Optimization Approaches for Self-Adaptive Systems

Tim Engbrocks

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

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 [3]).
This has reached a state where the "complexity appears to be approaching the limits of human capability" (Kephart and Chess, 2003 [3]). 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 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.
- decide how to react to faults and changes in the systems environment.
- make changes to itself.

To model these abilities Kephart and Chess developed the MAPE-K (Monitor-Analyze-Plan-Execute with Knowledge) feedback loop [3].
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 planning step plans which changes have to be made. 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 because of their ability to adapt themselves.

This approach to managing software systems has advantages compared to the use of human operators:

-

While Self-Adaptive Systems are better at handling more complex systems, human operators are better at handling uncertainty.
This is because the rules and policies, used by Self-Adaptive Systems, are created at design time. The consequence of this is, that Self-Adaptive Systems can adapt themselves under a changing environment, but they can not adapt the rules and policies that are used for adaptation. In combination with the fact that not all environment changes can be predicted during design time,

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 paper are:

- FORMS from Weyns et al., 2012 [7]
- Raibulet's, 2018 taxonomy for self-* properties [5]
- Krupitzer's et al., 2015 taxonomy for Self-Adaptive Systems [4]

In their 2012 paper FORMS [7] Weyns et al. propose a formal reference model for describing Self-Adaptive Systems. The goal of FORMS is to provide a well defined basis for talking and reasoning about Self-Adaptive Systems. This is achieved by providing a definition of FORMS, which classifies Self-Adaptive Systems, using Z notation.

By composing a Self-Adaptive System from different levels of components called subsystems, where each level can adapt the level beneath, FORMS models the self-adaptive process using reflection. The bottom level is populated by base-level computations, base-level subsystems and domain models. On top of these base-level components are reflective computations, reflective subsystems and reflection models.

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 1: self-* properties by Berns and Ghosh, 2009 [1]

While some of the first ideas for Self-Adaptive Systems focused on self-adaptation, like "The Vision of Autonomic Computing" by Kephart and Chess [3], there are other

aspects of software systems that can benefit from the ideas of Self-Adaptive Systems. These aspects are called self-* properties. Berns and Ghosh, 2009 [1] identified and described the self-properties in Figure 1. These self-* properties can be useful to quickly communicate the abilities and goals of a Self-Adaptive System.

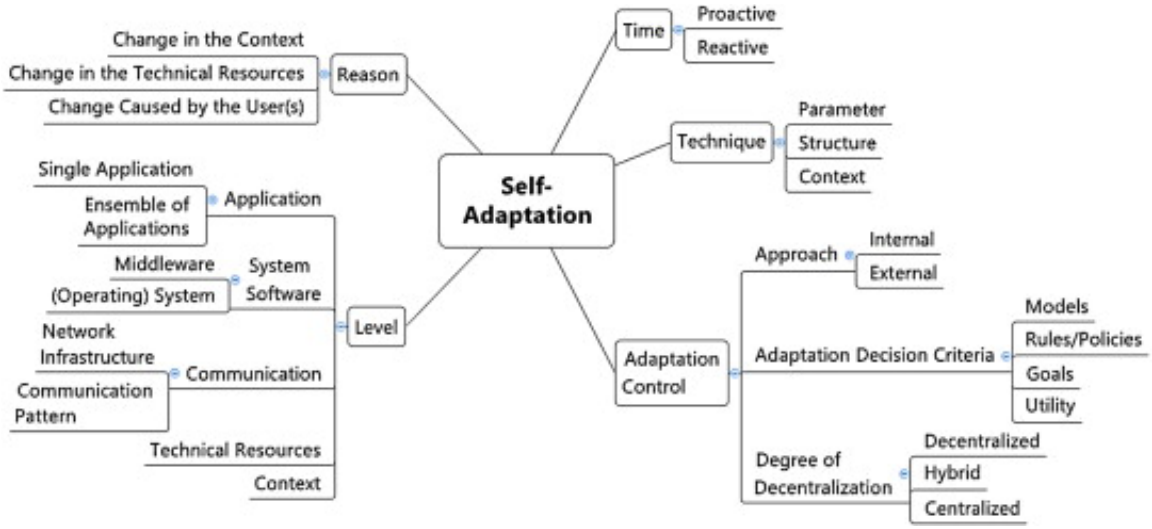

Figure 2: Taxonomy for Self-Adaptive Software by Krupitzer et al., 2015 [4]

The taxonomy for Self-Adaptive Systems by Krupitzer's et al., 2015 [4] in Figure 2 is based upon the 5W+1H questions by Salehie and Tahvildari, 2009 [6]. These questions are: Where, When, What, Why, Who and How. Each of these questions is responsible for a different aspect of Self-Adaptive Systems and corresponds to a dimension of the taxonomy.

First there needs to be a reason for a Self-Adaptive System 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.

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.

While the original When-Question by Salehie and Tahvildari tries to understand all temporal aspects of Self-Adaptive Systems, 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.

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.

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 [4]).

Lastly, after determining the where, when, what, why and by 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 factors of the Adaptation Control are the degree of decentralization, the adaptation decision criteria and the approach taken by the system, which divides Self-Adaptive Systems into those where the adaptation logic is part of the application logic and those with separated adaptation and application logic.

After classifying Self-Adaptive Systems. The following question can be asked: Which parts of a Self-Adaptive System can be optimized? To answer this we will start by looking at which parts can not be optimized.

The first part that can not be optimized is the environment which provides the Reason dimension. While the environment for a Self-Adaptive System can be choosen in a way which is most beneficial for the system and can be influenced by actors, the behaviour of the systems environment can generally not be controlled completly.

Another dimension of Self-Adaptive Systems that can not be optimized, or is not useful to optimize, is the Time dimension. This dimension is mostly 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 degredation 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 different or opposite to the currently needed change. 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 also predict that change.

The Level dimension is mostly a design decision?

Lastly the question of who is responsible can not be optimized because it is trivially answered b

The two remaining dimensions can be optimized. These are the Adaptation Control 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. But 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.

3 Proposal for classification of optimization approaches

When thinking about how to classify optimization approaches for Self-Adaptive Systems, one must realize that an optimization approach for Self-Adaptive Systems is really just a Self-Adaptive System of a Self-Adaptive System.

Because of this the proposed classification for optimization approaches for Self-Adaptive Systems should be based upon the same principles as the classification for Self-Adaptive Systems and it should use the same terminology.

Following Krupitzer's et al, 2015[4] taxonomy for Self-Adaptive Software, a classification for Optimization Approach of SAS should answer the 5W+1H questions by Salehie and Tahvildari, 2009[6]. These questions are:

- Where is the need for change?
- When should a change occur?
- What should be changed?
- Why should something be changed?
- Who should change something?
- How should something be changed?

The process of optimizing Self-Adaptive Software can be split into multiple parts using the MAPE-K (Monitor-Analyze-Plan-Execute with Knowledge) feedback loop by Kephart and Chess, 2003[3]. Each part of the MAPE-K feedback loop can be mapped to the process of optimization in the following way:

- Firstly the optimization approach has to constantly monitor the Self-Adaptive System and its context.
- The data gathered from monitoring can then be analyzed to decided 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.
- For all of this the optimization approach requires knowledge about the Self-Adaptive System and its context.

Location Where in the system is an optimization necessary?

- Adaptation control
- Level
- Technique
- Time

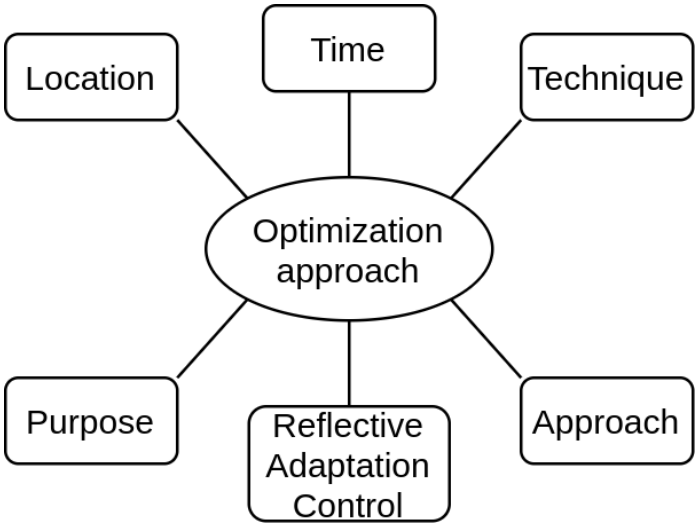

Figure 3: The proposed classification for Optimization Approaches for Self-Adaptive Systems

Time When are optimizations performed?
Similarly to SAS, Optimization Approaches can be differentiated by comparing when they perfom optimizations. There are three different phases during the lifetime of a Self-Adaptive System where optimizations can occur. These are:

- at the runtime of the system
- during the design time of the system
- while training the system

The training phase can happen in parallel to the run and design time of the system. Examples for this would be when the inital parameters for a Self-Adaptive System are choosen 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?

- —

Purpose Why should an optimization occur?
The most important question for any optimization is: What influences the need for an optimization. Just like with Self-Adaptive Systems this can be caused by changes in context, the system itself or updated requirements for the system. Additionally an optimization might be necessary if for example the system detects that its current adaptation control can be improved.

- Changes in context, the system itself or requirements for the system.
- Detection of suboptimal adaptation logic.

Approach Who is responsible for performing the optimization?

- An internal component of the system
- An external actor

Reflective Adaptation Control How is the optimization applied to the system?
Just like the Self-Adaptive System can be described by its Adaptation Control, which is responsible for applying changes to the system, an optimization approach can also be described by how it applies its optimizations to the system. In reference to FORMS by Weyns et al., 2012[7], which uses reflective operations to change components in the system, the Adaptation Control of the optimization approach will be called Reflective Adaptation Control.

- —

MAPE Mapping

MAPE	OA
Monitor	
Analyze	
Plan	Technique
Execute	Reflective Adaptation Control

5W+1H Mapping

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

4 Classifying existing optimization approaches

FUSION by Elkhodary et al, 2010 [2] 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: Adaptation Control + Technique ?
- 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:
- Purpose: The goal of FUSION is to
- Approach: FUSION acts as an external actor.
- Reflective Adaptation Control:

FIoT by

- Location:
- Time:
- Technique:
- Purpose:
- Approach:
- Reflective Adaptation Control:

FUSION

- Location:
- Time:
- Technique:
- Purpose:
- Approach:
- Reflective Adaptation Control:

FUSION

- Location:
- Time:
- Technique:
- Purpose:
- Approach:
- Reflective Adaptation Control:

FUSION

- Location:
- Time:
- Technique:
- Purpose:
- Approach:
- Reflective Adaptation Control:

5 Conclusion

References

[1] 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.

[2] Ahmed Elkhodary, Naeem Esfahani, 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. URL: https://doi.org/10.1145/1882291.1882296.

[3] 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.

[4] 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: https://doi.org/10.1016/j.pmcj.2014.09.009. URL: https://www.sciencedirect.com/science/article/pii/S157411921400162X.

[5] Claudia Raibulet. "Towards a Taxonomy for the Evaluation of Self-* Software". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2018, pp. 22–23. DOI: 10.1109/FAS-W.2018.00020.

[6] 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. URL: https://doi.org/10.1145/1516533.1516538.

[7] Danny Weyns, Sam Malek, 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. URL: https://doi.org/10.1145/2168260.2168268.