally, for stop-and-go traffic the speed limit is reduced to 80 km/h. Congested traffic and stop-and-go traffic situations are detected using measurements of the amount of vehicles on the track as well as their average velocity.

As a second measurement, we measured the performance of the parametric self-improvement, hence, the rule learner. The recommended speed limits are learned while the system is running. The system starts with an empty rule set and the rule set evolves over time.

It is not preferable to learn rules for a spontaneous, non-durable event as time is needed for the simulation to learn rules. Contrary, structural adaptation can react fast to changing conditions. Therefore, we decided to introduce additional events to the evaluation setting that trigger structural self-improvement of the adaptation logic. Reaction to events is not restricted to reactive adaptation only, but includes a proactive adaptation as reaction to forecasted events. We additionally simulate an accident that leads to closing the ramp of the junction in section 1. Additionally, we introduce a daily road work in the last section of the track. Further, as structural adaptation is event-based, we simulated the road work during the morning rush hour between 6am and 11am. The blocked parts of the highway are shaded in Figure 8.10. For the detailed evaluation parameters, the interested reader is referred to [223].

Evaluation Results & Discussion

Table 8.6 subsumes the results of the evaluation. The integral of the baseline without the adaptation logic is 21,308, the one with the adaptation logic having a fixed rule set decreases to 14,950, which is an improvement of 30 %. However, as traffic jams happen even with the adaptation logic, this is an indicator that the adaptation logic could be improved though meta-adaptation. Adding the parametric self-improvement module for rule learning, the integral of the measurement decreases by further 1,139 points which is a 7.6 % improvement compared to a fixed rule set and an improvement of 35 % compared to the baseline.

Table 8.6.: Aggregated average waiting time

Setting	Integral	Decrease
Without AL (scenario 1)	21,308	—
Static AL (scenario 1)	14,950	(−30 %)
Parameter	13,811	(−35 %)
Without AL (scenario 2)	18,128	—
Static AL (scenario 2)	13,032	(−28 %)
Structure Neo4j	14,055	(−23 %)
Structure TARL	13,713	(−24 %)

For the setting with the daily road work, the integral of the baseline without the adaptation logic is 18,128, the one with the static adaptation logic decreases to 13,032. This represents an improvement of 28 %. For measuring the effects of structural self-improvement, we compared the Neo4j and the TARL modules. The Neo4j module has an integral of 14,055, the TARL module performs slightly better with 13,713. Both, Neo4j and TARL, do not outperform the static adaptation logic. Several optimizations are possible: (i) learn new rules for the situation, (ii) optimize the speed limits with variable speed limits for the sections of a region, and (iii) optimize the parameters for prediction. However, they still improve the traffic compared to the situation without adaptation by more than 23 %. Additionally, the regions homogenize the traffic flow which increases safety [364].

The results support the benefits of self-improvement in the studied use case as well as the concept of the ALM. Accordingly, for the studied scenario, it can be concluded that parametric self-improvement increased the performance (cf. proposition $PR_{DA}1$) while structural self-improvement do not improve the quantitative performance, however, still offers benefits (cf. proposition $PR_{DA}2$). As for both studied types of self-improvement the performance of the SAS does not decrease and both offer advantages, proposition $PR_{DA}3$ holds. [223] provides a detailed discussion of the performance of the modules.

8.4. Assessment-based Descriptive Evaluation

This section presents *informed arguments* (cf. [180]) – a type of descriptive evaluations – for assessing the FESAS Framework and the ALM. Therefore, this section compares them with the knowledge base (cf. Section 4.2 & 4.3), i.e., the related work from [224, 226, 227]. We neglect the FESAS IDE here as the tools in literature show a high variance regarding their functionality and many tools are not publicly available for testing. Next, Section 8.4.1 presents the assessment of the FESAS Framework. Afterwards, Section 8.4.2 evaluates the ALM.

8.4.1. Informed Argument: Assessment of the FESAS Framework

This section provides an informed argument (cf. [180]) evaluation that compares the FESAS Framework to related work discussed in [224] (cf. Section 4.2).

Evaluation Design

Following the protocol presented in the introduction of this chapter, this section presents the metrics for the evaluation, categorizes the FESAS Framework using the taxonomy from [224] (cf. Section 4.2), and compares it to related work.

Evaluation Questions: The evaluation covers the following questions:

$EQ_{IAF}1$ Does the FESAS Framework differ from related approaches?

$EQ_{IAF}2$ Does the FESAS Framework offer more reusability as related work?

Evaluation Proposition: The main proposition is that the FESAS Framework offers more reusability as related work. This can be refined to the criteria:

$PR_{IAF}1$ The FESAS Framework offers more reusability on the MAPE component level as related approaches.

$PR_{IAF}2$ The FESAS Framework offers more flexibility for developers in implementing the MAPE functionality as related approaches.

$PR_{IAF}3$ The FESAS Framework offers a better integration of design/development actives and runtime support as related approaches.

Units of Analysis: Relevant are the related work from [224] (cf. Section 4.2) and the taxonomy from [224] for the categorization of the FESAS Framework.

Linking Data to Propositions: We categorize the FESAS Framework using dimensions of the taxonomy from [224] that are related to the propositions:

- $PR_{IAF}1$ is linked to the dimensions *level of abstraction*, *reusability*, *libraries*, and *reference architecture*.

- $PR_{IAF}2$ is linked to the dimensions *type of support*, *support of adaptation mechanisms*, *type of approach*, *special demands on developer*, *applicability*, *type of adaptation*, and *language specificity.*
- $PR_{IAF}3$ is linked to the dimensions *temporal scope of support*, *use of processes*, *use of tools*, *involved roles*, and *development phase.*

Interpretation Criteria: Differing manifestations in the dimensions of the taxonomy for the FESAS Framework in contrast to related work are seen as support of the propositions.

Evaluation Execution

The execution followed a qualitative approach. The FESAS Framework was assessed in the aforementioned described categories of the taxonomy from [224].

Evaluation Results & Discussion

This section compares the FESAS Framework with related approaches. The dimension *engineering context* is not further analyzed as it is equal for almost every approach. Table 8.7 presents an overview of the categorization of the FESAS Framework and related approaches with similar manifestations.

Proposition $PR_{IAF}1$ is linked to the dimensions *level of abstraction*, *reusability*, *libraries*, and *reference architecture.* Whereas several approaches offer a similar *level of abstraction* [5, 37, 66, 140, 163, 360] and also integrate a MAPE-K based reference architecture [65, 112, 215, 247, 360, 375, 377, 398] or a similar reference architecture [5, 20, 66, 255, 336], besides the FESAS Framework, only three approaches integrate reusable components with processes [255,336,347] and only one accompany them with a pattern library [347]. In contrast to the FESAS Framework, [347] does not integrate a reference architecture implementation. This fully supports proposition $PR_{IAF}1$. Only the FESAS Framework focuses on reusability on the intra-component level, offers an implementation of the reference architecture, and integrates reusable processes and design/development elements. This enables a higher degree of reusability in contrast to related approaches.

The dimensions *type of support*, *support of adaptation mechanisms*, *type of approach*, *special demands on developer*, *applicability*, *type of adaptation*, and *language specificity* are related to the proposition $PR_{IAF}2$. Model-based approaches abstract from implementation details and are used to transfer design knowledge

Table 8.7.: Comparison of the FESAS Framework with related work

Dimension	FESAS Framework	Similar approaches
Type of approach	model-based	[10,11,37,66,112,163,215, 242,377,398]
Type of support	framework, tools, design concept	[215,377]
Temporal scope	design, deployment, runtime	[112]
Involved roles	Developer, Designer	Only [163, 215, 377] addresses several characteristics.
Reusability	Reusable processes and components	Only [255,336,347,377] offers both.
Development phase	Design & Implementation	[5,10,20,37,65,102,163,215, 255,331,336,347,377]
Engineering context	Forward engineering	Only [13, 151] support reverse engineering
Applicability	SAS	All except of [66,151,331]
Special requirements for developers	none	Most do not have special demands except [10, 112, 140,242,271,336]
Level of abstraction	Low	[5,37,66,140,163,360]
Use of processes	Workflow for development & deployment	Only [102] addresses design and runtime.
Use of reference architecture	FESAS Adaptation Logic Template	MAPE-K based [65, 112, 215, 247, 360, 375, 377, 398] and others [5, 20, 66, 255, 336].
Use of libraries	FESAS Repository	Catalog of coordination pattern [347]
Use of tools	FESAS IDE	[10,20,37,65,112,140,163, 215,247,255,360,377]
Language specificity	Prototype in Java	[10,20,65,112,140,163]
Support of adaptation mechanisms	At runtime through self-improvement	Only [377] can support self-improvement
Type of adaptation	Parameter, structural, self-improvement	None integrate self-improvement
Evaluation	Case studies	Only [10, 37, 77, 102, 112, 215, 255] offer several real implementations.

to the runtime. The adaptation logic can use this information for decision-making. In line with related approaches [10,11,37,66,112,163,215,242,377,398], the FESAS Framework is model-based to benefit from the mentioned advantages. Uniquely, the FESAS Framework integrates a framework, tools, and a design concept. None of the other approaches offer this whole set of development support. Another unique feature is the support for parametric and structural self-adaptation as well as self-improvement by the ALM. None of the other approaches offer this type of end-to-end support. However, the support at design time is not integrated out of the box and depends on the available functional logics for the MAPE components. Further, FESAS is not restricted to a specific type of SAS. Whereas many approaches claim this, only some approaches [10, 37, 77, 102, 112, 215, 255] offer implementations in several different case studies analogous to the FESAS Framework. The FESAS Framework conceptually does not have special demands on the developers. Only the prototype implementation requires Java knowledge, however, this seems to be a common approach for developing SASs (cf. [10,20,65,112,140,163]). Especially taking the exceptional characteristics of the FESAS Framework for the dimensions *type of support*, *type of adaptation*, and *support of adaptation mechanisms* into account, proposition $PR_{IAF}2$ is strengthened by the analysis.

Proposition $PR_{IAF}3$ is linked to *temporal scope of support*, *use of processes*, *use of tools*, *involved roles*, and *development phase*. Besides [102], only the FESAS Framework addresses development and runtime (through the ALM) in an integrated solution. In the development phase, the FESAS Framework supports design and implementation activities. Other approaches supports also both types [5,10,20,37,65,102,163,215,255,331,336,347,377]. However, except of the FESAS Framework, only [163] splits the development responsibilities into two roles. For the two activities, the FESAS Framework integrates two roles: developers and designers. This enables fine granular support of the development of SASs. Several approaches [10,20,37,65,112,140,163,215,247,255,360,377] offer tool support for the development of SASs. However, most tools support specific aspects only, rather than offering an integrated approach for connecting development and deployment/runtime. Accordingly, the FESAS Framework does not exclusively cover characteristics in all dimensions. *MUSIC* [163], *DESCARTES* [190, 215], and *EUREMA* [377,380] offer a broad support for designers and developers, however, they focus on a specific modeling approach and do not offer a high flexibility for integrating existing works. The combination of characteristics that the FESAS Framework offers, provides the broadest coverage of design and development activities and more flexibility as other approaches. The FESAS IDE integrates these activities. Hence, proposition $PR_{IAF}3$ is supported.

8.4.2. Informed Argument: Assessment of the Adaptation Logic Manager

This section provides an informed argument (cf. [180]) evaluation of the ALM through a comparison with related work [226, 227] (cf. Section 4.3).

Evaluation Design

The evaluation follows the protocol presented in the introduction of this chapter. After the presentation of the relevant metrics, this section categorizes the ALM using the taxonomy on self-adaptation from [229] (cf. Section 2.1.3) and compares it with the related work presented in [226, 227] (cf. Section 4.3).

Evaluation Questions: As Section 8.3.2 evaluated the functionality of the ALM, this assessment is centered around the following evaluation question:

$\boldsymbol{EQ_{IAA}1}$ Does the ALM offer more flexibility than related approaches?

Evaluation Proposition: The main proposition is that the ALM offers a higher flexibility as related approaches. This is refined to:

$\boldsymbol{PR_{IAA}1}$ The ALM supports reactive and proactive self-improvement.

$\boldsymbol{PR_{IAA}2}$ The ALM supports parametric and structural self-improvement.

$\boldsymbol{PR_{IAA}3}$ The ALM supports several adaptation decision criteria.

$\boldsymbol{PR_{IAA}4}$ The ALM supports centralized and decentralized decision making.

Units of Analysis: The collection of related work [226, 227] (cf. Section 4.3) and the taxonomy on self-adaptation from [229] (cf. Section 2.1.3) for the categorization of the ALM are the units used within the analysis.

Linking Data to Propositions: All dimensions of the taxonomy on self-adaptation from [229] (cf. Section 2.1.3) are used for categorizing the ALM. Especially, the following propositions and dimensions are linked to each other:

- $PR_{IAA}1$ is linked to the dimension *time*.
- $PR_{IAA}2$ is linked to the dimension *technique*.
- $PR_{IAA}3$ is linked to the dimension *decision criteria*.
- $PR_{IAA}4$ is linked to the dimension *degree of (de)centralization*.

Interpretation Criteria: Different manifestations in the dimensions for the ALM in contrast related work are seen as support of the propositions.

Evaluation Execution

This evaluation compares the categorization of the ALM to the categorizations of the other related approaches [226, 227] for analyzing the fulfillment of the propositions. Therefore, the ALM is analyzed and evaluated qualitatively. The results are presented in the following.

Evaluation Results & Discussion

In the following, this section discusses the ALM in contrast to related work from the overviews on self-improvement in [226, 227] (cf. Section 4.3). Table 8.8 shows the categorization of the ALM.

Table 8.8.: Comparison of the ALM with related work.

Dimension	**Characteristic ALM**	**Similar approaches**
Time	Reactive & Proactive	[9, 115, 363]
Reason	Implemented:Context & MR; Supported: User	Both are supported by [116, 121, 149, 194, 218, 273, 291, 342, 377].
Technique	Parameter & Structure	[273, 342, 363]
Adaptation Control: Approach	External	All, except of [121, 287, 377]
Adaptation Control: Decision Criteria	Implemented: Rules/Utility/Models; Supported: Goals	None support more than two
Adaptation Control: (De)centralization	Centralized (Prototype) & Decentralized (Concept)	None offer both

The ALM is in line with research for the dimensions *reason* and *approach for adaptation control*. Eight other approaches [116, 121, 149, 194, 218, 273, 291, 342] supported context and managed resources as reasons for self-improvement. Four of them [194, 218, 273, 291] even support all three possible reasons. All others (except [274]) support either context or managed resources. Except of two approaches [121, 287], all analyzed approaches for self-improvement follows the external design of separating adaptation logic and self-improvement logic.

As discussed in [226, 229], a proactive reaction might shorten delays and avoids different issues in the system, but should always be supported by a reactive one as backup. The ALM support reactive and proactive self-improvement simultaneously. Only three other approaches [9, 115, 363] offer this flexibility for the *time* dimension. All others support only one of the two characteristics. Hence, proposition $PR_{IAA}1$ is partly supported as only three approaches cover the *time* dimension identically. Similarly, the ALM supports both meta-adaptation *techniques*. Again, only three other approaches support also parametric and structural self-improvement [273,342,363]. For offering the highest flexibility to developers, both approaches should be supported. Further, the modularity of the ALM MAPE components additionally offers a unique flexibility through its reusability and extensibility. Accordingly, the analysis of the dimension *techniques* mainly supports proposition $PR_{IAA}2$.

Regarding the remaining two dimensions, the ALM provides unique features. The ALM is the only approach for self-improvement that is not restricted in its support of *decision criteria for adaptation control*. Again, this enables maximum flexibility for developers as they are not restricted to a given approach. Besides the offered three modules for self-improvement, the modularization enables them to integrate any approach they want to use. This way, many approaches studied for self-adaptation might be integrated and also used for self-improvement. Finally, the ALM is the only approach that support a centralized and decentralized/distributed self-improvement module. As we used the FESAS Framework prototype to build the ALM, this is supported. Decentralized decision making leads to faster reactions and locally optimized (meta-)adaptation decisions [401]. However, the main reason for the restriction of most approaches to *centralized adaptation control* for self-improvement might be the fact that self-improvement needs global knowledge for optimal decision making [226]. For that reason, we also focused on centralized decision making in our prototype of the ALM. A mechanism for supporting the decentralized decision making and sharing the knowledge for a global view is part of future work. However, the discussion of the dimensions *decision criteria for adaptation control* ($PR_{IAA}3$) and *(de)centralized adaptation control* ($PR_{IAA}4$) fully supports the remaining propositions.

As seen in the discussion of related work, the ALM provides several features which are unique or at least not commonly present in related work. From the studied approaches, [363] is the closest as it offers flexibility by integrating proactive and reactive self-improvement as well as offering parametric and structural self-improvement. However, the ALM is unique in the support of adaptation decision criteria and the supported degree of (de)centralization.

9. Discussion

The evaluation showed that the FESAS Framework with the ALM offers a unique type of support for developing SASs. Similar approaches like *EUREMA* [377,380] or *MUSIC* [163] also target the full stack of development support – design of the SAS and the development of the adaptation logic's components – or runtime support for self-improvement (e.g., [112, 360, 377]). However, they do not provide the same flexibility in development as the FESAS Framework. The FESAS Framework abstracts from any specific modeling approaches and enables the inclusion of existing work as long as it can be transformed to the MAPE model. In contrast, related approaches target directly to model and implement support for adaptation reasoning, i.e., targets the MAPE functionality. They (i) come with the necessity to learn and apply a specific modeling approach, (ii) are limited to a specific component model, or (iii) are limited in their support for adaptation techniques. The FESAS Framework is more flexible and enables developers to integrate different approaches and concepts for adaptation reasoning through its openness. Further, the FESAS Repository acts as an "app store" and offers MAPE functionality to developers. Accordingly, it is possible to build the adaptation logic without the need to implement the MAPE functionality.

Based on the evaluation presented in the previous section, this section discusses the results of this thesis. It is structured similar to [372] in theoretical contributions, limitations, theoretical implications, and practical implications. Section 9.1 explains the theoretical contributions, i.e., how this thesis contributes to the research questions and the identified requirements derived from the knowledge base. It further analyzes threats for validity. Based on that, Section 9.2 explains limitations of the prototypes for the FESAS Framework, the FESAS IDE, and the ALM and how to address them in future work. Section 9.3 relates the results of the thesis to the research trends for SASs. Therefore, it presents current challenges in the field and how the FESAS Framework contributes to them. Last, Section 9.4 explains contributions of this thesis for practitioners.

9.1. Theoretical Contributions

This section describes the theoretical contributions of this thesis to extend the knowledge base, i.e., the compliance with the requirements and the research questions. First, Section 9.1.1 discusses the results of the evaluations and the fulfillment of the evaluation propositions and maps the evaluation results to the research questions. Second, based on the evaluation results, Section 9.1.2 analyzes the coverage of the requirements defined in Chapter 5. Third, Section 9.1.3 describes relevant threats for validity of the evaluation results. Based on these three elements, the following Section 9.2 depicts potential future work for the FESAS Framework, the FESAS IDE, and the ALM.

9.1.1. Discussion of Evaluation Results

This section discusses the results of the evaluation. Next, it summarizes the support of the evaluation results for the evaluation propositions. Afterwards, it maps the support of evaluation propositions to the research questions defined in Section 1.2.

Discussion of the Evaluation Propositions

Table 9.2 provides an overview of the evaluation propositions defined for the analytical, observational, experimental, and descriptive evaluations of the FESAS Framework, the FESAS IDE, and the ALM presented in the Sections 8.2-8.4. As the column "Support" shows, none of the evaluation propositions is discarded by the evaluation results. For a detailed discussion of the fully supported propositions (marked with ✓), the reader is referred to the presentations of the corresponding evaluation in the Sections 8.2-8.4. Here, we discuss the six partly supported evaluation propositions (indicated by the (✓) symbol) out of the 21 evaluation propositions in total.

Proposition $\boldsymbol{PR_{EE}4}$. The FESAS IDE offers a mode for testing the adaptation logic without the need to set up the whole SAS, i.e., avoiding a connection to managed resources. The results of the experimental evaluation (cf. Section 8.2.1) indicate that the users of the FESAS IDE see potential for improvement in the

Table 9.1.: Coverage of the evaluation propositions for the evaluations of the FESAS Framework, the FESAS IDE, and the ALM (✓indicates full support, (✓) indicates partly support).

Identifier		**Proposition**	**Support**
Usability FESAS IDE	$\boldsymbol{PR_{EE}1}$	The FESAS IDE does not require developers to learn the FESAS Framework workflow.	✓
	$\boldsymbol{PR_{EE}2}$	The FESAS IDE is easy to learn and easy to use.	✓
	$\boldsymbol{PR_{EE}3}$	The FESAS IDE supports the development of reusable SASs.	✓
	$\boldsymbol{PR_{EE}4}$	The FESAS IDE supports the testing of SASs.	(✓)
	$\boldsymbol{PR_{EE}5}$	The FESAS IDE accelerates the development of SASs.	✓
Applicability FESAS IDE	$\boldsymbol{PR_{FS}1}$	The FESAS IDE simplifies the initialization and configuration of the FESAS development environment.	✓
	$\boldsymbol{PR_{FS}2}$	The FESAS IDE simplifies the development of SASs compared to manually using the FESAS Framework components and corresponding workflow.	✓
	$\boldsymbol{PR_{FS}3}$	The perceived benefits of using the FESAS Framework for developing SASs are increased when using the FESAS IDE.	✓
Reusability FESAS	$\boldsymbol{PR_{SA}1}$	Except the functional logic code, most parts of SASs implemented using the FESAS Framework are reusable.	✓
	$\boldsymbol{PR_{SA}2}$	The MAPE components generated with the FESAS Framework are reusable with the exception of clearly defined parts.	✓
	$\boldsymbol{PR_{SA}3}$	The system models are reusable with minor customizations, only.	(✓)
Perf. ALM	$\boldsymbol{PR_{DA}1}$	Parametric self-improvement enhances the system performance.	✓
	$\boldsymbol{PR_{DA}2}$	Structural self-improvement enhances the system performance.	(✓)
	$\boldsymbol{PR_{DA}3}$	Self-improvement in general enhances the system performance.	(✓)
Assessment FESAS	$\boldsymbol{PR_{IAF}1}$	The FESAS Framework offers more reusability on the MAPE component level as related approaches.	✓
	$\boldsymbol{PR_{IAF}2}$	The FESAS Framework offers more flexibility for developers in implementing the MAPE functionality as related approaches.	✓
	$\boldsymbol{PR_{IAF}3}$	The FESAS Framework offers a better integration of design/development actives and runtime support as related approaches.	(✓)
Assessm. ALM	$\boldsymbol{PR_{IAA}1}$	The ALM supports reactive and proactive self-improvement.	✓
	$\boldsymbol{PR_{IAA}2}$	The ALM supports parametric and structural self-improvement.	✓
	$\boldsymbol{PR_{IAA}3}$	The ALM supports several adaptation decision criteria.	✓
	$\boldsymbol{PR_{IAA}4}$	The ALM supports centralized and decentralized decision making.	(✓)

usability of the testing mode (proposition $PR_{EE}4$). One reason might be that the mode is not perfectly integrated into the FESAS IDE. The FESAS IDE just creates the relevant Java classes. Running the testing mode and automatic analysis of the results are not integrated into the FESAS IDE.

Proposition $\boldsymbol{PR_{SA}3}$. The static analysis (cf. Section 8.3.1) of the degree of reusability with the FESAS Framework shows that the system models can be reused (proposition $PR_{SA}3$). This is supported by an analysis of the use case systems presented in Section 8.1 as well as by the interviews with developers who used the FESAS Framework or the FESAS IDE. However, this bases on rather small systems. For large, highly distributed adaptation logics, this might not be the case, hence, the proposition can only be partly supported by the evaluation results.

Propositions $\boldsymbol{PR_{DA}2}$&$\boldsymbol{PR_{DA}3}$. We analyzed the performance of the ALM and the implemented modules for analyzing and planning self-improvement in a dynamic analysis (cf. Section 8.3.2). There, we found that parametric self-improvement offers a quantitative measurable performance gain in the use case, whereas structural self-improvement offers only a qualitative deducible advantage: increased safety through homogenizing the traffic flow. Hence, the proposition $PR_{DA}2$ – describing that structural self-improvement enhances the system performance – is only partly supported by the results of the SmartHighway use case as only qualitative benefits are perceived. Accordingly, also proposition $PR_{DA}3$ – which targets the improvement for self-improvement in general – can only be partly supported.

Proposition $\boldsymbol{PR_{IAF}3}$. Proposition $PR_{IAF}3$ claims that the FESAS Framework offers a better integration of design/development activities with runtime support compared to related approaches. This was evaluated based on a comparison with the knowledge base (cf. Section 8.4.1). The evaluation indicates that the FESAS Framework offers a better integration than most of the related approaches. Some approaches also integrate design/development activities with runtime support, e.g., *DESCARTES* [190, 215], *EUREMA* [377, 380], *FUSION* [112], or *ArchStudio* [278, 279]. However, these approaches are either more specialized (e.g., *EUREMA* [377, 380] or *FUSION* [112]) which limit their flexibility to integrate other existing approaches or do not offer self-contained runtime support comparable to the ALM (e.g., *DESCARTES* [190, 215] or *Arch-*

Studio [278, 279]). Still, we acknowledge that these approaches have capabilities similar to the FESAS Framework, such as model-based development, implementation of adaptation logic components, or (partly) support for self-improvement. Accordingly, proposition $PR_{IAF}3$ is only partly fulfilled.

Proposition $\boldsymbol{PR_{IAA}4}$. Section 8.4.1 compares the ALM to the knowledge base focusing on the dimensions adaptation decision criteria, adaptation time, adaptation technique, and adaptation control. Whereas the first three factors are fully supported by the ALM, the last dimension is only partly supported (proposition $PR_{IAA}4$). The ALM supports centralized and decentralized distribution of the decision making through the implementation using the FESAS Framework. However, we did not evaluate decentralized decision making in the ALM within this thesis.

Discussion of the Research Questions

All of the former presented evaluation propositions contribute to answer one of the three research questions we motivated in Section 1.2:

***RQ*1 Reusability**: How to make the adaptation logic more reusable?

***RQ*2 Self-improvement**: How to adapt the adaptation logic at runtime?

***RQ*3 Integrated development**: How to support the development of SASs with tools and processes throughout the complete lifecycle?

RQ1 Reusability. First, research question $RQ1$ targets the most important concern for this thesis: *reusability* of elements of the adaptation logic. Within the evaluation, we proved that the FESAS Framework supports this reusability aspect. The static analysis of the degree of reusability achieved with the FESAS Framework (cf. Section 8.3.1) clearly proves that using the FESAS Framework, developers are able to reuse (i) code for the functional logics (proposition $PR_{SA}1$), (ii) the MAPE components of the reference implementations (proposition $PR_{SA}2$), and (iii) the system models with only minor customization (proposition $PR_{SA}3$). Further, the descriptive evaluation (cf. Section 8.4.1) shows that the FESAS Framework does not only focus more on the reusability aspect than other related approaches (proposition $PR_{IAF}1$), but also targets a higher flexibility for enabling developers to better integrate existing approaches (proposition $PR_{IAF}2$). Accordingly, research question $RQ1$ is successfully addressed in this thesis.

RQ2 Self-improvement. The second research question $RQ2$ demands the inclusion of an approach for *self-improvement*. Therefore, we implemented the ALM. It is self-contained, but well integrated with the FESAS Framework. We evaluated the ALM in the SmartHighway scenario. The support of the evaluation propositions $PR_{DA}1$-$PR_{DA}3$ shows that all implemented modules for planning self-improvement influence the qualitative aspects of the adaptation logic. The parametric self-improvement module also significantly enhances the performance in quantitative measurements. Further, the ALM offers a higher flexibility than other approaches as it supports (i) reactive and proactive self-improvement (proposition $PR_{IAA}1$), (ii) parametric and structural self-improvement (proposition $PR_{IAA}2$), (iii) several adaptation decision criteria (proposition $PR_{IAA}3$), and (iv) centralized and decentralized decision making (proposition $PR_{IAA}4$). Except for the decentralized decision making for self-improvement, the evaluation results supported all the propositions. Accordingly, this thesis answers research question $RQ2$.

RQ3 Integrated development. Third, research question $RQ3$ focuses on the *integration* of development activities with runtime support. In this thesis, we presented three elements: (i) the FESAS IDE for the design and development of SAS, (ii) the FESAS Framework for integrating a reference implementation of the adaptation logic and deployment support, as well as (iii) the ALM for runtime support. These elements are connected with the FESAS Workflow. The evaluation showed that the FESAS IDE is a central element for connecting all these elements (propositions $PR_{FS}1$-$PR_{FS}3$) and increasing their usability by abstracting from learning details of them (propositions $PR_{EE}1$-$PR_{EE}5$). Compared to related work, the FESAS Framework's components in combination with the FESAS IDE and the ALM offer (i) a better integration of design/development actives and runtime support as most related approaches and (ii) an integration of these elements while observing the flexibility to integrate different development paradigms and existing approaches (proposition $PR_{IAF}3$). Consequently, research question $RQ3$ is confirmed by the evaluation propositions.

Table 9.2.: Mapping of the evaluation propositions for the evaluations of the FESAS Framework, the FESAS IDE, and the ALM to the research questions (✓indicates full support, (✓) indicates partly support).

Research Question		Proposition	Support
RQ1 Reusability	$PR_{SA}1$	Except the functional logic code, most parts of SASs implemented using the FESAS Framework are reusable.	✓
	$PR_{SA}2$	The MAPE components generated with the FESAS Framework are reusable with the exception of clearly defined parts.	✓
	$PR_{SA}3$	The system models are reusable with minor customizations, only.	(✓)
	$PR_{IAF}1$	The FESAS Framework offers more reusability on the MAPE component level as related approaches.	✓
	$PR_{IAF}2$	The FESAS Framework offers more flexibility for developers in implementing the MAPE functionality as related approaches.	✓
RQ2 Self-improvement	$PR_{DA}1$	Parametric self-improvement enhances the system performance.	✓
	$PR_{DA}2$	Structural self-improvement enhances the system performance.	(✓)
	$PR_{DA}3$	Self-improvement in general enhances the system performance.	(✓)
	$PR_{IAA}1$	The ALM supports reactive and proactive self-improvement.	✓
	$PR_{IAA}2$	The ALM supports parametric and structural self-improvement.	✓
	$PR_{IAA}3$	The ALM supports several adaptation decision criteria.	✓
	$PR_{IAA}4$	The ALM supports centralized and decentralized decision making.	(✓)
RQ3 Integrated development	$PR_{EE}1$	The FESAS IDE does not require developers to learn the FESAS Framework workflow.	✓
	$PR_{EE}2$	The FESAS IDE is easy to learn and easy to use.	✓
	$PR_{EE}3$	The FESAS IDE supports the development of reusable SASs.	✓
	$PR_{EE}4$	The FESAS IDE supports the testing of SASs.	(✓)
	$PR_{EE}5$	The FESAS IDE accelerates the development of SASs.	✓
	$PR_{FS}1$	The FESAS IDE simplifies the initialization and configuration of the FESAS development environment.	✓
	$PR_{FS}2$	The FESAS IDE simplifies the development of SASs compared to manually using the FESAS Framework components and corresponding workflow.	✓
	$PR_{FS}3$	The perceived benefits of using the FESAS Framework for developing SASs are increased when using the FESAS IDE.	✓
	$PR_{IAF}3$	The FESAS Framework offers a better integration of design/development actives and runtime support as related approaches.	(✓)

9.1.2. Requirements Coverage

Based on the evaluation results, this section discusses the requirements coverage of the FESAS Framework and the ALM. As assessment, the requirements from Section 5.2 and Section 5.3 are discussed individually.

Requirements Coverage: FESAS Framework

$\boldsymbol{R_{Dev}}$1: External Control. The FESAS Adaptation Logic Template supports the separation of adaptation logic elements and the managed resources (requirement $R_{Dev}1$). Moreover, it specifies clear interfaces for interaction between both parts. This increases reusability, as the adaptation logic is not adjusted to specific managed resources.

$\boldsymbol{R_{Dev}}$2: Reference Architecture. The FESAS Adaptation Logic Template enables the fulfillment of requirement $R_{Dev}2$. It integrates the well-known MAPE control functionality (requirement $R_{Dev}2.i$) with additive components for supporting the deployment of the adaptation logic and offers runtime support for the MAPE components (requirement $R_{Dev}2.ii$). Furthermore, the FESAS Component Template abstracts from unnecessary implementation details and enables the developers to focus on the MAPE components' functionality.

$\boldsymbol{R_{Dev}}$3: Flexible Adaptation Control. The evaluation results for the example cases show that the reference system can be reused without adjustments. This significantly increases the level of code reuse within the different systems as well as reduces complexity in the development of SASs. The FESAS IDE hides the complexity in using the reference system. For the adaptation logic, developers only have to implement functional logic elements. Further, the reused code handles issues that the developers do not have to cover, such as the communication between the MAPE elements (requirement $R_{Dev}3.i$) and the deployment of the adaptation logic (requirement $R_{Dev}3.ii$). As it is highly reusable, it reduces the complexity of the development of SASs significantly.

$\boldsymbol{R_{Dev}}$4: Context Management. An open issue is the context management (requirement $R_{Dev}4$). The idea of the context manager is to systematically abstract the adaptation logic from specific context information of managed resources. Therefore, a generic context model is needed as well as a mechanism for handling the context data. We experimented with a JMS-based approach. However, this is only suitable for fully fledged systems due to the need of a data base and JMS broker. Further, the approach is not fully integrated into the FESAS Middleware and we did not integrate a systematical mapping of context information to a context model, such as the one presented in [329].

$\boldsymbol{R_{Dev}}$5: Generic Adaptation Support. The high variety of use cases shows the generic applicability of the FESAS Framework. Within these use cases, different types of adaptations can be encountered: parametric, structural, and contextual adaptation as well as reactive and proactive adaptation. Further, the use cases integrated all adaptation decision criteria, i.e., rules/policies, goals, models, and utility functions. The FESAS Framework and the FESAS IDE support all of these facets through the modularization offered by the FESAS Component Template (requirements $R_{Dev}5.i-a$ – $R_{Dev}5.i-c$). Furthermore, the FESAS Component Template enables the encapsulation of the adaptation mechanisms (requirement $R_{Dev}5.ii$) which also increases their reusability. Additionally, the analysis of the example systems shows that the integration of existing code is simplified through the use of the *FESAS Development Tool* as it offers a clear interface for adding code based on the modularization principle.

$\boldsymbol{R_{Dev}}$6: Connection to Managed Resources. For each use case in Section 8.1, the adaptation logic is implemented in Java using the FESAS Framework and FESAS IDE. However, the implementation of the managed resources varies. We use Java, JavaEE, Python, C++, and Lejos (a JavaVM for Lego Mindstorms robots). For some systems, the managed resources are simulated in a simplified way. For others, we use fully fledged simulators, robots, or real life systems. The connection between managed resources and the adaptation logic follows various communication approaches: HTTP requests to web servers, XML-RPC, Java-based method calls, communication middlewares, and sockets connecting the Java-based adaptation logic with managed resources implemented in Java or other programming languages. The FESAS Framework offers various communication modules to support the variance of communication approaches (requirement $R_{Dev}6.i$). Through the generic FESAS Adaptation Logic Template and its interfaces, the connection between adaptation logic and managed resources can be adjusted by changing the functional logic and/or properties of a sensor/effector component using the FESAS IDE (requirement $R_{Dev}6.ii$).

Table 9.3.: Coverage of the requirements by the implementation of the FESAS Framework / FESAS IDE and the results of the evaluation.

Requirement	Explanation	Support
R_{Dev}1: External control	Implemented as external layer.	✓
R_{Dev}2: Reference architecture	The FESAS Adaptation Logic Template separates MAPE components and additive one.	✓
R_{Dev}3: Flexible adaptation control	Different patterns supported and deployment mechanism integrated.	✓
R_{Dev}4: Context management	Prototype exists; context modeling missing.	(✓)
R_{Dev}5: Generic adaptation support	Generated code supports the full spectrum of adaptation mechanisms.	✓
R_{Dev}6: Connection to managed resources	The FESAS Framework offers various communication modules. The FESAS IDE supports the configuration of the communication.	✓
R_{Dev}7: Integrated development	FESAS IDE, FESAS Middleware, and ALM support the whole lifecycle and are integrated.	✓

R_{Dev}7: Integrated Development. Without the FESAS IDE, developers are required to learn the syntax for the metadata to describe code of the functional logics. The same is true for the system designer who would need to learn the syntax of the configuration files. Having both integrated into Eclipse enables to hide the details of the syntax. Further, the integration into Eclipse enables the use of implementation workflows known from IDEs. This reduces the learning time. The FESAS Middleware supports the deployment of the adaptation logic at runtime and the connection to the managed resources. Further, the ALM adds self-improvement. Accordingly, requirement $R_{Dev}7$ is fully covered.

Requirements Coverage: Adaptation Logic Manager

R_{SI}1: Meta-adaptation Mechanisms. The evaluation shows that the ALM enables self-improvement with different types of meta-adaptation of the adaptation logic (requirement $R_{SI}1.i$). To consider application-specific characteristics, we propose the integration of structural and parameter meta-adaptation of the adaptation logic. This increases the flexibility for developers. The modularization concept enables to exchange analyzing and planning algorithms (requirement $R_{SI}1.ii$).

***R_{SI}*2: External Control.** The idea of adding a self-improvement layer to an SAS offers a generic approach for self-improvement that can be customized through plugging in modules for reasoning. Through the introduction of the ALM as an external layer (requirement $R_{SI}2$), we improved its maintainability and reusability.

***R_{SI}*3: Monitoring.** The ALM monitors the adaptation logic and captures data of the managed resources (requirement $R_{SI}3$). The Proxy ALM periodically sends the relevant data. The managed resources' data is stored in a database using a generic approach. However, it needs minor configuration by specifying the expected data structure. The information of the adaptation logic's structure is stored in a graph and accessible for modules for reasoning. Developers of modules can rely on these mechanisms and do not need to implement monitoring.

***R_{SI}*4: Proactive Reasoning.** An approach for self-improvement should combine reactive as well as proactive reasoning for compliance in various applications. Proactive self-improvement is supported by the prediction module, that offers a prediction of future states (requirement $R_{SI}4$). The encapsulated prediction, based on WEKA [185], can be used out of the box by developers with minor configuration only.

***R_{SI}*5: Flexible Control.** Existing approaches for self-improvement focus on centralized reasoning [226]. Our system design supports the definition of a decentralized self-improvement layer as the various elements are encapsulated. Furthermore, the FESAS Framework and its system model offer the possibility to distribute the components, hence, they support different degrees of distribution and decentralization (requirement $R_{SI}5$).

***R_{SI}*6: Integration.** As the ALM is self-contained, all frameworks for building SASs that offer interfaces for changing the adaptation logic and a Proxy ALM implementation can be connected to the ALM (requirement $R_{SI}6$). In the evaluation, we showed this for the FESAS Framework.

We discussed in this thesis the layered version, only. The modularized version from [230] offers similar characteristics. Additionally, it is a second use case which supports the claim for flexibility (addressing requirements $R_{SI}1$, $R_{SI}2$, and $R_{SI}6$). Table 9.4 summarizes the coverage of the requirements.

Table 9.4.: Coverage of the requirements by the implementation of the ALM and the results of the evaluation.

Requirement	Explanation	Support
R_{SI}1: Meta-adaptation mechanisms	Modules for parametric and structural self-improvement tested. Different modules integrated in ALM.	✓
R_{SI}2: External control	Implemented as external layer.	✓
R_{SI}3: Monitoring	Generic monitoring integrated in ALM.	✓
R_{SI}4: Proactive reasoning	Prediction integrated in ALM.	✓
R_{SI}5: Flexible control	Decentralization supported by design; not evaluated.	✗
R_{SI}6: Integration	Self-contained, clearly defined interfaces; not evaluated with other frameworks.	(✓)

9.1.3. Threats to Validity

In line with other works (e.g. [241]), this section discusses potential threats for validity of the evaluation results based on the principles of experiments[1] in software engineering as presented in [406].

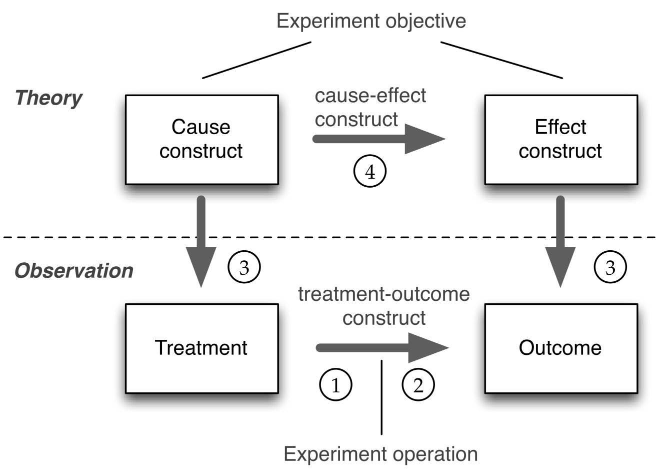


Figure 9.1.: Principles of experiments (taken from [241, p. 203], based on [406]). The theoretically proposed cause-effect relation is mapped to observations, that are analyzed to confirm the proposed relations.

The upper part of Figure 9.1 shows the objective of an experiment. With the help of an experiment, scientists want to prove the relation between a *cause* and

[1]Note: The term "experiment" differs from the more narrow meaning of the term "experiment" used in Chapter 8. Here, experiment refers to an evaluation in general.

an *effect* as proposed in their theory, the so called *cause-effect construct*. The lower part of Figure 9.1 depicts the mapping of the evaluation objective to the actual execution of the experiment. For the execution, different *treatments* can be defined and executed. The *outcomes* of these treatments must be analyzed to identify the support of them for the proposed theory. Several types of validity are present, each indicated by the corresponding number in Figure 9.1 [241, 406]:

1. **Conclusion Validity:** This describes a statistically proven relation between the treatment and the outcome of an experiment.
2. **Internal Validity:** Given conclusion validity, this property ensures that only variables under control influence the results.
3. **Construct Validity:** This type of validity describes the correct reflection of cause to treatment and effect to outcome.
4. **External Validity:** This type of validity is concerned with the generalization of the results.

Threats to Conclusion Validity

Parts of the evaluation are based on qualitative measurements and, hence, subjective (cf. [20, 374, 377]). To address this, we try to provide quantitative evidence if possible. Additionally, for the results of the interviews with developers especially for the evaluation of the FESAS IDE we emphasize that the results are subjective as perceived by the participant of the studies. This subjectivity of the metrics for the FESAS IDE might also support "fishing" for results. To avoid that, we used standard measurements in form of Likert scales to codify the answers of students and provide the full results for transparency.

While being accepted in literature (e.g., [77, 377]), the metric SLoC has some drawbacks. First, code can be written differently which influences the results. As mainly Master students participated, one can assume that these students have a comparable level of development knowledge and, hence, the SLoC numbers are comparable. Thus, SLoC can act as a quantitative measurement. Ideally, we could analyze the "intelligence" of the code that has been produced with the FESAS IDE. On the one hand, this is difficult to measure as the quality of the adaptations is highly use case specific. On the other hand, the intention was not to evaluate the performance of systems generated with the FESAS Framework but to show that the developers only have to implement a minor part of the adaptation logic. Therefore, we think SLoC is an acceptable indicator.

Threats to Internal Validity

The heterogeneity of the groups of participants can be a threat to internal validity. Indeed, the students have different study backgrounds and experiences with development. However, we did not account any effects. In general, to address this, control groups can be used. This is the case for the field study presented in Section 8.2.2. There, we tried to avoid differences in the control groups by composing the two groups in equal sizes and with students that attended the same lecture.

The instrumentation for the studies can be an additional threat for internal validity. This includes the design of the questionnaires as well as the setup of the analytical evaluations. We tried to tackle this by using references for the design of the questionnaires, such as the *ISO 9241-11 Guidance on Usability* standard and the definition of usability in the *ISO/IEC 9126-1 Software Product Quality Model* standard. For the static analysis of the degree of reusability, we integrated various systems from different application domains. The dynamic analysis is based on waiting time as the performance metric since it describes the system performance relevant to the users. Additionally, we used real traffic data for a realistic setting. Further, the heterogeneity of the various evaluations is a threat for internal validity. To handle this, we applied the structured evaluation protocol from Luckey [241] which is based on [408] in all evaluations except of testing.

Threats to Construct Validity

Mono-operation bias is a common threat for construct validity. To avoid this, most evaluations base on several use case systems for different application domains. Exceptional for the ALM, we analyzed a single case only in this work. However, [230] presents the application of the ALM in a second use case. Also for the FESAS IDE, we performed a single case analysis. A comparison with other tools might be an option for future work. This was already planned for this thesis, however, this failed due to several issues: The access to some of the tools was restricted as they are not publicly available, whereas other tools were not usable as the documentation was incomplete or their code dependencies were outdated.

As subjects in tests usually know the purpose of the evaluations, it might be possible that they want to support the evaluation with suitable answers. In case of students as participants, this effect can be strengthen as they are graded. To avoid this, the questionnaires were anonymized for interpretation and, furthermore, the work of the students was not graded based on their quality. Students only received a bonus for participation.

An additional threat is the expectancies of the researcher. Often, a single person interprets the results. The interpretations might be biased by the expectations of this person. In combination with subjectivity of the metrics introduced by the qualitative evaluations, this threat can be intensified. For this study, we reduced the risk of a threat for construct validity introduced by the expectancies of the researcher through the application of the evaluation protocol, transparency of results and interpretations, as well as the use of quantitative metrics if applicable.

Besides possible threats resulting from the chosen metrics, another threat results from the mapping of the objectives to the treatment in the evaluation. One possible issue here can be the fact that some aspects of the FESAS Framework are evaluated using the FESAS IDE. This issue is reduced by having two evaluations for separating the usability of the FESAS IDE from the applicability of the FESAS Framework. Additionally, the comparison of two control groups – one using the FESAS Framework components, one the FESAS IDE – reduces this threat.

Threats to External Validity

Threats to external validity influence the generalization of the results. One issue for generalization of the results of the evaluation is the target group. In this work, we evaluated the FESAS IDE and the FESAS Framework with students and researchers. However, the target group also includes practitioners. An analysis of the generalizability of the results also for practitioners is an open issue.

In the research domain of SASs, the evaluation of frameworks focuses on the performance of the SASs (e.g. [177,374,377]). This is not the case in this thesis. Thus, it seems likely that it might be a threat for validity as the results of other publications cannot be compared with this thesis. However, the performance of the SASs tests the performance of MAPE algorithms. This is mapped to the performance of functional logics. Accordingly, this does not measure the performance of the support for development offered by the FESAS Framework. As we focus in this work on the development support, the performance of the developed SASs is a secondary issue and, hence, not the target of the evaluation.

As described in Section 9.1.3, the amount of example systems is large and covers a high variability, but the sizes of the systems are rather small. The same is true for the groups of developers that participated in the evaluations. Both can be potential threats for validity, especially for the measurements of the degree of reusable code and the usability as well as the applicability of the FESAS IDE. It might be possible that these results are not transferable to large systems or do not prove their statistical significance when integrating larger user groups. Accordingly, one stream for improvements can be to integrate larger systems and more users for more reliable results. One additional use case in which the FESAS Framework is currently applied is the distributed adaptive authentication system presented in [19].

Often, the applicability of development approaches for SASs is shown in only one or two use cases. This increases the threats to external validity as the results might be use case specific. In this work, we applied the FESAS Framework in nine different application domains to reduce the threat for external validity.

9.2. Limitations of the Prototypes

Based on the previously discussed theoretical contributions of the thesis to the research questions and the requirements derived from the related work, this section describes limitations and potential for future work to enhance the prototypes. The section has three parts. First, Section 9.2.1 presents the potential future work for the FESAS Framework. Second, Section 9.2.2 describes potential future work to improve the usability of the FESAS IDE. Third, Section 9.2.3 derives future work for the ALM from its requirement coverage.

9.2.1. Limitations of the FESAS Framework Prototype

Context Manager. Requirement $R_{Dev}4$ for the FESAS Framework describes the handling of context information. The idea is to separate the information sensed from the managed resources with the information used for reasoning in the adaptation logic. Therefore, the information needs to be mapped to a generic context model, so that the adaptation logic can reason on the generic structure. However, the definition of this context model is future work. Existing context models might be integrated, e.g., [329]. Additionally, the context manager's reference implementation is a prototype so far, which requires a fully-fledged device for the JMS server. This might contradict some use cases, e.g., in the IoT domain. Again, it might be reasonable to integrate related approaches for managing the context information especially in distributed settings with resource-poor devices, e.g., the context broker from the *PROACTIVE* framework [370].

Functional Logic Contracts. So far, the description of the contracts for functional logics of the MAPE components is not restricted. This provides developers the highest possible degree of flexibility and enables to integrate different approaches for modeling the contracts (e.g., [33, 44]). However, this comes with the disadvantage that the same properties might be described differently. For a more systematic mapping of functional logic attributes to the components, it might be beneficial to define a SAS component ontology. This can also improve the reasoning process for finding functional logic components in the FESAS Repository.

Pub/Sub Approach. Table 7.1 shows the trade-off between deployment knowledge and domain knowledge for topic registrations. Adding more deployment knowledge enables to register for information of specific components only, e.g., for a specific monitoring component instead of a specific type of monitoring data. However, for that, the mapping of responsibilities in the monitoring process to components must be known. Furthermore, this responsibility might change at runtime. Hence, we added the possibility to register for topics. Therefore, the developers do not require to have distribution knowledge. However, this might result in communication overhead and receiving information that an instance is not interested in. The analysis of this trade-off is part of future work.

Adaptation Execution. One important functionality for performing adaptation is quiescence detection. *Quiescence* describes a state of the system in which all processes are passive and no activation message is in transit [217, 252]. Weyns and Iftikhar [396] state that a SAS should be adapted only when it is in a quiescent state. This is especially important for structural adaptations, as otherwise components might refer to non-existing components. Contrary, as strong quiescence may decrease system performance because components that are not involved in the adaptation process might also need to reach a passive state, Ramirez *et al.* [304] argue to weaken this requirement. In accordance with Vandewoude *et al.* [367], they propose *tranquility* as a weaker requirement as it "*does not require neighboring components to reach passive states before a component undergoes a reconfiguration*" [304, p. 227]. So far, the FESAS Framework does not automatically offer quiescence detection to developers. One type of development support could be to add a functional logic based on [387], which enables execution of adaptation in decentralized settings while ensuring quiescence.

9.2.2. Limitations of the FESAS IDE Prototype

Self-* Patterns IDE. The evaluation of the FESAS Framework and the FESAS IDE is based on rather qualitative metrics, such as SLoC and interviews with developers. In an optimal case, the quality of the adaptation logic's code would be the metric. A possible metric for the quality of the resulting software can be the self-* properties. These properties are part of the functional logics that are implemented by the developer. We plan to extend the FESAS IDE with a mechanism to offer further support for the decisions of the developer, e.g., regarding self-* properties. With this, we could evaluate the performance of the FESAS IDE for automated development of the resulting SASs. Therefore, the FESAS IDE can be a first step to an IDE for self-* properties and could be used to learn, how to capture the relevant properties. For this vision, we need to find a way to map system requirements to adaptation requirements and system goals to adaptation goals. This includes the definition of metrics for adaptation requirements as well as the inclusion of a generic goal modeling approach and metrics for the goals.

Integrating Design and Development. As the design and the development of the adaptation logic is divided into two parts, changes in the design do not necessarily lead to changes in the development of code and vice versa. The need for changes in the code after changes in the design depends on the specific implementation of the logic elements. In case the design switches from a decentralized planner to a central approach, it can be necessary to change the planner's functional logic as it now relies on different analyzers' values and needs to combine their results. For future work, we plan to elaborate this and provide code fragments that support a specific type of design and, therefore, a specific decentralization pattern.

Testing. The evaluation showed that the use of the testing mode in the FESAS IDE was not fully satisfiable for the developers. So far, the developer can configure the test system by specifying which functional logics should be tested. Further, the developer has to define the test data. The FESAS IDE than creates the test system which can be started by using Eclipse's functionality. However, the specification of data as a JSON String could be improved. Further, the use of the test mode and the analysis of the results are not integrated into the FESAS IDE. Optimization of the workflow for the testing mode is part of future work. Additionally, it might be possible to better integrate the debug mode of Eclipse and *JUnit* tests.

Integration of the ALM. As a second stream for future work related to the FESAS IDE, we identified the integration of the ALM into the FESAS IDE. So far, the components of the ALM are well integrated into the FESAS reference systems. Further, the FESAS IDE can be used to implement functional logics for the ALM. However, the support for modules for analyzing and planning of self-improvement can be extended. So far, this support is limited to code examples and the existing modules from the SmartHighway scenario. On the one hand, we plan to further abstract the reusable parts of the existing modules and better separate them from specific code. Additionally, this involves clear interfaces for changing the elements that require customization, e.g., the simulator in the rule-based module for parametric self-improvement or the definition of rule models for the Neo4j module. On the other hand, we plan to simplify the development of new modules by further assistance for developers, e.g., through a better integration of the functionalities that the ALM offers, such as the access to captured data.

9.2.3. Limitations of the Adaptation Logic Manager Prototype

Decentralized Decision Making. One important requirement for the ALM is the flexibility in the adaptation control (requirement $R_{SI}5$). Whereas most of the related approaches support only central decision making for self-improvement, the system model for the ALM is not limited to that and supports decentralized decision making, too. As we used the FESAS Framework for implementation of the ALM, decentralization through distribution of the ALM's MAPE components is supported. However, distribution of the MAPE components is only one side of the coin. Having a decentralized decision making might also involve trade-offs in the components for analyzing and planning self-improvement. We did not implement a use case system with ALM in a decentralized setting, yet. This is part of future work.

Supported Frameworks. The ALM prototype is implemented as a self-contained module which is decoupled from the adaptation logic for fulfilling the requirement $R_{SI}6$. Therefore, we specified (i) interfaces for the necessary components that the adaptation logic has to offer and (ii) the ALM protocol (cf. Appendix C.1) for interaction between adaptation logic and ALM. Accordingly, the ALM can be connected with every adaptation logic that integrates the necessary elements for the interaction with the ALM. However, in this thesis we only used the FESAS Framework for the implementation of the adaptation logic. Connecting the ALM with adaptation logics implemented using other frameworks is part of future work.

Modules for Self-improvement. In this thesis, we presented the implementation of three modules for planning self-improvement. These modules are reusable. However, for future work, we plan to offer developers additional modules for analyzing and planning. Possible approaches exist in related work. Ramirez *et al.* presented approaches for generating target system configurations [308] and automatically generating adaptation paths to achieve the targeted system configuration [304]. Gerostathopoulos *et al.* [149] derived components from meta-adaptation strategies for automatically adjusting knowledge distribution of sensor values, configuring parameters for scheduling, and changing analyzing parameters. Further, it is possible to integrate one of the approaches presented in Section 4.3. Additionally, we have to solve complexity and concurrency issues with the self-improvement layer itself. In the evaluation, we focused on the different self-improvement modules separately. This is reasonable because the modules have different timescales and perform well in different settings. However, we did not test an integrated version which has to choose the best module for a specific setting. This requires balancing of trade-offs. The ALM supports this through the different runtime modes for planning self-improvement. For future work, we plan to evaluate trade-offs that the ALM Planner has to consider for different runtime modes and concurrent planning modules.

9.3. Theoretical Implications

The research in the field of software engineering for SASs tremendously changed in the previous years. These changes are triggered by an increasing interest in the field, which arose through Dagstuhl seminars but also the success of conferences like the *International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (SEAMS) or the increased interests of main conferences like the *International Conference on Autonomic Computing* (ICAC) and the *International Conference on Self-Adaptive and Self-Organizing Systems* (SASO) for software engineering. This leads to a change of the research foci. The participants of the first Dagstuhl seminar in software engineering for SASs in 2008 [76] and the second one in 2010 [96] identified several challenges related to modeling, requirements engineering, adaptation logic engineering, and development processes. In contrast, the last seminar in this row in 2013 [94] focused on assurance and control theory; both had a minor importance before. This shows that research shifts from basic issues – such as how to build the adaptation logic – to issues in composition and verification of the adaptation logic.

A recent overview by Weyns [394] identified several waves of research in the field of software engineering for SASs. According to this overview, the upcoming big challenges in research for the development of SASs are related to unanticipated change and control theory as foundation for SASs. This section describes the challenges *uncertainty* – which triggers the need for unanticipated change – and *assurance*, which includes control theory. Second, it explains how the FESAS Framework contributes to these challenges.

9.3.1. Uncertainty

The dynamic nature of the environment of SASs leads to a gap between design time and runtime [361]. This gap is further strengthened due to *uncertainty* at runtime. Ramirez *et al.* provide a definition of uncertainty:

> *Uncertainty is a system state of incomplete or inconsistent knowledge such that it is not possible for a [SAS] to know which of two or more alternative environmental or system configurations hold at a specific point.* [307, p. 101]

Further, Ramirez *et al.* categorize reasons for uncertainty into a taxonomy. Triggers for this uncertainty are ambiguously defined requirements, wrong design assumptions, unpredictable elements or unexpected environment states, and situations that the adaptation logic cannot handle due to incomplete and inconsistent information as consequence of imprecise, inaccurate, or unreliable sensors or monitoring functionality [307].

The FESAS Framework addresses uncertainty with the ALM to adjust the adaptation logic to performance and adaptation issues related to uncertainty. Additionally, the flexibility of the FESAS Framework enables to integrate additional support for handling uncertainty in the adaptation logic.

For taming uncertainty, different requirement modeling languages exist, e.g., A-LTL [413], FLAGS [29], CARE [301, 302], Tropos4AS [258, 259], and RELAX [404, 405]. These languages are customized to handle degrees of freedom resulting from the uncertainty due to the gap between design time and runtime. As part of future work, it might be interesting to integrate such a language into the FESAS Design tool to support handling uncertainty.

Some approaches integrate such modeling languages to provide a more holistic approach for handling uncertainty. *LOREM* integrates the RELAX language and the KAOS modeling approach and provides a multi-level process to requirements modeling based on the level of requirements engineering from [42]. Ramirez *et al.* combines the RELAX language with the concept of *claims*, which are "*markers of uncertainty that document how design assumptions affect goals*" [306, p. 53]. *POISED* [117] offers an approach to evaluate positive and negative consequences of uncertainty using possibility theory. Additionally, *POISED* integrates this assessment in the adaptation logic to include reasoning on uncertainty for the adaptation decision. Casanova *et al.* describe an approach to derive the health state of components in case of incomplete monitoring information [67]. Moreno *et al.* recently presented tactics to reduce uncertainty depending on uncertainty coming from simplified design assumptions, noise, model drift, context issues, human-in-the-loop, or decentralization [260]. Through the flexibility and openness of the FESAS Framework, it is possible to integrate such approaches for handling uncertainty as functional logics or into the context manager and offering those as building blocks to developers of SASs.

9.3.2. Assurance

Another challenge that recently gained increased importance in the research domain of SASs is *assurance*. The participants of the Dagstuhl seminar on assurance in SASs in 2013 defined assurance accordingly:

> *[Assurance in SASs describes] the collection, analysis and synthesis of evidence that the system satisfies its stated functional and non-functional requirements during its operation in the presence of self-adaptation.* [94, p. 3]

First, perpetual assurance describes the evidence that the SAS still complies to its requirements [94]. Due to uncertainty, the monitored information might be incomplete, outdated, or unreliable. Accordingly, assurance must ensure the reliability of the monitoring information as it also describes the solution space for possible adaptations. Second, decentralization leads to the need of composing and decomposing assurances for goals and tactics as they are applied in a decentralized way, so that an entire re-validation of the whole system is not required at runtime [328]. Third, runtime verification and validation is an important aspect to guarantee the correct adaptation of a system [95]. For this purpose, [94] proposes a mixture of discrete and continuous control with procedures from control theory in CPSs. Further, adaptive control is recommended. [94] describes these three challenges – perpetual assurance, (de)composition of assurance, and control theory for SASs – in more detail.

On the one hand, the generic modules enabled by the FESAS Framework improve the reusability. However, this comes with the costs of abstractions. For composition of an adaptation logic with such modules, assurance is necessary to guarantee the compliance of all MAPE procedures. On the other hand, meta-adaptation of the adaptation logic involves complexity due to specifics of use cases and interaction of the control loops within the adaptation logic. Hence, an approach for validation of self-improvement has to be integrated. Control theory might contribute to this issue [124]. Results from the research on assurance in SASs could be integrated into the FESAS Framework and the ALM as future work. Additionally, also the proposed extension of the FESAS IDE to a self-* properties IDE requires the inclusion of assurance. The FESAS Framework is flexible enough so that assurance can be integrated at several points.

9.4. Practical Implications

Several developments influence the current trends in software engineering and development from a point of view of practitioners. This thesis addresses the development of SASs mainly in the domain of CPSs. In the following, this section describes different trends in industry related to the development of CPSs and which practical implications the results of this thesis and the FESAS Framework have related to the domain of CPSs.

As mentioned in the introduction of this thesis, the importance and influence of IoT devices for our daily lives increased significantly in recent years and will further increase in future. These devices pervade all aspects of human living and support their users as ubiquitous, invisible helpers. *Gartner Inc.* estimated that 8.4 billion devices are connected in the IoT worldwide in 2017. Estimations for the next three years range from 20.4 billion IoT devices by 2020 [144] up to 50 billion[2] IoT devices by 2020 [119]. The heterogeneity of IoT devices – ranging from small, mobile devices over embedded systems to large-scale cloud resources – as well as the heterogeneity of their execution environments lead to the need of (self-)adaptation. The FESAS Framework can significantly contribute as it offers an open, flexible platform for engineering reusable SASs. Further, it enables the consolidation of different development approaches.

[2]However, this number is doubted recently as being too high [277].

To keep abreast of the fast technological changes triggered by IoT hardware, the development processes need to be flexible and agile. Accordingly, the importance of agile software development methods as *Scrum* or *Kanban* increases. The FESAS Framework also contributes to this development. The flexibility in the exchange of the functional logics in combination with the provided code for generic tasks as communication not only accelerates development but also fosters the exchange of code as demanded by rapid prototyping methods. This suits agile methods for software development well. However, so far we focused on evaluating the FESAS Framework with students. An evaluation with practitioners in the field of developing SASs or IoT devices is part of future work.

Another type of practical implications is related to the users of the systems. SASs try to exclude the user per definition from active participation in the adaptation process. As described in Section 2.2, the user might influence this process by changing the system goals. However, a SAS requires procedures to capture the relevant information, such as the preferences of the users. Further, the adaptation logic has to include reasoning on this information. The FESAS Framework supports this through a simplified integration of sensors, the context manager, and the possibility to add the necessary functionality for reasoning as functional logic. As IoT systems are often CPSs which integrate humans and machines seamlessly, this requires another type of human-computer interaction and human-in-the-loop integration. Munir *et al.* [269] motivate the need for integrating models that capture human behavior. Due to the flexibility of the FESAS Framework and as it is not restricted to a specific modeling approach, such models can be integrated into the FESAS Framework either in the context manager or as functional logics. The participants of the Dagstuhl seminar on "Self-aware computing systems" defined several challenges that arise from human-in-the-loop integration [45]. First, the systems need to integrate confidential data for reasoning of adaptation, which might result in privacy issues. Second, trust of the users in the system capabilities can be an issue. Third, as the SAS performs autonomic adaptations, accountability and liability issues might arise. Development processes and frameworks for SASs have to take these issues into account. For the FESAS Framework, this influences the context manager since it integrates potentially confidential data and the ALM because it might influence the design of the adaptation logic and, hence, accountability of the meta-adaptations must be traceable.

10. Conclusion

This thesis presented a framework for developing SASs which focuses on reusability. Following the *Design Science Research Methodology Process Model* of Peffers *et al.* [289], this thesis grounds on a thorough analysis of approaches in the field of developing SASs. The analysis identified a lack of approaches that combine reusable processes and implementation artifacts with tools. Further, approaches do often not address self-improvement. If an approach addresses self-improvement, this is not combined with an approach for supporting the developer throughout the lifecycle of a SAS. Accordingly, the objective of this thesis is:

*The **integrated support** of the development of a **reusable** and **improvable** adaptation logic for SASs throughout the whole lifecycle.*

This thesis derived requirements from the analysis that an approach has to fulfill to support this thesis' objective. Those are centered around the issues *reusability*, *self-improvement*, and *integrated development*. Correspondingly, this thesis defined two artifacts: the Framework for Engineering Self-adaptive Systems (FESAS) and the Adaptation Logic Manager (ALM).

The FESAS Framework is a component framework supporting the development of SASs. It focuses on the reusability of the developed SASs. Therefore, the FESAS Framework offers reusable processes, the FESAS Repository component library, and an implementation of reference architecture components. These elements support developers and designers during the whole lifecycle of a SAS, from design and implementation, to deployment, and, additionally, at runtime through self-improvement. Further, it supports developers in adding all types of self-adaptation, i.e., adaptation of parameters, structure, and context. The separation in (i) designing the interaction in the adaptation logic and (ii) developing the functional logics for the MAPE components is supported by the FESAS IDE. The FESAS IDE also abstracts from the FESAS workflow for SAS development. Additionally, the FESAS Framework supports the development of MAPE algorithms through focusing on low level code activities and dividing generic, reusable code elements and specific ones. The concept of the FESAS Framework neither restricts the developers to specific development concepts (except of using forward engineering approaches) nor requires the developers to use/learn any concepts. As presented in the former sections, the FESAS Framework was evaluated in different case studies focusing the implementation of real SASs.

The FESAS Framework is complemented by the second artifact of this thesis, the ALM. The ALM supports both, reactive and proactive self-improvement. In the example system from [223], we combined the reactive structural approach with the proactive rule learning approach. As a trigger for self-improvement, the ALM reacts to changes in the managed resources as well as the context. The ALM supports parametric and structural self-improvement. Therefore, the reference implementation of the ALM offers one module for parametric self-improvement in the form of rule learning and two modules for model-based/rule-based structural self-improvement. The adaptation control for meta-adaptation through the ALM follows the external approach to separate a specific adaptation logic from a generic self-improvement layer. This improves maintainability leading to improved reusability. The ALM is not restricted to specific adaptation decision criteria. In the prototype of the ALM, rules and utilities are used to control the execution of the self-improvement modules. Further, the ALM modules themselves integrate rules (TARL module), models (Neo4j module), and utility functions (rule learner module). Last, the prototype implementation offers a centralized approach for self-improvement. However, as the ALM prototype itself is implemented using the FESAS Framework, distribution is supported out of the box. Therefore, the concept of the ALM as well as the foundation for the implementation enables centralized and decentralized patterns for the MAPE components of the self-improvement layer. The ALM is well integrated with the FESAS Framework. However, as it is self-contained and as the interaction between ALM and adaptation logic is well-defined with (i) the *ALM Protocol* (cf. Appendix C.1) as well as (ii) the interfaces for the necessary components, the ALM can be integrated with other frameworks for developing SASs.

This thesis evaluated the artifacts in a *proof by prototyping* approach. As part of this approach, we implemented prototypes for the FESAS Framework, the FESAS IDE, and the ALM. Those are evaluated using observational, analytical, experimental, testing, and descriptive evaluation methods with the objectives to analyze (i) the correct functionality, applicability, and achievable reusability of the FESAS Framework, (ii) the usability of the FESAS IDE, as well as (iii) the performance of self-improvement achieved with the ALM. The evaluation results support most of the evaluation propositions. As these are directly linked to the research questions, this indicates that the thesis contributes to them. Based on the results of these evaluations, we derived limitations and future work.

The prototypes cover most of the requirements defined in Chapter 5. For the FESAS Framework, an open issue is a fully integrated context manager in combination with a standardized context modeling approach. Further, a systematic ontology for functional logic contracts and integration of functional logics for quiescence detection might be potential future work. The FESAS IDE proved its usability and supported the applicability of the FESAS Framework. The evaluations identified possible improvements for the usability: better integration of the testing mode, extended support for designing self-* properties, and integrated development support for ALM modules. Regarding the ALM, we claim that decentralized settings are supported. From a technical point of view, this is given since the underlying FESAS Framework supports this. However, we did not investigate the issues that arise from decentralized reasoning. Further, testing the ALM with other frameworks for implementing the adaptation logic instead of the FESAS Framework and offering additional modules for analyzing and planning of self-improvement are potential future work.

Additionally, we discussed the relation of the theoretical contributions to current trends in the research field. We focused on the challenges of uncertainty and assurance. Works from these areas can be integrated into the FESAS Framework as functional logic elements. As complementary perspective, we also derived practical implications. The FESAS Framework contributes to the current trend of IoT systems because it supports the integration of different platforms and fosters implementation of adaptive behavior. Further, development cycles for these systems are short and the use of agile development methods is state of the art. The FESAS Framework fits these methods since it enables rapid prototyping. Additionally, necessary human-in-the-loop integration is possible in SASs that are implemented with the FESAS Framework.

Bibliography

[1] N. Abbas and J. Andersson. Architectural Reasoning for Dynamic Software Product Lines. In *Proceeding of the International Systems and Software Product Line Conference (SPLC) Workshops*, pages 117–124. IEEE, 2013.

[2] S. Abdelwahed, N. Kandasamy, and S. Neema. A Control-based Framework for Self-managing Distributed Computing Systems. In *Proceeding of the Workshop on Self-managed systems (WOSS)*, pages 3–7. ACM, 2004.

[3] U. A. Acar, G. E. Blelloch, and R. Harper. Adaptive functional programming. *ACM Transactions on Programming Languages and Systems*, 28(6):990–1034, 2006.

[4] M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, and J.-P. Rigault. Modeling Context and Dynamic Adaptations with Feature Models. In *Proceedings of the International Workshop for Models@run.time (Models@run.time)*, volume 509, pages 89–98. CEUR-WS.org, 2009.

[5] F. J. Affonso and E. Y. Nakagawa. Self-adaptive Software : Development Approach and Automatic Process for Adaptation at Runtime. In *Revista Brasileira de Computação Aplicada*, pages 68–84, 2015.

[6] M. Aksit and Z. Choukair. Dynamic, adaptive and reconfigurable systems overview and prospective vision. In *Proceeding of the International Conference on Distributed Computing Systems (ICDCS) Workshops*, pages 84–89. IEEE, 2003.

[7] R. Allen, R. Douence, and D. Garlan. Specifying Dynamism in Software Architectures. In *Proceeding of the Workshop on Foundations of Component-Based Software Engineering*, pages 11–22, 1997.

[8] F. Alvares, E. Rutten, and L. Seinturier. Behavioural Model-based Control for Autonomic Software Components. In *Proceeding of the International Conference on Autonomic Computing (ICAC)*, pages 187–196. IEEE, 2015.

[9] S. Amir Molzam, A. Metzger, C. Quinton, L. Baresi, and K. Pohl. Learning and Evolution in Dynamic Software Product Lines. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 158–164. ACM, 2016.

[10] M. Amoui, M. Derakhshanmanesh, J. Ebert, and L. Tahvildari. Achieving dynamic adaptation via management and interpretation of runtime models. *Journal of Systems and Software*, 85(12):2720–2737, 2012.

[11] J. Andersson, L. Baresi, N. Bencomo, R. de Lemos, A. Gorla, P. Inverardi, and T. Vogel. Software Engineering Processes for Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *LNCS*, pages 51–75. Springer, 2013.

[12] J. Andersson, R. de Lemos, S. Malek, and D. Weyns. Modeling Dimensions of Self-Adaptive Software Systems. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 27–47. Springer, 2009.

[13] J. Andersson, R. de Lemos, S. Malek, and D. Weyns. Reflecting on Self-Adaptive Software Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 38–47. IEEE, 2009.

[14] S. S. Andrade and J. de A. Macêdo. A Search-Based Approach for Architectural Design of Feedback Control Concerns in Self-Adaptive Systems. In *Proceeding of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 61–70. IEEE, 2013.

[15] S. S. Andrade and R. J. de Araújo Macêdo. Do Search-Based Approaches Improve the Design of Self-Adaptive Systems ? A Controlled Experiment. In *Proc. Brazilian Symp. Softw. Eng.*, pages 101–110. IEEE, 2014.

[16] K. Angelopoulos, V. E. S. Souza, and J. Pimentel. Requirements and architectural approaches to adaptive software systems: a comparative study. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 23–32. IEEE, 2013.

[17] R. J. Anthony. A Policy-Definition Language and Prototype Implementation Library for Policy-based Autonomic Systems. In *Proceeding of the International Conference on Autonomic Computing (ICAC)*, pages 265–276. IEEE, 2006.

[18] P. Arcaini, E. Riccobene, and P. Scandurra. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In *Proceeding of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 13–23. IEEE, 2015.

[19] P. Arias-Cabarcos and C. Krupitzer. On the design of distributed adaptive authentication systems. In *Proceeding of the Symposium on Usable Privacy and Security (SOUPS)*. USENIX, 2017.

[20] R. Asadollahi, M. Salehie, and L. Tahvildari. StarMX: A framework for developing self-managing Java-based systems. In *Proceeding of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 58–67. IEEE, 2009.

[21] U. Aßmann, S. Götz, J.-M. Jézéquel, B. Morin, and M. Trapp. A Reference Architecture and Roadmap for Models@run.time Systems. In *Models@run.time: Foundations, Applications, and Roadmaps*, pages 1–18. Springer, Cham, 2014.

[22] M. Autili, P. Di Benedetto, and P. Inverardi. A Programming Model for Adaptable Java Applications. In *Proceedings of the International Conference on the Principles and Practice of Programming in Java (PPPJ)*, pages 119–128. ACM, 2010.

[23] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes. Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):26–66, 2006.

[24] O. Babaoglu and H. E. Shrobe. Foreword from the General Co-Chairs . In *Proceeding of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages ix–x. IEEE, 2007.

[25] T. Bäck and H.-P. Schwefel. An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[26] C. Ballagny, N. Hameurlain, and F. Barbier. MOCAS: A State-Based Component Model for Self-Adaptation. In *Proceeding of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 206–215. IEEE, 2009.

[27] J.-P. Banâtre and T. Priol. Chemical Programming of Future Service-oriented Architectures. *Journal of Software*, 4(7):738–746, 2009.

[28] L. Baresi, S. Guinea, and L. Pasquale. Service-Oriented Dynamic Software Product Lines. *IEEE Computer*, 45(10):42–48, 2012.

[29] L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy Goals for Requirements-Driven Adaptation. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 125–134. IEEE, 2010.

[30] M. Bashari, E. Bagheri, and W. Du. Dynamic software product line engineering: a reference framework. *International Journal of Software Engineering and Knowledge Engineering*, 27(2), 2017.

[31] R. Baskerville. What design science is not. *European Journal of Information Systems*, 17:441–443, 2008.

[32] D. Batory. Feature Models, Grammars, and Propositional Formulas. In *Software Product Lines*, volume 3714 of *LNCS*, pages 7–20. Springer, 2005.

[33] C. Becker, M. Handte, G. Schiele, and K. Rothermel. PCOM — A Component System for Pervasive Computing. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*, pages 67–76. IEEE, 2004.

[34] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel. BASE – a Micro-broker-based Middleware for Pervasive Computing. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*, pages 443–451. IEEE, 2003.

[35] N. Bencomo. On the use of software models during software execution. In *Proceedings of the Modeling in Software Engineering (MISE) Workshop*, pages 62–67. IEEE, 2009.

[36] N. Bencomo and G. Blair. Using Architecture Models to Support the Generation and Operation of Component-Based Adaptive Systems. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 183–200. Springer, 2009.

[37] N. Bencomo, P. Grace, C. Flores, D. Hughes, and G. Blair. Genie: Supporting the Model Driven Development of Reflective, Component-based

Adaptive Systems. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 811–814. ACM, 2008.

[38] N. Bencomo, P. Sawyer, G. Blair, and P. Grace. Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. In *Proceedings of the International Workshop on Dynamic Software Product Lines (DSPL)*, pages 23–32. ACM, 2008.

[39] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier. Requirements Reflection: Requirements As Runtime Entities. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 199–202. ACM/IEEE, 2010.

[40] C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard. Tools for Self-Organizing Applications Engineering. In *Engineering Self-Organizaning Systems*, volume 2977 of *LNCS*, pages 283–298. Springer, 2004.

[41] A. Berns and S. Ghosh. Dissecting Self-* Properties. *International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 10–19, 2009.

[42] D. M. Berry, B. H. C. Cheng, and J. Zhang. The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems. In *Proceedings of the International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ)*, 2005.

[43] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. A Survey of Context Modelling and Reasoning Techniques. *Pervasive and Mobile Computing Journal*, 6(2):161–180, 2010.

[44] A. Beugnard, J.-M. Jezequel, N. Plouzeau, and D. Watkins. Making components contract aware. *IEEE Computer*, 32(7):38–45, 1999.

[45] R. Birke, J. Cámara, L. Y. Chen, L. Esterle, K. Geihs, E. Gelenbe, H. Giese, A. Robertsson, and X. Zhu. Self-aware computing systems: Open challenges and future research directions. In *Self-Aware Computing Systems*, pages 709–722. Springer, 2017.

[46] G. Blair, N. Bencomo, and R. B. France. Models@ run.time. *IEEE Computer*, 42(10):22–27, 2009.

[47] G. Blair, T. Coupaye, and J.-B. Stefani. Component-based architecture: the Fractal initiative. *annals of telecommunications – annales des télécommunications*, 64(1-2):1–4, 2009.

[48] G. S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, and K. Saikoski. The Design and Implementation of Open ORB 2. *IEEE Distributed Systems Online*, 2(June), 2001.

[49] D. G. Bobrow, R. P. Gabriel, and J. L. White. CLOS in Context: the Shape of the Design Space. In *Object-Oriented Programming*, pages 29–61. MIT Press, 1993.

[50] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, 1999.

[51] J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach.* ACM/Addison-Wesley, 2000.

[52] C. Bourjot, V. Chevrier, and V. Thomas. A New Swarm Mechanism Based on Social Spiders Colonies: From Web Weaving to Region Detection. *Web Intelligence and Agent Systems*, 1:47–64, 2003.

[53] W. Brockmann, N. Rosemann, and E. Maehle. A Framework for Controlled Self-optimisation in Modular System Architectures. In *Organic Computing – A Paradigm Shift for Complex Systems*, pages 281–294. Springer, 2011.

[54] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. EasyLiving: Technologies for Intelligent Environments. In *Handheld and Ubiquitous Computing*, pages 12–29. Springer, 2000.

[55] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit. A Design Space for Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *LNCS*, pages 33–50. Springer, 2013.

[56] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. A. Müller, M. Pezzè, and M. Shaw. Engineering Self-Adaptive Systems through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 48–70. Springer, 2009.

[57] E. Cakar, N. Fredivianus, J. Hähner, J. Branke, C. Müller-Schloer, and H. Schmeck. Aspects of Learning in OC Systems. In *Organic Computing – A Paradigm Shift for Complex Systems*, pages 237–251. Springer, 2011.

[58] R. Calinescu, S. Gerasimou, and A. Banks. Self-Adaptive Software with Decentralised Control Loops. In *Fundamental Approaches to Software Engineering*, pages 235–251. Springer, 2015.

[59] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-Adaptive Software Needs Quantitative Verification at Runtime. *Communications of the ACM*, 55(9):69–77, 2012.

[60] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, 2011.

[61] J. Cámara, K. L. Bellman, J. O. Kephart, M. Autili, N. Bencomo, A. Diaconescu, H. Giese, S. Götz, P. Inverardi, S. Kounev, and M. Tivoli. Self-aware Computing Systems: Related Concepts and Research Areas. In *Self-Aware Computing Systems*, pages 17–49. Springer, 2017.

[62] C. Canal, E. Pimentel, and J. M. Troya. Specification and Refinement of Dynamic Software Architectures. In *Software Architecture*, volume 12 of *IFIP – The International Federation for Information Processing*, pages 107–126. Kluwer, 1999.

[63] M. Caporuscio, M. D'Angelo, V. Grassi, and R. Mirandola. Reinforcement Learning Techniques for Decentralized Self-adaptive Service Assembly. In *Service-Oriented and Cloud Computing*, pages 53–68. Springer, 2016.

[64] L. Capra, W. Emmerich, and C. Mascolo. CARISMA: Context-Aware Reflective Middleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, 29(10):929–944, 2003.

[65] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola. MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems. *IEEE Transactions on Software Engineering*, 38(5):1138–1159, 2012.

[66] V. P. Carlos Cetina, Pau Giner, Joan Fons. A Model-Driven Approach for

Developing Self-Adaptive Pervasive Systems. In *Proceedings of the International Workshop for Models@run.time (Models@run.time)*, 2008.

[67] P. Casanova, D. Garlan, B. Schmerl, and R. Abreu. Diagnosing unobserved components in self-adaptive systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 75–84. ACM, 2014.

[68] G. Castelli, M. Mamei, A. Rosi, and F. Zambonelli. Engineering Pervasive Service Ecosystems: The SAPERE Approach. *ACM Transactions on Autonomous and Adaptive Systems*, 10(1):1–27, 2015.

[69] W. Cazzola. Evaluation of Object-Oriented Reflective Models. In *Object-Oriented Technology: ECOOP'98 Workshop Reader*, volume 1543 of *LNCS*, pages 386–387. Springer, 1998.

[70] M. Ceriotti, M. Corrà, L. D'Orazio, R. Doriguzzi, D. Facchin, S. T. Gună, G. P. Jesi, R. L. Cigno, L. Mottola, A. L. Murphy, M. Pescalli, G. P. Picco, D. Pregnolato, and C. Torghele. Is there light at the ends of the tunnel? wireless sensor networks for adaptive lighting in road tunnels. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 187–198. ACM/IEEE, 2011.

[71] H. Cervantes and R. S. Hall. Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 614–623. IEEE, 2004.

[72] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. *IEEE Computer*, 42(10):37–43, 2009.

[73] A. Charfi, T. Dinkelaker, and M. Mezini. A Plug-in Architecture for Self-Adaptive Web Service Compositions. In *Proceeding of the International Conference on Web Services (ICWS)*, pages 35–42. IEEE, 2009.

[74] A. Chawla and A. Orso. A generic instrumentation framework for collecting dynamic information. *ACM SIGSOFT Software Engineering Notes*, 29(5):1–4, 2004.

[75] B. Chen, X. Peng, Y. Yu, B. Nuseibeh, and W. Zhao. Self-Adaptation

through Incremental Generative Model Transformations at Runtime. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 676–687. ACM/IEEE, 2014.

[76] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 1–26. Springer, 2009.

[77] S.-W. Cheng. *Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation.* PhD thesis, Carnegie Mellon University, 2008.

[78] S.-W. Cheng and D. Garlan. Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software*, 85(12):2860–2875, 2012.

[79] S.-W. Cheng, D. Garlan, B. R. Schmerl, J. P. Sousa, B. Spitnagel, and P. Steenkiste. Using Architectural Style as a Basis for System Self-repair. In *Proceedings of the World Computer Congress - TC2 Stream / Conference on Software Architecture: System Design, Development and Maintenance (WISCA)*, pages 45–59. Kluwer, 2002.

[80] S.-W. Cheng, D. Garlan, B. R. Schmerl, J. P. Sousa, B. Spitznagel, P. Steenkiste, and N. Hu. Software Architecture-Based Adaptation for Pervasive Systems. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS): Trends in Network and Pervasive Computing*, pages 67–82. Spinger, 2002.

[81] S.-W. Cheng, D. Garlan, B. R. Schmerl, P. Steenkiste, and N. Nu. Software Architecture-based Adaptation for Grid Computing. In *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, pages 389–398. IEEE, 2002.

[82] S.-W. Cheng, V. V. Poladian, D. Garlan, and B. R. Schmerl. Improving Architecture-Based Self-Adaptation through Resource Prediction. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 71–88. Springer, 2009.

[83] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond.* Addison-Wesley, 2 edition, 2010.

[84] Z. Coker, D. Garlan, and C. Le Goues. SASS: Self-adaptation using stochastic search. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 168–174. IEEE, 2015.

[85] M. Cordy, A. Classen, P. Heymans, A. Legay, and P.-Y. Schobbens. Model Checking Adaptive Software with Featured Transition Systems. In *Assurances for Self-Adaptive Systems*, volume 7740 of *LNCS*, pages 1–29. Springer, 2013.

[86] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan. A Generic Component Model for Building Systems Software. *ACM Transactions on Computer Systems*, 26(1):Art. 1, 2008.

[87] M. H. Cruz Torres, T. Van Beers, and T. Holvoet. (No) more design patterns for multi-agent systems. In *Proceedings of the SPLASH Workshops*, pages 213–220. ACM, 2011.

[88] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley, 2000.

[89] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.

[90] D. Dasgupta. Advances in Artificial Immune Systems. *IEEE Computational Intelligence Magazine*, 1(4):40–49, 2006.

[91] D. G. De La Iglesia. *A Formal Approach for Designing Distributed Self-Adaptive Systems* . Phd thesis, Department of Media Technology, Linnaeus University, 2014.

[92] D. G. De La Iglesia and D. Weyns. MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems. *ACM Transactions on Autonomous and Adaptive Systems*, 10(3):1–31, 2015.

[93] R. de Lemos and J. L. Fiadeiro. An Architectural Support for Self-adaptive Software for Treating Faults. In *Proceedings of the Workshop on Self-managed systems (WOSS)*, pages 39–42. ACM, 2002.

[94] R. de Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo, Y. Brun, J. Camara, R. Calinescu, M. B. Cohen, A. Gorla, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jezequel, S. Malek, R. Mirandola, M. Mori, H. A. Müller, R. Rouvoy, C. M. F. Rubira, E. Rutten, M. Shaw, G. Tamburrelli, G. Tamura, N. M. Villegas, T. Vogel, and F. Zambonelli. Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In *Software Engineering for Self-Adaptive Systems III. Assurances*, volume 9640 of *Lecture Notes in Computer Science (LNCS)*, pages 3–30. Springer, 2017.

[95] R. de Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo, Y. Brun, J. Camara, R. Calinescu, M. B. Cohen, A. Gorla, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jezequel, S. Malek, R. Mirandola, M. Mori, H. A. Müller, R. Rouvoy, C. M. F. Rubira, E. Rutten, M. Shaw, G. Tamburrelli, G. Tamura, N. M. Villegas, T. Vogel, and F. Zambonelli. Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *LNCS*. Springer, 2013.

[96] R. de Lemos, H. Giese, H. Müller, M. Shaw, J. Andersson, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cikic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Goeschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, M. Litoiu, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, B. Schmerl, D. B. Smith, J. P. Sousa, G. Tamura, L. Tahvildari, N. M. Villegas, T. Vogel, D. Weyns, K. Wong, and J. Wuttke. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *LNCS*, pages 1–32. Springer, 2013.

[97] T. De Wolf and T. Holvoet. Towards Autonomic Computing: Agent-Based Modelling, Dynamical Systems Analysis, and Decentralised Control. In *Proc. INDIN*, pages 470–479. IEEE, 2003.

[98] T. De Wolf and T. Holvoet. Emergence Versus Self-Organisation Different

Concepts but Promising When Combined. In *Engineering Self-Organising Systems*, volume 3464 of *LNCS*, pages 1–15. Springer, 2005.

[99] T. De Wolf and T. Holvoet. Towards a Methodology for Engineering Self-Organising Emergent Systems. In *Proc. SOAS*, pages 18–34. IOS Press, 2005.

[100] T. De Wolf and T. Holvoet. Design Patterns for Decentralised Coordination in Self-organising Emergent Systems. In *Engineering Self-Organising Systems*, volume 4335 of *LNCS*, pages 28–49. Springer, 2007.

[101] A. K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[102] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky. MetaSelf — An Architecture and a Development Method for Dependable Self-* Systems. In *Proc SAC*, pages 457–461. ACM, 2010.

[103] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15(3-4):313–341, 2008.

[104] Y. Diao, J. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung. Self-Managing Systems: A Control Theory Foundation. In *Proc. ECBS*, pages 441–448. IEEE, 2005.

[105] S. Dobson, F. Zambonelli, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, and N. Schmidt. A Survey of Autonomic Communications. *ACM Transactions on Autonomous and Adaptive Systems*, 1(2):223–259, 2006.

[106] M. Dorigo, G. A. Di Caro, and L. M. Gambardella. Ant Algorithms for Discrete Optimization. *Artificial life*, 5(2):137–172, 1999.

[107] J. Dowling and V. Cahill. The K-Component Architecture Meta-Model for Self-Adaptive Software. In *Metalevel Architectures and Separation of Crosscutting Concerns*, volume 2192 of *LNCS*, pages 81–88. Springer, 2001.

[108] J. Dowling and V. Cahill. Self-managed Decentralised Systems Using K-components and Collaborative Reinforcement Learning. In *Proceedings of the Workshop on Self-managed systems (WOSS)*, pages 39–43. ACM, 2004.

[109] F. Duarte. Using machine learning to revise adaptation models under non-determinism. Technical report, Departamento de Engenharia Informática, Instituto Superior Técnico (IST), Universidade de Lisboa, 2016.

[110] E. H. Durfee and V. R. Lesser. Negotiating task decomposition and allocation using partial global planning. In *Distributed Artificial Intelligence (Vol. 2)*, pages 229–243. Morgan Kaufmann, 1989.

[111] J. Edinger, D. Schäfer, C. Krupitzer, V. Raychoudhury, and C. Becker. Fault-Avoidance Strategies for Context-Aware Schedulers in Pervasive Computing Systems. *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*, pages 79–88, 2017.

[112] A. Elkhodary, N. Esfahani, and S. Malek. FUSION: A Framework for Engineering Self-tuning Self-adaptive Software Systems. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, pages 7–16. ACM, 2010.

[113] A. Elkhodary, S. Malek, and N. Esfahani. On the Role of Features in Analyzing the Architecture of Self-Adaptive Software Systems. In *Proceedings of the International Workshop for Models@run.time (Models@run.time)*, pages 41–50. ACM/IEEE, 2009.

[114] M. Endler and J. Wei. Programming Generic Dynamic Reconfigurations for Distributed Applications. In *Proc. CDS*, pages 68–79, 1992.

[115] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 111–121. IEEE, 2009.

[116] N. Esfahani, A. Elkhodary, and S. Malek. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Transactions on Software Engineering*, 39(11):1467–1493, 2013.

[117] N. Esfahani, E. Kouroshfar, and S. Malek. Taming uncertainty in self-adaptive software. In *Proceedings of the ACM SIGSOFT Symposium and the European Conference on Foundations of Software Engineering (ES-EC/FSE)*, pages 234–244. ACM, 2011.

[118] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[119] D. Evans. The Internet of Things How the Next Evolution of the Internet Is Changing Everything. Technical report, Cisco Internet Business Solutions Group, 2011.

[120] J. L. Fernandez-Marquez, G. Di Marzo Serugendo, S. Montagna, M. Viroli, and J. L. Arcos. Description and composition of bio-inspired design patterns: A complete overview. *Natural Computing*, 12(1):43–67, 2013.

[121] N. Ferry, F. Chauvel, H. Song, and A. Solberg. Towards Meta-adaptation of Dynamic Adaptive Systems with Models@Runtime. In *Proceedings of the International Conference on Model-Driven Engineering and Software Development*, pages 503–508, 2017.

[122] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-Time Efficient Probabilistic Model Checking. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 341–350. ACM/IEEE, 2011.

[123] A. Filieri, H. Hoffmann, and M. Maggio. Automated Design of Self-Adaptive Software with Control-Theoretical Formal Guarantees. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 299–310. ACM/IEEE, 2014.

[124] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel. Software engineering meets control theory. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 71–82. IEEE, 2015.

[125] A. Filieri and G. Tamburrelli. Probabilistic Verification at Runtime for Self-Adaptive Systems. In *Assurances for Self-Adaptive Systems*, volume 7740 of *LNCS*, pages 30–59. Springer, 2013.

[126] D. Fisch, E. Kalkowski, and B. Sick. Collaborative Learning by Knowledge Exchange. In *Organic Computing – A Paradigm Shift for Complex Systems*, pages 267–280. Springer, 2011.

[127] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, and J.-M. Jézéquel. Modeling and Validating Dynamic Adaptation. In *Models in Software Engineering*, volume 5421 of *LNCS*, pages 97–108. Springer, 2009.

[128] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. Using Architecture Models for Runtime Adaptability. *IEEE Software*, 23(2):62–70, 2006.

[129] R. France and B. Rumpe. Model-driven Development of Complex Software: A Research Roadmap. In *Proceedings of the Future of Software Engineering (FOSE)*, pages 37–54. IEEE, 2007.

[130] J. M. Franco, F. Correia, R. Barbosa, M. Zenha-Rela, B. Schmerl, and D. Garlan. Improving self-adaptation planning through software architecture-based stochastic modeling. *Journal of Systems and Software*, 115:42–60, 2016.

[131] E. M. Fredericks, B. DeVries, and B. H. C. Cheng. Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 17–26, 2014.

[132] S. Frey, A. Diaconescu, and I. Demeure. Architectural Integration Patterns for Autonomic Management Systems. In *Proceedings of the International Workshop on Engineering of Autonomic and Autonomous Systems (EASe)*, 2012.

[133] S. Frey, A. Diaconescu, D. Menga, and I. Demeure. A Generic Holonic Control Architecture for Heterogeneous Multiscale and Multiobjective Smart Microgrids. *ACM Transactions on Autonomous and Adaptive Systems*, 10(2):1–21, 2015.

[134] A. Frömmgen, R. Rehner, M. Lehn, and A. Buchmann. Fossa: Learning ECA Rules for Adaptive Distributed Systems. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 207–210. IEEE, 2015.

[135] T. Gabor, L. Belzner, M. Kiermeier, M. T. Beck, and A. Neitz. A Simulation-Based Architecture for Smart Cyber-Physical Systems. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 374–379. IEEE, 2016.

[136] N. Gámez, L. Fuentes, and J. M. Troya. Creating Self-Adapting Mobile Systems with Dynamic Software Product Lines. *IEEE Software*, 32(2):105 – 112, 2015.

[137] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[138] E. Garcia, A. Giret, and V. Botti. Software Engineering for Service-Oriented MAS. In *Cooperative Information Agents XII*, volume 5180 of *LNCS*, pages 86–100. Springer, 2008.

[139] L. Gardelli, M. Viroli, and A. Omicini. Design Patterns for Self-Organizing Systems. In *Multi-Agent Systems and Applications V*, volume 4696 of *LNCS*, pages 123–132. Springer, 2007.

[140] D. Garlan, S.-W. Cheng, A.-C. Huang, B. R. Schmerl, and P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *IEEE Computer*, 37(10):46–54, 2004.

[141] D. Garlan, B. Schmerl, and J. Chang. Using gauges for architecture-based monitoring and adaptation. In *Working conference on complex and dynamic systems architecture*, 2001.

[142] D. Garlan and B. R. Schmerl. Using Architectural Models at Runtime: Research Challenges. In *Software Architecture*, volume 3047 of *LNCS*, pages 200–205. Springer, 2004.

[143] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, 2002.

[144] Gartner Inc. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. `https://www.gartner.com/newsroom/id/3598917`. Accessed: 2018-03-26.

[145] K. Geihs, R. Reichle, M. Wagner, and M. U. Khan. Modeling of Context-Aware Self-Adaptive Applications in Ubiquitous and Service-Oriented Environments. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 146–163. Springer, 2009.

[146] J. C. Georgas and R. N. Taylor. Policy-Based Architectural Adaptation Management: Robotics Domain Case Studies. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 89–108. Springer, 2009.

[147] J. C. Georgas, A. van der Hoek, and R. N. Taylor. Using Architectural Models to Manage and Visualize Runtime Adaptation. *IEEE Computer*, 42(10):52–60, 2009.

[148] I. Georgiadis, J. Magee, and J. Kramer. Self-Organising Software Architectures for Distributed Systems. In *Proceedings of the Workshop on Self-managed systems (WOSS)*, pages 33–38. ACM, 2002.

[149] I. Gerostathopoulos, T. Bures, P. Hnetynka, A. Hujecek, F. Plasil, and D. Skoda. Strengthening Adaptation in Cyber-Physical Systems via Meta-Adaptation Strategies. *ACM Transactions on Cyber-Physical Systems*, 1(3):1–25, 2017.

[150] I. Gerostathopoulos, T. Bures, P. Hnetynka, J. Keznikl, M. Kit, F. Plasil, and N. Plouzeau. Self-adaptation in software-intensive cyber–physical systems: From system goals to architecture configurations. *Journal of Systems and Software*, 122:378–397, 2016.

[151] C. Gershenson. *Design and Control of Self-organizing Systems (PhD Thesis)*. Phd thesis, Vrije Universiteit Brussel, 2007.

[152] C. Ghezzi, A. Mocci, and M. Monga. Synthesizing Intensional Behavior Models by Graph Transformation. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 430–440. IEEE, 2009.

[153] C. Ghezzi, A. Mocci, and M. Sangiorgio. Runtime Monitoring of Functional Component Changes with Behavior Models. In *Models in Software Engineering*, volume 7167 of *LNCS*, pages 152–166. Springer, 2012.

[154] H. Giese and W. Schäfer. Model-driven development of safe self-optimizing mechatronic systems with mechatronicuml. In *Assurances for Self-adaptive Systems*, volume 7740 of *LNCS*, pages 152–186. Springer, 2013.

[155] H. Goldsby, P. Sawyer, N. Bencomo, B. H. C. Cheng, and D. Hughes. Goal-Based Modeling of Dynamically Adaptive System Requirements. In *Proceedings of the International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, pages 36–45. IEEE, 2008.

[156] H. Gomaa, K. Hashimoto, M. Kim, S. Malek, and D. A. Menascé. Software Adaptation Patterns for Service-Oriented Architectures. In *Proceedings of the Symposium on Applied Computing (SAC)*, pages 462–469. ACM, 2010.

[157] H. Gomaa and M. Hussein. Dynamic Software Reconfiguration in Software Product Families. In *Software Product-Family Engineering*, volume 3014 of *LNCS*, pages 435–444. Springer, 2004.

[158] M. Gouda and T. Herman. Adaptive programming. *IEEE Transactions on Software Engineering*, 17(9):911–921, 1991.

[159] M. Güdemann, F. Nafz, F. Ortmeier, H. Seebach, and W. Reif. A Specification and Construction Paradigm for Organic Computing Systems. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 233–242. IEEE, 2008.

[160] M. Güdemann, F. Ortmeier, and W. Reif. Formal Modeling and Verification of Systems with Self-x Properties. In *Autonomic and Trusted Computing*, volume 4158 of *LNCS*, pages 38–47. Springer, 2006.

[161] N. Gui and V. De Florio. Towards Meta-Adaptation Support with Reusable and Composable Adaptation Components. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 49–58. IEEE, 2012.

[162] R. Haesevoets, E. Truyen, T. Holvoet, and W. Joosen. Weaving the Fabric of the Control Loop through Aspects. In *Self-Organizing Architectures*, volume 6090 of *LNCS*, pages 38–65. Springer, 2009.

[163] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. Papadopoulos. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *Journal of Systems and Software*, 85(12):2840–2859, 2012.

[164] S. Hallsteinsen, M. Hinchey, and K. Schmid. Dynamic Software Product Lines. *IEEE Computer*, 41(4):93–95, 2008.

[165] S. Hallsteinsen, E. Stav, and J. Floch. Self-Adaptation for Everyday Systems. In *Proceedings of the Workshop on Self-managed systems (WOSS)*, pages 69–74. ACM, 2004.

[166] S. Hallsteinsen, E. Stav, A. Solberg, and J. Floch. Using Product Line Techniques to Build Adaptive Systems. In *Proceedings of the International Systems and Software Product Line Conference (SPLC)*, pages 141–150. IEEE, 2006.

[167] M. Handte, G. Schiele, V. Matjuntke, C. Becker, and P. J. Marrón. 3PC: System Support for Adaptive Peer-to-Peer Pervasive Computing. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1):Art. 10, 2012.

[168] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer, 2003.

[169] M. Harman, E. Burke, J. A. Clark, and X. Yao. Dynamic adaptive search based software engineering. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–8. ACM, 2012.

[170] M. Harman, Y. Jia, W. B. Langdon, J. Petke, I. H. Moghadam, S. Yoo, and F. Wu. Genetic Improvement for Adaptive Software Engineering (Keynote). In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 1–4. ACM, 2014.

[171] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833–839, 2001.

[172] M. Harman, S. A. Mansouri, and Y. Zhang. Search-Based Software Engineering: Trends, Techniques and Applications. *ACM Computing Surveys*, 45(1):1–61, 2012.

[173] M. Harman, P. McMinn, J. Teixeira De Souza, and S. Yoo. Search Based Software Engineering: Techniques, Taxonomy, Tutorial. In *Empirical Software Engineering and Verification*, volume 7007 of *LNCS*, pages 1–59. Springer, 2012.

[174] W. Heaven, D. Sykes, J. Magee, and J. Kramer. A Case Study in Goal-Driven Architectural Adaptation. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 109–127. Springer, 2009.

[175] R. Hebig, H. Giese, and B. Becker. Making Control Loops Explicit when Architecting Self-adaptive Systems. In *Proceedings of the International Workshop on Self-Organizing Architectures (SOAR)*, pages 21–28. ACM, 2010.

[176] C. Heinzemann, J. Rieke, and W. Schäfer. Simulating Self-Adaptive Component-Based Systems using MATLAB/Simulink. In *Proceedings of*

the International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pages 71–80. IEEE, 2013.

[177] N. Herbst, S. Becker, S. Kounev, H. Koziolek, M. Maggio, A. Milenkoski, and E. Smirni. Metrics and Benchmarks for Self-aware Computing Systems. In *Self-Aware Computing Systems*, pages 437–464. Springer, Cham, 2017.

[178] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn. Self-adaptive workload classification and forecasting for proactive resource provisioning. *Concurrency and Computation: Practice and Experience*, 26(12):2053–2078, 2014.

[179] M. Hermann, T. Pentek, and B. Otto. Design principles for industrie 4.0 scenarios. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*, pages 3928–3937, 2016.

[180] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[181] J. Highsmith and A. Cockburn. Agile software development: the business of innovation. *IEEE Computer*, 34(9):120–127, 2001.

[182] J. Hillman and I. Warren. Meta-adaptation in autonomic systems. In *Proceedings of the International Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, pages 292–298. IEEE, 2004.

[183] M. Hinchey and R. Sterritt. Self-Managing Software. *IEEE Computer*, 39(2):107–109, 2006.

[184] R. Hirschfeld, P. Costanza, and O. Nierstrasz. Context-Oriented Programming. *Journal of Object Technology*, 7(3):125–151, 2008.

[185] G. Holmes, A. Donkin, and I. H. Witten. WEKA: a machine learning workbench. In *Proceedings of the Australian and New Zealand Conference on Intelligent Information Systems (ANZIIS)*, pages 357–361. IEEE, 1994.

[186] M. Hölzl, N. Koch, M. Puviani, M. Wirsing, and F. Zambonelli. *The Ensemble Development Life Cycle and Best Practices for Collective Autonomic Systems*, pages 325–354. Springer, 2015.

[187] G. Huang, H. Mei, and F.-Q. Yang. Runtime recovery and manipulation of software architecture of component-based systems. *Automated Software Engineering*, 13(2):257–281, 2006.

[188] T. Huang, G.-Q. Wu, and J. Wei. Runtime Monitoring Composite Web Services Through Stateful Aspect Extension. *Journal of Computer Science and Technology*, 24(2):294–308, 2009.

[189] N. Huber, F. Brosig, and S. Kounev. Model-based self-adaptive resource allocation in virtualized environments. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 90–99. ACM, 2011.

[190] N. Huber, F. Brosig, S. Spinner, S. Kounev, and M. Bähr. Model-based self-aware performance and resource management using the descartes modeling language. *IEEE Transactions on Software Engineering*, 43(5):432–452, 2017.

[191] M. C. Huebscher and J. A. McCann. A survey of Autonomic Computing – Degrees, Models, and Applications. *ACM Computing Surveys*, 40(3):1–28, 2008.

[192] M. C. Huebscher, J. A. McCann, and A. Hoskins. Context as autonomic intelligence in a ubiquitous computing environment. *International Journal of Internet Protocol Technology*, 2(1):30–39, 2007.

[193] IBM Corp. An Architectural Blueprint for Autonomic Computing. Technical report, IBM, 2005.

[194] M. U. Iftikhar and D. Weyns. Activforms: Active formal models for self-adaptation. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 125–134. ACM, 2014.

[195] F. Irmert, T. Fischer, and K. Meyer-Wegener. Runtime Adaptation in a Service-Oriented Component Model. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 97–104. ACM, 2008.

[196] D. A. Kafaf and D. K. Kim. A web service-based approach for developing self-adaptive systems. *Computational Electrical Engineering*, 63(1):260–276, 2017.

[197] R. Keays and A. Rakotonirainy. Context-Oriented Programming. In *Pro-*

ceedings of the International Workshop on Data Engineering for Wireless and Mobile Access (MobiDe), pages 9–16. ACM, 2003.

[198] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[199] J. O. Kephart, M. Maggio, A. Diaconescu, H. Giese, H. Hoffmann, S. Kounev, A. Koziolek, P. Lewis, A. Robertsson, and S. Spinner. Reference Scenarios for Self-aware Computing. In *Self-Aware Computing Systems*, pages 87–106. Springer, 2017.

[200] S. K. Khaitan and J. D. McCalley. Design Techniques and Applications of Cyberphysical Systems: A Survey. *IEEE Systems Journal*, 9(2):350–365, 2015.

[201] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proc. Robotics and Automation - Vol. 2*, pages 500–505. IEEE, 1985.

[202] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In *ECOOP 2001 – Object-Oriented Programming*, volume 2072 of *LNCS*, pages 327–354. Springer, 2001.

[203] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP'97 – Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer, 1997.

[204] D. Kim and S. Park. Reinforcement Learning-Based Dynamic Adaptation Planning Method for Architecture-based Self-Managed Software. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 76–85. IEEE, 2009.

[205] T. King, A. Ramirez, R. Cruz, and P. Clarke. An Integrated Self-Testing Framework for Autonomic Computing Systems. *Journal of Computers*, 2(9), 2007.

[206] T. M. King, A. Ramirez, P. J. Clarke, and B. Quinones-Morales. A Reusable Object-Oriented Design to Support Self-Testable Autonomic Software. In *Proceedings of the Symposium on Applied Computing (SAC)*, pages 1664–1669. ACM, 2008.

[207] M. Kit, I. Gerostathopoulos, T. Bures, P. Hnetynka, and F. Plasil. An Architecture Framework for Experimentations with Self-Adaptive Cyber-physical Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 93–96. ACM, 2015.

[208] B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University, UK, 2007.

[209] H. Klus, D. Niebuhr, and A. Rausch. A Component Model for Dynamic Adaptive Systems. In *Proceeding of the International Workshop on Engineering of Software Services for Pervasive Environments (ESSPE=*, pages 21–28. ACM, 2007.

[210] M. Kokar, K. Baclawski, and Y. Eracar. Control Theory-Based Foundations of Self-Controlling Software. *IEEE Intelligent Systems*, 14(3):37–45, 1999.

[211] F. Kon, F. Costa, G. Blair, and R. H. Campbell. The Case for Reflective Middleware. *Communications of the ACM*, 45(6):33–38, 2002.

[212] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, C. Magalhã, and R. H. Campbell. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In *Middleware 2000*, volume 1795 of *LNCS*, pages 121–143. Springer, 2000.

[213] S. Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Transactions on Software Engineering*, 32(7):486–502, 2006.

[214] S. Kounev, F. Brosig, and N. Huber. Self-aware QoS management in Virtualized Infrastructures. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 175–176. ACM, 2011.

[215] S. Kounev, N. Huber, F. Brosig, and X. Zhu. A model-based approach to designing self-aware it systems and infrastructures. *IEEE Computer*, 49(7):53–61, 2016.

[216] S. Kounev, P. Lewis, K. L. Bellman, N. Bencomo, J. Camara, A. Diaconescu, L. Esterle, K. Geihs, H. Giese, S. Götz, P. Inverardi, J. O. Kephart,

and A. Zisman. The Notion of Self-aware Computing. In *Self-Aware Computing Systems*, pages 3–16. Springer, Cham, 2017.

[217] J. Kramer and J. Magee. The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, 1990.

[218] J. Kramer and J. Magee. Self-Managed Systems: an Architectural Challenge. In *Proceedings of the Future of Software Engineering (FOSE)*, pages 259–268, 2007.

[219] C. Krupitzer. FESAS: A Framework for Engineering Self-Adaptive Systems. In *Proceedings of the First Organic Computing Doctoral Dissertation Colloquium (OC-DDC'13)*, pages 16–19, 2013.

[220] C. Krupitzer, M. Breitbach, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker. A Survey on Engineering Approaches for Self-Adaptive Systems (Extended Version). Technical report, Chair of Information Systems II, University of Mannheim, Germany, 2018.

[221] C. Krupitzer, M. Breitbach, J. Saal, C. Becker, M. Segata, and R. Lo Cigno. RoCoSys: A framework for coordination of mobile IoT devices. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom) Workshops*, pages 485–490. IEEE, 2017.

[222] C. Krupitzer, G. Drechsel, D. Mateja, A. Pollkläsener, F. Schrage, T. Sturm, A. Tomasovic, and C. Becker. Using Spreadsheet-defined Rules for Reasoning in Self-Adaptive Systems. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom) Workshops*, pages 462–467. IEEE, 2018.

[223] C. Krupitzer, J. Otto, F. M. Roth, A. Frommgen, and C. Becker. Adding Self-Improvement to an Autonomic Traffic Management System. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 209–214. IEEE, 2017.

[224] C. Krupitzer, M. Pfannemüller, V. Voss, and C. Becker. Comparison of Approaches for developing Self-adaptive Systems. Technical report, Chair of Information Systems II, University of Mannheim, Germany, 2018.

[225] C. Krupitzer, F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and

A. Schürr. FESAS IDE: An Integrated Development Environment for Autonomic Computing. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 15–24. IEEE, 2016.

[226] C. Krupitzer, F. M. Roth, M. Pfannemüller, and C. Becker. Comparison of Approaches for Self-Improvement in Self-Adaptive Systems. In *Proceedings of the International Conference on Autonomic Computing (ICAC))*, pages 308–314. IEEE, 2016.

[227] C. Krupitzer, F. M. Roth, M. Pfannemüller, and C. Becker. Comparison of Approaches for Self-Improvement in Self-Adaptive Systems (Extended Version). Technical report, Chair of Information Systems II, University of Mannheim, Germany, 2017.

[228] C. Krupitzer, F. M. Roth, S. VanSyckel, and C. Becker. Towards Reusability in Autonomic Computing. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 115–120. IEEE, 2015.

[229] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker. A Survey on Engineering Approaches for Self-Adaptive Systems. *Pervasive and Mobile Computing Journal*, 17(Part B):184–206, 2015.

[230] C. Krupitzer, T. Sztyler, J. Edinger, M. Breitbach, H. Stuckenschmidt, and C. Becker. Hips do lie! a position-aware mobile fall detection system. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*, pages 95–104. IEEE, 2018.

[231] C. Krupitzer, S. VanSyckel, and C. Becker. FESAS: Towards a Framework for Engineering Self-Adaptive Systems. In *Proceedings of the 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 263–264. IEEE, 2013.

[232] F. Křikava, P. Collet, and R. B. France. Actor-based Runtime Model of Adaptable Feedback Control Loops. In *Proceedings of the International Workshop for Models@run.time (Models@run.time)*, pages 39–44. ACM, 2012.

[233] R. Laddaga. Active Software. In *Self-Adaptive Software*, volume 1936 of *LNCS*, pages 11–26. Springer, 2001.

[234] P. Lalanda, J. A. McCann, and A. Diaconescu. *Autonomic Computing.* Springer, 2013.

[235] J. Lee and K. C. Kang. A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. In *Proceedings of the International Systems and Software Product Line Conference (SPLC)*, pages 131–140. IEEE, 2006.

[236] U. Lee, E. Magistretti, M. Gerla, P. Bellavista, P. Lio, and K.-W. Lee. Bio-Inspired Multi-Agent Data Harvesting in a Proactive Urban Monitoring Environment. *Ad Hoc Networks*, 7(4):725–741, 2009.

[237] G. Lehmann, M. Blumendorf, F. Trollmann, and S. Albayrak. Metamodeling Runtime Models. In *Models in Software Engineering*, volume 6627 of *LNCS*, pages 209–223. Springer, 2011.

[238] S. Lightstone. Foundations of Autonomic Computing Development. In *Proceedings of the International Workshop on Engineering of Autonomic and Autonomous Systems (EASe)*, pages 163–171. IEEE, 2007.

[239] H. Liu and M. Parashar. Accord: A Programming Framework for Autonomic Applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 36(3):341–352, 2006.

[240] L. Liu and H. Schmeck. A Roadmap towards Autonomic Service-Oriented Architectures. *International Transactions on Systems Science and Applications*, 2(3):245–254, 2006.

[241] M. Luckey. *Adaptivity Engineering: Modeling and Quality Assurance for Self-Adaptive Software Systems.* PhD thesis, Paderborn University, 2013.

[242] M. Luckey and G. Engels. High-Quality Specification of Self-Adaptive Software Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 143–152. ACM, 2013.

[243] D. Luckham, J. Kenney, L. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and Analysis of System Architecture Using Rapide. *IEEE Transactions on Software Engineering*, 21(4):336–354, 1995.

[244] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez. Self-

adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40:7267–7279, 2013.

[245] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying Distributed Software Architectures. In *Proceedings of the European Software Engineering Conference (ESEC)*, volume 989 of *LNCS*, pages 137–153. Springer, 1995.

[246] A. Malatras, F. Peng, and B. Hirsbrunner. A Self-Management Framework for Efficient Resource Discovery in Pervasive Environments. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 181–182. ACM, 2011.

[247] S. Malek, G. Edwards, Y. Brun, H. Tajalli, J. Garcia, I. Krka, N. Medvidovic, M. Mikic-Rakic, and G. S. Sukhatme. An architecture-driven software mobility framework. *Journal of Systems and Software*, 83(6):972–989, 2010.

[248] J. Malenfant, M. Jacques, and F. Demers. A Tutorial on Behavioral Reflection and its Implementation. In *Proceeding of the Reflection*, pages 1–20, 1996.

[249] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. Case Studies for Self-Organization in Computer Science. *Journal of Systems Architecture*, 52(8-9):443–460, 2006.

[250] M. Mamei and F. Zambonelli. Programming Pervasive and Mobile Computing Applications: the TOTA Approach. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*, pages 263–273. IEEE, 2004.

[251] V. Matena, T. Bures, I. Gerostathopoulos, and P. Hnetynka. Model Problem and Testbed for Experiments with Adaptation in Smart Cyber-Physical Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 82–88. ACM, 2016.

[252] F. Mattern. Global quiescence detection based on credit distribution and recovery. *Information Processing Letters*, 30(4):195–200, 1989.

[253] J. Mazzola Paluska, H. Pham, U. Saif, G. Chau, C. Terman, and S. Ward.

Structured Decomposition of Adaptive Applications. *Pervasive and Mobile Computing Journal*, 4(6):791–806, 2008.

[254] P. McKinley, S. Sadjadi, E. Kasten, and B. H. C. Cheng. Composing Adaptive Software. *IEEE Computer*, 37(7):56–64, 2004.

[255] D. Menasce, H. Gomaa, S. Malek, and J. P. Sousa. SASSY: A Framework for Self-Architecting Service-Oriented Systems. *IEEE Software*, 28(6):78–85, 2011.

[256] M. B. Miller and B. L. Bassler. Quorum Sensing in Bacteria. *Annual Reviews in Microbiology*, 55:165–199, 2001.

[257] M. Mongiello, P. Pelliccione, and M. Sciancalepore. AC-contract: Run-time verification of context-aware applications. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 24–34. ACM, 2015.

[258] M. Morandini, L. Penserini, and A. Perini. Modelling Self-Adaptivity: A Goal-Oriented Approach. In *Proceeding of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 469–470. IEEE, 2008.

[259] M. Morandini, L. Penserini, and A. Perini. Towards Goal-oriented Development of Self-adaptive Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 9–16. ACM, 2008.

[260] G. A. Moreno, J. Cámara, D. Garlan, and M. Klein. Uncertainty reduction in self-adaptive systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2018. To be published.

[261] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg. Models@ Run.time to Support Dynamic Adaptation. *IEEE Computer*, 42(10):44–51, 2009.

[262] B. Morin, O. Barais, G. Nain, and J.-M. Jézéquel. Taming Dynamically Adaptive Systems Using Models and Aspects. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 122–132. IEEE, 2009.

[263] B. Morin, F. Fleurey, N. Bencomo, J.-M. Jézéquel, A. Solberg, V. Dehlen, and G. Blair. An Aspect-Oriented and Model-Driven Approach for Managing Dynamic Variability. In *Model Driven Engineering Languages and Systems*, volume 5301 of *LNCS*, pages 782–796. Springer, 2008.

[264] R. Morrison, D. Balasubramaniam, F. Oquendo, B. Warboys, and R. M. Greenwood. An Active Architecture Approach to Dynamic Systems Coevolution. In *Software Architecture*, volume 4758 of *LNCS*, pages 2–10. Springer, 2007.

[265] J. Morse, D. Araiza-Illan, J. Lawry, A. Richards, and K. Eder. A formal approach to analysing requirements conformance in adaptive systems. Technical report, University of Bristol, 2017.

[266] M. Mubashir, L. Shao, and L. Seed. A survey on fall detection: Principles and approaches. *Neurocomputing*, 100:144–152, 2013.

[267] H. A. Müller, M. Pezzè, and M. Shaw. Visibility of Control in Adaptive Systems. In *Proceedings of the International Workshop on Ultra-large-scale Software-intensive Systems (ULSSIS)*, pages 23–26. ACM, 2008.

[268] C. Müller-Schloer, H. Schmeck, and T. Ungerer, editors. *Organic Computing – A Paradigm Shift for Complex Systems*. Springer, 2011.

[269] S. Munir, J. A. Stankovic, C.-J. M. Liang, and S. Lin. Cyber physical system challenges for human-in-the-loop control. In *Proceedings of the International Workshop on Feedback Computing*, 2013.

[270] A. Musil, J. Musil, D. Weyns, T. Bures, H. Muccini, and M. Sharaf. Patterns for Self-Adaptation in Cyber-Physical Systems. In *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, pages 331–368. Springer, 2017.

[271] F. Nafz, H. Seebach, J.-P. Steghöfer, S. Bäumler, and W. Reif. A Formal Framework for Compositional Verification of Organic Computing Systems. In *Autonomic and Trusted Computing*, volume 6407 of *LNCS*, pages 17–31. Springer, 2010.

[272] R. Nagpal. A Catalog of Biologically-Inspired Primitives for Engineering Self-Organization. In *Engineering Self-Organising Systems*, volume 2977 of *LNCS*, pages 53–62. Springer, 2004.

[273] L. Nahabedian, V. Braberman, N. D'Ippolito, S. Honiden, J. Kramer, K. Tei, and S. Uchitel. Assured and correct dynamic update of controllers. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 96–107. ACM, 2016.

[274] H. Nakagawa, A. Ohsuga, and S. Honiden. Towards dynamic evolution of self-adaptive systems based on dynamic updating of control loops. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 59–68. IEEE, 2012.

[275] V. H. Nguyen, F. Fouquet, N. Plouzeau, and O. Barais. A Process for Continuous Validation of Self-Adapting Component Based Systems. In *Proceedings of the International Workshop for Models@run.time (Models@run.time)*, pages 32–37. ACM, 2012.

[276] O. Nierstrasz, M. Denker, and L. Renggli. Model-Centric, Context-Aware Software Adaptation. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 128–145. Springer, 2009.

[277] A. Nordrum. Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated - IEEE Spectrum, 2016.

[278] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.

[279] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-Based Runtime Software Evolution. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 177–186. IEEE, 1998.

[280] P. Oreizy, N. Medvidovic, and R. N. Taylor. Runtime Software Adaptation: Framework, Approaches, and Styles . In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 899–910. ACM, 2008.

[281] A. Pandey, G. A. Moreno, J. Camara, and D. Garlan. Hybrid Planning for Decision Making in Self-Adaptive Systems. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 130–139, 2016.

[282] J. Panerati, M. Triverio, M. Maggio, and M. D. Santambrogio. On How to

Coordinate the Behavior of Independent Adaptive Systems. In *Proceedings of the International Workshop on Feedback Computing*, pages 1–6. ACM, 2012.

[283] V. Panzica La Manna, J. Greenyer, C. Ghezzi, and C. Brenner. Formalizing correctness criteria of dynamic updates derived from specification changes. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 63–72. IEEE, 2013.

[284] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, 2007.

[285] M. Parashar and S. Hariri. Autonomic Computing: An Overview. In *Unconventional Programming Paradigms: International Workshop UPP 2004, Le Mont Saint Michel, France, September 15-17, 2004, Revised Selected and Invited Papers*, pages 257–269. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[286] C. Parra, D. Romero, S. Mosser, R. Rouvoy, L. Duchien, and L. Seinturier. Using Constraint-based Optimization and Variability to Support Continuous Self-adaptation. In *Proceedings of the Symposium on Applied Computing (SAC)*, pages 486–491. ACM, 2012.

[287] L. Pasquale, L. Baresi, and B. Nuseibeh. Towards Adaptive Systems through Requirements@Runtime. In *Proceedings of the International Workshop for Models@run.time (Models@run.time)*, pages 13–24. CEUR-WS.org, 2011.

[288] T. Patikirikorala, A. Colman, J. Han, and L. Wang. A Systematic Survey on the Design of Self-Adaptive Software Systems Using Control Engineering Approaches. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 33–42. IEEE, 2012.

[289] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[290] X. Peng, B. Chen, Y. Yu, and W. Zhao. Self-tuning of software systems

through dynamic quality tradeoff and value-based feedback control loop. *Journal of Systems and Software*, 85(12):2707–2719, 2012.

[291] G. Perrouin, B. Morin, F. Chauvel, F. Fleurey, J. Klein, Y. Le Traon, O. Barais, and J.-M. Jezequel. Towards flexible evolution of Dynamically Adaptive Systems. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 1353–1356. IEEE, 2012.

[292] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 68–77. BCS Learning & Development Ltd., 2008.

[293] M. Pfannemüller, C. Krupitzer, M. Weckesser, and C. Becker. A dynamic software product line approach for adaptation planning in autonomic computing systems. In *Proceeding of the International Conference on Autonomic Computing (ICAC)*, pages 247–254. IEEE, 2017.

[294] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer, 2005.

[295] V. Poladian, J. P. Sousa, D. Garlan, and M. Shaw. Dynamic Configuration of Resource-Aware Services. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 604–613. ACM/IEEE, 2004.

[296] T. Preisler, T. Dethlefs, and W. Renz. Middleware for Constructing Decentralized Control in Self-Organizing Systems. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 325–330. IEEE, 2015.

[297] C. Priesterjahn, D. Steenken, and M. Tichy. Timed Hazard Analysis of Self-healing Systems. In *Assurances for Self-Adaptive Systems*, volume 7740 of *LNCS*, pages 112–151. Springer, 2013.

[298] H. Prothmann, F. Rochner, S. Tomforde, J. Branke, C. Müller-Schloer, and H. Schmeck. Organic Control of Traffic Lights. In *Autonomic and Trusted Computing*, volume 5060 of *LNCS*, pages 219–233. Springer, 2008.

[299] H. Prothmann, S. Tomforde, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck. Organic Traffic Control. In *Organic Computing — A Paradigm Shift for Complex Systems*, pages 431–446. Springer, 1 edition, 2011.

[300] M. Puviani, G. Cabri, and F. Zambonelli. A taxonomy of architectural patterns for self-adaptive systems. In *Proceedings of the International C* Conference on Computer Science and Software Engineering (C3S2E)*, pages 77–85, 2013.

[301] N. A. Qureshi, I. J. Jureta, and A. Perini. Towards a Requirements Modeling Language for Self-Adaptive Systems. In *Requirements Engineering: Foundation for Software Quality*, pages 263–279. Springer, 2012.

[302] N. A. Qureshi and A. Perini. Continuous Adaptive Requirements Engineering: An Architecture for Self-Adaptive Service-Based Applications. In *Proceedings of the First International Workshop on Requirements@Run.Time (RE@RunTime)*, pages 17–24. IEEE, 2010.

[303] N. A. Qureshi and A. Perini. Requirements Engineering for Adaptive Service Based Applications. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 108–111. IEEE, 2010.

[304] A. J. Ramirez, B. H. Cheng, P. K. McKinley, and B. E. Beckmann. Automatically Generating Adaptive Logic to Balance Non-functional Tradeoffs During Reconfiguration. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 225–234. ACM, 2010.

[305] A. J. Ramirez and B. H. C. Cheng. Design Patterns for Developing Dynamically Adaptive Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 49–58. ACM, 2010.

[306] A. J. Ramirez, B. H. C. Cheng, N. Bencomo, and P. Sawyer. Relaxing claims: Coping with uncertainty while evaluating assumptions at run time. In *Model Driven Engineering Languages and Systems*, pages 53–69. Springer, 2012.

[307] A. J. Ramirez, A. C. Jensen, and B. H. C. Cheng. A Taxonomy of Uncertainty for Dynamically Adaptive Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 99–108. IEEE, 2012.

[308] A. J. Ramirez, D. B. Knoester, B. H. Cheng, and P. K. McKinley. Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems.

In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 97–106. ACM, 2009.

[309] R. Razavi, J.-F. Perrot, and N. Guelfi. Adaptive Modeling: An Approach and a Method for Implementing Adaptive Agents. In *Massively Multi-Agent Systems I*, volume 3446 of *LNCS*, pages 136–148. Springer, 2005.

[310] J. H. Reif and H. Wang. Social Potential Fields: A Distributed Behavioral Control for Autonomous Robots. *Robotics and Autonomous Systems*, 27:171—194, 1999.

[311] P. Robertson and R. Laddaga. Model Based Diagnosis and Contexts in Self Adaptive Software. In *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations*, pages 112–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[312] T. Robinson, E. Chan, and E. Coelingh. Operating Platoons On Public Motorways: An Introduction To The SARTRE Platooning Programme. In *Proceedings of the Intelligent Transportation Society World Congress (ITSWC)*, 2010.

[313] M. Rohr, S. Giesecke, W. Hasselbring, M. Hiel, W.-J. van den Heuvel, and H. Weigand. A Classification Scheme for Self-adaptation Research. In *Proceedings of the International Conference on Self Organization and Autonomous Systems in Computing and Communications (SOAS)*, 2006.

[314] W. Romberg. Vereinfachte numerische integration. *Kongelige Norske Videnskabers Selskab Forhandlinger*, 28(7):30–36, 1955.

[315] F. M. Roth, C. Krupitzer, and C. Becker. Runtime evolution of the adaptation logic in self-adaptive systems. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 141–142. IEEE, 2015.

[316] F. M. Roth, C. Krupitzer, S. Vansyckel, and C. Becker. Nature-Inspired Interference Management in Smart Peer Groups. In *Proceedings of the International Conference on Intelligent Environments (IE)*, pages 132–139. IEEE, 2014.

[317] S. M. Sadjadi and P. McKinley. A Survey of Adaptive Middleware. Technical report, Michgan State University, 2003.

[318] S. M. Sadjadi, P. K. McKinley, B. H. C. Cheng, and R. E. K. Stirewalt. TRAP/J: Transparent Generation of Adaptable Java Programs. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, LNCS 3291, pages 1243–1261. Springer, 2004.

[319] N. Salazar, J. A. Rodriguez-Aguilar, and J. L. Arcos. Robust Coordination in Large Convention Spaces. *AI Communications*, 23(4):357–372, 2010.

[320] M. Salehie and L. Tahvildari. Self-Adaptive Software: Landscape & Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):Art. 14, 2009.

[321] K. Saller, M. Lochau, and I. Reimund. Context-Aware DSPLs: Model-Based Runtime Adaptation for Resource-Constrained Systems. In *Proceedings of the International Systems and Software Product Line Conference (SPLC) Workshops*, pages 106–113. ACM, 2013.

[322] G. Salvaneschi, C. Ghezzi, and M. Pradella. Context-oriented programming : a software engineering perspective. *Journal of Systems and Software*, 85(8):1801–1817, 2012.

[323] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 95–103. IEEE, 2010.

[324] D. Schäfer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker. Tasklets: "Better than Best-Effort" Computing. In *Proccedings of the International Conference on Computer Communication and Networks*, pages 1–11. IEEE, 2016.

[325] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proceedings of the First Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 85–90. IEEE, 1994.

[326] J. Schlaich and M. Friedrich. Staumeldungen und routenwahl in autobahnnetzen – teil 1: Analyse von staumeldungen. *Straßenverkehrstechnik*, 14(10):621–627, 1999.

[327] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter. Adap-

tivity and Self-Organization in Organic Computing Systems. *ACM Transactions on Autonomous and Adaptive Systems*, 5(3):1–32, 2010.

[328] B. Schmerl, J. Andersson, T. Vogel, M. B. Cohen, C. M. F. Rubira, Y. Brun, A. Gorla, F. Zambonelli, and L. Baresi. Challenges in Composing and Decomposing Assurances for Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems III. Assurances*, volume 9640, pages 64–89. Springer, 2017.

[329] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to Context than Location. *Computers & Graphics*, 23(6):893–901, 1999.

[330] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006.

[331] H. Seebach, F. Nafz, J.-P. Steghofer, and W. Reif. A Software Engineering Guideline for Self-Organizing Resource-Flow Systems. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 194–203. IEEE, 2010.

[332] M. Segata, S. Joerer, B. Bloessl, C. Sommer, F. Dressler, and R. Lo Cigno. PLEXE: A Platooning Extension for Veins. In *Proceedings of the Vehicular Networking Conference (VNC)*, pages 53–60. IEEE, 2014.

[333] S. Selvakennedy, S. Sinnappan, and Y. Shang. A Biologically-Inspired Clustering Protocol for Wireless Sensor Networks. *Computer Communications*, 30(14–15):2786–2801, 2007.

[334] M. Shackleton, F. Saffre, R. Tateson, E. Bonsma, and C. Roadknight. Autonomic Computing for Pervasive ICT - A Whole-System Perspective. In *Intelligent Spaces*, Computer Communications and Networks, pages 323–335. Springer, 2004.

[335] S. Sicard, F. Boyer, and N. De Palma. Using Components for Architecture-Based Management: The Self-Repair case. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 101–110. ACM, 2008.

[336] V. A. Silva Souza. *Requirements-based Software System Adaptation.* Phd thesis, University of Trento, 2012.

[337] H. A. Simon. The Science of Design: Creating the Artificial. *Design Issues*, 4(1/2):67–82, 1988.

[338] O. Simonin. Construction of Numerical Potential Fields with Reactive Agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1351–1352. ACM, 2005.

[339] C. Simpkins, S. Bhat, C. Isbell, and M. Mateas. Towards Adaptive Programming: Integrating Reinforcement Learning into a Programming Language. *ACM SIGPLAN Notices*, 43(10):603–614, 2008.

[340] G. Smith and J. W. Sanders. Formal Development of Self-organising Systems. In *Autonomic and Trusted Computing*, volume 5586 of *LNCS*, pages 90–104. Springer, 2009.

[341] P. L. Snyder, G. Valetto, J. L. Fernandez-Marquez, and G. Di Marzo Serugendo. Augmenting the Repertoire of Design Patterns for Self-Organized Software by Reverse Engineering a Bio-Inspired P2P System. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 199–204. IEEE, 2012.

[342] B. Solomon, D. Ionescu, M. Litoiu, and M. Mihaescu. A real-time adaptive control of autonomic computing environments. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*, page 124. ACM, 2007.

[343] M. Sommer, S. Tomforde, and J. Hähner. An organic computing approach to resilient traffic management. In *Autonomic Road Transport Support Systems*, pages 113–130. Springer, 2016.

[344] J. P. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw. Task-based Adaptation for Ubiquitous Computing. *IEEE Transactions on Systems, Man and Cybernetics – Part C Applications and Reviews*, 36(3):328–340, 2006.

[345] M. Stein, A. Frömmgen, R. Kluge, F. Löffler, A. Schürr, A. Buchmann, and M. Mühlhäuser. TARL: Modeling topology adaptations for networking applications. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 57–63. ACM, 2016.

[346] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland. A Concise Introduction to Autonomic Computing. *Advanced Engineering Informatics*, 19(3):181–187, 2005.

[347] J. Sudeikat, L. Braubach, A. Pokahr, W. Renz, and W. Lamersdorf. Systematically engineering self-organizing systems: The SodekoVS approach. *Electronic Communications of the EASST*, 17, 2009.

[348] J. Sun and I. Satoh. Specifying Distributed Adaptation through Software Component Relocation. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 337–342. IEEE, 2015.

[349] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[350] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue. Learning revised models for planning in adaptive systems. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 63–71. ACM, 2013.

[351] D. Sykes, W. Heaven, J. Magee, and J. Kramer. From Goals To Components: A Combined Approach To Self-Management. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 1–8. ACM, 2008.

[352] C. Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley, 2nd edition, 2002.

[353] H. Tajalli, J. Garcia, G. Edwards, and N. Medvidovic. PLASMA: A plan-based layered architecture for software model-driven adaptation. In *Proceedings of the International Conference on Automated Software Eengineering (ASE)*, pages 467–476. ACM, 2010.

[354] M. Tanabe, K. Tei, Y. Fukazawa, and S. Honiden. Learning environment model at runtime for self-adaptive systems. In *Proceedings of the Symposium on Applied Computing (SAC)*, pages 1198–1204. ACM, 2017.

[355] E. Tanter, J. Noyé, D. Caromel, and P. Cointe. Partial Behavioral Reflection: Spatial and Temporal Selection of Reification. In *Proceedings of the Conference on Object-oriented programming systems, languages and applications (OOPSLA)*, pages 27–46. ACM, 2003.

[356] R. Tateson. Self-Organising Pattern Formation: Fruit Flies and Cell Phones. In *Parallel Problem Solving from Nature – PPSN V*, volume 1498 of *LNCS*, pages 732–741. Springer, 1998.

[357] R. Tateson, S. Howard, and R. Bradbeer. Nature-Inspired Self-Organisation in Wireless Communications Networks. In *Engineering Self-Organising Systems*, volume 2977, pages 63–74. Springer, 2004.

[358] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White. A Multi-Agent Systems Approach to Autonomic Computing. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) – Vol. 1*, pages 464–471. IEEE, 2004.

[359] F. Tisato, A. Savigni, W. Cazzola, and A. Sosio. Architectural Reflection: Realising Software Architectures via Reflective Activities. In *Engineering Distributed Objects*, volume 1999 of *LNCS*, pages 102–115. Springer, 2001.

[360] S. Tomforde. *An Architectural Framework for Self-configuration and Self-improvement at Runtime.* Phd thesis, Universität Hannover, 2011.

[361] S. Tomforde and C. Müller-Schloer. Incremental Design of Adaptive Systems. *Journal of Ambient Intelligence and Smart Environments*, 6(2):179–198, 2014.

[362] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck. Observation and Control of Organic Systems. In *Organic Computing – A Paradigm Shift for Complex Systems*, pages 325–338. Springer, 2011.

[363] S. Tomforde, H. Prothmann, F. Rochner, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck. Decentralised Progressive Signal Systems for Organic Traffic Control. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 413–422. IEEE, 2008.

[364] M. Treiber and A. Kesting. *Traffic Flow Dynamics - Data, Models and Simulation.* Springer, 2013.

[365] H. Van Dyke Parunak. "Go to the Ant": Engineering Principles from Natural Multi-Agent Systems. *Annals of Operations Research*, 75:69–101, 1997.

[366] H. Van Dyke Parunak, J. Sauter, M. Fleischer, and A. Ward. The RAPPID Project: Symbiosis between Industrial Requirements and MAS Research. *Autonomous Agents and Multi-Agent Systems*, 2(2):111–140, 1999.

[367] Y. Vandewoude, P. Ebraert, Y. Berbers, and T. D'Hondt. Tranquility: A Low Disruptive Alternative to Quiescence for Ensuring Safe Dynamic Updates. *IEEE Transactions on Software Engineering*, 33(12):856–868, 2007.

[368] S. VanSyckel. *System Support for Proactive Adaptation.* Phd thesis, University of Mannheim, Germany, 2015.

[369] S. VanSyckel, D. Schäfer, G. Schiele, and C. Becker. Configuration Management for Proactive Adaptation in Pervasive Environments. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 131–140. IEEE, 2013.

[370] S. VanSyckel, G. Schiele, and C. Becker. Extending context management for proactive adaptation in pervasive environments. In *Ubiquitous Information Technologies and Applications*, pages 823–831. Springer Netherlands, 2013.

[371] E. Vassev and J. Paquet. ASSL – Autonomic System Specification Language. In *Proceedings of the Software Engineering Workshop (SEW)*, pages 300–309. IEEE, 2007.

[372] V. Venkatesh. *Road to Success: A Guide for Doctoral Students and Junior Faculty Members in the Behavioral and Social Sciences.* Dog Ear Publishing, 2011.

[373] C. Villalba and F. Zambonelli. Towards nature-inspired pervasive service ecosystems: Concepts and simulation experiences. *Journal of Network and Computer Applications*, 34(2):589–602, 2011.

[374] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas. A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 80–89. ACM, 2011.

[375] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas. *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems*, pages 265–293. LNCS. Springer, Berlin, Heidelberg, 2013.

[376] M. Viroli and M. Casadei. Biochemical Tuple Spaces for Self-organising Coordination. In *Coordination Models and Languages*, volume 5521 of *LNCS*, pages 143–162. Springer, 2009.

[377] T. Vogel. *Model-Driven Engineering of Self-Adaptive Software*. Phd thesis, University of Potsdam, 2018.

[378] T. Vogel and H. Giese. Adaptation and Abstract Runtime Models. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 39–48. ACM, 2010.

[379] T. Vogel and H. Giese. A Language for Feedback Loops in Self-Adaptive Systems: Executable Runtime Megamodels. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 129–138. IEEE, 2012.

[380] T. Vogel and H. Giese. Model-Driven Engineering of Self-Adaptive Software with EUREMA. *ACM Transactions on Autonomous and Adaptive Systems*, 8(4):18:1–18:33, 2014.

[381] T. Vogel and H. Giese. On Unifying Development Models and Runtime models. In *Proceedings of the International Workshop for Models@run.time (Models@run.time)*, pages 5–10. CEUR-WS.org, 2014.

[382] T. Vogel, A. Seibel, and H. Giese. The Role of Models and Megamodels at Runtime. In *Models in Software Engineering*, volume 6627 of *LNCS*, pages 224–238. Springer, 2011.

[383] M. Vrbaski, G. Mussbacher, D. Petriu, and D. Amyot. Goal Models as Runtime Entities in Context-Aware Systems. In *Proceedings of the International Workshop for Models@run.time (Models@run.time)*, pages 3–8. ACM, 2012.

[384] P. Vromant, D. Weyns, S. Malek, and J. Andersson. On interacting control loops in self-adaptive systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 202–207. ACM, 2011.

[385] L. Wang. Search-Based Adaptation Planning Framework for Self-Adaptive Systems. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 465–466. IEEE/ACM, 2017.

[386] L. Wang. Using Search-Based Software Engineering to Handle the Changes with Uncertainties for Self-Adaptive Systems. In *Proceedings of the ACM SIGSOFT Symposium and the European Conference on Foundations of Software Engineering (ESEC/FSE)*, pages 1014–1017. ACM, 2017.

[387] M. Weißbach, P. Chrszon, T. Springer, and A. Schill. Decentrally coordinated execution of adaptations in distributed self-adaptive software systems. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 111–120. IEEE, 2017.

[388] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, 1991.

[389] I. Welch and R. J. Stroud. Kava - Using Bytecode Rewriting to add Behavioral Reflection to Java. In *Proceedings of the Conference on Object-Oriented Technologies and Systems (COOTS)*, pages 119–130. USENIX, 2001.

[390] M. Wermelinger. Towards a Chemical Model for Software Architecture Reconfiguration. In *Proceedings of the International Conference on Configurable Distributed Systems (CDS)*, pages 111–118. IEEE, 1998.

[391] M. Wermelinger, A. Lopes, and J. L. Fiadeiro. A Graph Based Architectural (Re)configuration Language. *ACM SIGSOFT Software Engineering Notes*, 26(5):21–32, 2001.

[392] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 142–153. ACM, 2005.

[393] D. Weyns. *Architecture-Based Design of Multi-Agent Systems.* Springer, 2010.

[394] D. Weyns. Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges. In *Handbook of Software Engineering.* Springer, to be published.

[395] D. Weyns, N. Boucké, and T. Holvoet. A Field-Based Versus a Protocol-Based Approach for Adaptive Task Assignment. In *Autonomous Agents and Multi-Agent Systems*, volume 17, pages 288–319. Springer, 2008.

[396] D. Weyns and M. U. Iftikhar. Model-based Simulation at Runtime for Self-adaptive Systems. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 364–373. IEEE, 2016.

[397] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad. A Survey of Formal Methods in Self-adaptive Systems. In *Proceedings of the International C* Conference on Computer Science and Software Engineering (C3S2E)*, pages 67–79. ACM, 2012.

[398] D. Weyns, S. Malek, and J. Andersson. FORMS: A Formal Reference Model for Self-adaptation. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 205–214. ACM, 2010.

[399] D. Weyns, S. Malek, and J. Andersson. On Decentralized Self-Adaptation: Lessons from the Trenches and Challenges for the Future. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 84–93. ACM, 2010.

[400] D. Weyns, S. Malek, and J. Andersson. FORMS: Unifying Reference Model for Formal Specification of Distributed Self-Adaptive Systems. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1), 2012.

[401] D. Weyns, B. R. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka. On Patterns for Decentralized Control in Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *LNCS*, pages 76–107. Springer, 2013.

[402] J. White, D. C. Schmidt, K. Czarnecki, C. Wienands, G. Lenz, E. Wuchner, and L. Fiege. Automated Model-based Configuration of Enterprise Java Applications. In *Proceedings of the International Enterprise Distributed Object Computing Conference, (EDOC)*, pages 301–312. IEEE, 2007.

[403] S. White, J. Hanson, I. Whalley, D. Chess, and J. Kephart. An architectural approach to autonomic computing. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 2–9. IEEE, 2004.

[404] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J. Bruel. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 79–88. IEEE, 2009.

[405] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel. RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15(2):177–196, 2010.

[406] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering.* Springer, 2012.

[407] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy. Traffic Routing for Evaluating Self-Adaptation. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 27–32. IEEE, 2012.

[408] R. K. Yin. Advancing rigorous methodologies: A review of "towards rigor in reviews of multivocal literatures... ". *Review of Educational Research*, 61(3):299–305, 1991.

[409] E. S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 226–235. IEEE, 1997.

[410] X. Yu. Approaches and principles of fall detection for elderly and patient. In *Proceedings of the International Conference on e-health Networking, Applications and Services (HealthCom)*, pages 42–47. IEEE, 2008.

[411] F. Zambonelli and M. Viroli. A survey on nature-inspired metaphors for pervasive service ecosystems. *International Journal of Pervasive Computing and Communications*, 7(3):186–204, 2011.

[412] H. Zhang and J. Llorca. Nature-Inspired Self-Organization, Control, and Optimization in Heterogeneous Wireless Networks. *IEEE Mobile Computing*, 11(7):1207 – 1222, 2012.

[413] J. Zhang and B. H. C. Cheng. Specifying Adaptation Semantics. *SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.

[414] J. Zhang and B. H. C. Cheng. Model-Based Development of Dynamically Adaptive Software. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 371–308. ACM, 2006.

[415] J. Zhang, H. J. Goldsby, and B. H. C. Cheng. Modular Verification of Dynamically Adaptive Systems. In *Proceedings of the International Con-*

ference on Aspect-oriented Software Development (AOSD), pages 161–172. ACM, 2009.

[416] T. Zhao, W. Zhang, H. Zhao, and Z. Jin. A Reinforcement Learning-based Framework for the Generation and Evolution of Adaptation Rules. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 103–112. IEEE, 2017.

[417] P. Zoghi, M. Shtern, and M. Litoiu. Designing Search Based Adaptive Systems: A Quantitative Approach. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 7–16. ACM, 2014.

[418] P. Zoghi, M. Shtern, M. Litoiu, and H. Ghanbari. Designing Adaptive Applications Deployed on Cloud Environments. *ACM Transactions on Autonomous and Adaptive Systems*, 10(4):1–26, 2016.

Appendix

A. Appendix to the Related Work

This chapter present further information regarding the analysis of related work from Chapter 4. First, Section A.1 presents the overview on the analyzed approaches from [220] and [229]. Second, Section A.2 introduces the dimensions of the taxonomy from [224]. Last, Section A.3 shows an overview of the comparison of development approaches from [224].

A.1. Overview of the Engineering Approaches from [229] and [220]

Table A.1 shows an overview of the most relevant approaches that we presented in Section 4.1 and shows their relation to MAPE activities, the taxonomy presented in Section 2.1.3, and the issues for implementing an adaptation logic of Section 2.2. It is based on [220] and [229]. Within the table, we used the following abbreviations:

- **M/A/P/E** = Monitoring, Analyzing, Planning, Executing
- **Re** = Reactive; **Pro** = Proactive;
- **Ctx** = Context; **TR** = Technical resources; **U** = User(s);
- **App** = Application; **Com** = Communication; **Sys** = System Software;
- **Tec** = Technique; **Par** = Parameter; **Str** = Structure;
- **Appr** = Approach; **Ext** = External; **Int** = Internal;
- **DC** = Decision Criteria; **G** = G; **M** = M; **U** = U; **P** = P
- **DDec** = Degree of Decentralization; **Hyb** = Hybrid; **Dec** = Decentralized; **Cen** = Centralized

Cell marked with "-" indicates that the approach does not have a specific requirement or that no further information is given.

A.1. Overview of the Engineering Approaches from [229] and [220]

	Approach	MAPE	Time	Reason	Level	Tec	Appr	DC	DDec
Model-based	(Dynamic) Software Product Lines [1,4,28,30,38,51,72,85,136,157,164, 166,235,262,293,294,321]	A/P	R	Ctx/ TR/ U	App/ Com/ TR	Par/ Str	Ext	G, P, U	-
	MUSIC [128,145,163]	All	R	Ctx/ TR	App/ Com	Par/ Str	Ext	M, U	Hyb/ Cen
	Meta-M/ MegaM [378,379,381,382]	All	-	-	-	-	Ext	M	-
	MechatronicUML [154,176]	All	R	Ctx/ TR	TR	Par/ Str	Ext	M	All
Architecture-based	Rainbow Framework [140]	All	R	TR	App/ TR	Par/ Str	Ext	M, P	All
	3L Approach [218]	All	R	Ctx/ TR	App/ TR	Par/ Str	Ext	G	All
	Architectural Run-time Configuration Manager [147]	All	R	TR	App/ TR	Par/ Str	Ext	P, M	-
	Archstudio [278,279]	All	R	Ctx/ TR	App/ TR	Par/ Str	Ext	M	All
Reflection-based	Introspection [254]	M/A	-	Ctx/ TR/ U	-	-	Both	-	-
	Intercession [254]	P/E	-	-	All	All	Both	-	-
	Reflection Reference Model [13]	All	R	Ctx/ TR	App/ TR	Par/ Str	Ext	(Meta) M	-
	FORMS	See category "Formal Modelling and Verification Approaches"							
	Reflex [355]	All	R	TR	App/ TR	Par/ Str	Ext	(Meta) M	-
	Kava [389]	All	R	TR	App/ TR	Par/ Str	Ext	(Meta) M	-
	Reflective Middleware [48,64,212]	P/E	-	-	Sys/ Com	Par/ Str	-	-	-
	CARISMA [64]	All	R	Ctx/ TR	App/ TR	Par	Ext	P	Dec/ Hyb
Paradigms	Component-Based SE [6, 33, 36, 37, 47, 107, 187, 195, 209, 218, 240, 254, 275,335,351]	P/E	-	-	App/ TR	Str	Ext	All	All
	Aspect-Oriented Programming [162, 202,262,263]	P/E	-	-	App/ TR	Str	Ext	All	All

	Approach	MAPE	Time	Reason	Level	Tec	Appr	DC	DDec
	Generative Programming [254, 276, 402]	P/E	-	-	App/ TR	Par/ Str	Ext	All	All
	Adaptive Programming [3, 158, 339]	Addresses various aspects in the development of SAS							
	Context-oriented Programming [22, 184, 197]	A/P/ E	R	Ctx/ TR	App/ TR/ Ctx	All	-	M, G	-
Control Theory	Autonomic Computing [198]	All	R	TR/ Ctx	Sys	Par	Ext	P, G	Dec
	Autonomic Communication [105]	All	R	TR/ Ctx	Com	Par/ Ctx	-	-	-
	Control Loop Patterns [267]	All	R	Ctx	Sys/ TR	Par	-	M	-
	Control Loop UML Profile [175]	All	R	TR	Com	Par/ Str	Ext	-	Dec
	Control Theory Foundation [104]	All	R	TR/ Ctx	Sys	Par	Ext	M	-
Service-oriented	MUSIC	See category "Model-based Approaches"							
	SASSY Framework [156, 255]	All	R	Ctx/ TR	App	Str	Ext	G, U	Dec/ Hyb
	MetaSelf [102]	All	R	Ctx/ TR	App	Str	Ext	P, Rules	All
	QoSMOS [60]	All	R	Ctx/ TR	App	Str	Ext	M	Cen
	Aspect-oriented and Service-oriented Computing [73, 188]	P/E	-	-	App	Str	Ext	-	-
	Agent systems and SOA	See category "Agent-based Approaches"							
	Component M and services [71, 195]	All	R	TR	App	Str	Ext	-	-
	CARE method [303]	A,P,E	R	Ctx/ TR/ U	App	Str	Ext	G, U	Cen
	MOSES Framework [65]	All	R	TR	App	Str	Ext	G, U	Cen
Agent-based	MOCAS [26]	All	R	Ctx	Sys	Par	-	P	Dec
	Design Patterns [100]	P	R	Ctx	App	Par	-	M	Dec
	Agent-Based Modeling, Dynamical Systems Analysis, and Decentralized Control [97]	All	R	Ctx	App	Par	-	M	Dec/ Hyb

	Approach	**MAPE**	**Time**	**Reason**	**Level**	**Tec**	**Appr**	**DC**	**DDec**
	Unity	See category "Learning Approaches"							
Nature-inspired	Optimization [106, 333, 412]	P	R	TR/ Ctx	Sys	Par/ Str	-	U	Dec
	Autonomic Computing for Pervasive ICT [334]	P	R	TR	App/ Sys	Par/ Str	-	-	Dec
	Frequency Planning [356, 357]	P	R	TR	App	Par	-	-	Dec
	Data Harvesting [236]	P	R	Ctx	App	Par	-	U	Dec
	Network Synchronicity [392]	P	R	TR	Sys	Par	-	-	Dec
	Region Detection [52]	P	R	Ctx	App	Par	-	-	Dec
	Immune System [90]	P	R	Ctx	Sys	Par	-	-	Dec/ Hyb
	Obstacle Avoidance [201]	P	R	Ctx	App	Par	Int	U	-
	Potential Fields [68, 310, 338]	P	R	Ctx	App	Par	-	-	Dec/ Hyb
	Task Assignment [395]	P	R	Ctx	App	Par	-	U	Dec
	Ecosystem Framework [373]	-	R	Ctx	App	Par	-	U	Dec
	RAPPID [366]	P	R	Ctx	App	Par	-	-	Dec
	Social Conventions [319]	P	R	Ctx	App	Par	-	-	Dec
Formal Methods	FORMS [400]	All	R	TR/ Ctx	Sys/ TR	Par/ Str	Ext	M	-
	Restore-Invariant Approach [159, 160, 271]	A	R	TR	TR	Str	-	-	-
	Self-Testing Framework [205, 206]	A	R	TR/ Ctx	Sys	Par	Ext	-	Dec
	Stochastic Modeling [130]	M , A, P	R / Pro	TR/ Ctx	TR	Par/ Str	Ext	M, U	Cen
	Markov Chains [122, 125]	A, P	R	TR/ Ctx	TR	Par/ Str	Ext	M	Cen
	Abstract State Machines [18]	All	R	TR/ Ctx	TR	Par/ Str	Ext	M	All
	Timed Hazard Analysis [297]	A/P	R	TR	TR	Str	Ext	M	-
	MAPE-K Formal Templates [92]	All	R	TR/ Ctx / U	TR	Par/ Str	Ext	M	-
	ActiveFORMS [194, 396]	All	R	TR/ Ctx / U	App/ Sys	Par/ Str	Ext	M	Cen

	Approach	**MAPE**	**Time**	**Reason**	**Level**	**Tec**	**Appr**	**DC**	**DDec**
Learning	Unity [358]	P	R	TR	Sys	Par	-	G, U	Dec
	FUSION [112, 113]	P	R	TR/ Ctx	App/ Sys	Par/ Str	-	G, U	-
	Reinforcement Learning [63, 108, 204, 282, 339, 349, 416]	P	R	TR/ Ctx	App	Par/ Str	Ext	U	All
	Evolutionary Algorithms [25,57,298, 362, 363]	All	R	Ctx	App	Par	Ext	U	All
	Control-Based Framework [2]	P	R	Ctx	Sys	Par	Ext	M	Cen
	Fossa	A, P	Provides learning of rules at design time						
	SBSE [14, 15, 84, 170, 172, 173, 255, 308, 385, 386, 417, 418]	P	R	All	App	Par / Str	Ext	M, U	Cen
RE-based	LoREM [155]	All	R	-	-	Par	-	M	-
	Requirements@Runtime [287]	A/P	R	U	App	Par/ Str	-	G	-
	Zanshin [336]	All	R	Ctx	App	Par	-	G	-
Further	Task-based adaptation [143, 295, 344]	A/P	-	-	App/ TR	-	-	G, U	-
	Middleware-centric adaptation [165, 262, 317]	All	R	Ctx/ TR	App/ TR	Par/ Str	Ext	All	Dec/ Hyb
	Adaptation languages [8, 17, 78, 348]	Different type of support: Some support reasoning on adaptation, some the design of adaptive systems							
	Processes [11, 231, 238, 331, 361, 414]	Does support the design/development activities of all types of SAS; does not influence the actual system at run-time adaptation							
	Modeling Dimensions [12, 55]	Support of design, rather than the specific implementation of the adaptation logic							
	Design patterns [23, 87, 100, 120, 132, 133, 139, 231, 270, 305, 341, 401]	Does support many different aspects, e.g., design of the adaptation logic, information dissemination, reasoning, or execution							

Table A.1.: Overview of the Approaches of the Surveys in [220] and [229].

A.2. Taxonomy of the Development Approaches from [224]

In the following, this section presents the dimensions of the taxonomy. Figure A.1 provides of the taxonomy. Table A.2 summarizes the 18 dimensions of the taxonomy as well as their characteristics. In the following, this section presents the dimensions of the taxonomy.

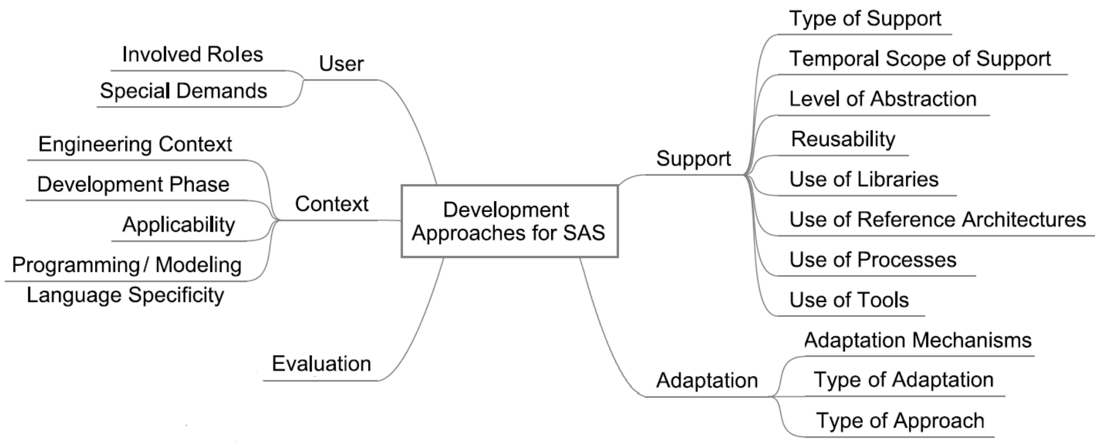


Figure A.1.: Taxonomy for comparing the development approaches for SASs.

Type of approach describes the underlying key concept the approach is based on. In accordance with [229], possible manifestations are *"model-based", "architecture-based", "control-based", "service-oriented", "agent-based", "nature-inspired",* and *"design concepts"*.

Type of support helps to find a suitable approach for a certain problem. Due to the diversity of development approaches, also the form of support is very diverse. It includes *"framework", "tools", "design principles", "guidelines",* and *"methodologies"*.

Temporal scope of support signifies the temporal scope of the different components that provide support. The temporal scope can be *"design-time", "run-time"* or both.

Involved roles describes which parties are involved in an approach. Some approaches make a clear and precise statement regarding involved parties, some only distinguish between designers and developers.

Reusability refers to the reusability an approach offers. This includes, e.g., reusable process elements and components, reference architectures, component and design libraries, generic middleware, modeling languages, and design concepts. Some approaches consider reusability at a high abstraction level neglecting lower abstraction levels [228], others do not consider reusability at all.

Development phase states in which phase of the software engineering process the approach should be applied. This covers phases of traditional and modern engineering processes.

Engineering context describes the engineering context and, thus, the integrability of an approach with the chosen software development process. Several approaches limit their application possibilities to traditional forward engineering, some broaden the applicability to modern engineering contexts, and others do not limit them at all.

Applicability of an approach can be general or specific. SAS are deployed in many different system domains. Hence, SAS can differ in their structure and functionality, based on specific system domain requirements. To meet these requirements, several approaches support the design and construction of specific system types.

Special demands on developer covers requirements a developer or designer must possess, such as specific programming / modeling languages or others.

Level of abstraction describes the degree of abstraction of the provided support. Design principles have a high-level abstraction and do not offer concrete implementations, whereas tools and frameworks provide a low-level abstraction, as they directly facilitate the construction of software artifacts.

Use of processes describes the existence and content of processes. Several approaches provide new software development processes tailored to the development of SAS. By contrast, others do neither name nor explain their processes.

Use of reference architectures describes whether an approach makes use of a reference architecture and how it is used. Reference architectures serve as architectural templates for the construction of software systems with self-adaptivity properties. However, the structure and functionality can differ.

Use of libraries contains information about libraries, such as component libraries (e.g., based MAPE [198]), design pattern libraries (e.g., [23]), and coordination mechanisms (cf. [401]).

Use of tools can support the specification of requirements, the system design, the implementation, or the system validation. Some approaches include proprietary tools that support the software system development, whereas others reference common and open-source software or do not specify tools.

Language specificity states whether an approach is bound to any specific programming or modeling language. Often, frameworks and run-time oriented approaches integrate adaptation logic components and a middleware that are programming language specific. In addition, some approaches require the use of a specific modeling language for designing the SAS.

Support of adaptation mechanisms describes how the approach supports adaptation. Dependent on the temporal scope of the support, the approaches consider adaptation at design- or run-time. Furthermore, adaptation can be achieved e.g., through adaptation and coordination patterns, middleware services, design principles, or the refinement of a model.

Type of adaptation states the granularity of adaptation. It can be *"compositional adaptation"*, *"parameter adaptation"*, or a combination of both.

Evaluation captures the type and extent of evaluation. It is inevitable to use the approaches for developing real-world systems for examining their benefits and challenges. Proofs of concept include case studies, prototypes, or interviews.

Dimension	**Captured Information**	**Characteristics**
Type of approach	What is the key concept? What aspects does it focus on?	model-based, architecture-based, control-based, service-oriented, agent-based, nature-inspired, design concept, verification
Type of support	What kind of support does it provide? What elements does the approach include?	framework, tools, design concept, guidelines, methodology
Temporal scope of support	Which temporal scope does the support by the approach affect?	design-time, run-time, both
Involved roles	Which kind of parties are involved in the development process? What people does the approach aim at?	Designer, developer, tester, not specified

Dimension	Captured Information	Characteristics
Reusability	Is reusability considered? How is it achieved?	reusable process elements, reusable components, reference architectures, component libraries, design patterns, generic middleware, modeling languages, design concepts
Development phase	In which step(s) of the software development process can it be applied?	design, implementation, both
Engineering context	Which software engineering context does it suit?	Forward Eng., reverse engineering, not specified
Applicability	Which systems can the approach be applied on?	SAS, CPS, adaptive systems
Special demands on developer	What requirements does the developer have to fulfil? What type of and how much knowledge is demanded in order to use the approach?	none, modeling languages, programming languages, not specified
Level of abstraction	What is the level of abstraction of the approach? Does it solve certain development issues explicitly?	High, medium, low, not specified
Use of processes	Is(are) there any process(es) determined?	Provided, not provided
Use of reference architecture	Does the approach provide a reference architecture? How is it integrated and what is its purpose?	Provided, not provided
Use of libraries	What do they consist of? How are they used?	provided, not provided, not specified
Use of tools	How do the tools support the development? When are they applied?	proprietary tools, open-source tools, no tools
Language specificity	Does the approach require a specific programming or modeling language?	programming language, modeling language, independent
Support of adaptation mechanisms	How does the approach handle the system‘s adaptation? What mechanisms does it utilize?	At design-time (requirements), at run-time (adaptation logic)
Type of adaptation	What is the granularity of the adaptation?	Compositional adaptation, parameter adaptation, both
Evaluation	Has the approach already been evaluated? How is it tested?	Case studies, Industry cooperations, prototyping, surveys, no evaluation

Table A.2.: Taxonomy of Development Approaches [224]

A.3. Overview of the Development Approaches from [224]

The appendix presents the analysis of development approaches from [224]. Additionally, for this thesis, EUREMA [377, 380] and DESCARTES [190, 215] are added. The comparison is based on the aforementioned taxonomy. Tools are omitted due to the variety of tools but also the fact that many tools are not publicly available.

Table A.3.: Analysis of Development Approaches (1)

Title of Approach	**Year**	**Type of Approach**	**Type of Support**	**Temporal Scope**
Rainbow [77, 140]	2004	Architecture-based	Framework, Tools	Run-time
Model-Driven Approach [66]	2008	Model-based	Framework	Run-time
Meta-Self [102]	2008	Service-oriented	Framework	Design-time, Run-time
SodekoVS [347]	2009	Agent-based	Framework	Design-time, Run-time
MUSIC [163]	2012	Model-based, Service-oriented	Framework, Tools, Modeling language	Design-time, Run-time
Arch. Fr. for Self-Conf. & Self-Impr. [362]	2011	Architecture-based	Framework	Run-time
FUSION [112]	2010	Model-based	Framework	Run-time
SASSY [255]	2011	Service-oriented	Framework, Tools	Design-time, Run-time
Zanshin [336]	2012	Control-based	Framework	Design-time, Run-time
StarMX [20]	2009	Architecture-based	Framework	Design-time, Run-time
MOSES [65]	2012	Service-oriented	Framework	Design-time, Run-time
Software Mobility Framework [247]	2010	Architecture-based	Framework	Run-time
GRAF [10]	2012	Model-based	Framework	Design-time, Run-time
DESCARTES [190, 215]	2016/17	Model-based	Framework	Design-time, Run-time
EUREMA [377, 380]	2018	Model-based	Framework	Design-time, Run-time
Software Engineering Guideline [331]	2010	Agent-based	Guideline, Pattern	Design-time, Run-time
Development Approach and Automatic Process [5]	2015	Architecture-based	Guidelines, Reference Architecture	Design-time, Run-time
SE Processes for SAS [11]	2013	Model-based	Process	Design-time
Modeling Dimensions [13]	2009	Design Concept	Design principles and concepts	Design-time
Design Space [55]	2013	Design Concept	Design principles and concepts	Design-time
High Quality Specification [242]	2013	Model-based	Specification methodology	Design-time
Behavioral corridors [271]	2010	Verification-based	Methodology	Design-time
General Meth. for Designing SOSs [151]	2007	Design concept	Methodology	Design-time
FORMS [398]	2010	Model-based	Methodology	Design-time
DYNAMICO [375]	2010/ 2013	Control-based	Methodology	Design-time

Table A.4.: Analysis of Development Approaches (2)

Title of Approach	**Involved Parties**	**Reusability**	**Development Phase**	**Engi-neering**
Rainbow [77, 140]	Developer	Infrastructure, Architecture	Implementation	not specified
Model-Driven Approach [66]	Developer	Architecture, Processes	Implementation	not specified
Meta-Self [102]	Designer, Developer	Infrastructure	Design, Implementation	not specified
SodekoVS [347]	Developer, Designer	Components, Architecture, Library	Entire development process	Forward
MUSIC [163]	Developer, Designer	Middleware	Entire development	Forward
Arch. Fr. for Self-Conf. & Self-Impr. [362]	Developers	Architecture	Implementation	Forward
FUSION [112]	Developers	Architecture	Implementation	Forward
SASSY [255]	Developer, Designer	Processes, Components, Architecture	Design, Implementation	Forward
Zanshin [336]	Developer, Designer	Processes, Components, Reference Architecture	Design, Implementation	Forward
StarMX [20]	Developer, Designer	Components	Design, Implementation	Forward
MOSES [65]	Developer	Methods, Components	Design, Implementation	Forward
Software Mobility Framework [247]	Developers	Architecture	Implementation	Forward
GRAF [10]	Developer, Designer	Architecture	Design, Implementation	Forward
DESCARTES [190, 215]	Designer, Developer	Processes, Components	Design	Forward
EUREMA [377, 380]	Designer, Developer	Method, Components	Design	Forward
Software Engineering Guideline [331]	Designer, Developer	Design Pattern, Method	Design, Implementation	Forward
Development Approach and Automatic Process [5]	Developer, Designer	Architecture, Guidelines	development process	Forward
SE Processes for SAS [11]	Designer	Process	Design	Forward
Modeling Dimensions [13]	Designer	Concepts	Design	Forward, Reverse
Design Space [55]	Designer	Concepts	Design	not specified
High Quality Specification [242]	Designer	Modeling Language	Design	Forward
Behavioral corridors [271]	Designer	Concepts	Design	Forward
General Meth. for Designing SOSs [151]	Design-time	not specified	Design	Forward, Reverse
FORMS [398]	Design-time	not specified	Design	Forward
DYNAMICO [375]	Design-time	not specified	Design	Forward

Table A.5.: Analysis of Development Approaches (3)

Title of Approach	**Applicability**	**Special Demands on Developer**	**Level of Abstraction**
Rainbow [77, 140]	SASs	Mathematical knowledge	Low
Model-Driven Approach [66]	Pervasive Systems	not specified	Low
Meta-Self [102]	SASs and SOSs	not specified	not specified
SodekoVS [347]	SASs	not specified	not specified
MUSIC [163]	SASs	not specified	Low
Arch. Fr. for Self-Conf. & Self-Impr. [362]	SASs and SOSs	not specified	Low
FUSION [112]	SASs	Machine Learning	High
SASSY [255]	SASs	not specified	High
Zanshin [336]	SASs	Requirements Modeling	High
StarMX [20]	SASs	not specified	High
MOSES [65]	SASs	not specified	High
Software Mobility Framework [247]	SASs	not specified	High
GRAF [10]	SASs	TGraphs	Medium
DESCARTES [190, 215]	SASs	Descartes Modelling Language	High
EUREMA [377, 380]	SASs	Eurema Modelling Language	High
Software Engineering Guideline [331]	Resource Flow Systems	none	Medium
Development Approach and Automatic Process [5]	SASs	not specified	Low
SE Processes for SAS [11]	SASs	none	High
Modeling Dimensions [13]	SASs	none	High
Design Space [55]	SASs	none	High
High Quality Specification [242]	SASs	Modeling language (ACML)	High
Behavioral corridors [271]	SASs and SOSs	ITL+, Simple Programming Language	High
General Meth. for Designing SOSs [151]	SOSs	none	High
FORMS [398]	SASs	none	High
DYNAMICO [375]	SASs	none	High

Table A.6.: Analysis of Development Approaches (4)

Title of Approach	**Use of Processes**	**Use of Reference Architecture**	**Use of Libraries**
Rainbow [77, 140]	not specified	not specified	none
Model-Driven Approach [66]	not specified	Architecture based on communication channels	not specified
Meta-Self [102]	Design- and run-time activities	not specified	none
SodekoVS [347]	Self-Organized Coordination Engineering	Configuration and integration of self-organizing processes	Catalog of coordination patterns
MUSIC [163]	Model-driven development methodology with tasks for every development phase	none	none
Arch. Fr. for Self-Conf. & Self-Impr. [362]	not specified	Observer/ Controller architectures	None
FUSION [112]	not specified	MAPE-K based	None
SASSY [255]	Model-driven development methodology	Service-oriented architecture	none
Zanshin [336]	not specified	Automated monitoring	None
StarMX [20]	Workflow for the entire development process	Execution Chain Architecture	none
MOSES [65]	not specified	MAPE-K based	-
Software Mobility Framework [247]	not specified	MAPE-K based	None
GRAF [10]	For modeling	Transformation of runtime models in rules	None
DESCARTES [190, 215]	For modeling	MAPE-K based	None
EUREMA [377, 380]	not specified	MAPE-K based	None
Software Engineering Guideline [331]	Development guideline with different steps	none	none
Development Approach and Automatic Process [5]	Workflow for the entire development process	Reference architecture consisting of modules	none
SE Processes for SAS [11]	Workflow for the entire development process	none	none
Modeling Dimensions [13]	Application of the design dimensions	none	none
Design Space [55]	Application of the design space principles	none	none
High Quality Specification [242]	Application of the ACML within the requirements specification, analysis and design phases	none	none
Behavioral corridors [271]	Workflow for entire development process	none	none
General Meth. for Designing SOSs [151]	Workflow for entire development process	none	none
FORMS [398]	Workflow for entire development process	MAPE-K based	none
DYNAMICO [375]	Workflow for entire development process	MAPE-K based	none

Table A.7.: Analysis of Development Approaches (5)

Title of Approach	**Support of Adaptation Mechanisms**	**Programming Language Specifity**
Rainbow [77, 140]	not specified	Java, XML
Model-Driven Approach [66]	At run-time, Adaptation through models	none
Meta-Self [102]	At run-time, Adaptation through application of coordination/adaptation services	none
SodekoVS [347]	At design-time, Adaptation through coordination mechanisms	not specified
MUSIC [163]	At run-time, Adaptation through MUSIC middleware	Java, OSGi
Arch. Fr. for Self-Conf. & Self-Impr. [362]	At run-time, Adaptation through modules	not specified
FUSION [112]	At run-time, Adaptation through modules	Java
SASSY [255]	At run-time, Adaptation through modules	not specified
Zanshin [336]	At run-time, Adaptation through modules	not specified
StarMX [20]	At run-time, Adaptation through modules	J2EE
MOSES [65]	At design- and run-time, Composition of SOAs	J2EE
Software Mobility Framework [247]	At run-time, Adaptation through modules	not specified
GRAF [10]	At design- modeling; at run-time, Adaptation through modules	Java, XML
DESCARTES [190, 215]	At run-time, Adaptation through modules	not specified
EUREMA [377, 380]	At run-time, Adaptation through modules	not specified
Software Engineering Guideline [331]	At design- and run-time, Adaptation through construction and execution of Organic Design Pattern	not specified
Development Approach and Automatic Process [5]	At run-time, Adaptation through modules	not specified
SE Processes for SAS [11]	not specified	none
Modeling Dimensions [13]	At design-time, Adaptation through design dimension exploration	none
Design Space [55]	At design-time, Adaptation through design space principles	none
High Quality Specification [242]	At design-time, Adaptation through separation of self-adaptivity concerns	UML
Behavioral corridors [271]	At design-time, definition of adaptive behavior	none
General Meth. for Designing SOSs [151]	At design-time, definition of adaptive behavior	none
FORMS [398]	At design-time, definition of adaptive behavior	none
DYNAMICO [375]	At design-time, definition of adaptive behavior	none

Table A.8.: Analysis of Development Approaches (6)

Title of Approach	Use of Tools	Granularity Adaptation	Evaluation
Rainbow [77, 140]	Stitch script editor, Rainbow development toolkit	Compositional	Case studies
Model-Driven Approach [66]	not specified	Compositional	not specified
Meta-Self [102]	none	Compositional	Case studies
SodekoVS [347]	not specified	Compositional	not specified
MUSIC [163]	Tools for modeling, implementation, testing and validation	Compositional, Parameter	Trial development, Testing of a collection of applications
Arch. Fr. for Self-Conf. & Self-Impr. [362]	MASON simulation tool	Compositional, Parameter	Case study
FUSION [112]	XTEAM, WEKA, PRISM-MW	Compositional	Case studies
SASSY [255]	XTEAM, xADL, GME, GReAT	Compositional	Case studies
Zanshin [336]	not specified	Parameter	Case study
StarMX [20]	IBM ABLE, Imperius	Parameter	Case study
MOSES [65]	BPEL	Compositional, Parameter	Case study
Software Mobility Framework [247]	XTEAM, DeSI, PRISM-MW	Compositional, Parameter	Case study
GRAF [10]	TGraph	Parameters	Case studies
DESCARTES [190, 215]	Various tools	Compositional, Parameter	Case study
EUREMA [377, 380]	Modeling tool based on EMF	Compositional	Case study
Software Engineering Guideline [331]	None	not specified	Case study
Development Approach and Automatic Process [5]	not specified	not specified	Case study
SE Processes for SAS [11]	none	not specified	Case study
Modeling Dimensions [13]	none	Compositional, Parameter	Case studies
Design Space [55]	none	none	Case study
High Quality Specification [242]	none	none	Case studies, Projects involving students
Behavioral corridors [271]	none	Parameter	Case study
General Meth. for Designing SOSs [151]	none	not specified	Case study
FORMS [398]	none	not specified	Case study
DYNAMICO [375]	none	not specified	Case study

B. FESAS Documentation

This chapter provides documentation for the FESAS Framework. Sections B.1 and B.2 describes the interfaces for adaptation logic components and functional logic elements, respectively. Section B.3 provides a description of the metadata of functional logics. Section B.4 presents the syntax of the system model files and is split in a description of the devices' information and the communication information. Section B.5 describes the functional logic contracts. Section B.6 presents the interface of the FESAS Repository. Section B.7 outlines the syntax of configuration files for configuring the FESAS components, the SAS, and the connection between SAS and FESAS Repository. Section B.8 explains the interface of the SAS Setup Service.

B.1. Adaptation Logic Component Interface

The following listing presents the interface for components of the adaptation logic, especially MAPE components. It is written in Java. The FESAS Framework and the reference systems provides implementations of all methods.

```java
public interface IAdaptationLogic {

  /** This method is called by the Setup service, after the initialization. */
  public void start();

  /** This method is called by the setup service before the element is
      destroyed. */
  public boolean stop();

  /** Implement/deplement the specified logic or change an algorithm. */
  public void implementLogic(Contract contract) throws
      LogicRepositoryNotFoundException;
  public boolean deplementLogic(Contract contract);
  public boolean changeAlgorithm(String logicString, String algorithm);
```

```

    /** Implements the communication logic for the input/output channel. */
    public void implementInputCommunicationLogic(String communicationType, String communicationID) throws Exception;
    public void implementOutputCommunicationLogic(String communicationType, String communicationID) throws Exception;

    /** Called by predecessor for push of data. */
    public Object receiveData(String sender, String informationType, Object data);

    /** Called by successor for pull of data from element. */
    public void externalRequest(String sucessor);

    /** Called for sending data. */
    public void prepareDataForSending(Object data, String type);

    /** Initialize communication channels. */
    public void addCommunicationToByType(String receiver, CommunicationInformation properties);
    public void addCommunicationFromByType(String sender, CommunicationInformation properties);
    public void addCommunicationToByCategory(String receiver, CommunicationInformation properties);
    public void addCommunicationFromByCategory(String sender, CommunicationInformation properties);

    /** Knowledge management. */
    public String saveKnowledge (IKnowledgeRecord knowledge);
    public IKnowledgeRecord getKnowledge(String knowledgeID);

    /** Get information about service. */
    public String getName();
    public String getFesasID();
    public HashMap<String, String> getProperties();
    public void setProperties(HashMap<String, String> properties);
    public void addProperty(String key, String value);
}
```

Listing B.1: Interface of the FESAS Repository

B.2. Functional Logic Interface

The following listing presents the interface for functional logic components. It is written in Java. The FESAS Framework and the reference systems provides implementations of all methods except of `callLogic(..)`. Developers might overwrite the `initializeLogic(..)` method if specific actions should be triggered while initialization of the functional logic element.

```java
public interface ILogic {

  /** Call Logic Method performing the functionality. */
  public String callLogic(IKnowledgeRecord data);

  /** Setter for the corresponding adaptation logic element. */
  public void setAL(IAdaptationLogic al);

  /** Checks wheter the logic is compatible with the specified data type. */
  public boolean isCompatibleDataType(String dataType);

  /** Called when the element is initialized. */
  public void initializeLogic(HashMap<String, String> properties);

  /** Called for changing the algorithm within the functional logic. */
  public void changeAlgorithm(String algorithm);

  /** Getter for the logic type. */
  public LogicType getLogicType();

  /** Getter for the logic short name. */
  public String getShortName();

  /** Getter for the information category. */
  public InformationCategory getInformationCategory();

  /** Getter for the information category. */
  public String getInformationCategoryAsString();

  /** Setter for the information category. */
  public void setInformationCategory(InformationCategory informationCategory);

  /** Getter for the information type. */
  public InformationType getInformationType();

  /** Getter for the information type as String. */
  public String getInformationTypeAsString();
}
```

Listing B.2: Interface of the FESAS Repository

B.3. Metadata Description of Functional Logics

The following information is part of a JSON file for defining the metadata of a functional logic.

$< ID >$ = String with ID (assigned by the remote repository).
$< NAME >$ = String with a name.
$< SHORT\_NAME >$ = String with the class name only.
$< DESCRIPTION >$ = Object with Description.
$< LOGIC\_TYPE >$ = Logic Type.
$< INFORMATION\_TYPE >$ = String with the information type of generated output.
$< PROGRAMMING\_LANGUAGE >$ = String with the programming language.
$< DEPENDENCIES >$ = Array with dependent classes.
$< SUPPORTED\_INFO\_TYPES >$ = Array with the supported information types (input).
$< PROPERTIES >$ = Array with properties as tuples (key [String], value [String]).

Example for a metadata description in JSON:

```
{
  "ID": "",
  "NAME": "de.test.Monitoring_123",
  "SHORT_NAME": "Monitoring_123",
  "DESCRIPTION": "Bla Blubb",
  "LOGIC_TYPE": "Monitoring",
  "INFORMATION_TYPE": "Monitoring_DEFAULT",
  "PROGRAMMING_LANGUAGE": "Java",
  "DEPENDENCIES": ["Class A", "Class B"],
  "SUPPORTED_INFO_TYPES": "[Sensor_DEFAULT,Sensor_CLOUD]",
  "PROPERTIES": [{"key": "a_k","value": "a_v"},{"key": "b_k","value": "b_v"}]
}
```

Listing B.3: Example for a Metadata Description in JSON

B.4. Syntax of the System Model Files

This section provides the syntax of the system model.

B.4.1. Adaptation Logic System Model

The following information is part of a JSON configuration file for specifying the adaptation logic of a (sub)system:

$< AL\_ADAPTATIONLOGIC >$ = An array containing the AL component objects.
$< AL\_ELEMENT >$ = One AL component object.
$< AL\_TYPE >$ = The type of the AL component.
$< AL\_ID >$ = The FESAS ID of the AL component.
$< AL\_NAME >$ = The name of the AL component.
$< AL\_LOGIC >$ = An array with multiple contracts for specifying logic elements that should be loaded (cf. Section B.5 for teh structure of a contract).
$< AL\_PROPERTIES >$ = Array with properties as key value pair (both are Strings).

The following excerpt shows an example for a JSON file specifying the AL (containing one AL component):

```
{
  "AL_ADAPTATIONLOGIC": [
    {
    "AL_ELEMENT": {
      "AL_TYPE": "MONITOR",
      "AL_ID": "fesasID-123_1_001",
      "AL_NAME": "1Var_monitor",
      "AL_LOGIC": [
        {
          "LOGIC_TYPE":"MONITOR",
          "PROGRAMMING_LANGUAGE":"Java",
          "CONTRACT_PROPERTIES": [
            {
              "VALUE":"MonitorLogicDummy_1Var",
              "UTILITY":0.75,
              "KEY":"SHORT_NAME"
            }
          ]
        }
      ],
      "AL_PROPERTIES" :[
        {
          "KEY":"port",
          "VALUE":"12345"
        }
      ]
      }
    },//Definitions of further AL elements omitted
  ]
}
```

Listing B.4: Example System Model for the Adaptation Logic

B.4.2. Communication System Model

The following information is part of a JSON configuration file for specifying the adaptation logic of a (sub)system:

$< COMM\_ADAPTATIONLOGIC >$ = An array containing the connections' specifications.
$< COMM\_ELEMENT >$ = One communication element for each connection.
$< COMM\_TYPE >$ = The type of the communication mechanism.
$< COMM\_RECEIVER >$ = The receiver of information (String with FESAS ID).
$< COMM\_SENDER >$ = The sender of information (String with FESAS ID).
$< COMM\_INFO\_TYPE >$ = The type of information sent.
$< COMM\_INFO\_CATEGORY >$ = The category of information sent.

The following excerpt shows an example for a JSON file specifying the communication for one AL component (containing one connection):

```
{
  "COMM_ADAPTATIONLOGIC": [
    {
      "COMM_ELEMENT": {
        "COMM_TYPE": "PUBSUB",
        "COMM_RECEIVER": "fesasID-123_1_002",
        "COMM_SENDER": "fesasID-123_1_001",
        "COMM_INFO_TYPE": "Monitoring_SIMPLESAS",
        "COMM_INFO_CATEGORY": "MONITOR"
      }
    },//Definitions of further AL elements omitted
  ]
}
```

Listing B.5: Example System Model for the Communication Structure

B.5. Functional Logic Contract

The adaptation logic uses the SAS Setup Service to generate contracts that describe the properties of a functional logic that should be loaded. These contracts are specified in JSON. The following information is part of such a JSON contract file.

$< LOGIC\_TYPE >$ = Logic Type.

$< INFORMATION\_TYPE >$ = String with the information type of generated output.

$< SUPPORTED\_INFORMATION\_TYPES >$ = Array with the supported information types (input).

$< PROGRAMMING\_LANGUAGE >$ = String with language.

$< CONTRACT\_PROPERTIES >$ = Array with properties as triples (key [String], value [String], utility [double between 0.0 and 1.0]).

Example for a JSON contract file:

```
{
  "LOGIC_TYPE": "Monitoring",
  "INFORMATION_TYPE": "Monitoring_DEFAULT",
  "SUPPORTED_INFORMATION_TYPES": "[Sensor_DEFAULT,Sensor_CLOUD]",
  "PROGRAMMING_LANGUAGE": "Java",
  "CONTRACT_PROPERTIES": [{"KEY": "a_k","VALUE": "a_v","UTILITY": 0.5},
    {"KEY": "b_k","VALUE": "b_v","UTILITY": 0.8}]
}
```

Listing B.6: Example of a Functional Logic Contract

B.6. Interface of the FESAS Repository

The remote repository `RemoteLogicRepository` implement the interface `IRemoteLogicRepository`, which provides methods for adding and removing logic elements, finding a logic element that fits a contract, delivering the path to `.class` files, and loading logic classes and their dependencies. The following listing shows the code of `IRemoteLogicRepository`. The interface is written in the Java syntax.

```
public interface IRemoteLogicRepository extends Remote {

  /**
   * This method returns a logic element as JSON string that suits
   * to the specified logic description (as JSON string).
   */
```

```
	public String findLogicElement(String contractString) throws RemoteException
	    ;

	/**
	 * Delivers the path to the .class file of a specified class name
	 * (full name including package).
	 */
	public String getPathToClass(String className, String type) throws
	    RemoteException;


	/**
	 * This methods expects a JSON string with the logic metadata and
	 * loads the specified logic or callLogic method, respectively.
	 */
	public byte[] loadLogicFromRepository(String jsonString) throws
	    RemoteException, LogicNotFoundException;


	/**
	 * This methods expects a string specifying a dependent class
	 * loads the specified logic or callLogic method, respectively.
	 */
	public byte[] loadDependencyFromRepository(String className) throws
	    RemoteException, LogicNotFoundException;

	/**
	 * Used for adding a logic to the repository at runtime.
	 */
	public boolean addLogicToRepository(LogicElementMetadata metadata, File file
	    )
	    throws RemoteException, LogicNotFoundException;

	/**
	 * Used for removing a logic from the repository at runtime.
	 */
	public boolean removeLogic() throws RemoteException;
}
```

Listing B.7: Interface of the FESAS Repository

B.7. Syntax of the Configuration Parameters

The wrapper projects contain three configuration files that are used for configuration of the adaptation logic. These files contains configuration parameters regarding the FESAS Framework, parameters for the adaptation logic as well as parameter to configure the connection to the FESAS Repository. In the following, the section describes these parameters.

B.7.1. FESAS Constants

FESAS constants are parameters, that are relevant for the FESAS middleware, e.g., for handling the configuration files, for the logic repository (such as connection information or location of the remote repository), or for the initialization process of the developed SASs. The class `FESASConstants` in the package `de.mannheim.wifo2.fesas.settings` saves these parameters. The following listing presents these parameters.

```java
public static final String PARSER_TYPE = "JSONFileParser";

// (Temporary) config files
/** Link to the config files directory. */
public static final String CONFIG_FILE_PATH = "C:" + File.separator + "
    ConfigFiles"
    + File.separator;
/** Link to the location of the logic binaries. */
public static final String LOGIC_ELEMENTS_PACKAGE_PATH = "de" + File.
    separator +
    "mannheim" + File.separator + "wifo2" + File.separator + "fesas" +
    File.separator + "logicRepository" + File.separator + "logicElements" +
        File.separator ;
/** Link to the location of the JDK for compiling logic binaries. */

public static final String JDK_JAVA_HOME = "C:\\Program Files"
    + File.separator + "Java" + File.separator
    + "jdk1.8.0_31" + File.separator + "jre";

/** Link to the location of binaries for the logic data types. */
public static final String DEPENDENCIES_PACKAGE_PATH = "de" + File.separator
     +
    "mannheim" + File.separator + "wifo2" + File.separator + "fesas" +
    File.separator + "logicRepository" + File.separator + "dependencies" +
        File.separator ;

// System Deployment
/** Timeout between creation of the elements and establishment of
    connections. */
public static final long INITIALIZATION_WAITING_TIME = 3000;

// Logic Repository
/** IP of the logic repository. */
public static final String LOGIC_REPOSITORY_ADRESS = "127.0.0.1"; //
    localHost
/** Port of the logic repository. */
public static final int LOGIC_REPOSITORY_PORT = 9999;
/** Logic repository registry RMI identifier. */
public static final String LOGIC_REPOSITORY_IDENTIFIER = "LogicRepository";
/** Port of the logic repository registry. */
public static final int LOGIC_REPOSITORY_REGISTRY_PORT = 1099;

// ALM
/** IP of the logic repository. */
public static final String ALM_ADRESS = "127.0.0.1"; //localHost
/** Port of the ALM for registration. */
public static final int ALM_REGISTRATION_PORT = 5555;
/** Port of the ALM proxy. */
public static final int ALM_PROXY_PORT = 7777;

// Logging
/** Enable / Disable logging. */
public static final boolean USE_LOGGING_IN_FILE = false;
/** Path to the log files. */
public static final String LOGGING_IN_FILE_PATH = "C:" + File.separator + "
    LogFiles"
    + File.separator;
```

Listing B.8: Syntax of the Configuration Files for the FESAS Constants

B.7.2. SAS Constants

SAS constants are parameters, that are relevant for an SAS, e.g., initializign the communication middleware or logging settings. The class `Constants` in the package `de.mannheim.wifo2.fesas.settings` saves these parameters. The following listing presents these parameters.

```java
// BASE

/** Timeout between creation of the BASE structure and establishment of
    connections. */
public static final int DEVICE_TIMEOUT = 5000;
/**
 * The amount of time that the service will wait after
 * it has been started (BASE).
 */
public static final long PERIOD_INIT = 3000;
/**
 * The amount of time that the service will wait between
 * the lookup calls (BASE).
 */
public static final long PERIOD_CYCLE = 1500;
/**
 * The amount of time that the pub/sub service will wait between
 * the lookup calls (BASE).
 */
public static final long COMMUNICATION_PERIOD_CYCLE = 500;

/** Specifies, if the ALM is run locally. In this case, a random port is
    used by the ALM proxy. */
public static final boolean LOCAL_ALM_DEBUG_MODE = true;


// Settings of the adaptation logic

/**
 * The amount of time between two requests of the sensor to the MRs.
 */
public static final long ENV_SENSOR_PERIOD_CYCLE = 1000;


// Debugging
```

```

  /** Debug the initialization of the adaptation logic. */
  public static final boolean DEBUG_FESASSETUP = false;
  /** Debug the saving and getting of knowledge. */
  public static final boolean DEBUG_KNOWLEDGE = false;
  /** Debug the communication and pub sub component. */
  public static final boolean DEBUG_COMMUNICATION = false;
  /** Debug the ID conversion. */
  public static final boolean DEBUG_IDCONVERSION = false;
  /** Debug the logic element. */
  public static final boolean DEBUG_LOGIC = true;
  /** Debug the remote logic repository. */
  public static final boolean DEBUG_REMOTE_LOGIC_REPOSITORY = false;
  /** Debug the GUI of the remote logic repository. */
  public static final boolean DEBUG_REMOTE_LOGIC_REPOSITORY_GUI = false;
  /** Debug the logic logic repository. */
  public static final boolean DEBUG_LOCAL_LOGIC_REPOSITORY = true;
  /** Debug the result of the input for a logic (in callLogic()). */
  public static final boolean DEBUG_LOGIC_RESULT = false;
  /** Debug the result of the JSON parser. */
  public static final boolean DEBUG_JSON_PARSER = true;
  /** Debug the ALM. */
  public static final boolean DEBUG_ALM = true;
```

Listing B.9: Syntax of the Configuration Files for the SAS Constants

B.7.3. Logic Repository Constants

Logic repository constants are parameters, that are relevant for the configuration of the FESAS repository. The class `LogicRepositoryConstants` in the package `de.mannheim.wifo2.fesas.settings` saves these parameters. The following listing presents these parameters.

```
/** Path to the temporary destination for zip files. */
  public static final String ZIP_TEMP_PATH = "res" + File.separator +
      "temp_zip" + File.separator;
  /** Path to the temporary destination for JSON files. */
  public static final String JSON_TEMP_PATH = "res" + File.separator +
      "temp_json" + File.separator;
  /** Path to the temporary destination for .java files. */
  public static final String JAVA_TEMP_PATH = "res" + File.separator +
        "temp_java" + File.separator;

    /** Link to the folder containing the logic elements that should be loaded
        at start of the repository. */
    public static final String REPOSITORY_START_ELEMENTS_FOLDER= "C:\\
        logicElements\\";

    /** Logic elements that should be loaded at start of the repository. */
    public static final String[] REPOSITORY_START_LOGIC_ELEMENTS = new String[]{
        REPOSITORY_START_ELEMENTS_FOLDER + "AnalyzerLogicDummy_1VarLargerX.zip",
        REPOSITORY_START_ELEMENTS_FOLDER + "ExecutorLogicDummy_Var.zip",
        REPOSITORY_START_ELEMENTS_FOLDER + "KnowledgeLogicWithHashMap.zip",
        REPOSITORY_START_ELEMENTS_FOLDER + "MonitorLogicDummy_1Var.zip",
        REPOSITORY_START_ELEMENTS_FOLDER + "ExecutorLogicDummy_Var1.zip",
        REPOSITORY_START_ELEMENTS_FOLDER + "PlannerLogicDummy_VarDecentralized.
            zip",
        REPOSITORY_START_ELEMENTS_FOLDER + "ALMAnalyzerLogicDummy.zip",
        REPOSITORY_START_ELEMENTS_FOLDER + "ALMExecutorLogicDummy.zip",
        REPOSITORY_START_ELEMENTS_FOLDER + "ALMMonitorLogicDummy.zip",
        REPOSITORY_START_ELEMENTS_FOLDER + "ALMPlannerLogicDummy.zip"
    };
```

Listing B.10: Syntax of the Configuration Files for the FESAS Repository

B.8. Interface of the SAS Setup Service

The following listing show the interface for the SAS Setup Service. The interface is written in the Java syntax.

```
public interface ISASSetupService {

  /**
   * Becomes active, if the middleware is running and all data is transmitted.
   */
  public void configureSetupService(FesasDeviceID fesasID, IDeviceType type);

  /**
   * Get the FESAS ID assigned from the config file
   */
  public String getFESASID();

public void setFESASID(String id);

public IDeviceType getDeviceType();

public void setDeviceType(int type) throws InitializationException;

public String getDeviceName();

public void setDeviceName(String deviceName);

public void setDeviceProperty(KeyValueProperty p);

public void setElementProperty(String fesasID, KeyValueProperty p);

public void changeAlgorithmOfLogic(String ALObject, String logic, String
    algorithm);

/**
 * Start the SAS's AL after the initialization.
 */
public void startAdaptationLogic();

/**
 * This method start a new AL element.
 */
public String implementAdaptationLogicElement(AdaptationLogicElement
    adaptationLogicElement, boolean startDirectly);

public boolean deactivateAdaptationLogicElement(AdaptationLogicElement
    adaptationLogicElement);

public boolean activateAdaptationLogicElement(AdaptationLogicElement
    adaptationLogicElement);

/**
 * Inializes a communication channel.
 */
public void implementCommunicationStructure(Connection connection);

public void destroyCommunicationStructure(Connection connection);
```

```

  /*
   * Change the implemented logics
   */
  public void addFunctionalLogicToComponent(String fesasID, Contract contract)
      ;

  public void removeFunctionalLogicFromComponent(String fesasID, Contract
      contract);

  /*
   * Change the rule base
   */
  public void addRule(String ALObject, String rule);

  public void removeRule(String ALObject, String rule);
}
```

Listing B.11: Interface of the SAS Setup Service

C. ALM Documentation

This chapter provides a documentation of various aspects of the ALM. First, Section C.1 provides an overview on the ALM protocol. Second, Section C.3 presents the implementation of the TARL module and Section C.4 outlines a TARL transition rule Last, Section C.5 presents the implementation of the Neo4j module.

C.1. ALM Protocol

The data exchanged between the ALM and the ALM proxy has a specified protocol, the *ALM protocol*. It can be expressed in JSON. In the reference implementation, it is represented at runtime as Java objects. The header is represented as `PROTOCOL - TYPE -COMMAND`; the body as `INFORMATION` with command-specific information. So far, the `PROTOCOL`[1] has version 0.9. The `TYPE` can be:

- `REQ` for request,
- `RESP` for a reply of a request, and
- `ERR` for an error.

The `COMMAND` can be:

- `REG` for registration at the ALM,
- `SYS` for system state information of a SAS,
- `COM` for changes of the communication structure,
- `LOG` for changes of an logic element, and
- `RUL` for changes of rules.

[1]The use of the `PROTOCOL` parameter is omitted so far.

INFORMATION is represented as key-value pairs. For transfer between components, the information can be represented as JSON string having the following properties:

$< ALM\_MSG\_TYPE >$ = Type of the message (can be REQ for request, RESP for response,or ERR for an error).
$< ALM\_COMMAND >$ = Command describing the action (can be REG, SYS, COM, LOG, and RUL).
$< PROPERTIES >$ = Array with properties as tuples (key [String], value [String]). These properties are the actual information.

The following excerpt shows an example for a metadata description in JSON. This string is a request for registration, send from the ALM proxy to the ALM. As information, the IP and the port of the ALM proxy is added.

```json
{
  "ALM_MSG_TYPE": "REQ",
  "ALM_COMMAND": "REG",
  "PROPERTIES": [
    {
      "key": "IP",
      "value": "123.123.123.123"
    },{
      "key": "Port",
      "value": "12345"
    }
  ]
}
```

C.2. Data Adapter

Listing C.1 shows the implementation of the IDataAdapterMongoDB interface within the class DataAdapterSHData.

```java
public class DataAdapterSHData implements IDataAdapterMongoDB {
  public List<Document> convertData(String dataElement) {
    [...] // Attribute declaration
    Pattern pattern = Pattern.compile("(fesasID-\\d+_\\d+_\\d+)\\s:\\s(\\d+)\\
        s:((?:\\s\\d+\\s#\\s\\((\\d+,\\s)+\\d+\\)\\s#\\s\\[([\\d\\.-]+\\,\\s)
        +[\\d\\.-]+\\]\\s-)*)");
    Matcher matcher = pattern.matcher(dataElement);

    while (matcher.find()) {
      fesasID = matcher.group(1);
      timestamp = Integer.parseInt(matcher.group(2));
      detectors = matcher.group(3);
    }

    pattern = Pattern.compile("(\\d+)\\s#\\s\\((\\d+),\\s(\\d+),\\s(\\d+),\\s
        (\\d+)\\)\\s#\\s\\[([\\d\\.-]+)\\,\\s([\\d\\.-]+)\\,\\s([\\d\\.-]+)
        \\,\\s([\\d\\.-]+)\\]\\s-");
    matcher = pattern.matcher(detectors);

    while (matcher.find()) {
      doc = new Document("fesasID", fesasID).append("timestamp", timestamp);
      numOfVehicles = Integer.parseInt(matcher.group(6));
      meanSpeed = Double.parseDouble(matcher.group(7));
      jamLengthVehicles = Integer.parseInt(matcher.group(8));
      jamLengthMeters = Double.parseDouble(matcher.group(9));
      doc.append("detector", matcher.group(1));
      doc.append("numOfVehicles", numOfVehicles);
      doc.append("meanSpeed", meanSpeed);
      doc.append("jamLengthVehicles", jamLengthVehicles);
      doc.append("jamLengthMeters", jamLengthMeters);
      trafficAnalzyer = new TrafficAnalyzer(meanSpeed, numOfVehicles);
      doc.append("density", trafficAnalzyer.getDensity());
      doc.append("flow", trafficAnalzyer.getFlow());
      doc.append("velocity", trafficAnalzyer.getVelocity());
      doc.append("trafficStatus", trafficAnalzyer.getStatus().toString());
      docs.add(doc);
    }
    return docs;
  }
}
```

Listing C.1: `DataAdapterSHData` implementing the `IDataAdapterMongoDB` interface.

The conversion of the data format shown in Listing C.2 is carried out utilizing a two stage regular expression. Line 11 of Listing contains the first pattern which matches the `fesasID`, `timestamp`, and the actual detector values using capture groups. The second pattern, also employs subgroups that match an arbitrary amount of `Strings` like `2\#(16,17,24,25)\#[10,14.96,0,0.0]-` corresponding to atomic sets of sensors ids and sensor values per detector (in the example 2 is the id of the detector), respectively (cf. line 21 of Listing C.1). Finding multiple concurrences of this pattern is mandatory as each system sends the data of all managed detectors in one line for each `timestamp`. This way, lines 25 - 48 of Listing C.1 subsequently take care of the creation of an `Object` of the type `Document` for every parsed detector and map the sensor ids to properly named and typed attributes. Furthermore, the parsed raw data of the sensor is used to calculate some further features like `density`, `flow`, `velocity`, and the `trafficStatus` of this highway section. Finally, the `Document` is added to a `List` that is returned after all items have been parsed and enriched. Hence, the caller of the `convertData` method can store the data in the MongoDB, which is in this case the ALM Monitor.

```
fesasID-123_2_001 :64148:2# (16, 17,24,25) # [10,14.96, 0,0.0] - 3# (16, 17,24,25) # [18,11.39, 0,0.0] - 4# (16, 17,24,25) # [31,4.5, 3,20.53] - 5# (16, 17,24,25) # [26,7.47, 6,45.65] - 6# (16, 17,24,25) # [2,19.67, 0,0.0] -
fesasID-123_2_002 :64148:7# (16, 17,24,25) # [6,17.90, 0,0.0] - 8# (16, 17,24,25) # [6,17.34, 0,0.0] - 9# (16, 17,24,25) # [17,15.09, 0,0.0] - 10# (16, 17,24,25) # [0,-1.0, 0,0.0] -
fesasID-123_2_003 :64148:11# (16, 17,24,25) # [6,25.99, 0,0.0] - 12# (16, 17,24,25) # [8,26.25, 0,0.0] - 13# (16, 17,24,25) # [6,26.71, 0,0.0] -
```

Listing C.2: Example of sensor values captured from the cameras for the SmartHighway scenario.

C.3. TARL Module Implementation

Listing C.3 illustrates the `call()` method of the `RegionalPlannerAdaptationTarl` class from [223]. In order to match the requirements of TARL, a `TopologyProvider` is created for the manipulation of the graph in line 3. The class `FesasTopologyProvider` contains the information about how to add and remove nodes to and from the `TARLGraph`, respectively. Lines 14-38 of Listing C.3 show how the result of the `AccidentAdaptationTarl` class, which contains the matching of the rule, is processed in terms of assigning the regional planner, new and obsolete connections, as well as the generation of these connections.

```
1  public IRuntimeOptimization call() {
2    final Host host = new DummyHost(42);
3    final FesasTopologyProvider topProv = new FesasTopologyProvider(host);
4    final AccidentTransitionTarl tarl = new AccidentTransitionTarl(host, topProv
         , topProv);
5
6    Node fromNode = null;
7    Node toNode = null;
8    Node regionalPlanner = null;
9    Multimap<String, String> nodes;
10
11   tarl.checkAccident();
12   nodes = topProv.getNewConnections();
13
14   Iterator<Entry<String, String>> iterator = nodes.entries().iterator();
15
16   while (iterator.hasNext()) {
17     Map.Entry<String, String> entry = (Map.Entry<String, String>) iterator.
           next();
18   fromNode = FesasGraph.getInstance().getNode(entry.getKey());
19   toNode = FesasGraph.getInstance().getNode(entry.getValue());
20
21     if (fromNode.getAttribute("type").equals("ANALYZER")) {
22       addConnectedAnalyzer(fromNode);
23       if (regionalPlanner == null) {
24         regionalPlanner = toNode;
25         setRegionalPlanner(toNode);
26         }
27       }else {
```

```java
        addConnectedExecutor(toNode);
        if (regionalPlanner == null) {
          regionalPlanner = fromNode;
          setRegionalPlanner(fromNode);
        }
      }
    }
  }

  generateNewConnections();
  generateObsoleteConnections();

  return this;
}
```

Listing C.3: TARL-based implementation of a structural adaptation extending the class `AbstractRegionalPattern`.

C.4. TARL Transition Rule

Listing C.4 shows the rule for the TARL module that triggers a switch from the fully decentralized to the Regional Planning pattern.

```java
public AccidentTransitionTarl(final Host host, final AdaptableTopologyProvider provider,
    final ObservableTopologyProvider obProvider) {
  super(new LocalTransitionEngine(host));

  this.host = host;
  this.provider = provider;

  logger.info("Starting accident transition");

  localTransitionEngine.setConnectsWithGlobalTransitionEngine(false);

  final TransitionModel transitionModel = new TransitionModel(host, MODEL_FILE);
  localTransitionEngine.setTransitionModel(transitionModel);
  localTransitionEngine.getEventConditionEvaluator().getExecutionManager().setTopologyProvider(provider);
```

```java

// Construct a new TARL Rule
TopologyGraphPatternMatchTerm tgpmt1 = new TopologyGraphPatternMatchTerm(provider, TOP_PROVIDER_NAME,
    new ConstantStringTerm("FesasTopology"),
    new DirectedEdgeTerm("eefst0", new NodeTerm("ef0"), new NodeTerm("st0")), 1);

StringGraphAttribute sgaACC = new StringGraphAttribute("st0", "jam");
StringGraphAttribute sgaEF0 = new StringGraphAttribute("ef0", "type");
StringGraphAttribute sgaST0 = new StringGraphAttribute("st0", "type");

StringEqualTerm setACC = new StringEqualTerm(sgaACC, new ConstantStringTerm("true"));
StringEqualTerm setEF0 = new StringEqualTerm(sgaEF0, new ConstantStringTerm("EFFECTOR"));
StringEqualTerm setST0 = new StringEqualTerm(sgaST0, new ConstantStringTerm("DOMAIN"));

BooleanAndTerm batFilter1 = new BooleanAndTerm(setST0, setACC);
BooleanAndTerm batFilter0 = new BooleanAndTerm(setEF0, batFilter1);

ApplyBooleanTerm abt0 = new ApplyBooleanTerm(tgpmt1, batFilter0);

TopologyGraphPatternMatchTerm tgpmt2 = new TopologyGraphPatternMatchTerm(provider, TOP_PROVIDER_NAME,
    new ConstantStringTerm("FesasTopology"),
    new TopologyAndTerm(new DirectedEdgeTerm("ecef0", new NodeTerm("c0"), new NodeTerm("ef0")),
        new DirectedEdgeTerm("eexc0", new NodeTerm("ex0"), new NodeTerm("c0"))),
    1);

StringGraphAttribute sgaC0 = new StringGraphAttribute("c0", "type");
StringGraphAttribute sgaEX0 = new StringGraphAttribute("ex0", "type");

StringEqualTerm setC0 = new StringEqualTerm(sgaC0, new ConstantStringTerm("CONTEXTMANAGER"));
StringEqualTerm setEX0 = new StringEqualTerm(sgaEX0, new ConstantStringTerm("EXECUTOR"));

```

```
BooleanAndTerm batFilter3 = new BooleanAndTerm(setC0, setEF0);
BooleanAndTerm batFilter2 = new BooleanAndTerm(setEX0, batFilter3);

ApplyBooleanTerm abt1 = new ApplyBooleanTerm(tgpmt2, batFilter2);

TopologyGraphPatternMatchTerm tgpmt3 = new TopologyGraphPatternMatchTerm(provider, TOP_PROVIDER_NAME,
    new ConstantStringTerm("FesasTopology"),
    new DirectedEdgeTerm("eplex0", new NodeTerm("pl0"), new NodeTerm("ex0")), 1);

StringGraphAttribute sgaPL0 = new StringGraphAttribute("pl0", "type");

StringEqualTerm setPL0 = new StringEqualTerm(sgaPL0, new ConstantStringTerm("PLANNER"));

BooleanAndTerm batFilter4 = new BooleanAndTerm(setPL0, setEX0);

ApplyBooleanTerm abt2 = new ApplyBooleanTerm(tgpmt3, batFilter4);

JoinMatchTerms jmt1 = new JoinMatchTerms(abt0, abt1);
JoinMatchTerms jmt2 = new JoinMatchTerms(abt2, jmt1);

TopologyGraphPatternMatchTerm tgpmt4 = new TopologyGraphPatternMatchTerm(provider, TOP_PROVIDER_NAME,
    new ConstantStringTerm("FesasTopology"),
    new TopologyAndTerm(
        new TopologyAndTerm(new DirectedEdgeTerm("estst0", new NodeTerm("st1"), new NodeTerm("st0")),
            new DirectedEdgeTerm("estst1", new NodeTerm("st2"), new NodeTerm("st1"))),
        new DirectedEdgeTerm("estst2", new NodeTerm("st3"), new NodeTerm("st2"))),
    1);

StringGraphAttribute sgaST3 = new StringGraphAttribute("st3", "type");
StringEqualTerm setST3 = new StringEqualTerm(sgaST3, new ConstantStringTerm("DOMAIN"));

BooleanAndTerm batFilter5 = new BooleanAndTerm(setST3, setST0);
BooleanAndTerm batFilter51 = new BooleanAndTerm(setACC, batFilter5);
```

```

ApplyBooleanTerm abt3 = new ApplyBooleanTerm(tgpmt4, batFilter51);

TopologyGraphPatternMatchTerm tgpmt6 = new TopologyGraphPatternMatchTerm(
    provider, TOP_PROVIDER_NAME,
    new ConstantStringTerm("FesasTopology"),
    new TopologyAndTerm(
        new TopologyAndTerm(new DirectedEdgeTerm("eefst1", new NodeTerm("ef1"
            ), new NodeTerm("st3")),
            new DirectedEdgeTerm("ecef1", new NodeTerm("c1"), new NodeTerm("
                ef1"))),
        new DirectedEdgeTerm("eexc1", new NodeTerm("ex1"), new NodeTerm("c1")
            )),
    1);

StringGraphAttribute sgaC1 = new StringGraphAttribute("c1", "type");
StringGraphAttribute sgaEX1 = new StringGraphAttribute("ex1", "type");
StringGraphAttribute sgaEF1 = new StringGraphAttribute("ef1", "type");

StringEqualTerm setEF1 = new StringEqualTerm(sgaEF1, new ConstantStringTerm(
    "EFFECTOR"));
StringEqualTerm setC1 = new StringEqualTerm(sgaC1, new ConstantStringTerm("
    CONTEXTMANAGER"));
StringEqualTerm setEX1 = new StringEqualTerm(sgaEX1, new ConstantStringTerm(
    "EXECUTOR"));

BooleanAndTerm batFilter6 = new BooleanAndTerm(setEF1, setST3);
BooleanAndTerm batFilter7 = new BooleanAndTerm(setC1, batFilter6);
BooleanAndTerm batFilter8 = new BooleanAndTerm(setEX1, batFilter7);

ApplyBooleanTerm abt5 = new ApplyBooleanTerm(tgpmt6, batFilter8);

TopologyGraphPatternMatchTerm tgpmt7 = new TopologyGraphPatternMatchTerm(
    provider, TOP_PROVIDER_NAME,
    new ConstantStringTerm("FesasTopology"),
    new TopologyAndTerm(new DirectedEdgeTerm("eplex1", new NodeTerm("pl1"),
        new NodeTerm("ex1")),
        new DirectedEdgeTerm("eanpl1", new NodeTerm("an1"), new NodeTerm("pl1
            "))),
    1);

```

```
StringGraphAttribute sgaPL1 = new StringGraphAttribute("pl1", "type");
StringGraphAttribute sgaAN1 = new StringGraphAttribute("an1", "type");

StringEqualTerm setPL1 = new StringEqualTerm(sgaPL1, new ConstantStringTerm(
    "PLANNER"));
StringEqualTerm setAN1 = new StringEqualTerm(sgaAN1, new ConstantStringTerm(
    "ANALYZER"));

BooleanAndTerm batFilter9 = new BooleanAndTerm(setPL1, setEX1);
BooleanAndTerm batFilter10 = new BooleanAndTerm(setAN1, batFilter9);

ApplyBooleanTerm abt6 = new ApplyBooleanTerm(tgpmt7, batFilter10);

JoinMatchTerms jmt3 = new JoinMatchTerms(abt3, jmt2);
JoinMatchTerms jmt4 = new JoinMatchTerms(abt5, jmt3);
JoinMatchTerms jmt5 = new JoinMatchTerms(abt6, jmt4);

TopologyGraphPatternMatchTerm tgpmt8 = new TopologyGraphPatternMatchTerm(
    provider, TOP_PROVIDER_NAME,
    new ConstantStringTerm("FesasTopology"),
    new TopologyAndTerm(
        new TopologyAndTerm(new DirectedEdgeTerm("eststst3", new NodeTerm("st4"
            ), new NodeTerm("st3")),
            new DirectedEdgeTerm("estst4", new NodeTerm("st5"), new NodeTerm(
                "st4"))),
        new DirectedEdgeTerm("estst5", new NodeTerm("st6"), new NodeTerm("st5
            "))),
    1);

StringGraphAttribute sgaST6 = new StringGraphAttribute("st6", "type");
StringEqualTerm setST6 = new StringEqualTerm(sgaST6, new ConstantStringTerm(
    "DOMAIN"));

BooleanAndTerm batFilter11 = new BooleanAndTerm(setST6, setST3);

ApplyBooleanTerm abt7 = new ApplyBooleanTerm(tgpmt8, batFilter11);

TopologyGraphPatternMatchTerm tgpmt9 = new TopologyGraphPatternMatchTerm(
    provider, TOP_PROVIDER_NAME,
    new ConstantStringTerm("FesasTopology"),
    new TopologyAndTerm(
        new TopologyAndTerm(new DirectedEdgeTerm("eefst2", new NodeTerm("ef2"
            ), new NodeTerm("st6")),
            new DirectedEdgeTerm("ecef2", new NodeTerm("c2"), new NodeTerm("
                ef2"))),
        new DirectedEdgeTerm("eexc2", new NodeTerm("ex2"), new NodeTerm("c2")
            )),
    1);

StringGraphAttribute sgaC2 = new StringGraphAttribute("c2", "type");
StringGraphAttribute sgaEX2 = new StringGraphAttribute("ex2", "type");
StringGraphAttribute sgaEF2 = new StringGraphAttribute("ef2", "type");

StringEqualTerm setEF2 = new StringEqualTerm(sgaEF2, new ConstantStringTerm(
    "EFFECTOR"));
StringEqualTerm setC2 = new StringEqualTerm(sgaC2, new ConstantStringTerm("
    CONTEXTMANAGER"));
StringEqualTerm setEX2 = new StringEqualTerm(sgaEX2, new ConstantStringTerm(
    "EXECUTOR"));

BooleanAndTerm batFilter13 = new BooleanAndTerm(setEF2, setST6);
BooleanAndTerm batFilter14 = new BooleanAndTerm(setC2, batFilter13);
BooleanAndTerm batFilter15 = new BooleanAndTerm(setEX2, batFilter14);

ApplyBooleanTerm abt8 = new ApplyBooleanTerm(tgpmt9, batFilter15);

TopologyGraphPatternMatchTerm tgpmt10 = new TopologyGraphPatternMatchTerm(
    provider, TOP_PROVIDER_NAME,
    new ConstantStringTerm("FesasTopology"),
    new TopologyAndTerm(new DirectedEdgeTerm("eplex2", new NodeTerm("pl2"),
        new NodeTerm("ex2")),
        new DirectedEdgeTerm("eanpl2", new NodeTerm("an2"), new NodeTerm("pl2
            "))),
    1);

StringGraphAttribute sgaPL2 = new StringGraphAttribute("pl2", "type");
StringGraphAttribute sgaAN2 = new StringGraphAttribute("an2", "type");

StringEqualTerm setPL2 = new StringEqualTerm(sgaPL2, new ConstantStringTerm(
    "PLANNER"));
StringEqualTerm setAN2 = new StringEqualTerm(sgaAN2, new ConstantStringTerm(
    "ANALYZER"));
```

```

BooleanAndTerm batFilter17 = new BooleanAndTerm(setPL2, setEX2);
BooleanAndTerm batFilter18 = new BooleanAndTerm(setAN2, batFilter17);

ApplyBooleanTerm abt9 = new ApplyBooleanTerm(tgpmt10, batFilter18);

JoinMatchTerms jmt6 = new JoinMatchTerms(abt7, jmt5);
JoinMatchTerms jmt7 = new JoinMatchTerms(abt8, jmt6);
JoinMatchTerms jmt8 = new JoinMatchTerms(abt9, jmt7);

Condition condition = new Condition(jmt8);

EdgeRemoveTopologyTransitionInstance erttiPl1Ex1 = new
    EdgeRemoveTopologyTransitionInstance(
    new TopologyTransitionDescription("Remove Edge"), "ex1", new
        TopologyTransitionAddressing("pl1",
        new ConstantStringTerm(TOP_PROVIDER_NAME), new ConstantStringTerm("
            FesasTopology")));

EdgeRemoveTopologyTransitionInstance erttiAn1Pl1 = new
    EdgeRemoveTopologyTransitionInstance(
    new TopologyTransitionDescription("Remove Edge"), "pl1", new
        TopologyTransitionAddressing("an1",
        new ConstantStringTerm(TOP_PROVIDER_NAME), new ConstantStringTerm("
            FesasTopology")));

EdgeRemoveTopologyTransitionInstance erttiPl2Ex2 = new
    EdgeRemoveTopologyTransitionInstance(
    new TopologyTransitionDescription("Remove Edge"), "ex2", new
        TopologyTransitionAddressing("pl2",
        new ConstantStringTerm(TOP_PROVIDER_NAME), new ConstantStringTerm("
            FesasTopology")));

EdgeRemoveTopologyTransitionInstance erttiAn2Pl2 = new
    EdgeRemoveTopologyTransitionInstance(
    new TopologyTransitionDescription("Remove Edge"), "pl2", new
        TopologyTransitionAddressing("an2",
        new ConstantStringTerm(TOP_PROVIDER_NAME), new ConstantStringTerm("
            FesasTopology")));

EdgeAddTopologyTransitionInstance eattiAn1Pl0 = new
```

```
    EdgeAddTopologyTransitionInstance(
    new TopologyTransitionDescription("Add Edge"), "pl0", new
        TopologyTransitionAddressing("an1",
        new ConstantStringTerm(TOP_PROVIDER_NAME), new ConstantStringTerm("
            FesasTopology")));

EdgeAddTopologyTransitionInstance eattiPl0Ex1 = new
    EdgeAddTopologyTransitionInstance(
    new TopologyTransitionDescription("Add Edge"), "ex1", new
        TopologyTransitionAddressing("pl0",
        new ConstantStringTerm(TOP_PROVIDER_NAME), new ConstantStringTerm("
            FesasTopology")));

EdgeAddTopologyTransitionInstance eattiAn2Pl0 = new
    EdgeAddTopologyTransitionInstance(
    new TopologyTransitionDescription("Add Edge"), "pl0", new
        TopologyTransitionAddressing("an2",
        new ConstantStringTerm(TOP_PROVIDER_NAME), new ConstantStringTerm("
            FesasTopology")));

EdgeAddTopologyTransitionInstance eattiPl0Ex2 = new
    EdgeAddTopologyTransitionInstance(
    new TopologyTransitionDescription("Add Edge"), "ex2", new
        TopologyTransitionAddressing("pl0",
        new ConstantStringTerm(TOP_PROVIDER_NAME), new ConstantStringTerm("
            FesasTopology")));

SequentialTransitionInstance sti7 = new SequentialTransitionInstance(new
    TopologyTransitionDescription("Seq 7"),
    eattiAn2Pl0, eattiPl0Ex2);

SequentialTransitionInstance sti6 = new SequentialTransitionInstance(new
    TopologyTransitionDescription("Seq 6"),
    eattiPl0Ex1, sti7);

SequentialTransitionInstance sti5 = new SequentialTransitionInstance(new
    TopologyTransitionDescription("Seq 5"),
    eattiAn1Pl0, sti6);

SequentialTransitionInstance sti4 = new SequentialTransitionInstance(new
    TopologyTransitionDescription("Seq 4"),
        erttiAn2Pl2, sti5);

    SequentialTransitionInstance sti3 = new SequentialTransitionInstance(new
        TopologyTransitionDescription("Seq 3"),
        erttiPl2Ex2, sti4);

    SequentialTransitionInstance sti2 = new SequentialTransitionInstance(new
        TopologyTransitionDescription("Seq 2"),
        erttiAn1Pl1, sti3);

    SequentialTransitionInstance sti1 = new SequentialTransitionInstance(new
        TopologyTransitionDescription("Seq 1"),
        erttiPl1Ex1, sti2);

    ECARule rule = new ECARule(0, condition, sti1, new EveryMatchRepetitionMode
        (), new EveryMatchExecutionMode(),
        Location.local);

    // Register Rule
    localTransitionEngine.getEventConditionEvaluator().addECARule(rule);

    host.registerComponent(localTransitionEngine);
    host.registerComponent(transitionModel);

    obProvider.addTopologyObserver(this);
}
```

Listing C.4: TARL Transition Rule

C.5. Neo4j Module Implementation

Listing C.5 shows the query for switching the current pattern to a *Regional Planning* pattern in the mentioned SmartHighway scenario from [223]. At first, a highway section which includes the attribute `jam="true"` has to be found. This is realized by matching a node `s` labeled as `:DOMAIN` for which `where s.jam=true` applies. In case a match exists, the corresponding node is stored in the variable `s`. Afterwards, the regional planner has to be determined. The most posterior section with a jam on the highway becomes the regional planner. This section is searched with the query in line 12. Once the query is defined, *Neo4j* tries to find a match on the current graph. In case a situation is present in which a jam section exists and the other parts of the query also apply, the result has to be processed. Lines 17-37 of Listing C.5 show the processing of the result which consists of the regional planner and the analyzers and executors that have to be connected to the planner. Our system uses this module for structural adaptation of the adaptation logic. However, the modules can integrate other types of adaptation or combinations thereof.

```
1  public IRuntimeOptimization call() {
2    GraphDatabaseService graphDb = Neo4jService.getInstance().getGraphDb();
3    Node node = null;
4    Relationship relationship;
5    List<Node> toPlanner = new ArrayList<Node>();
6    List<Node> fromPlanner = new ArrayList<Node>();
7    Node centralPlanner = null;
8    String fesasID = "";
9    Result result = null;
10
11   try (Transaction tx = graphDb.beginTx()) {
12     result = graphDb.execute("match (x)-[*1..3]->(c:CONTEXTMANAGER)-[*2]->(t:DOMAIN)-[*1..6]->(s:DOMAIN)<-[*4]-(p:PLANNER) where s.jam=\"true\" and ((x:ANALYZER)OR(x:EXECUTOR)) return distinct p,x");
13
14     while (result.hasNext()) {
15       Map<String, Object> row = result.next();
16       for (Entry<String, Object> column :row.entrySet()) {
17         node = (Node) column.getValue();
18         logger.info(column.getKey().toString() + ": " + node.toString() + node.getProperty("fesasID") + node.getLabels().toString());
19
20         if (column.getKey().equals("p")) {
21           centralPlanner = node;
22           fesasID = (String) centralPlanner.getProperty("fesasID");
23           setRegionalPlanner(FesasGraph.getInstance().getGraph().getNode(fesasID));
24         }else if (node.getLabels().toString().equals("[ANALYZER]")) {
25           fesasID = (String) node.getProperty("fesasID");
26           addConnectedAnalyzer(FesasGraph.getInstance().getGraph().getNode(fesasID));
27           toPlanner.add(node);
28         }else if (node.getLabels().toString().equals("[EXECUTOR]")) {
```

```
          fesasID = (String) node.getProperty("fesasID");
          addConnectedExecutor(FesasGraph.getInstance().getGraph().getNode(
              fesasID));
          fromPlanner.add(node);
          }
        }
      }

    generateNewConnections();
    generateObsoleteConnections();
  }
}
```

Listing C.5: Neo4j-based implementation of a structural adaptation extending the class `AbstractRegionalPattern`.

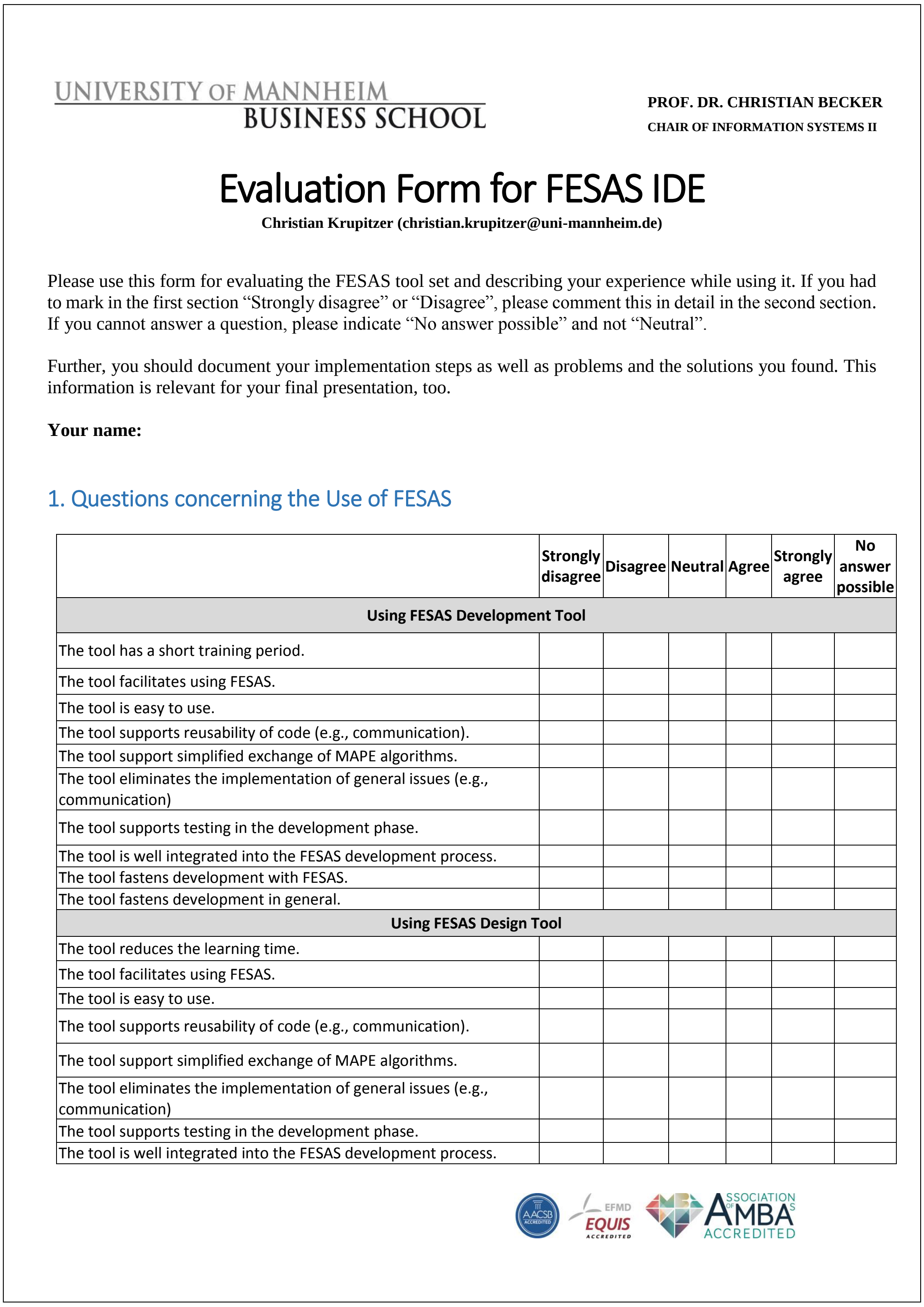
D. Evaluation Questionnaires

For capturing the answer of the student for the evaluations in Section 8.2, we used the following questionnaires.

D.1. Questionnaire FESAS IDE

This appendix presents the questionnaire used for the analysis of the suitability and usability of the FESAS IDE in Section 8.2.

We derived the following question items from the *ISO 9241-11 Guidance on Usability* standard and the definition of usability in the *ISO/IEC 9126-1 Software Product Quality Model* standard



UNIVERSITY OF MANNHEIM
BUSINESS SCHOOL

PROF. DR. CHRISTIAN BECKER
CHAIR OF INFORMATION SYSTEMS II

Evaluation Form for FESAS IDE

Christian Krupitzer (christian.krupitzer@uni-mannheim.de)

Please use this form for evaluating the FESAS tool set and describing your experience while using it. If you had to mark in the first section "Strongly disagree" or "Disagree", please comment this in detail in the second section. If you cannot answer a question, please indicate "No answer possible" and not "Neutral".

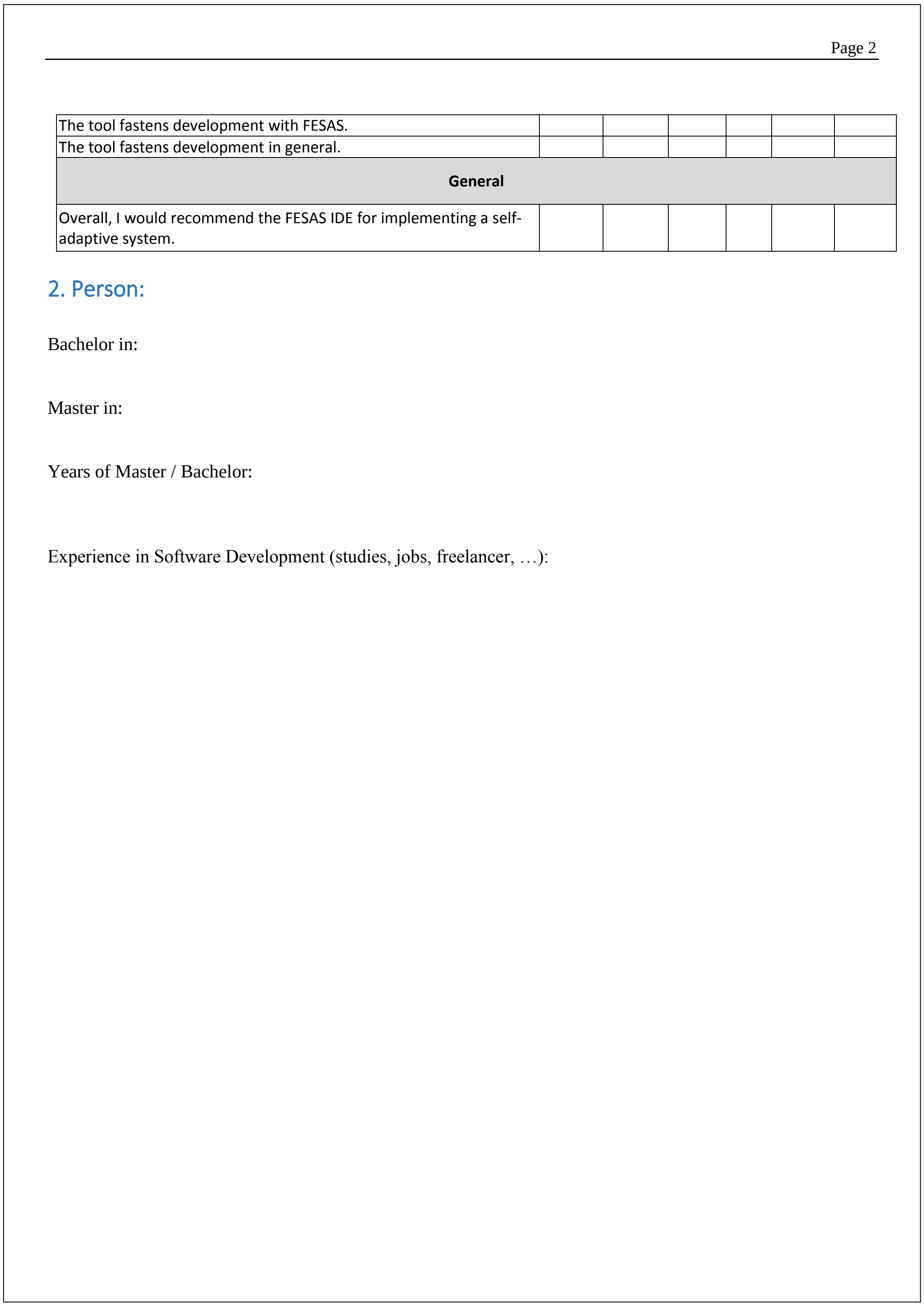
Further, you should document your implementation steps as well as problems and the solutions you found. This information is relevant for your final presentation, too.

Your name:

1. Questions concerning the Use of FESAS

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	No answer possible
Using FESAS Development Tool						
The tool has a short training period.						
The tool facilitates using FESAS.						
The tool is easy to use.						
The tool supports reusability of code (e.g., communication).						
The tool support simplified exchange of MAPE algorithms.						
The tool eliminates the implementation of general issues (e.g., communication)						
The tool supports testing in the development phase.						
The tool is well integrated into the FESAS development process.						
The tool fastens development with FESAS.						
The tool fastens development in general.						
Using FESAS Design Tool						
The tool reduces the learning time.						
The tool facilitates using FESAS.						
The tool is easy to use.						
The tool supports reusability of code (e.g., communication).						
The tool support simplified exchange of MAPE algorithms.						
The tool eliminates the implementation of general issues (e.g., communication)						
The tool supports testing in the development phase.						
The tool is well integrated into the FESAS development process.						

Figure D.1.: Questionnaire for the FESAS IDE (1)



Page 2

The tool fastens development with FESAS.						
The tool fastens development in general.						
General						
Overall, I would recommend the FESAS IDE for implementing a self-adaptive system.						

2. Person:

Bachelor in:

Master in:

Years of Master / Bachelor:

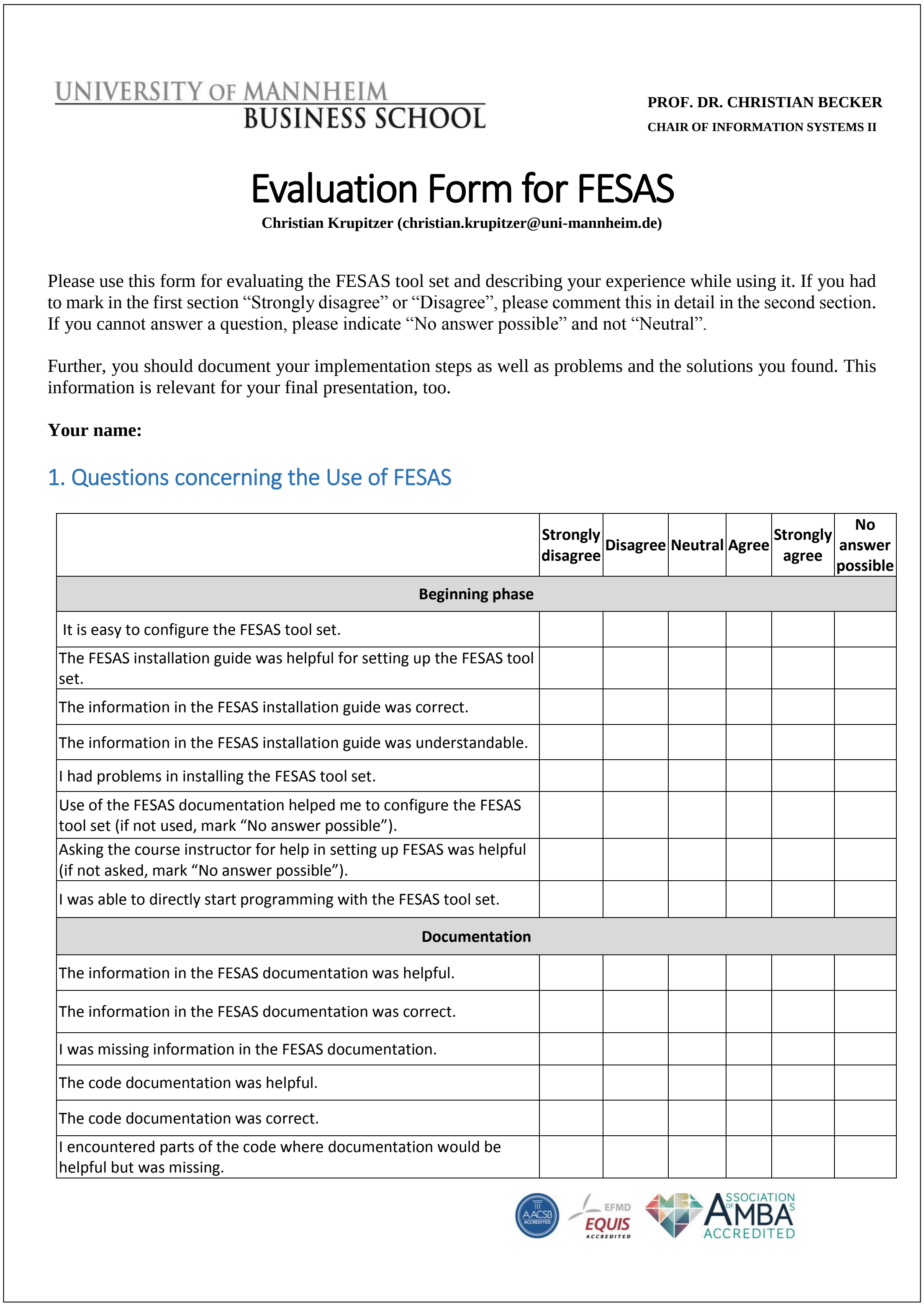
Experience in Software Development (studies, jobs, freelancer, …):

Figure D.2.: Questionnaire for the FESAS IDE (2)

D.2. Questionnaire FESAS Framework

This appendix presents the questionnaire used for the comparison of the FESAS Framework and the FESAS IDE as well as the analysis of the applicability of the FESAS Framework in general in Section 8.2.2.

We derived the following question items from the *ISO 9241-11 Guidance on Usability* standard and the definition of usability in the *ISO/IEC 9126-1 Software Product Quality Model* standard



UNIVERSITY OF MANNHEIM
BUSINESS SCHOOL

PROF. DR. CHRISTIAN BECKER
CHAIR OF INFORMATION SYSTEMS II

Evaluation Form for FESAS

Christian Krupitzer (christian.krupitzer@uni-mannheim.de)

Please use this form for evaluating the FESAS tool set and describing your experience while using it. If you had to mark in the first section "Strongly disagree" or "Disagree", please comment this in detail in the second section. If you cannot answer a question, please indicate "No answer possible" and not "Neutral".

Further, you should document your implementation steps as well as problems and the solutions you found. This information is relevant for your final presentation, too.

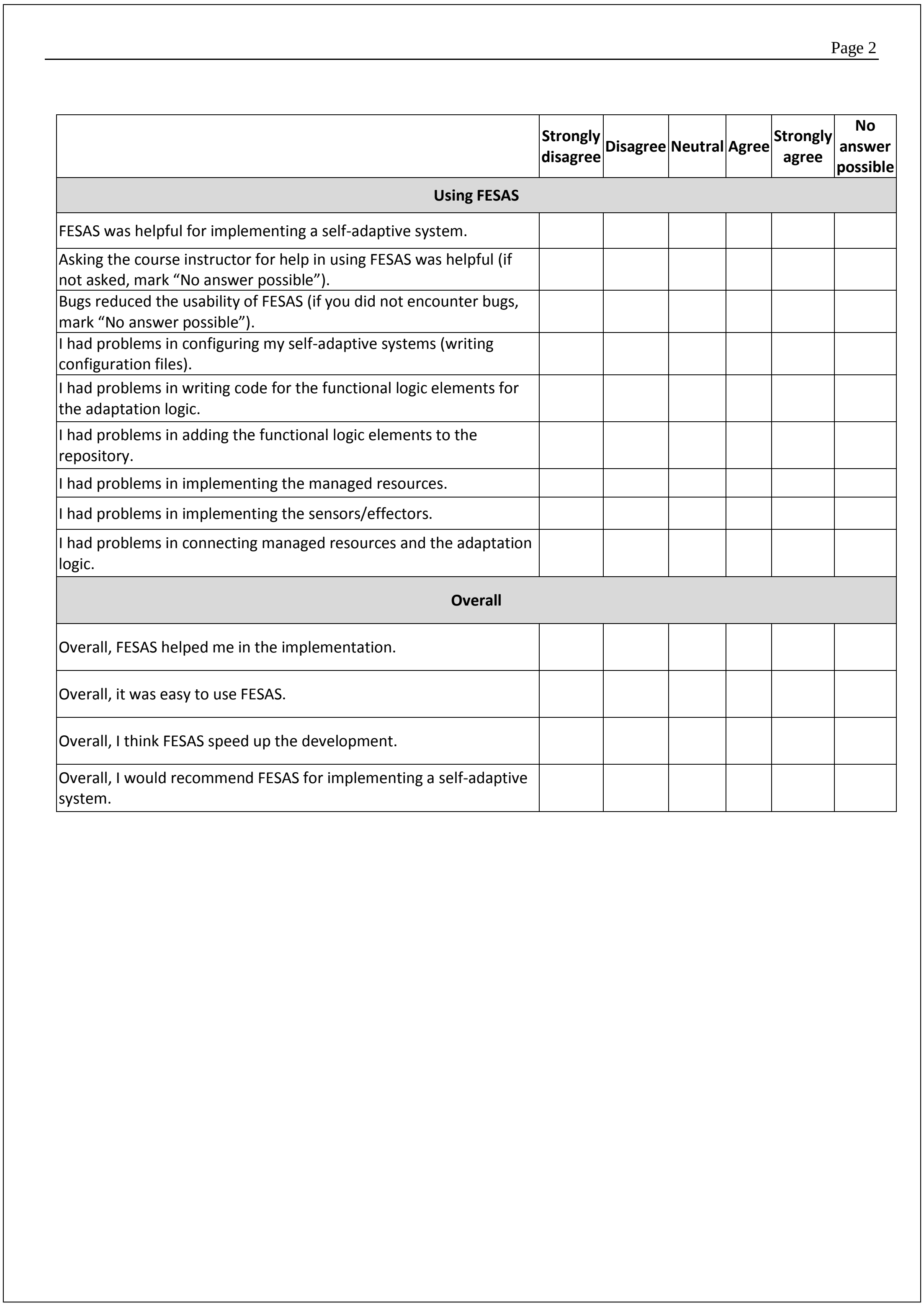
Your name:

1. Questions concerning the Use of FESAS

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	No answer possible
Beginning phase						
It is easy to configure the FESAS tool set.						
The FESAS installation guide was helpful for setting up the FESAS tool set.						
The information in the FESAS installation guide was correct.						
The information in the FESAS installation guide was understandable.						
I had problems in installing the FESAS tool set.						
Use of the FESAS documentation helped me to configure the FESAS tool set (if not used, mark "No answer possible").						
Asking the course instructor for help in setting up FESAS was helpful (if not asked, mark "No answer possible").						
I was able to directly start programming with the FESAS tool set.						
Documentation						
The information in the FESAS documentation was helpful.						
The information in the FESAS documentation was correct.						
I was missing information in the FESAS documentation.						
The code documentation was helpful.						
The code documentation was correct.						
I encountered parts of the code where documentation would be helpful but was missing.						

AACSB ACCREDITED EFMD EQUIS ACCREDITED ASSOCIATION OF MBAS ACCREDITED

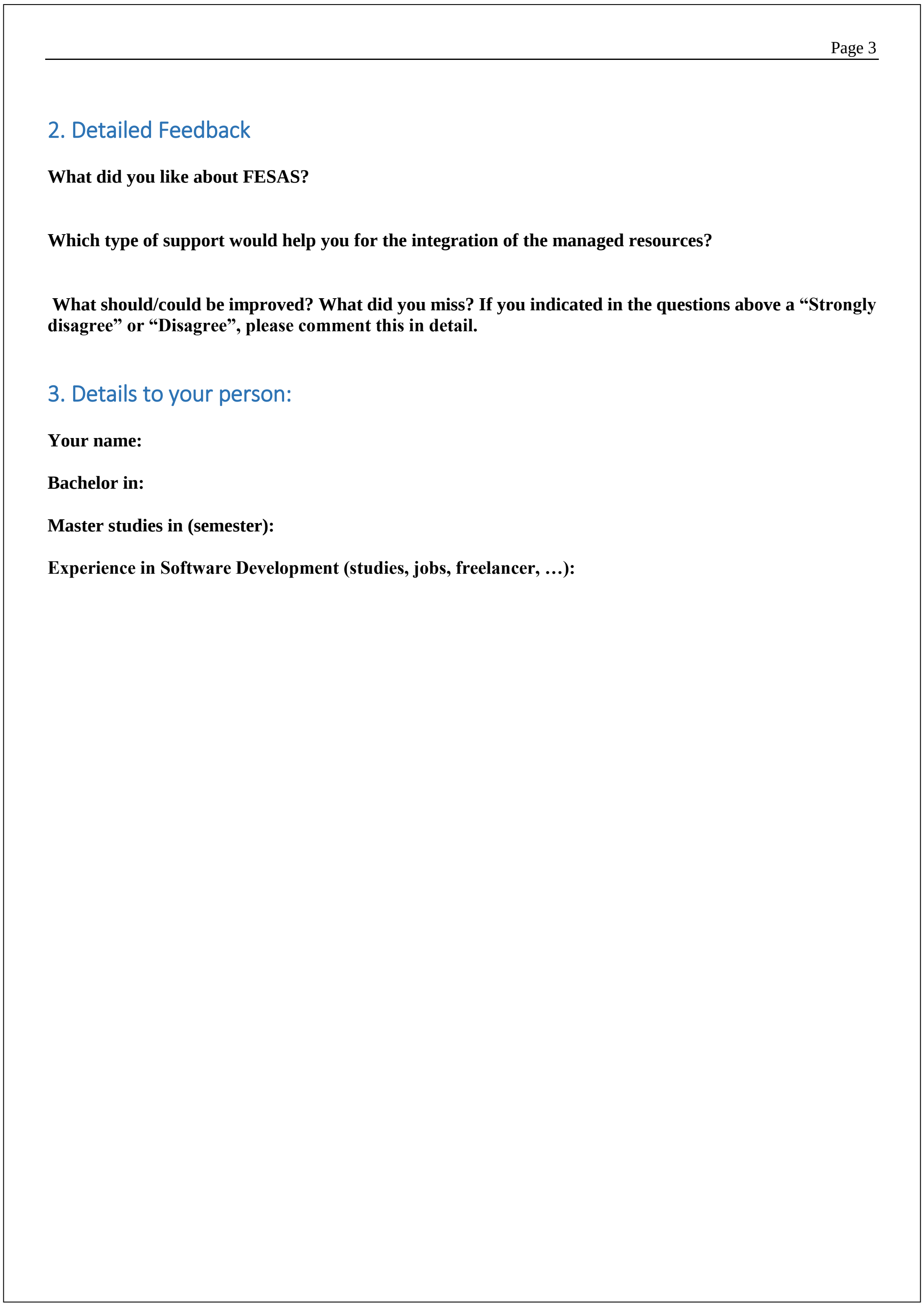
Figure D.3.: Questionnaire for the FESAS Framework (1)



Page 2

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	No answer possible
Using FESAS						
FESAS was helpful for implementing a self-adaptive system.						
Asking the course instructor for help in using FESAS was helpful (if not asked, mark “No answer possible”).						
Bugs reduced the usability of FESAS (if you did not encounter bugs, mark “No answer possible”).						
I had problems in configuring my self-adaptive systems (writing configuration files).						
I had problems in writing code for the functional logic elements for the adaptation logic.						
I had problems in adding the functional logic elements to the repository.						
I had problems in implementing the managed resources.						
I had problems in implementing the sensors/effectors.						
I had problems in connecting managed resources and the adaptation logic.						
Overall						
Overall, FESAS helped me in the implementation.						
Overall, it was easy to use FESAS.						
Overall, I think FESAS speed up the development.						
Overall, I would recommend FESAS for implementing a self-adaptive system.						

Figure D.4.: Questionnaire for the FESAS Framework (2)



Page 3

2. Detailed Feedback

What did you like about FESAS?

Which type of support would help you for the integration of the managed resources?

What should/could be improved? What did you miss? If you indicated in the questions above a “Strongly disagree” or “Disagree”, please comment this in detail.

3. Details to your person:

Your name:

Bachelor in:

Master studies in (semester):

Experience in Software Development (studies, jobs, freelancer, …):

Figure D.5.: Questionnaire for the FESAS Framework (3)

E. Results of the ALM Evaluation

This chapter provides the detailed results of the dynamic analysis of the ALM evaluation from Section 8.3.2. First, Appendix E.1 describes the baseline measurements. Next, Appendix E.2 presents the measurments for parametric self-improvement. Last, Appendix E.3 shows the measurments for structural self-improvement. All measurements are originally presented in [223].

E.1. Baseline Measurement Results

In order to have baseline measurements of the adaptation logic's performance, we first ran two series of 50 simulations each without an adaptation logic as well as an adaptation logic using a fixed rule set for both tracks. The self-improvement layer is inactive. Hence, there are no rules added during the simulation and structural self-improvement is not active. The adaptation logic is able to set three different speed limits. The speed limit is set to unrestricted if the status of the highway is free. In case of stop-and-go traffic, a limit of 120 km/h is set and finally, for traffic jams the speed limit is reduced to 80 km/h. Stop-and-go traffic situations and traffic jams are detected using measurements of the amount of vehicles on the track as well as their average speed. The integrals of the baseline without the adaptation logic is 21,308, the one with the adaptation logic having a fixed ruleset decreases to 14,950. This is equal to decrease of the aggregated waiting by 30 %. For the setting with the daily road work, the integral of the baseline without the adaptation logic is 18,128, the one with the static adaptation logic decreases to 13,032. This results in a decrease of 28 %. However, as traffic jams happen even with the adaptation logic, this is an indicator that the adaptation logic could be improved though meta-adaptation.

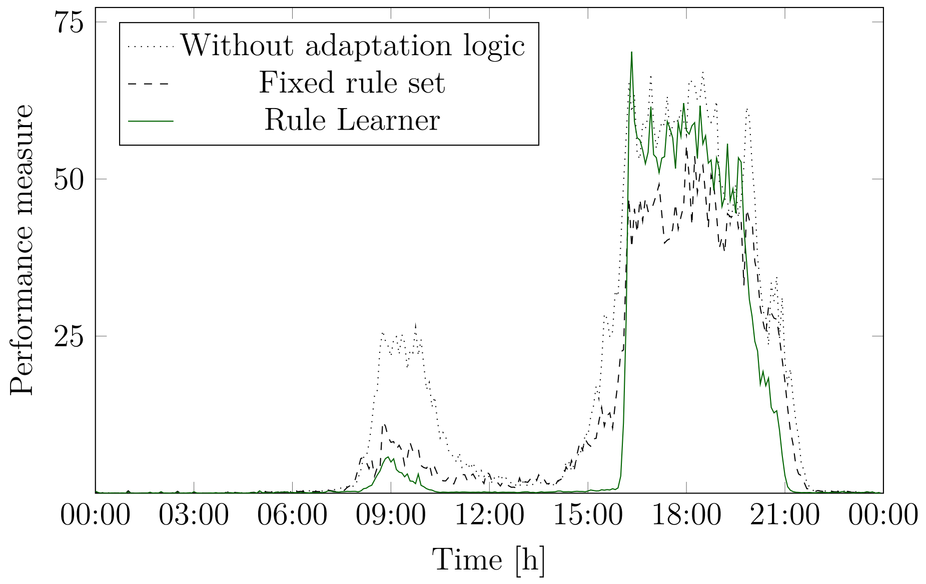


Figure E.1.: Measurements for the parametric self-improvement compared to the baseline measurements. The x-axis shows the time and the y-axis the corresponding waiting time.

E.2. Evaluation Results of Parametric Self-improvement

As a second measurement, we measured the performance of the parametric self-improvement, hence, the rule learner. The recommended speed limits are learned while the system is executed. The system starts with an empty rule base and the rule base evolves over time. Figure E.1 shows the results. For comparison, the results of the baseline evaluation – without adaptation logic and with an adaptation logic having a static rule set – are added.

The integral of the measurement decreases by 1,139 points which is a 7.6 % improvement compared to the fixed rule set and an improvement of 35 % compared to the scenario without adaptation logic. When looking at the progression of the two curves in detail, both start with the same performance. At the beginning of the day, the highway is free. Hence, no speed limits are set. As new rules are learned, the performance of the online learner increases. For the first fifteen and the last four hours of the day, the performance of the parametric self-improvement is constantly better than the baseline with a static adaptation logic. This means that for the traffic conditions in this time frame the learning module found better speed limits than the rules of the static adaptation logic. It can also be concluded that traffic jams form later and resolve earlier which means that the traffic remains longer in a flowing state.

E.3. Evaluation Results of Structural Self-improvement

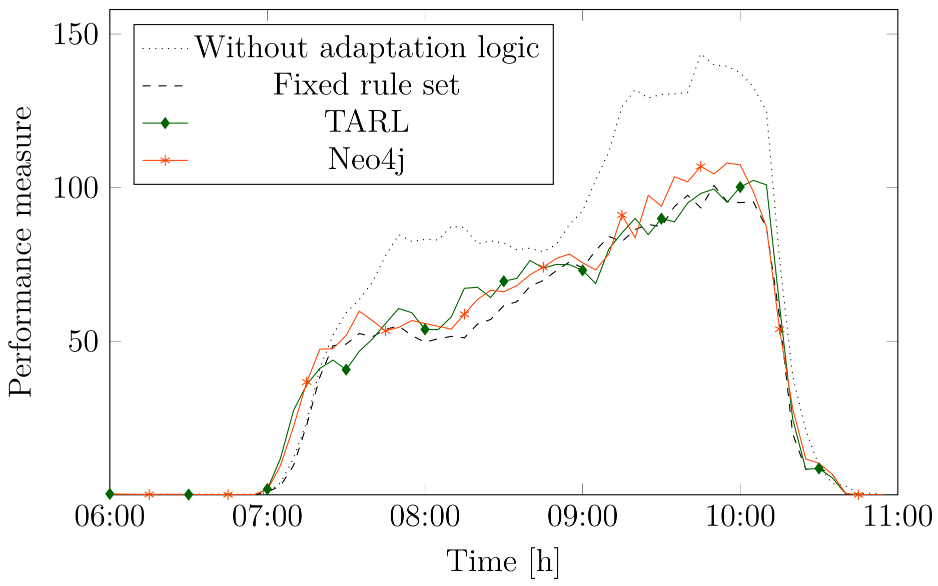


Figure E.2.: Measurements for the structural self-improvement compared to the baseline measurements. The x-axis shows the time and the y-axis the corresponding waiting time.

It is not preferable to learn rules for a spontaneous, non-durable event as time is needed for the simulations to learn rules. Contrary, structural adaptation can react fast to changing conditions. Therefore, we decided to introduce additional events to the evaluation setting that trigger structural self-improvement of the adaptation logic. Reaction to events is not limited to reactive adaptation only, but includes a proactive adaptation as reaction to forecasted events. We additionally simulate an accident that leads to closing the second street in section 1. Additionally, we introduce a daily road work in the last section of the track. Further, as structural adaptation is event-based, we simulated having the road work during the morning rush hour between 6am and 11am.

We configured the following parameters of the prediction module: size of the training set (1,000), amount of forecasted time steps (200), and classifier (time series forecast based on Support Vector Machines for Regression). We performed

50 runs using the `PREDEFINED` execution strategy in the ALM Planner. As TARL needs more time than Neo4j and both have the same utility, we decided to use this execution strategy and changed the order after 25 runs. The Neo4j module has an integral of 14,055, the TARL module performs slightly better with 13,713. Both approaches perform similarly to the static adaptation logic. Figure E.2 shows the results of the evaluation of the structural self-improvement.

Images Integrated in This Thesis

This list presents the origin of images that are integrated in this thesis. All other symbol are own drawings in Visio, Powerpoint, or Paint. The origin of the Visio car shape is unknown.

Figure 2.5

- Server: `https://lizenzbilliger.de/img/cms/SERVER.png`
- Smartphone: `http://picscdn.redblue.de/doi/pixelboxx-mss-68958264/fee_786_587_png/APPLE-iPhone-6-32-GB-Spacegrau-`
- Production robot: `http://www.k-aktuell.de/wp-content/uploads/2015/06/kuka150617-214x300.jpg`
- Robot: `https://assets.cdn.moviepilot.de/files/fd5aefcbf20ab88ea57eb4b1032a8c630bd391e59630784de38805aa03b9/limit/960/600/Blech_08.jpg`
- Car: `https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSwIMipHMGTASKBfY3fDGrD3UL-7BECSj-nCsh1f62R-yIxyWJIBg`

Figure 6.3

- User: `http://1.bp.blogspot.com/-JGddZ9etXnE/Uhw15ev4vSI/AAAAAAAAFk0/MvwD1k8eH-k/s1600/excited.jpg`
- Toolset: `https://files.encuentra24.com/large/28/02/2802564_c69953.jpg`
- Eclipse Screenshot: `http://www.irisa.fr/espresso/Polychrony/images/sme_environment.png`

For further images see list for Figure 2.5.

Figure 7.2

- Traffic sign accident: `https://de.fotolia.com/id/7299484`
- Wifi symbol: `https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcStq6sruhl4Yo8a7ibf_ws5-nengYEu2IYyUKbTgwJ5TmHkC9-g`

Figure 7.15

- Traffic simulation: `http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/`
- Robot simulation: `https://www.controleng.com/fileadmin/_processed_/f/e/csm_CTL1712_MAG_F2_RIA_Simulation_Fig1-Genesis-LaserWeld_8942e8caf1.jpg`
- Network simulation: `https://thomazchamberlain.files.wordpress.com/2012/02/screenshot-at-2012-02-08-171227.png`

Figure 7.17

- User group: `http://www.wisetronic.com/wp-content/uploads/2017/01/icon-slider-image-one.png`

For further images see list for Figure 2.5.

Figure 7.18

For all images see list for Figure 2.5.

Figure 8.2

- Wifi symbol: `https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcStq6sruhl4Yo8a7ibf_ws5-nengYEu2IYyUKbTgwJ5TmHkC9-g`
- Visio truck shape: `https://leannekroll.wordpress.com/2008/06/09/d-truck-orthos-isos/`

Figure 8.3

- Load balancer: `https://www.1001freedownloads.com/free-cliparts/?order=popular&tag=server`

Figure 8.5

- iCasa screenshot: `http://adeleresearchgroup.github.io/iCasa/snapshot/index.html`

Figure 8.7

- Gyroscope: `https://upload.wikimedia.org/wikipedia/commons/thumb/e/e2/3D_Gyroscope.png/1200px-3D_Gyroscope.png`
- Magnetometer: `https://www.nxp.com/videos/poster/NXP-SENSOR-PROCESSING-MOTI`
`jpg`

The origin of the other symbols is unknwon.

Publications Contained in This Thesis

[19] P. Arias-Cabarcos and C. Krupitzer. On the design of distributed adaptive authentication systems. In *Proceeding of the Symposium on Usable Privacy and Security (SOUPS)*. USENIX, 2017.
My Contribution: Mapping of the concept to the Design for a SAS (Section IV).

[111] J. Edinger, D. Schäfer, C. Krupitzer, V. Raychoudhury, and C. Becker. Fault-Avoidance Strategies for Context-Aware Schedulers in Pervasive Computing Systems. *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*, pages 79–88, 2017.
My Contribution: Implementation of the simulation, Design of the System (Section IV).

[219] C. Krupitzer. FESAS: A Framework for Engineering Self-Adaptive Systems. In *Proceedings of the First Organic Computing Doctoral Dissertation Colloquium (OC-DDC'13)*, pages 16–19, 2013.
My Contribution: Responsible author of the paper.

[220] C. Krupitzer, M. Breitbach, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker. A Survey on Engineering Approaches for Self-Adaptive Systems (Extended Version). Technical report, Chair of Information Systems II, University of Mannheim, Germany, 2018.
My Contribution: Coordination of the extension, Search of the new material.

[221] C. Krupitzer, M. Breitbach, J. Saal, C. Becker, M. Segata, and R. Lo Cigno. RoCoSys: A framework for coordination of mobile IoT devices. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom) Workshops*, pages 485–490. IEEE, 2017.
My Contribution: Responsible author of the paper.

[222] C. Krupitzer, G. Drechsel, D. Mateja, A. Pollkläsener, F. Schrage,

T. Sturm, A. Tomasovic, and C. Becker. Using Spreadsheet-defined Rules for Reasoning in Self-Adaptive Systems. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom) Workshops*, pages 462–467. IEEE, 2018.
My Contribution: Responsible author of the paper, Coordination of Section II & V.A.

[223] C. Krupitzer, J. Otto, F. M. Roth, A. Frommgen, and C. Becker. Adding Self-Improvement to an Autonomic Traffic Management System. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 209–214. IEEE, 2017.
My Contribution: Responsible author of the paper.

[224] C. Krupitzer, M. Pfannemüller, V. Voss, and C. Becker. Comparison of Approaches for developing Self-adaptive Systems. Technical report, Chair of Information Systems II, University of Mannheim, Germany, 2018.
My Contribution: Responsible author of the paper.

[225] C. Krupitzer, F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schürr. FESAS IDE: An Integrated Development Environment for Autonomic Computing. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 15–24. IEEE, 2016.
My Contribution: Responsible author of the paper.

[226] C. Krupitzer, F. M. Roth, M. Pfannemüller, and C. Becker. Comparison of Approaches for Self-Improvement in Self-Adaptive Systems. In *Proceedings of the International Conference on Autonomic Computing (ICAC))*, pages 308–314. IEEE, 2016.
My Contribution: Responsible author of the paper, Coordination of the writing.

[227] C. Krupitzer, F. M. Roth, M. Pfannemüller, and C. Becker. Comparison of Approaches for Self-Improvement in Self-Adaptive Systems (Extended Version). Technical report, Chair of Information Systems II, University of Mannheim, Germany, 2017.
My Contribution: Responsible for the extension.

[228] C. Krupitzer, F. M. Roth, S. VanSyckel, and C. Becker. Towards Reusability in Autonomic Computing. In *Proceedings of the International Confer-*

ence on Autonomic Computing (ICAC), pages 115–120. IEEE, 2015.
My Contribution: Responsible author of the paper.

[229] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker. A Survey on Engineering Approaches for Self-Adaptive Systems. *Pervasive and Mobile Computing Journal*, 17(Part B):184–206, 2015.
My Contribution: Responsible author of the paper, Wrote most of the paper.

[230] C. Krupitzer, T. Sztyler, J. Edinger, M. Breitbach, H. Stuckenschmidt, and C. Becker. Hips do lie! a position-aware mobile fall detection system. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*, pages 95–104. IEEE, 2018.
My Contribution: Implementation of the fall detection system, Responsible author for all sections except of Section V.

[231] C. Krupitzer, S. VanSyckel, and C. Becker. FESAS: Towards a Framework for Engineering Self-Adaptive Systems. In *Proceedings of the 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 263–264. IEEE, 2013.
My Contribution: Responsible author of the paper.

[293] M. Pfannemüller, C. Krupitzer, M. Weckesser, and C. Becker. A dynamic software product line approach for adaptation planning in autonomic computing systems. In *Proceeding of the International Conference on Autonomic Computing (ICAC)*, pages 247–254. IEEE, 2017.
My Contribution: Outcome of a Master thesis that I supervised, Supervision of paper writing process.

[315] F. M. Roth, C. Krupitzer, and C. Becker. Runtime evolution of the adaptation logic in self-adaptive systems. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 141–142. IEEE, 2015.
My Contribution: Contribution in the design of the approach.

[316] F. M. Roth, C. Krupitzer, S. Vansyckel, and C. Becker. Nature-Inspired Interference Management in Smart Peer Groups. In *Proceedings of the International Conference on Intelligent Environments (IE)*, pages 132–139. IEEE, 2014.

My Contribution: Outcome of a Master thesis that I supervised, Supervision of paper writing process.

Lebenslauf

Seit 11/2012	Akademischer Mitarbeiter Lehrstuhl für Wirtschaftsinformatik II Universität Mannheim
08/2011 – 01/2012	Informatik Vrije Universiteit Amsterdam
07/2007 – 10/2012	Bachelor & Master of Science Wirtschaftsinformatik Universität Mannheim