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 (SAS) 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 environment. This can be solved by using Optimization Approaches (OA) for SAS which can optimize these systems. There are already many OA, but no way to classify them. This paper proposes a classification for OA for SAS.

---

1 Introduction

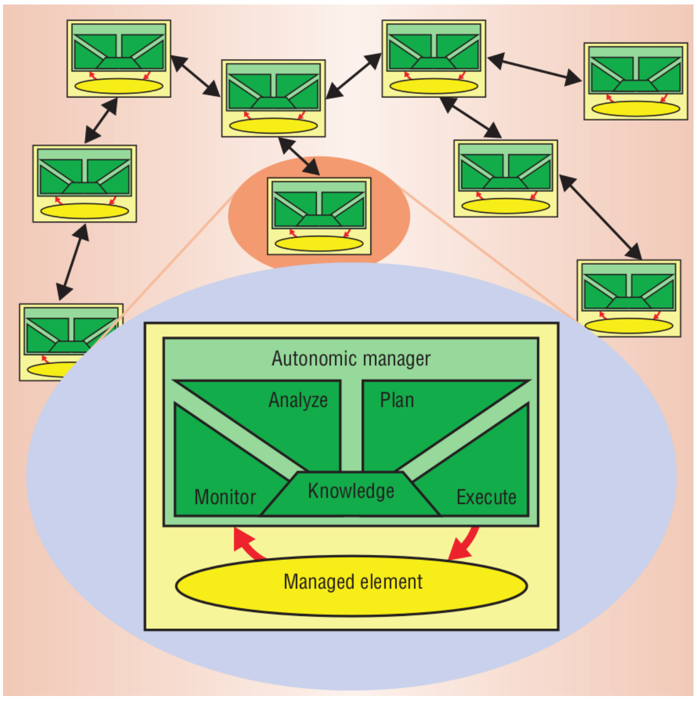


Figure 1: The MAPE-K (Monitor-Analyze-Plan-Execute with Knowledge) feedback [10]

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" [10]. This has reached a state where the "complexity appears to be approaching the limits of human capability" [10]. 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.
To model these abilities Kephart and Chess developed the Monitor, Analyze, Plan, Execute with Knowledge (MAPE-K) feedback loop [10] 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. This feedback loop can be used in a centralized way where one *Autonomic manager* manages the complete system. It can also be used in decentralized way with one *Autonomic manager* per system

component. In this case the different *Autonomic managers* can communicate with each other to exchange information about the system.
Software systems that can autonomously manage themselves are called self-adaptive because of their ability to adapt themselves.
While SAS are better at handling more complex systems, human operators are better at handling uncertainty. This is because SAS can adapt the software that they are managing, but they can not adapt themselves, because the adaptation rules and policies used by SAS are statically created at design time. 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.
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.
While there are already many 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 OA for SAS. The proposed classification will then be used to compare existing OA.
Chapter 2 provides some foundational knowledge about SAS. To derive a classification for OA for SAS, chapter 3 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 4. In chapter 5 the proposed classification will be applied to some existing OA. Lastly, chapter 6 will finish with a conclusion and recommendations for future research directions.

2 Foundations

In essence SAS are systems which change themselves and therefore their own behavior according to predefined rules and policies when they detect that predefined conditions are met or states have been reached.
To better understand SAS, let us take a look at two examples.

Example 1 Imagine you are managing an online store. Your store offers multiple services. Customers can browse your selection and buy your products. In addition to that customers can select products for which they want to receive a notification when they drop to a certain price. Business customers have the option of receiving their bill via Email. Lastly you manage your own advertisements.
Based on these services there are some system parameters that you can control. You can change which products and advertisements are served to a customer. Based on how customers set price alerts you can change the price of a product to potentially increase

sales. Lastly you can change how often your system sends Emails to decrease the load of your Email server.
While you could perform all these tasks on your own, it would be more efficient to automate them. For this purpose you choose to implement a SAS. This SAS automatically adapts the price of products based on sales performance metrics that you define. It also chooses which products and advertisements to serve to a customer based on their purchase history and marketing policies that you create. Additionally the SAS controls the frequency of sending Emails by monitoring the load of the Email server and a goal that you selected.
In summary, this SAS monitors itself and its environment and then adapts itself according to rules and policies that you defined.

Example 2 Imagine you are developing the software for a robot. The robot should be able to navigate through partially known terrain, collect data based on some metrics and be able to update its software when a new version is released.
To navigate through terrain that is only partially known, the robot has to be able to adapt its movement to an environment which it has not experienced before. To collect data based on some performance metrics, the robot needs to change its method of data collection when the selected metrics change. Lastly, to update its software autonomously, the robots needs to be able to modify its software and reconfigure itself.
All these tasks can be implemented as a SAS. Firstly, the robot might use a system that selects which strategy should be used for navigation based on images of its environment. You could then define a set of different strategies for the robot's navigation. For example, one strategy for steep terrain and one for terrain with an even ground.
For the data collection you could define different policies that decide which data the robot should collect if it detects that a metric has reached a threshold. An example could be that if the robot is certain that it detected a human with its camera, that it should not store images of that person to avoid privacy conflicts.
The self-updating behavior only requires the monitoring of one environment attribute: If a new version of the robot's software has been released. When this is the case, you define a policy that the robot should stop moving, download the new software and adapt itself.
These two examples illustrate how different systems can use self-adaptiveness. In most cases it is enough to define simple rules and policies that should be executed when some condition is met.

3 Classification of Self-Adaptive Systems

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

- FORMS [4]
- Berns' and Ghosh's definition of self-* properties [9]
- Krupitzer's et al. taxonomy for SAS [2]

These approaches contain ideas that will be used to construct the classification for OA for SAS in the next chapter.

With FORMS Weyns et al. propose a formal reference model for describing SAS [4]. The goal of FORMS is to provide a well defined basis for talking and reasoning about SAS. This is achieved by constructing SAS from a set of primitives which are defined in 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.

Figure 2 shows all the primitives used by FORMS and their relationships. The SAS is split into different layers of subsystems. Base-level refers to subsystems on the bottom most layer of the system. Subsystems in layers above the base-level layer are called reflective. Reflection refers to the ability of a system to inspect and change itself.

Each subsystem consists of two parts: its computation and its model. Base-level subsystems have domain models which contain data about the environment. In addition to that, reflective subsystems also have a reflection model which contains data about the system itself. While base-level subsystems can use their computation to affect the environment, reflective subsystems can affect the subsystems in the layer below themselves.

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

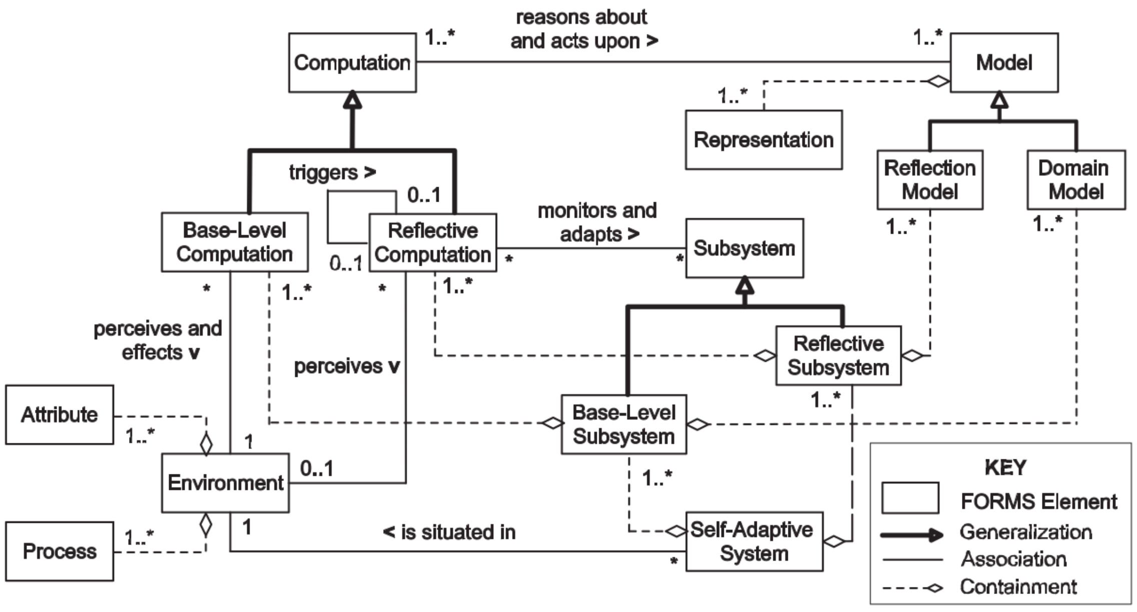
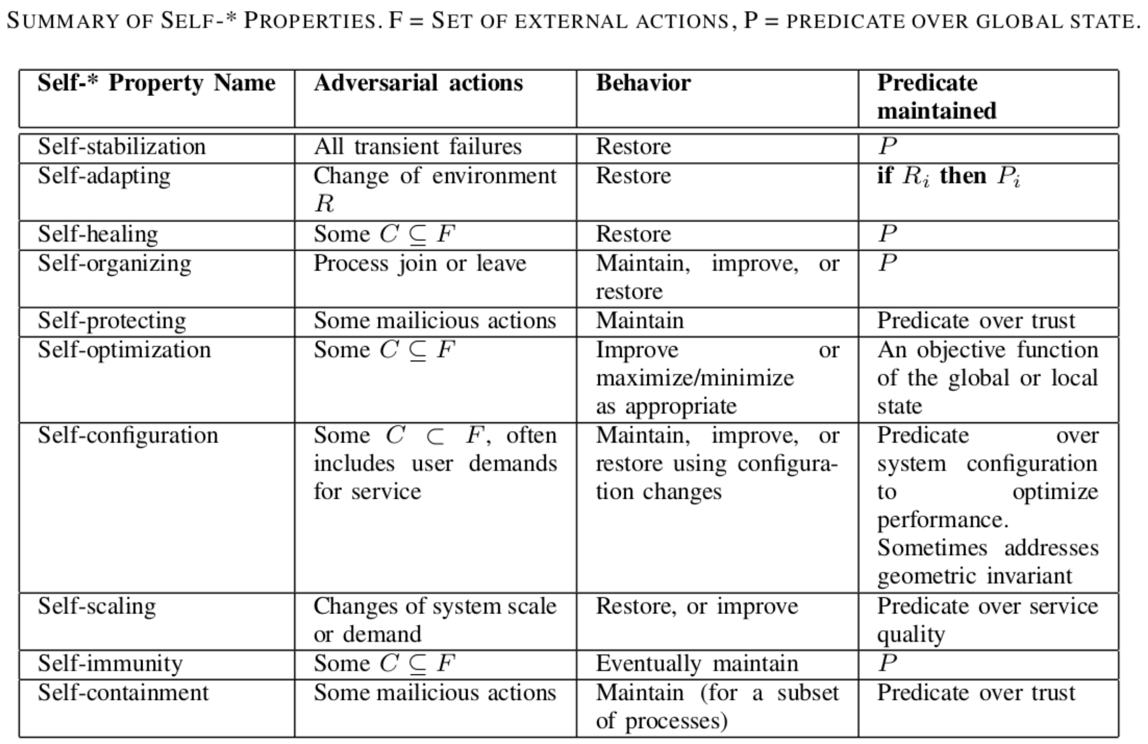


Figure 2: FORMS primitives [4]


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 [9]

robot would be placed in the base-level layers. The sensors would be placed in the base-level layer as well. These components are responsible for observing the environment and interacting with it. On top of the base-level 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.

The concept of different layers of reflection will be useful later to distinguish OA from SAS.

Although some of the first papers on SAS, for example, "The Vision of Autonomic Computing" by Kephart and Chess [10] focused on self-adaptation, there are other aspects of software systems that can benefit from the ideas introduced by SAS. Berns and Ghosh identified and described such aspects which they call self-* properties [9]. They found the self-* properties that are depicted in Figure 3.

Berns and Ghosh describe SAS in terms of their interaction with their environment and their reaction to external actions [9]. External actions are actions stemming from the systems environment which affect the system directly. Safety predicates are used to describe goals of the system. These external actions are called malicious if the intent of the action is the violation of a safety predicate. A system also has a configuration which is the collection of its parameters. Using these definitions, Berns and Ghosh describe their self-* properties in the following way:

Self-stabilization A system has the self-stabilization property if it can get from any starting configuration to a configuration in which its safety predicates are fulfilled and stay in such a configuration afterwards. In other words: The system can complete its starting procedure without requiring assistance.

Self-adapting A system is self-adapting (or managing) if it maintains, improves or restores safety predicates without human intervention. This definition matches the definition proposed by Kephart and Chess [10] of systems which can autonomously manage themselves without human intervention.

Self-healing A system is self-healing if an external action can only cause a temporary violation of safety predicates. This means that a self-healing system can recover from external actions on its own. An example would be a system which can restart itself after a power outage.

Self-organizing A system is self-organizing if it maintains, improves or restores safety predicates after an external action which involves processes joining or leaving the system. The recovery time per join or leave should be sublinear. Peer-to-peer networks can be examples for self-organizing systems.

Self-protecting A system is self-protecting if it maintains one chosen safety predicate even when malicious external actions occur. Meaning that a self-protecting system guarantees that it will never violate the chosen safety predicate. A real world example for this would be a system which guarantees that it will not disclose personal data to unauthorized users.

Self-optimization When a system is self-optimizing, it can maximize or minimize the value of a utility function. A system which starts and stops services based on demand to reduce energy usage is an example for a self-optimizing system.

Self-configuration A system possesses the self-configuration property if it can change its configuration to restore or improve a safety predicate.

Self-scaling A self-scaling system can maintain or improve a system property while an external action affects its scale. This means that a system which regulates the number of instances of one of its services based, for example, on demand is self-scaling.

Self-immunity A system with self-immunity can restore safety predicates after the occurrence of external actions and return to a state where no safety predicate is violated as if the external action never happened.

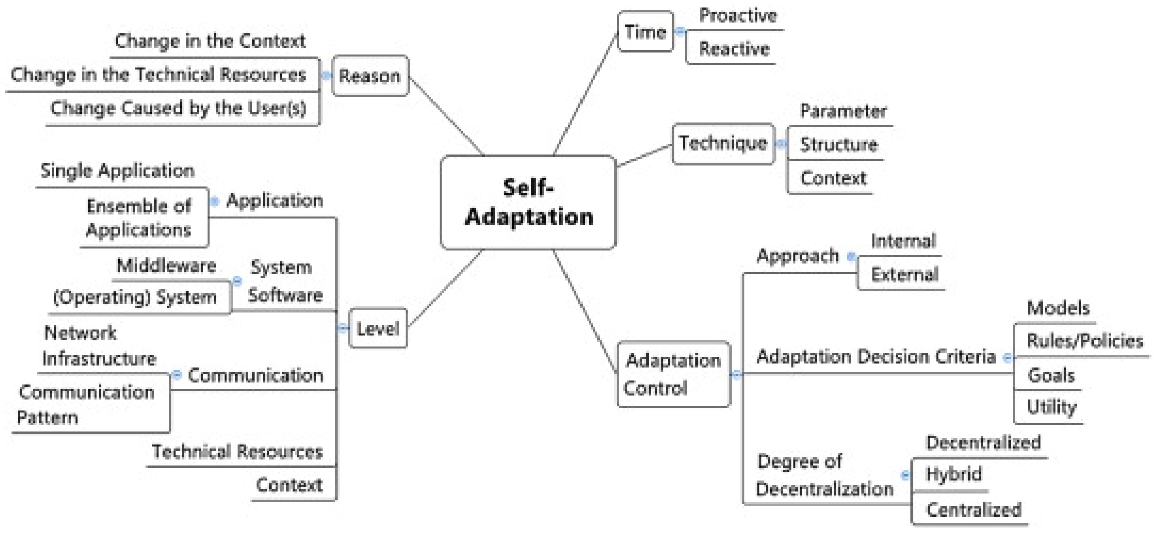


Figure 4: Taxonomy for SAS [2]

Self-containment A system is self-containing if a malicious external action can not compromise the whole system and the system is able to eventually return to its normal operating state.
These self-* properties are useful to:

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

They show that there are aspects besides the management of software which can benefit from the ideas of self-adaptation. This can be used as it distinguishes SAS from system with other self-* properties. The only other self-* property that will be important for the rest of this paper is the self-optimization property.
The taxonomy for SAS in Figure 4 is based upon the 5W+1H questions by Salehie and Tahvildari [14]. 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.

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)" [2].

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 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.

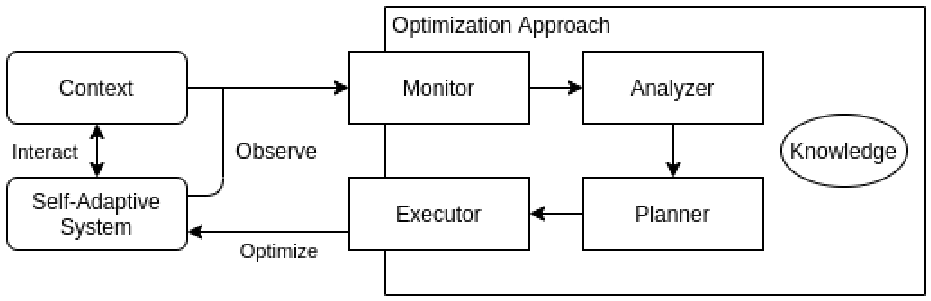


Figure 5: Mapping the optimization process to the MAPE-K feedback loop.

4 Proposal for Classification of Optimization Approaches

This chapter will derive and propose a classification for OA for SAS based on concepts that were discussed in chapter 3. 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 SAS operates
- 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

Figure 5 shows how an OA for SAS can be understood as an additional reflective layer above the SAS. Just like a SAS, the OA has to monitor its environment or context and its subject. In contrast to a SAS the OA observes a SAS whose top level is a reflective layer and not a system whose top level is a base-level layer. Similarly to the SAS the OA has to analyze the data gathered by the *Monitor* which is done by the *Analyzer*. After this the *Planner* can plan an optimization which is then performed by the *Executor*. This optimization process happens with *Knowledge* of the SAS and its environment.

This means that with the concept of layers of reflectiveness from FORMS [4] and the MAPE-K feedback loop [10], an OA for a SAS can be understood as an SAS of a SAS. Following this logic, a classification for OA should be able to answer the same questions as a classification for SAS. The main classification for SAS that this paper will focus on is the taxonomy for SAS by Krupitzer et al. [2]. Because of this there are six questions that the classification for OA has to be able to answer. These are the 5W+1H questions by Salehie and Tahvildari [14]: 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 (RAC). 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 3. These are:

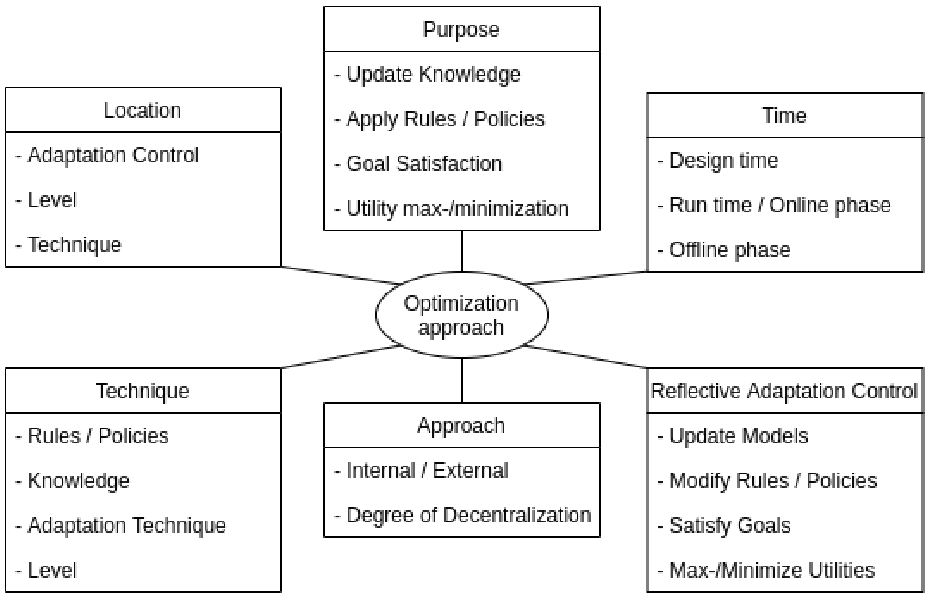


Figure 6: The proposed classification for OA for SAS

- 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
- During the system's *offline phase*

The offline 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. [4], which uses reflective operations to change components in the system, the Adaptation Control of the OA will be called RAC. The RAC 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. This reflects the self-optimization property [9].

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.
The *Location* determines where optimizations are necessary which requires both monitoring and analyzing. The *Time* questions ask when optimizations should be performed. This

information is required by both the executing and monitoring step as it also determines when data from monitoring is required. The *Technique* determines what gets optimized which is the planning step. The *Purpose* determines how data is analyzed as it provides answers to the question why optimizations should be performed. Who is responsible for the execution step is answered by the *Approach* dimension. The *RAC* decides how optimizations are performed which is equivalent to the planning step of the MAPE-K feedback loop. These relations between an OA, the MAPE-K feedback loop and the 5W+1H questions are depicted in Figure 1.

OA	**MAPE-K**	**5W+1H questions**
Location	Monitor, Analyze	Where
Time	Monitor, Execute	When
Technique	Plan	What
Purpose	Analyze	Why
Approach	Execute	Who
RAC	Plan	How

Table 1: How the MAPE-K feedback loop [10] and the 5W+1H questions [14] relate to the dimensions of the classification for OA for SAS.

5 Classifying Existing Optimization Approaches

This chapter will classify and compare existing OA for SAS using the proposed classification. The comparison can be found in Table 2.
Most of the following existing OA were found through a literature study conducted by Saputri and Lee [15]. The goal of the literature study was to give an overview over the application of machine learning in SAS. Because of this it is important to acknowledge that they all use some form of machine learning or similar technique to perform their optimizations. Generally most currently existing use machine learning which is why the conclusions drawn from this comparison should not be influenced by the fact that they all use machine learning to perform optimizations.
The first conclusion that can be drawn from this comparison of OA is that most approaches only use the run time of the system. This is interesting because most of these approaches use machine learning to perform optimizations as mentioned previously. Most machine learning algorithms require trainings time before working effectively. The consequence of directly using an untrained system is that the system will behave in an unexpected or undesired manner in the beginning. This is especially true for approaches that actively explore the problem space to find optimal solutions like Q-learning [17].
However there is a tradeoff that these OA have to deal with. On one hand the system will behave in an unexpected or undesired manner, on the other hand there is no need for simulating the system to train the OA. Simulating the system for training purposes can be difficult as it also requires the simulation of the systems environment. If the developers of a SAS could predict the environment perfectly, they would not need an OA because

Reference	Location	Purpose	Time	Technique	Approach	RAC
[1]	A	R	D, R	R, K	E - C	M, R
[3]	A	R, M	R	R, K	I - C	R
[13]	L	R	R	L, A, R	E - D	R
[16]	T	K, U	R	K	I - D	M, U
[7]	A, L	U	R	A, L	E - D	U
[12]	A	K, U	R	R, K	I - C	M, R, U
[5]	A, L	K, R	R, O	R, K	E - C	M, R
[6]	A	K, G	D	K	E - C	M, G
[8]	A	U	R	R	E - C	R, U
[11]	A	U	R	R	I - C	R, U

Table 2: **Location**: **A** = Adaptation Control, **L** = Level, **T** = Technique
Purpose: **K** = Update Knowledge, **R** = Apply Rules / Policies,
G = Goal Satisfaction, **U** = Utility max-/minimization
Time: **D** = Design time, **R** = Run time / Online phase, **O** = Offline phase
Technique: **R** = Rules / Policies, **K** = Knowledge,
A = Adaptation Technique, **L** = Level
Approach: **I** = Internal actor, **E** = External actor,
C = Centralized, **D** = Decentralized, **H** = Hybrid
RAC: **M** = Update Models, **R** = Modify Rules / Policies,
G = Satisfy Goals, **U** = Max-/Minimize Utilities

they could just define adaptation rules and policies that are suited to all ways in which the environment may develop. This means that training the OA before the systems runtime can decrease the time that it takes until the system behaves in an expected manner, but it also can not guarantee that the system will be perfectly trained.
The second conclusion that can be drawn is that most OA only search for problems in the Adaptation Control. But the *Location* dimension offers other possibilities as well. In addition to the Adaptation Control, the Level or the Technique may require optimizations.
The third conclusion that can be drawn is that only a few OA use decentralized approaches. Using a OA with a decentralized approach can have the advantage that each area can be optimized specifically. As Nascimento and Lucena showed, decentralized approaches can also be used to create self-organizing systems [13]. But it also has the disadvantage of increasing the complexity of the OA.

6 Conclusion

While there are different approaches to classify and reason about SAS, three of which were highlighted by this paper, there is no classification for OA for SAS. Because of this it is hard to compare different OA and identify areas which require further research. To solve this problem this paper focused on proposing a classification for OA for SAS.
To accomplish this, chapter 2 started by explaining how SAS work with two short examples and also showed some of their limitations. This was followed by chapter 3 which explored three different approaches for classifying and reasoning about SAS. Chapter 4 then derived and proposed a classification for OA based on the principles shown in the previous chapter. Lastly, chapter 5 compared a selection of existing OA by using the proposed classification. The comparison was then used to draw some initial conclusions about OA.
The first conclusion was that most OA only use the systems runtime to perform optimizations. This has the advantage of not needing to simulate the system and its environment which can be complex. But because most OA use machine learning for their optimizations, this can lead to unexpected or unwanted behavior at the beginning of the systems runtime. This disadvantage is especially present in OA that use machine learning approaches like Q-learning which actively explores the problem space to find optimal solutions.
The second conclusion was that only a few OA search for problems which require optimization in areas of a SAS besides the Adaptation Control. The other two areas that can be optimized: the Level and the Technique still require research into how effective their optimization can be.
The last conclusion was that most OA use centralized approaches for performing optimizations. Because of this the usage of decentralized approaches still requires further research.
Even though the proposed classification helps with comparing the existing OA, it does not help with comparing their performance. This is a problem not only for OA but for SAS as well. There is currently no scientific way of comparing the performance of SAS. Because of this the effect of OA on SAS can also not be quantified.
Another problem that SAS and their OA have is that most of them are highly domain specific and only try to solve a single domain problem. This means that domain independent SAS and OA for them still require research.

References

[1] Ahmed Elkhodary et al. “FUSION: a framework for engineering self-tuning self-adaptive software systems”. In: *FSE ’10: Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. 2010, pp. 7–16. DOI: 10.1145/1882291.1882296.

[2] Christian Krupitzer et al. “A survey on engineering approaches for self-adaptive systems”. In: *Pervasive and Mobile Computing* 17.B (2015), pp. 184–206. DOI: 10.1016/j.pmcj.2014.09.009.

[3] 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.

[4] Danny Weyns et al. “FORMS: Unifying reference model for formal specification of distributed self-adaptive systems”. In: *ACM Transactions on Autonomous and Adaptive Systems* 7.1 (2012), pp. 1–61. DOI: 10.1145/2168260.2168268.

[5] Giljong Yoo et al. “Hybrid Prediction Model for improving Reliability in Self-Healing System”. In: *Fourth International Conference on Software Engineering Research, Management and Applications (SERA’06)*. 2006, pp. 108–116. DOI: 10.1109/SERA.2006.40.

[6] Heather Goldsby et al. “Digitally Evolving Models for Dynamically Adaptive Systems”. In: *International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS ’07)*. 2007, p. 13. DOI: 10.1109/SEAMS.2007.6.

[7] Iara Augustin et al. “ISAM, a software architecture for adaptive and distributed mobile applications”. In: *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*. 2002, pp. 333–338. DOI: 10.1109/ISCC.2002.1021698.

[8] Mehdi Amoui et al. “Adaptive Action Selection in Autonomic Software Using Reinforcement Learning”. In: *Fourth International Conference on Autonomic and Autonomous Systems (ICAS’08)*. 2008, pp. 175–181. DOI: 10.1109/ICAS.2008.35.

[9] 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.

[10] Jeffrey Kephart and David Chess. “The vision of autonomic computing”. In: *Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055.

[11] Dongsun Kim and Sooyong Park. “Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software”. In: *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. 2009, pp. 76–85. DOI: 10.1109/SEAMS.2009.5069076.

[12] Elise Langham. “A Fuzzy Evolutionary System for Concept Formation and Adaptive Behaviour in Software Agents”. In: *The 14th IEEE International Conference on Fuzzy Systems, 2005. FUZZ ’05.* 2005, pp. 631–635. DOI: 10.1109/FUZZY.2005.1452467.

[13] Nathalia Moraesdo Nascimento and Carlos José Pereirade 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. DOI: 10.1016/j.ins.2016.10.031.

[14] Mazeiar Salehie and Ladan Tahvildari. "Self-adaptive software: Landscape and research challenges". In: *ACM Transactions on Autonomous and Adaptive Systems* 4.2 (2009), pp. 1–42. DOI: 10.1145/1516533.1516538.

[15] Theresia Ratih Dewi Saputri and Seok-Won Lee. "The Application of Machine Learning in Self-Adaptive Systems: A Systematic Literature Review". In: *IEEE Access* 8 (2020), pp. 205948–205967. DOI: 10.1109/ACCESS.2020.3036037.

[16] Alvaro Soto and Pradeep Khosla. "Adaptive Agent Based System for State Estimation Using Dynamic Multidimensional Information Sources". In: *Self-Adaptive Software: Applications*. Springer, Berlin, Heidelberg, 2003, pp. 66–83. DOI: 10.1007/3-540-36554-0_6.

[17] Christopher Watkins. "Learning from delayed rewards." University of Cambridge, 1989.