Optimization Approaches for Self-Adaptive Systems

Tim Engbrocks

Institute for Program Structures and Data Organization (IPD)
Advisor: Dipl.-Inform. Martina Rapp

Abstract The complexity of modern software systems is constantly growing. This requires new ways to be found to better manage large scale software systems. Self-Adaptive Systems which autonomously manage themselves using well-defined rules are a solution to this problem. Even though these systems are able to adapt themselves, they struggle with unpredicted changes in their context. This can be solved by using Optimization Approaches for Self-Adaptive Systems which can optimize these systems. There are already many Optimization Approaches, but no way to classify them. This paper proposes a classification for Optimization Approaches for Self-Adaptive Systems.

1 Introduction

The complexity of modern software systems is constantly growing. Most of this growth in complexity stems from the "need to integrate several heterogeneous environments into corporate-wide computing systems, and to extend that beyond company boundaries into the Internet" (Kephart and Chess, 2003 [5]). This has reached a state where the "complexity appears to be approaching the limits of human capability" (Kephart and Chess, 2003 [5]). In combination with the uncertainty about a software systems future operations and environment, that the developers of such complex systems face, it becomes uneconomical to purely operate such a system by human operators.
From this the need for software systems which can autonomously manage themselves arises. In order to achieve this task of autonomous self-management, the system has to be able to:

- detect faults and changes in its environment,
- analyze them,
- decide how to react to them
- and make changes to itself.

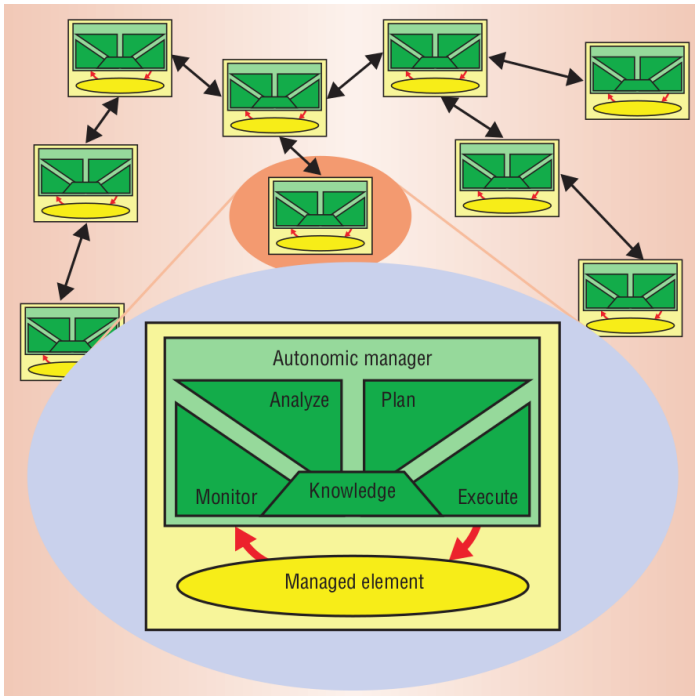

Figure 1: The MAPE-K (Monitor-Analyze-Plan-Execute with Knowledge) feedback loop by Kephart and Chess, 2003 [5]

To model these abilities Kephart and Chess developed the Monitor, Analyze, Plan, Execute with Knowledge (MAPE-K) feedback loop [5] shown in Figure 1.

First the system has to *monitor* itself and its environment. The data, gathered by the monitoring step, has to be *analyzed* to detect changes and faults. If the analyzing step detects, that an adaptation is necessary, the system has to *plan* how to perform the necessary changes. After the changes have been planned, they need to be *executed*. All of this happens with *knowledge* of the environment and the system.

Software systems that can autonomously manage themselves are called Self-Adaptive Systems (SAS) because of their ability to adapt themselves.

To better understand SAS, let us take a look at a commonly used example. Imaging you are the system administrator of a large scale online store. This store has four parameters: site traffic, number of purchases, number of active server instances and the number of served advertisements. During your day to day operations you encounter a common type of task: Update some system parameter X based on some metric Y. To make your job easier, you decide to use a SAS for these tasks and come up with the following generalized adaptation rule: If metric Y crosses threshold Z, update the system parameter X.

In this case the usage of a SAS was beneficial because it could easily automate a general set of tasks. A human operator might have been able to perform these tasks on his own, if the number of system parameters was sufficiently small. But the SAS is better at handling large numbers of system parameters.

While SAS are better at handling more complex systems, human operators are better at handling uncertainty. This has two reasons. Firstly, the adaptation rules and policies used

by SAS are statically created at design time. Secondly, SAS can adapt the software that they are managing but they can not change their adaptation process. Over time this leads to an increasing divergence between the expected results of adaptations and the actual results, when the environment changes in ways that were not predicted by developers during the design time of the system.
This can be illustrated by the previous example. Imagine that the SAS has been in operation for some and you collected data on the systems performance. You notice that the SAS changes some system parameters too aggressively, because it performs an adaptation as soon as a metrics threshold has been violated. As a human operator you would have waited some time to see how the metric develops before performing an adaptation which would result in a smoother operation. The SAS can not handle this type of uncertainty and only reacted to the current state of its environment.
Optimizations are necessary to improve the performance and effectiveness of SAS in situations like these. There are already many approaches on how to optimize SAS. Some of them focus on updating adaptation rules and policies during the systems runtime. Others dynamically change at which level of the system adaptations are performed. Generally most optimizations target static components of SAS. These components can be improved by making them more dynamic, which is often achieved by applying modern learning methods.
The previous example could benefit from such optimizations by dynamically updating adaptation rules to better reflect the systems changing environment.
While there are already many Optimization Approaches (OA) for SAS, there is no classification for them. Because of this, the existing OA can not be easily compared and it is difficult to identify areas which require further research. This paper aims to provide such a classification for Optimization Approaches for Self-Adaptive Systems.
To derive a classification for OA for SAS, chapter 2 will first explain how SAS are classified using three different approaches. Based on these approaches, a classification for OA for SAS will be derived and proposed in chapter 3. In chapter 4 the proposed classification will be applied to some existing OA. Lastly, chapter 5 will finish with a conclusion and recommendations for future research directions.

2 Classification of Self-Adaptive Systems

There are different approaches on how to classify and describe Self-Adaptive Systems which all focus on different usages. The three approaches that will be highlighted by this section are:

- FORMS from Weyns et al., 2012 [3]
- Berns and Ghosh, 2009 definition of self-* properties [2]
- Krupitzer's et al., 2015 taxonomy for Self-Adaptive Systems [6]

These approaches contain ideas that will be used to construct the classification for OA for SAS in the next chapter.
In their 2012 paper FORMS [3] Weyns et al. propose a formal reference model for describing SAS. The goal of FORMS is to provide a well defined basis for talking and reasoning about SAS. This is achieved by providing a definition of FORMS which classifies SAS using Z

notation. Z notation is used for formal specifications and is based upon Zermelo-Fraenkel set theory. It is mathematically well-defined and often used to describe software systems.

FORMS describes SAS by dividing them into components which are arranged over different layers. The bottom most layer is called the base layers. This layer contains domain models and base components. These are directly responsible for interacting with the systems environment. On the layers above the base layer are reflective layers with reflective components. These components can adapt components in the layer beneath them. Reflection refers to the ability of a system to examine and modify itself.

This approach can be understood with the example of a robot that uses motorized wheels and some sensors to observe its environment. The motor control software used by the robot would be placed in the base layers. The sensors would be placed in the base layer as well. These components are responsible for observing the environment and interacting with it. On top of the base layer the robot might use a reflective layer which changes how the wheels are operated based on the current weather conditions that it observes. The top layer of the robots software could be an automatic updater which automatically updates the robots software to the latest version. For this, the updater has to examine the robots software and modify it. In other words, the updater is a reflective component.

Another example for composing SAS from base and reflective components and layers from the original paper by Weyns et al. is depicted in Figure 2.
The concept of different layers of reflection will be useful later to distinguish OA for SAS from SAS.

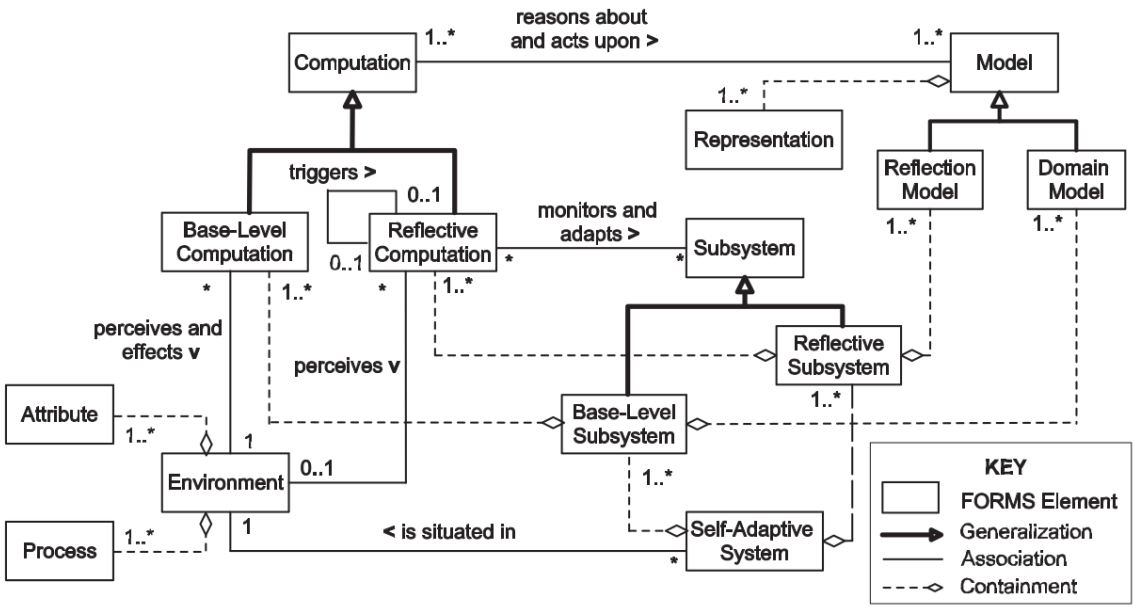

Figure 2: FORMS primitives by Weyns et al., 2012 [3]

SUMMARY OF SELF-* PROPERTIES. F = SET OF EXTERNAL ACTIONS, P = PREDICATE OVER GLOBAL STATE.

Self-* Property Name	**Adversarial actions**	**Behavior**	**Predicate maintained**
Self-stabilization	All transient failures	Restore	P
Self-adapting	Change of environment R	Restore	**if** R_i **then** P_i
Self-healing	Some $C \subseteq F$	Restore	P
Self-organizing	Process join or leave	Maintain, improve, or restore	P
Self-protecting	Some mailicious actions	Maintain	Predicate over trust
Self-optimization	Some $C \subseteq F$	Improve or maximize/minimize as appropriate	An objective function of the global or local state
Self-configuration	Some $C \subset F$, often includes user demands for service	Maintain, improve, or restore using configuration changes	Predicate over system configuration to optimize performance. Sometimes addresses geometric invariant
Self-scaling	Changes of system scale or demand	Restore, or improve	Predicate over service quality
Self-immunity	Some $C \subseteq F$	Eventually maintain	P
Self-containment	Some mailicious actions	Maintain (for a subset of processes)	Predicate over trust

Figure 3: self-* properties by Berns and Ghosh, 2009 [2]

Although some of the first papers on SAS, for example, "The Vision of Autonomic Computing" by Kephart and Chess [5] focused on self-adaptation, there are other aspects of software systems that can benefit from the ideas introduced by SAS. Berns and Ghosh, 2009 [2] identified and described such aspects which they call self-* properties. These self-* properties can be seen in Figure 3. They are useful to:

- Communicate the abilities and goals of a system.
- Establish well-defined goals for the system.

These self-* properties show that there are aspects besides the management of software which can benefit from the ideas of Self-Adaptation. This will be useful for the proposed classification as it gives a clear setting for the OA to work within.

The taxonomy for SAS by Krupitzer's et al., 2015 [6] in Figure 4 is based upon the 5W+1H questions by Salehie and Tahvildari, 2009 [7]. These questions are: Where, When, What, Why, Who and How. Each of these questions is responsible for a different aspect of SAS and corresponds to a dimension of the taxonomy.

Why First there needs to be a reason for a SAS to adapt. Why an adaptation should be performed is answered by the Reason dimension. According to the taxonomy reasons for an adaptation can be changes in either the context, a technical resource or changes caused by the user.

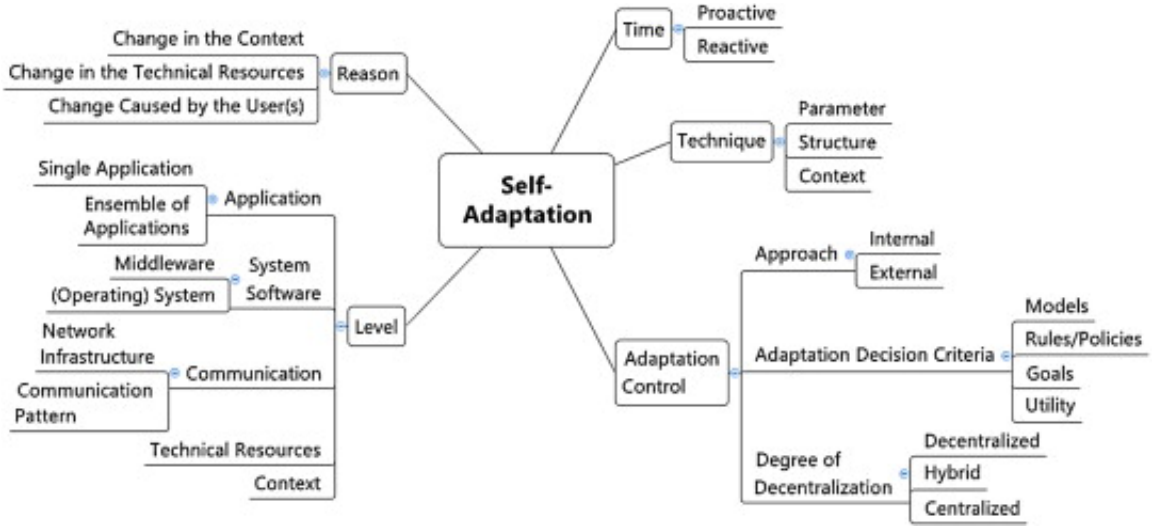

Figure 4: Taxonomy for SAS by Krupitzer et al., 2015 [6]

Where The question of where asks on which level of the system changes need to occur. The different levels on which changes can occur include:

- Different levels of applications from the operating system to a user application.
- How systems communicate with each other.
- The technical resources that are needed by the system.
- The context in which the system operates.

When While the original When-Question by Salehie and Tahvildari tries to understand all temporal aspects of SAS, including how frequently changes should occur and if they happen continously, the taxonomy only answers the question when changes should be performed. For this purpose the Time dimension differentiates between systems that perform changes proactively or reactively. Reactive changes occur after, for example, a system metric has been violated. Proactive changes occur before a system metric can be violated.

What In addition to the question of where and when changes should occur, it is also important to know what changes should occur. There are different techniques that can be used. The Technique dimension of the taxonomy differentiates between systems that change parameters, their structure or their context. A system that changes its structure could, for example, be a datacenter, which starts new server instances on demand. Robots are an example for systems that change their context. This is achieved by, for example, moving the robot around.

Who After answering where, when, what and why changes should be performed, it is necessary to select who is responsible for these changes. According to Salehie and Tahvildari it is also important to establish if the changes can be performed fully autonomous or if the involvement of human operators is necessary. The taxonomy does not directly

address all of these concerns but states that: "N/A (nature of a SAS leads to an automatic type of adaptation)" (Krupitzer et al., 2015 [6]).

How Lastly, after determining the where, when, what, why and who, there needs to be a way to perform the required changes. This is answered by asking how the changes should be performed and corresponds to the Adaptation Control. The three main aspects of the Adaptation Control are the degree of decentralization, the adaptation decision criteria and the approach taken by the system. The degree of decentralization ranges from systems which perform adaptations through a central component (fully centralized) to systems where each component is responsible for its own adaptation (fully decentralized). In between these two degrees are systems that employ a hybrid approach. The adaptation decision criteria controls how the adaptation is achieved. Krupitzer et al. propose the following possibilities:

- Models: The SAS updates e.g. domain models which results in behavioral changes.
- Rules/Policies: The SAS changes its behavior based on rules or policies.
- Goals: The SAS changes its behavior to meet some predefined set of goals.
- Utility: The SAS changes its behavior to maximize or minimize a utility function.

The approach divides SAS into those where the adaptation logic is part of the application logic and those with separated adaptation and application logic.

After classifying SAS the following question can be asked: Which parts of a SAS can be optimized? To answer this we will start by looking at which parts can not be optimized or do not benefit from optimization.

The first part that can not be optimized is the environment which provides the Reason dimension. While the environment for a SAS can be chosen in a way which is most beneficial for the system and can be influenced by actors, the behavior of the systems environment can generally not be controlled.

Another dimension of SAS that can not be optimized, or is not useful to optimize, is the Time dimension. This dimension is mostly a design decision on how the system should behave and be constructed. It is also a question of how to handle uncertainty and the level of accepted risk. A proactive system can prevent faults and degradation in Quality-of-Service metrics, but it can also predict the wrong changes which can lead to a situation where the system itself generates faults by reacting in a way that is contradictive to its goals. A reactive system can not prevent faults like a proactive system, but its behavior can be much more stable, because it only has to react to a change and not predict that change as well.

Lastly, the question of who is responsible can not be optimized, because the taxonomy simply answers it, by referring to the name "SAS" which implies that the system itself is responsible for managing adaptations.

The three remaining dimensions can be optimized or can benefit from being adapted dynamically. These are the Adaptation Control, the Level and the Technique.

The Approach and the Degree of Decentralization used by the Adaptation Control can not be optimized because they are design decisions of how the system is built. However, the Adaptation Decision Criteria can be optimized. An optimization of the Adaptation

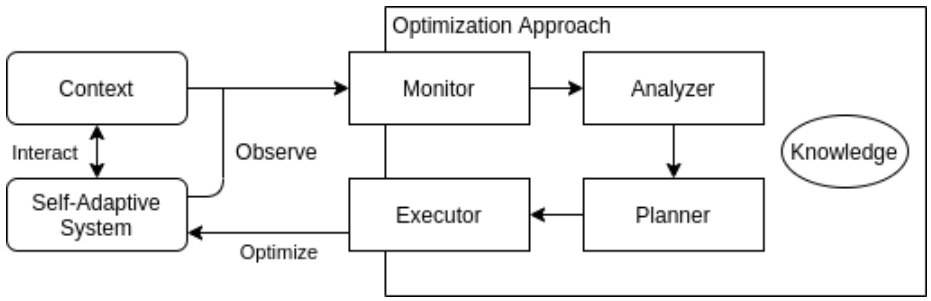

Figure 5: Mapping the optimization process to the MAPE-K feedback loop.

Decision Criteria could, for example, be to dynamically adapt the rules and policies at runtime to better reflect a changing environment.
Another dimension that can be optimized is the Technique. This can be optimized by changing what gets adapted by the system.
The last dimension that can be optimized is the Level, which can be done by dynamically changing at which level of the system adaptations should be performed.
This explanation of how SAS are classified, leads us to how we can classify OA for SAS.
The main concepts that will be used for the classification of OA are:

- The concept of using reflectiveness to describe SAS.
- The 5W+1H questions to determine how a SASoperates.
- The MAPE-K feedback loop as a general base for understanding SAS.
- The concept of self-* properties which enables us to differentiate between self-adaptation and other self-* abilities.
- A taxonomy for SAS which will be the basis for the proposed classification of OA, because SAS are so closely related to their OA.

3 Proposal for Classification of Optimization Approaches

This chapter will derive and propose a classification for OA for Self-Adaptive System based on concepts that were discussed in chapter 2.
Using the idea from Weyns et al. 2012 paper FORMS [3] to compose SAS from layers of base and reflective components, an Optimization Approach for SAS can be thought of as a reflective component in a layer above the SAS.
This allows the application of multiple OA to SAS and even the optimization of OA.
It also establishes a connection between OA and SAS. This leads to the realization that a classification for OA of SAS should be based upon the same principles as the classifications for SAS.
Another benefit of this approach is a clear distinction between SAS and OA for SAS. SAS and OA are both composed of reflective components and not base components.
Yet the SAS adapts base components, while the OA adapts other reflective components.
Just like the adaptation process of SAS can be understood using the MAPE-K feedback loop by Kephart and Chess, 2003 [5]. The process of optimizing a SAS can also be expressed using the MAPE-K feedback loop:

- Firstly the OA has to constantly monitor the SAS and its context.
- The data gathered from monitoring can then be analyzed to decide wether or not an optimization is necessary and what should be optimized.
- After deciding that something should be optimized, there needs to be a plan on how to optimize.
- Lastly the planned optimization can be executed.
- During all of these steps the OA requires knowledge about the SAS and its context.

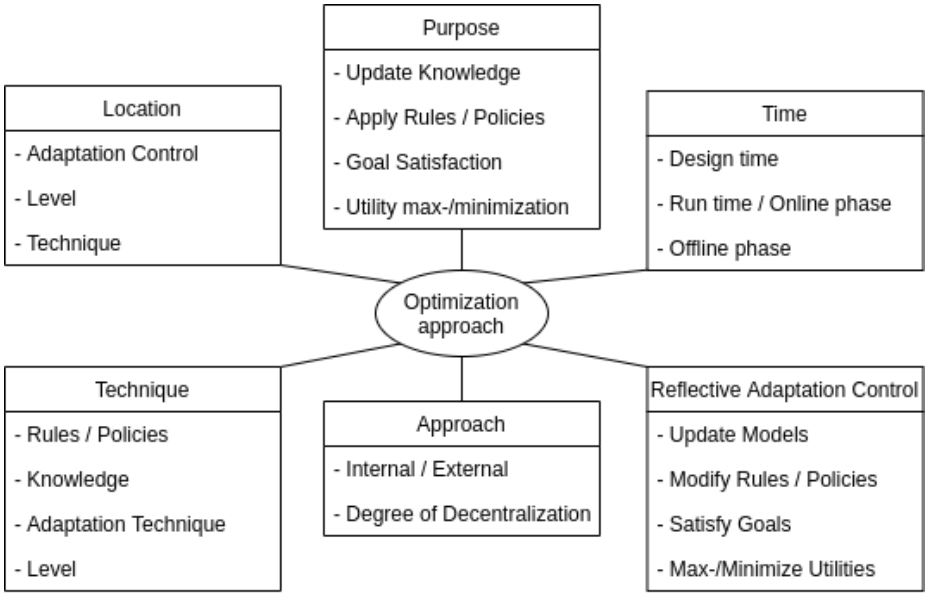

Figure 6: The proposed classification for OA for SAS

Following Krupitzer's et al, 2015 [6] taxonomy for SAS, a classification for OA of SAS should answer the 5W+1H questions by Salehie and Tahvildari, 2009 [7]. These questions are: Where, When, What, Why, Who and How?
Figure 6 shows the proposed classification for OA for SAS. The classification consists of six dimensions: Location, Time, Purpose, Approach, Technique and Reflective Adaptation Control. Each dimension is related to one of the 5W+1H questions.

Location Where in the system is an optimization necessary?
There are three main parts in a SAS that can be optimized as explained in chapter 2. These are:

- The Adaptation Control which can, for example, be optimized by changing or generating new adaptation rules.
- The Level on which adaptations are performed.
- The Technique that is used to perform adaptations.

Time When are optimizations performed?
Similarly to SAS, OA can be differentiated by comparing when they perform optimizations. There are three different phases during the lifetime of a SAS where optimizations can occur. These are:

- At the runtime of the system, which can also be called the online phase.
- During the design time of the system.
- While training the system or during its offline phase.

The training phase can happen in parallel to the run and design time of the system. Examples for this would be when the initial parameters for a SAS are chosen by training a domain model or when generating a new model for the system by training an updated domain model parallel to the running system.

Technique What gets changed to perform the optimization?
What gets changed in a SAS relates to where an optimization is necessary. Thus, the following parts can be changed:

- The Rules / Policies used by the Adaptation Decision Criteria.
- The Knowledge of the system, which for example can consist of domain models.
- The Adaptation Technique used by the SAS.
- The Level on which the SAS performs changes.

One could argue that the Goals or especially Utility functions of a SAS could also be changed to optimize its performance. In most cases this is not advisable because Goals and Utility functions can encode information like legal or safety parameters, of which the system has no intrinsic knowledge.

Purpose Why should an optimization occur?
This is perhaps the most important question for any OA as it determines if an optimization is necessary. There can be several reasons for performing an optimization in a SAS:

- The system learned new information about, for example, its environment and needs to update its domain models.
- Due to external or internal changes, a Goal might not be satisfiable anymore.
- The difference of the actual versus the expected outcome of a rule / policy has exceeded a threshold.
- A Utility function needs to be maximized or minimized.

Approach Who is responsible for performing the optimization?
The original "Who?" question by Salehie and Tahvildari focuses on the level of human intervention that is need by the system. SAS have no need for human intervention, because they operate on clearly defined rule sets. It is in the nature of OA for SAS to change these rule sets. This can lead to problems that make human intervention necessary. A result of this is that, in contrast to the the taxonomy for SAS, the Approach for OA should have its own dimension.

- Is there an internal component that performs changes or are they performed by an external actor.
- Which degree of decentralization is used? Is each component responsible for itself (fully decentralized), are all components optimized by a central entity (fully centralized) or is a hybrid approach used?

Reflective Adaptation Control How is the optimization applied to the system?
Just like the SAS can be described by its Adaptation Control, which is responsible for applying changes to the system, an OA can also be described by how it applies its optimizations to the system. In reference to FORMS by Weyns et al., 2012[3], which uses reflective operations to change components in the system, the Adaptation Control of the OA will be called Reflective Adaptation Control. The Reflective Adaptation Control can be classified by the following behaviors:

- Models: The system updates its models after receiving new information about its models.
- Rules / Policies: If the system detects, that it is in a state for which an adaptation rule/policy exists, it should perform that adaptation.
- Goals: Adaptation should be performed if the system does not meet pre-defined goals.
- Utility: An adaptation occurs to maximize or minimize a utility function.

Because the proposed classification is based upon the MAPE-K feedback loop and the 5W+1H questions, it is useful to understand how they relate to each other. This is depicted in Figures 7 and 8.

Optimization Approach	MAPE-K
Location	Monitor, Analyze, Execute
Time	Monitor, Execute
Technique	Plan, Execute
Purpose	Analyze
Approach	Plan, Execute
Reflective Adaptation Control	Plan, Execute

Figure 7: How the MAPE-K feedback loop by Kephart and Chess, 2003[5] relates to the dimensions of the classification for OA for SAS.

Optimization Approach	5W+1H questions
Location	Where
Time	When
Technique	What
Purpose	Why
Approach	Who
Reflective Adaptation Control	How

Figure 8: How the 5W+1H questions by Salehie and Tahvildari, 2009 [7] relate to the dimensions of the classification for OA for SAS.

4 Classifying Existing Optimization Approaches

FUSION by Elkhodary et al, 2010 [1] uses a learning and an adaptation cycle. The adaptation cycle corresponds to the Self-Adaptive part of the system and does not directly affect the learning cycle. The learning cycle is the Optimization Approach used by FUSION, it changes the behavior of the adaptation cycle. Both of these cycles continuously run in parallel to each other.

- Location: FUSION optimizes the Adaptation Control.
- Time: FUSION uses both the design and run time to perform optimizations. The design time is used to generate an initial model for the learning cycle. During the run time the learning cycle monitors adaptations and updates the Knowledge Base to improve future adaptations.
- Technique: FUSION updates the rules used by the SAS and the shared knowledge base.
- Purpose: FUSION aims to decrease the difference between the expected and actual outcome of adaptation rules.
- Approach: FUSION acts as an external actor.
- Reflective Adaptation Control: FUSION applies optimizations by updating models and modifying rules.

Learning Revised Models for Planning in Adaptive Systems by Sykes et al., 2013 [8] uses "a non-monotonic probabilistic rule learning technique, NoMPRoL, that finds hypotheses consisting of new rules specifying the probability of an action achieving its specified outcome under particular conditions" (Sykes et al., 2013 [8])

- Location: The optimizations focus on the Adaptation Control.
- Time: The run time is used to perform optimizations.
- Technique: This approach optimizes the adaptation rules and domain models.
- Purpose: The optimizations aim to generate new and better adaptation rules and domain models.
- Approach: The optimizations are performed by an internal component that sits between the domain model and the SAS.
- Reflective Adaptation Control: This approach generates new adaptation rules.

FIoT by Moraes do Nascimento and Pereira, 2016 [4] is a framework for using agent-based systems to construct self-organizing systems for the Internet of Things (IoT). For this they use three different types of agents: God, Observer and Adaptive agents. The God agents are responsible for establishing connections to IoT devices. The adaptive agent represent the Self-Adaptive System. The observer agents observe the adaptations that are made by the adaptive agents and change their behavior.

- Location: FIoT optimizes three different levels: The physical, communication and application layer.
- Time: FIoT performs its optimizations during the run time of the system.
- Technique: FIoT changes the Level, Adaptation Technique and Rules / Policies of a SAS.

- Purpose: Generate self-organizing behavior in a SAS.
- Approach: FIoT uses a decentralized agent-based approach.
- Reflective Adaptation Control: The observer agents change the behavior of the adaptive agents. This is equivalent to modifying rules or policies.

5 Conclusion

This paper proposed a classification for Optimization Approaches for Self-Adaptive Systems. The classification was derived from the same principles as the taxonomy for Self-Adaptive Systems by Krupitzer et al., 2015 [6]. The main principles that were used to derive the classification are:

- The 5W+1H questions by Salehie and Tahvildari, 2009 [7].
- The MAPE-K feedback loop which was created by Kephart and Chess, 2003 [5].

To classify OA the proposed classification provides six dimensions: Location, Time, Purpose, Approach, Technique and Reflective Adaptation Control. These are shown in Figure 6.
The Location dimension asks on which level optimizations in a Self-Adaptive System might be necessary. Time distinguishes between OA that are performed during the design time of the system, during the run time or online phase of the system or approaches that optimize systems during their offline phase. The dimension of Purpose provides a reason for Optimizations to be performed. Approach determines who is responsible for optimizations and Technique states which parts of a SAS can be optimized. Lastly, the Reflective Adaptation Control classifies how optimizations are performed.
After proposing a classification, it was exemplarily applied to three existing OA. This is a part where further research is required. The classification has to be applied to further existing approaches to verify its validity and completeness.
By applying the classification to more approaches, one could also gain a structured overview over existing approaches. This can be used to identify approaches that have not been explored yet.

References

[1] Naeem Esfahani Ahmed Elkhodary and Sam Malek. "FUSION: A Framework for Engineering Self-Tuning Self-Adaptive Software Systems". In: *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE '10. Santa Fe, New Mexico, USA: Association for Computing Machinery, 2010, pp. 7–16. ISBN: 9781605587912. DOI: 10.1145/1882291.1882296.

[2] Andrew Berns and Sukumar Ghosh. "Dissecting Self-* Properties". In: *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. 2009, pp. 10–19. DOI: 10.1109/SASO.2009.25.

[3] Sam Malek Danny Weyns and Jesper Andersson. "FORMS: Unifying Reference Model for Formal Specification of Distributed Self-Adaptive Systems". In: *ACM Trans. Auton. Adapt. Syst.* 7.1 (May 2012). ISSN: 1556-4665. DOI: 10.1145/2168260.2168268.

[4] Nathalia Moraes do Nascimento and Carlos José Pereira de Lucena. "FIoT: An agent-based framework for self-adaptive and self-organizing applications based on the Internet of Things". In: *Information Sciences* 378 (2017), pp. 161–176. ISSN: 0020-0255. DOI: 10.1016/j.ins.2016.10.031.

[5] J.O. Kephart and D.M. Chess. "The vision of autonomic computing". In: *Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055.

[6] Christian Krupitzer et al. "A survey on engineering approaches for self-adaptive systems". In: *Pervasive and Mobile Computing* 17 (2015). 10 years of Pervasive Computing' In Honor of Chatschik Bisdikian, pp. 184–206. ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2014.09.009.

[7] Mazeiar Salehie and Ladan Tahvildari. "Self-Adaptive Software: Landscape and Research Challenges". In: *ACM Trans. Auton. Adapt. Syst.* 4.2 (May 2009). ISSN: 1556-4665. DOI: 10.1145/1516533.1516538.

[8] Daniel Sykes et al. "Learning revised models for planning in adaptive systems". In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 63–71. DOI: 10.1109/ICSE.2013.6606552.