FESAS: Towards a Framework for Engineering Self-Adaptive Systems

Christian Krupitzer, Sebastian VanSyckel and Christian Becker
University of Mannheim
Schloss, 68161 Mannheim, Germany
{christian.krupitzer, sebastian.vansyckel, christian.becker}@uni-mannheim.de

***Abstract*—The complexity and size of information systems are growing, resulting in an increasing effort for maintenance. Self-adaptive systems (SAS) that autonomously adapt to changes in the environment or in the system itself (e.g. disfunction of components) can be a solution. So far, the development of SAS is frequently tailored to specific use case requirements. The creation of frameworks with reusable process elements and system components is often neglected. However, with such a framework developing SAS would become faster and less error prone. This work addresses this gap by providing a framework for engineering SAS.**

I. Introduction

Recently, the complexity and size of information systems are increasing, especially in systems that are composed of heterogeneous devices, such as pervasive environments. This results in higher effort for maintenance. A solution is the design of self-adaptive systems (SAS) that autonomously perform tasks, known as the self-* properties [1]. The focus of this abstract is self-adaptation, which is the modification of system behavior in response to changes in its operating environment. Hence, self-adaptation is a basic but powerful mechanism for enabling self-organizing systems. In this abstract, our approach towards a model-driven framework for engineering SAS is presented.

II. Motivation

In general, there are two approaches existing for adaptation of software: parameter adaptation and compositional adaptation [2]. Parameter adaptation is concerned with adapting the system's behavior through changing parameters. The focus in this work is on compositional adaptation, which is described as exchange of components. Various approaches for compositional adaptation exist. Architecture-based approaches have specific architectural components or layers that control the adaptation. Other approaches are based on models for controlling adaptation. For the composition of artifacts, service-based techniques are beneficial. Here, services can be composed in different chronological orders and service implementations are exchangeable for handling changing requirements.

However, most systems based on the above mentioned approaches are lacking the ability of generalization, as well as reusable and well-defined processes and components for design and implementation. Designers often start from scratch when building a SAS. Reusability is challenging because of use case specific design decisions [3]. Furthermore, there is a lack regarding generic process elements for SAS design and engineering [3]. Integrating process definitions and a library of components for an adaptation logic into a framework that is enhanced with tools would simplify the development of SAS and result in faster and less error prone development [3].

III. Our Approach

Our goal is to simplify the development of SAS with a framework for engineering SAS with reusable processes and components, as it has been identified as a gap in the research landscape of SAS engineering [3]. Designers should be supported in abstracting from the low-level requirements of specific systems and offer a high-level abstraction for the adaptation requirements. These high-level requirements will be translated to requirements for the adaptation logic of SAS. The framework will offer solutions to these high-level requirements, autonomously determine the needed system infrastructure, and build the adaptation logic.

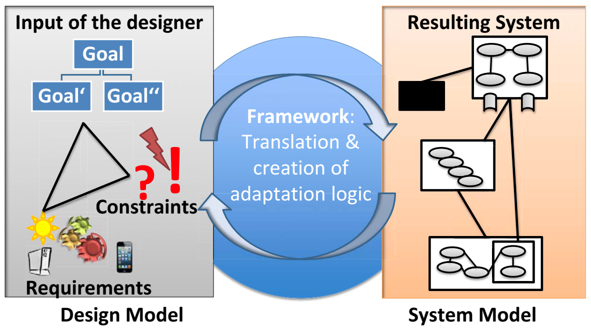

Fig. 1: Design time elements of the framework

Figure 1 shows how the framework supports developers at design time. The framework provides tools for designing the system and transforms the model into a system model, using a reference architecture for SAS and a library with reusable components for the adaptation logic. In the design phase, requirements, goals, and constraints are captured using a requirements engineering approach tailored to developing SAS [4]. Subsequently, the design model can automatically be transformed into a run-time system model. The challenge regarding the design is how to support system developers in specifying the requirements and how to transfer these into requirements for the adaptation logic. These adaptation requirements are first class entities [5] that are present throughout the lifecycle of the system.

978-0-7695-5129-6/13 $26.00 © 2013 IEEE
DOI 10.1109/SASO.2013.36

The system model offers an initialization of the system based on elements of a component library, which are attached to a reference architecture. The reference architecture consists of a middleware for communication between the different elements and a container for the adaptation logic. Therefore, it will be necessary to construct a suitable adaptation logic based on the captured adaptation requirements. The adaptation logic is based on the MAPE-K principle [1]. Issues for the adaptation logic can be the degree of decentralization of the MAPE-K elements and their distribution among devices [6]. All elements for the adaptation logic are modeled in the component library. Another challenge is the design of the component library in an extendable manner. The distribution of the elements is determined by the framework based on distribution and design patterns. Here, a major challenge will be the creation of adaptation logic patterns based on the adaptation requirements of a specific use case. Furthermore, the implementation of MAPE-K loops can be done using different principles. One issue is the communication between the loop elements, e.g. event-based versus a push model. Additional complexity will arise from the decision of a concrete feedback loop initialization based on the adaptation requirements.

Context-awareness is essential because of the necessity to respond to environment changes with self-adaptive mechanisms. Therefore, a context manager (CM) will be part of the component library. The library contains components like (i) control loop elements, (ii) design and distribution patterns, and (iii) a learning component (LC) for improving the quality of adaptation. These are modeled in the component library and used for building the adaptation logic. The component library is modeled as API, so that independence of the underlying infrastructure can be achieved. Communication between the different system artifacts is facilitated by an object-oriented middleware (OOMW). Such a SAS is shown in Figure 2.

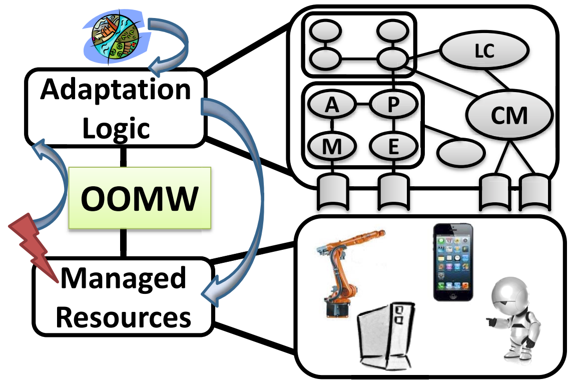

Fig. 2: Self-adaptive system designed with the framework

The creation and adaptation of the system is done according to the design model. Once the system is running, new detected requirements and constraints for the system are used for refinement of the system model (and verifying it against the design model). A reorganization of components is determined by the adaptation logic and triggered by environment changes or corruption of system components. Through this reorganization, compositional adaptation is achieved, which enables adaptation in changing environments.

IV. Current Activities and Future Work

Currently, we are developing a service-oriented version of the component library in Java. In this first prototype the BASE middleware is used [7]. The BASE middleware supports the use of services on remote devices. It has been developed to the needs of pervasive environments with many heterogeneous devices. Each element of the component library, as well as additional functionalities, are modeled as BASE services in Java. With the middleware, the system is modeled as a connection of services, and compositional adaptation as exchange of services.

Components are implemented as BASE services. The elements of the MAPE-K cycle as control structure has been implemented as BASE services. Additionally we implemented two patterns for decentralized control (based on [6]) so far. Additional structures for the MAPE-K elements will be integrated and evaluated against self-adaptive scenarios. Using the results, we want to build a mechanism for determining the need of decentralization based on adaptation requirements, so that the framework can implement it into the adaptation logic.

One of the challenges is the modeling of requirements, goals, and constraints in an appropriate modeling language. The dynamic environment of SAS results in uncertainty during design time. Consequently, the requirements engineering of SAS needs a special handling [4]. Therefore, a suitable requirements engineering process must be constructed. We want to achieve this by a definition of adaptation requirements that can be used at design time for designing the adaptation logic and later at runtime for identifying adaptation need. Afterwards, a component for transforming the design model into a run-time model and an adaptation logic must be implemented.

Due to the requirement that the framework should be as generic as possible, it sounds reasonable that the evaluation should be done in different scenarios. As such scenarios, intelligent hospital management, road traffic management, or smart logistics are conceivable. For evaluation of the adaptation and for learning purposes during runtime, a metric for the adaptation quality will be defined.

Main challenges of our work will be the design of adaptation requirements as first class entities and the construction of an appropriate adaptation logic based on adaptation requirements, the component library, and design and distribution patterns.

References

[1] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

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

[3] R. de Lemos et al., "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap," in *LNCS 7475*. Springer, 2013, pp. 1–32.

[4] B. H. Cheng et al., "Software Engineering for Self-Adaptive Systems: A Research Roadmap," in *LNCS 5525*. Springer, 2009, pp. 1–26.

[5] N. Bencomo et al., "Requirements Reection: Requirements as Runtime Entities," in *Proc. of ICSE*, 2010, pp. 199–202.

[6] D. Weyns et al., "On Patterns for Decentralized Control in Self-Adaptive Systems," in *LNCS 7475*. Springer, 2013, pp. 76–107.

[7] C. Becker et al., "BASE - a Micro-broker-based Middleware for Pervasive Computing," in *Proc. of Percom*, 2003, pp. 443–451.