A Vision for Explainability of Coordinated and Conflicting Adaptions in Self-Adaptive Systems

Sandro Speth[1], Sarah Stieß[1], and Steffen Becker[1]

Institute of Software Engineering, University of Stuttgart, Stuttgart, Germany
{sandro.speth,sarah.stiess,steffen.becker}@iste.uni-stuttgart.de

Abstract. In the microservice domain, self-adaptive systems exist that reconfigure themselves to adhere to their guaranteed quality of service in the face of a changing environment. Constant changes in the environment enforce continuous adaptations of the system. Especially, different and potentially conflicting adaptations might interact, making it challenging to explain the decision and rationale behind the overall reconfiguration. In this paper, we discuss different approaches for the explainability of self-adaptive systems. Furthermore, we propose our approach to achieve a good trade-off between explainability and its performance impact for the mandatory data gathering. The approach encompasses eliciting requirements regarding explanations and their representations and experimenting on reference architectures for insights into the data required to fulfil the requirements.

Keywords: Microservices · Self-adaption · Explainability.

1 Introduction

Modern Cloud-native applications increasingly consist of self-adaptive microservice systems to better cope with constant changes in the environment and demands [10]. To achieve a better overall resilience, services adapt themselves through reconfigurations of their architecture, e.g., by scaling or by replacing entire failed services [1, 4]. Therefore, a self-adaptive system must monitor and analyze its current state, plan on which adaptions to take, if any required, and execute these actions without human intervention [6].

Due to the varying amount of users in the cloud, the workload changes constantly and enforces continuous adaptions, which may, either accidentally or on purpose, happen simultaneously and, therefore, influence or even conflict with each other. As an example, for two dependent services that are part of a larger architecture, scaling out the consuming service increases the incoming load of the consumed one, causing the consumed one to scale as well. Regarding conflicts, a service might define adaptation rules based on different metrics, e.g., response time and CPU load. In case the response time increases while the CPU load does not, e.g., if the increased response time is caused by waiting for another service, the response time rule triggers a scale out, followed by the CPU load rule trying to scale back in. In these examples, the behaviour deviates from the

J. Manner, D. Lübke, S. Haarmann, S. Kolb, N. Herzberg, O. Kopp (Eds.): 14th ZEUS Workshop, ZEUS 2022, Bamberg, held virtually due to Covid-19 pandemic, Germany, 24–25 February 2022, published at http://ceur-ws.org

expected or is sub-optimal. To comprehend the behaviour and to improve and fix the system, a DevOps engineer must understand the rationale behind the performed self-adaptations [12], especially if they are frequently recurring [13]. However, the interactions between potentially conflicting adaptations are challenging to understand, especially if the adaptation rules are more elaborated. Consequently, the need to explain the interactions between various adaptations arises. To create an explanation that DevOps engineers easily understand, we must identify which information the explanation should contain. Furthermore, for the sake of performance, we must find a reasonable trade-off between the amount of gathered data and the granularity of the explanation. This leads to our problem statements:

Problem 1. What is mandatory information to explain the coordination of and the interactions between multiple, perhaps conflicting reconfigurations and their impact on the system's overall adaptation behaviour?

Problem 2. How can we obtain the components of such an explanation while keeping a trade-off between the quality of the explanation and the performance impact of the data gathering?

2 Related Work

Explainability is becoming increasingly popular and essential in many research fields as it allows developers to understand systems more efficiently [5, 12]. In the context of cyber-physical systems, Bohlender et al. [3] characterise an explanation as a collection of information that has a target group and a subject and improves the target groups' understanding of the subject [3]. As the usefulness of the explanation depends on the target group, this endorses the importance of our first problem.

Klös et al. [8, 9] consider the explainability of self-learning self-adaptive systems. Their system adapts based on timed adaption rules and improves them with a genetic learning algorithm. It records various information, such as which condition in the system or environment triggered the adaption, the adaptation's expected effects and its actual effects, and feeds these information into a learning algorithm [9, 8]. Furthermore, they state that the collected information may serve as explanations of the system's adaptions or as a foundation to create further explanations for specific target groups [7]. In contrast to our problem, their initial focus is on explaining the self-learning aspect. In addition, they focus on single rules only instead of coordinated reconfigurations.

Blumreiter et al. [2] propose the reference framework MAB-EX for self-explaining systems. Their framework consists of four steps: (1) *Monitor*, (2) *Analyze*, (3) *Build* and (4) *EXplain*. *Monitor* and *Analyze* are analogous to the steps from the MAPE-K [6] loop. *Build* creates the explanation, and *Explain* transforms the explanation into a representation befitting the receiver and transmits it to the receiver [2]. The last step emphasises the importance of the target group. MAB-EX proposes two realisations for assisted driving systems [2]. In contrast to that, we focus on self-adaptive microservice systems.

3 Proposed Approach

In compliance with Bohlender et al. [3], we define DevOps engineers as the target group for explanations of self-adaptations. Furthermore, we identified three subjects: (1) (non-)application of a single reconfiguration, (2) coordination of reconfigurations, and (3) influences and relations between reconfigurations.

For our first problem statement, we already conducted an expert survey regarding reconfiguration on a Kubernetes cluster and found out that DevOps engineers consider Kubernetes' primarily textual representations and logs challenging to understand and, therefore, preprocessed cognitive effective representations are needed. Next, we plan to conduct an expert survey on DevOps engineers to identify requirements, mandatory information, and suitable representations, e.g. text or visual, interactive or static, which improve the DevOps engineers' understanding of the self-adaptations. We expect that an explanation requires at least information about (1) the components which were adapted, (2) the configuration of the components before the adaption, (3) the time of the adaption, and (4) the environmental change stimuli, e.g., the workload for the affected components triggering the adaption. Based on the elicited requirements, we decide on a fitting representation for the explanations. For example, explanations could be reported as cross-component issues [14] in Gropius [15], as issues are an already well-established natural platform to explain problems. This way, the explanations would be available in the developer's IDE to reduce context-switches [16].

For the second problem statement, we need a reference architecture for self-adaptive systems to evaluate our solution approach. The system is required to execute not only single reconfigurations but multiples in coordination while providing various metrics and data for the explanations. We plan to conduct a literature survey to identify suitable reference architectures, starting with the list provided by Taibi[1]. To monitor environmental change-stimuli to simulate and gain required information to explain adaptions, we plan to instrument OpenAPM [11] solutions. Especially, the monitoring solution should provide data and insights about the system's behaviour after a reconfiguration to assert the correct execution of adaption. However, deciding on the monitored metrics, their level of detail, and how long to preserve the data depends on the requirements collected in problem 1. Finally, we plan to evaluate explanations created from our reference architecture's adaptions for their comprehensibility by performing expert surveys with DevOps engineers as representatives of our target group.

4 Conclusion

Interactions between self-adaptations and potential conflicts between them are difficult to understand. Therefore, the need for explaining the rationale behind such adaptations arises. However, current approaches focus on explaining single adaptations only. Therefore, we propose our ideas of improving the DevOps engineers' understanding of a self-adaptive system by explaining single system

[1] https://github.com/davidetaibi/Microservices_Project_List

reconfiguration decisions as well as coordinated reconfiguration decisions and their influences on and relations with each other. Our ideas include (1) determining the requirements for explanations in self-adaptive systems and (2) how to create a suitable explanation.

References

1. Aderaldo, C.M., et al.: Kubow: An architecture-based self-adaptation service for cloud native applications. In: Proceedings of the 13th European Conference on Software Architecture - Volume 2. p. 42–45. ACM (2019)
2. Blumreiter, M., Greenyer, J., Chiyah Garcia, F.J., Klös, V., Schwammberger, M., Sommer, C., Vogelsang, A., Wortmann, A.: Towards self-explainable cyber-physical systems. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 543–548 (2019)
3. Bohlender, D., Köhl, M.A.: Towards a characterization of explainable systems (2019)
4. Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer 37(10), 46–54 (2004)
5. Greenyer, J., Lochau, M., Vogel, T.: Explainable software for cyber-physical systems (ES4CPS): report from the GI dagstuhl seminar 19023, january 06-11 2019, schloss dagstuhl. CoRR abs/1904.11851 (2019), http://arxiv.org/abs/1904.11851
6. Kephart, J., Chess, D.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
7. Klös, V.: Explainable self-learning self-adaptive systems. In: Explainable Software for Cyber-Physical Systems (ES4CPS): Report from the GI Dagstuhl Seminar 19023, January 06-11 2019, Schloss Dagstuhl. pp. 46–47 (2019)
8. Klös, V., Göthel, T., Glesner, S.: Comprehensible and dependable self-learning self-adaptive systems. Journal of Systems Architecture 85-86, 28–42 (2018)
9. Klös, V., Göthel, T., Glesner, S.: Comprehensible decisions in complex self-adaptive systems. In: Software Engineering und Software Management 2018. pp. 215–216. Gesellschaft für Informatik, Bonn (2018)
10. Newman, S.: Building Microservices: Designing Fine-Grained Systems. O'Reilly, 2nd edn. (2021)
11. Novatec GmbH: Openapm: Landscape for apm tools, obervability tools and monitoring tools, https://openapm.io/landscape
12. Sadeghi, M., Klös, V., Vogelsang, A.: Cases for explainable software systems: Characteristics and examples. In: Proceedings of 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW). pp. 181–187. IEEE (2021)
13. Speth, S.: Semi-automated cross-component issue management and impact analysis. In: Proceedings of 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 1090–1094. IEEE (2021)
14. Speth, S., Becker, S., Breitenbücher, U.: Cross-component issue metamodel and modelling language. In: Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER 2021). pp. 304–311. SciTePress (2021)
15. Speth, S., Breitenbücher, U., Becker, S.: Gropius — a tool for managing cross-component issues. In: Software Architecture. vol. 1269, pp. 82–94. Springer (2020)
16. Speth, S., Krieger, N., Breitenbücher, U., Becker, S.: Gropius-vsc: Ide support for cross-component issue management. In: Companion Proceedings of the 15th European Conference on Software Architecture. CEUR (2021)