Runtime Evolution of the Adaptation Logic in Self-Adaptive Systems

Felix Maximilian Roth, Christian Krupitzer, and Christian Becker
University of Mannheim
Schloss, 68131 Mannheim, Germany
{felix.maximilian.roth, christian.krupitzer, christian.becker}@uni-mannheim.de

***Abstract*—Self-adaptive systems, which are highly related to Autonomic Computing, are a response to the increasing complexity and size of information systems. They are able to adapt their behavior to changes in the environment or system resources. A self-adaptive system consists of managed resources that realize functionality and an adaptation logic that controls the adaptations. So far, many research has been performed on adapting the managed resources. However, only few works cover adapting the adaptation logic, which might be necessary in several cases, e.g., when the architecture of the managed resources changes. This work adresses why adaptation of the adaptation logic might be beneficial, how it can be achieved, and what challenges arise.**

I. Introduction

Due to an increasing complexity and size of information systems, the maintenance effort for such systems gets higher. Self-adaptive systems (SASs), which are highly related to Autonomic Computing, address this issue. A SAS is able to adapt its behavior to changes in the system resources or in the context [1]. It consists of managed resources (MRs), such as hardware or software, and the adaptation logic (AL) or autonomic manager [2]. The MRs realize functionality whereas the AL controls the adaptation and implement changes accordingly. So far, most researchers focus on adapting the MRs, e.g., in order to optimize performance or recover from defects. However, only few work is done in adapting the AL, which means optimizing how the AL decides on modifying the MRs. Also in [3], support for co-evolution of the MRs and the AL is mentioned to be a challenge in SAS. In this abstract, we sketch our ideas on how to adapt the AL.

II. Motivation and Status Quo

Adaptations of the AL might be necessary due to changes in the system resources, the environment, or the system goals. For example, consider an adaptive production cell [4] where the interaction scheme of robots is dynamic. Assuming a master/slave pattern [5], slaves are not able to coordinate if the master crashes. Therefore, a new master has to be selected - an adaptation of the AL is required.

To the best of the authors' knowledge, so far, only few approaches are made in order to adapt the AL at runtime. Iftikhar and Weyns have proposed ActivFORMS [6] where goals of the SAS can be changed at runtime. This is done with the help of a *goal management module* responsible for finding a goal model adaptation for the AL. The goal management module is located on top of the *active model module* which incorporates the AL. Kramer and Magee present a three layer architecture model for self-management [7]. They also focus on modifying the adaptation logic due to goal changes. Elkhodary et al. developed FUSION where the AL is automatically fine-tuned to unforeseen changes in the environment [8]. This is achieved by introducing a learning cycle responsible for inducing and updating the AL's system model to unanticipated conditions. In KAMI, a Bayesian estimator is applied for proactive adaptation in order to satisfy non-functional requirements by updating transition probabilities at runtime [9]. In [10], new classifiers are generated using offline simulation and added to the classifier system in the AL. These approaches only focus on one reason to adapt the adaptation logic. However, there is no approach that combines several reasons for AL adaptation and offers a generic tool. This abstract presents challenges of AL adaptation as well as an approach to encounter them.

III. Goals and Challenges

Goals of adapting the AL can be, e.g., to satisfy a certain quality level (such as quality of service or output quality), to reduce communication overhead (i.e., messages sent between AL components on the same machine or on remote machines) by structural adaptation of the AL in response to a change in the MRs' structure, or to increase performance.

To achieve these goals, we introduce a new component responsible for AL adaptation. We will refer to this component as *adaptation logic manager* (ALM) in the remainder of this abstract. For implementing the ALM, several challenges exist.

Adaptation reason [11]: The ALM must be able to identify the reason why an adaptation is necessary. Reasons might be *explicit* or *implicit*. An explicit reason can be monitored by the ALM whereas an implicit reason results from an adaptation of MRs. *Explicit* reasons can be changes caused by the user (e.g., goal changes), changes in the AL (e.g., component defects), or changes in the context. An *implicit* reason can be an adaptation of MRs which results in a need for adaptation of the AL. For example, in a cloud environment, if a component is replicated, the same or different AL can manage it. In order to detect a need for change, the AL must provide mechanisms that allow the ALM to monitor the AL. Thus, the AL needs to be extended with either components that can send information about itself, or interfaces that allow to request information.

Adaptation techniques [11]: As in classical adaptation, different adaptation techniques can be applied: parameter adaptation or compositional adaptation [12]. Compositional adaptation is concerned with the modification or exchange of structural or algorithmic components. The techniques work respectively in the case of AL adaptation. The ALM needs to decide if a parameter, structure, or algorithm change is required. For reasoning on adaptation different criteria can be applied, such as models, policies, goals, or utility functions [13]. However, choosing an adequate method depends on the use case.

Adaptation control [11]: First, the ALM can be internal - directly intertwined in the AL - or external - separated from the AL. The external approach has the advantage of a more independent ALM and, thus, of higher maintainability [14]. Furthermore, different degrees of (de)centralization are possible: decentralized, hybrid, or centralized. A decentralized approach benefits from high robustness. However, it introduces high coordination overhead between ALMs. In a centralized approach coordination overhead between ALMs becomes redundant whereas a bottleneck might emerge. A hybrid approach is also possible. Interaction patterns, proposed in [5], can also be applied to the adaptation control for the AL. A suitable approach, again, depends on the use case.

IV. Our Approach

This section presents our approach of the ALM. The approach is visualized in Figure 1.

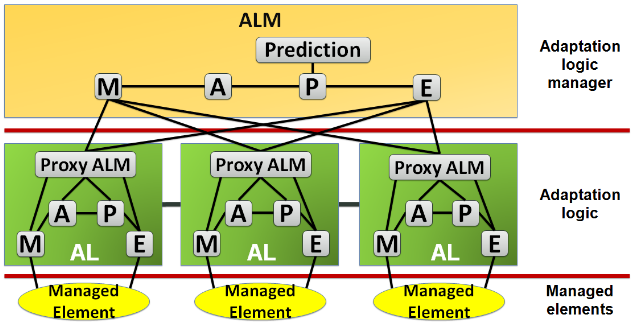

Fig. 1. Approach of AL adaptation using an ALM.

As an external approach benefits from higher maintainability and reduced dependability, we applied the external approach. The ALM is located on top of the ALs. For the sake of simplicity, we start with a centralized approach.

The ALM, consists of a feedback loop represented by the MAPE components. It monitors the state of the AL (M), analyzes it (A), plans adaptations (P), and executes them on the AL (E). So in other terms, we added a MAPE loop on top of the AL. For this MAPE loop, the AL represents the managed element. This approach has the benefit of reusing the structure and components of the AL, only algorithms need to be adjusted. Moreover, orchestration is not restricted to self-* properties as in [15].

Further, we introduce a prediction component. This component is responsible for continuously searching for undiscovered states using models of the AL. Proactive configuration techniques, such as [16], can be applied in order to find adequate configurations for new states ahead of time. With this, calculated configurations can be requested by the planning component if an unknown state occurs that is known to the prediction component.

In order to get information on the AL and implement changes in the AL, we use proxies (Proxy ALM). A Proxy ALM is connected to the MAPE components of the AL and, further, to the M and E components of the ALM. That way, it gathers information about the AL and sends them to the monitoring component of the ALM. As an example, the Proxy ALM can notify the ALM of lower-level adaptations, i.e., adaptations of the managed resources by the AL. The executing component of the ALM can send adaptation plans to a Proxy ALM which applies them.

V. Current Activities and Future Work

Currently, we derive requirements based on scenarios. Based on this, we will design and implement the ALM in order to evaluate approaches based on their feasibility. Therefore, we will design reusable AL elements and procedures.

Furthermore, we will implement a prototype for the system infrastructure. One important requirement is an easy integration of the ALM in frameworks for engineering SAS, such as FESAS [17]. Thus, we will specify interfaces for the ALM.

The decision criteria (policy-based, model-based, goal-based, or utility-based) is still an open issue. An adequate method depends on the specific use case. Thus, we will develop an evaluation approach to map criteria to a use case.

References

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 14:1–14:42, 2009.

[2] Y. Brun et al., "A design space for self-adaptive systems," in *LNCS 7475*. Springer, 2013, pp. 33–50.

[3] M. U. Iftikhar and D. Weyns, "Assuring system goals under uncertainty with active formal models of self-adaptation," in *Proc. of ICSE Companion*, 2014, pp. 604–605.

[4] M. Güdemann et al., "A specification and construction paradigm for organic computing systems," in *Proc. of SASO*, 2008, pp. 233–242.

[5] D. Weyns et al., "On patterns for decentralized control in self-adaptive systems," in *LNCS 7475*. Springer, 2013, pp. 76–107.

[6] M. U. Iftikhar and D. Weyns, "Activforms: Active formal models for self-adaptation," in *Proc. of SEAMS*, 2014, pp. 125–134.

[7] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Proc. of FOSE*, 2007, pp. 259–268.

[8] A. Elkhodary et al., "Fusion: A framework for engineering self-tuning self-adaptive software systems," in *Proc. of FSE*, 2010, pp. 7–16.

[9] I. Epifani et al., "Model evolution by run-time parameter adaptation," in *Proc. of ICSE*, 2009, pp. 111–121.

[10] F. Rochner et al., "An organic architecture for traffic light controllers." in *GI Jahrestagung (1)*, 2006, pp. 120–127.

[11] C. Krupitzer et al., "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, Part B, pp. 184 – 206, 2015.

[12] P. K. McKinley et al., "Composing adaptive software," *IEEE Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[13] P. Lalanda et al., *Autonomic Computing: Principles, Design and Implementation*. Springer, 2013.

[14] J. Floch et al., "Using architecture models for runtime adaptability," *IEEE Software*, vol. 23, no. 2, pp. 62–70, 2006.

[15] IBM Group, "An architectural blueprint for autonomic computing," *IBM White paper*, 2005.

[16] S. VanSyckel et al., "Configuration management for proactive adaptation in pervasive environments," in *Proc. of SASO*, 2013, pp. 131–140.

[17] C. Krupitzer et al., "Fesas: Towards a framework for engineering self-adaptive systems," in *Proc. of SASO*, 2013, pp. 263–264.