FESAS: A Framework for Engineering Self-Adaptive Systems

Christian Krupitzer
University of Mannheim, Germany
christian.krupitzer@uni-mannheim.de

***Abstract*—The complexity and size of information systems are growing, resulting in an increasing effort for maintenance. Autonomic Computing, Organic Computing or self-adaptive systems (SAS) that autonomously adapt to environment changes or changes in the system itself (e.g. disfunction of components) can be a solution. So far, the development of a SAS is tailored to its 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. The framework is based on model-driven engineering and a service-oriented architecture. At the development stage, a design model is transformed into a system model. During runtime, this system model is mapped to a concrete system build of various heterogeneous devices with functionalities. The functionalities of the devices is represented as services. A reference architecture and a component library support this mapping and enable the deployment of an adaptation logic tailored to the system's use case. This reduces engineering time through a generic process with reusable elements.

***Index Terms*—Self-Adaptive Systems; Software Engineering; Model-Driven Engineering; Service-Oriented Architecture; Pervasive Computing; BASE Middleware**

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 (for a description of the self-* properties see [10]). A formal definition is given by Oreizy *et al.* [16]:

> Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation.

The focus of this proposal 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 Autonomic Computing, which is described in analogy to the autonomous nervous system as systems that perform tasks autonomously [10]. Another related research area of SAS research is Organic Computing. In Organic Computing concepts to achieve controlled self-organization are sought after [18].

The remainder of the proposal is structured as follows: in Section II the problem of missing reusability in the engineering processes for SAS is defined. The relevance of the problem is stated in Section III. Some approaches to the problem are given in Section IV. The framework is explained in Section V. Section VI presents challenges and the roadmap for the development of the framework.

II. Problem Definition

SAS research mainly addresses approaches for achieving self-adaptation by designing systems tailored to specific use cases. Design decisions related to the specific needs of the use cases complicates reusability of the resulting artifacts.

The creation of frameworks for designing and engineering SAS is often neglected [2]. Especially, there is a lack regarding generic process elements for SAS design and engineering [3]. Furthermore, a library of reusable components like control structure elements would support the development of adaptation logic for SAS [3].

Integrating the process definitions and a library of components into a framework enhanced with tools would simplify the development of SAS and result in faster and less error prone development.

III. Problem Relevance

In general, there are two approaches existing for adaptation of software: parameter adaptation and compositional adaptation [12]. Parameter adaptation is concerned with adapting the system's behavior through changing various parameters. The focus in this work is on compositional adaptation, which is described as exchange of components [12]. Additionally the compositional adaptation is divided in an internal approach, where application and adaptation logic are integrated, and an external one with a dedicated adaptation logic [17]. In this work the external adaptation approach is addressed. Different approaches for systems that are able to adapt their behavior via compositional adaptation exist. These are presented in this section.

Architecture-based approaches have in common that specific architectural components or layers control and execute the adaptation (e.g. layer approach [11], adaptation manager [16], and the Rainbow framework [6]). Other approaches are based on models for controlling adaptation (e.g. Models@Run.time

[14], architectural models [5], and adaptation models [20]). 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 (environmental) requirements. Therefore, service architectures are commonly used for simplifying the compositional adaptation (e.g. [4], [7], [13]).

Organic Computing [15] or Autonomic Computing [10] are derived from principles of adaptation found in biology, physics, or chemistry that are transfered to information technology.

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 [3]. Furthermore, the authors mostly focus on one of the categories (architecture-based, model-based, or service-based) instead of combining them in order to benefit from their advantages – e.g. the abstraction of service orientation and ease of model-driven design – while minimizing the respective drawbacks – e.g. the fixation on specific use case requirements. Additionally, the focus is mainly on addressing single self-* properties.

IV. Related Work

Different approaches already addressed issues in software engineering for SAS. Seebach *et al.* presented software engineering guidelines for self-organizing resource-flow systems [19]. The guidelines are rather generic and applicable in many different cases. Concrete and reusable elements are missing.

Frameworks are presented by Hallsteinsen *et al.* [8], Weyns, Malek and Andersson [21], and Menascé *et al.* [13]. These approaches address some of the issues that should be provided within the FESAS project.

Hallsteinsen *et al.* [8] focus in the MUSIC project on self-adapting applications in ubiquitous environments, a similar setting like in the FESAS project. Nevertheless in FESAS there will be requirements engineering activities integrated which is not explicitly mentioned in MUSIC.

Weyns *et al.* [21] provide with the FORMS reference model a meta-level description of SAS. Their notation can be used for comparing and analyzing different alternatives regarding self-adaptive elements. Nevertheless, concrete methods, tools, or practices are missing.

SASSY is a model-driven framework for supporting the development of systems in dynamic environments, e.g. SAS [13]. The focus here is on self-architecting software, whereas FESAS is concentrated on the adaptation mechanisms.

An approach to adaptation in pervasive environments, specialized in adaptation of distributed systems, offers PCOM [9]. PCOM is build on top of the BASE middleware [1], which should be used in the FESAS project, too, and could be partly integrated.

What is missing so far is mainly the reusability of concrete components and system parts as well as tools for supporting designers and engineers. This work wants to address the gap.

V. Approach

The goal is to create a generic framework for engineering SAS with reusable processes and components as it has been identified as gap in the research landscape of SAS engineering [3]. The term ”generic” means that the framework should enable the construction of self-adaptive systems with different requirements and in different settings. Therefore, it will be necessary that designers can use specific tools that are able to abstract from the low-level requirements of specific systems and offer a high-level abstraction for parts of the system requirements. These high-level requirements will relate to typical requirements for SAS. The framework will offer solutions to these high-level requirements and determine autonomously the system infrastructure needed and build these systems. Low-level requirements that are specific to a special SAS must be implemented separately, but – once implemented – will be integrated into the SAS autonomously.

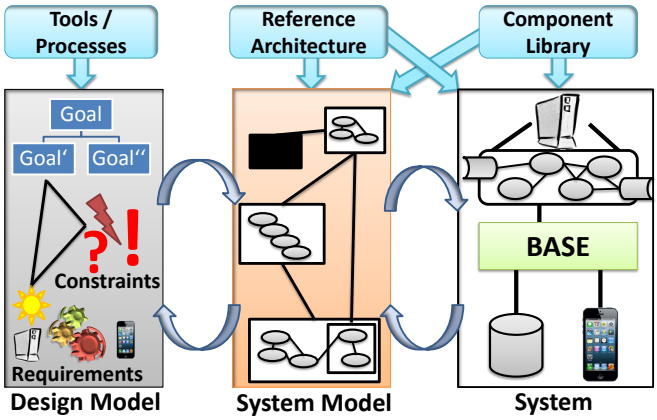

Fig. 1: Framework for Engineering Self-Adaptive Software

Figure 1 shows the conceptual design of the framework. The framework provides tools for designing the system, a reference architecture for SAS and a library with reusable components that are used for building the system. The design is based on model-driven engineering. In the design phase, requirements, goals, and constraints are captured in a design model using a SAS tailored requirements engineering approach [2]. The design model can automatically be transformed into a run-time system model.

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 the BASE middleware [1] for communication between the different elements and containers for self-adaptive elements which are filled in at run-time with components of the component library.

The reference architecture is designed in a general way in order to apply to many different use cases. Besides self-adaptation, the system has to handle different, heterogeneous devices. Context-awareness is essential because of the necessity to respond to environment changes with self-adaptive

mechanisms. Generic components like self-* properties containers, control loop elements (e.g. MAPE cycle [10]), context manager, communication infrastructure, adaptation manager, and Quality of service (QoS) manager (for non-functional requirements) are part of the reference architecture.

The elements needed for self-adaptation are arranged in one or more adaptation logic components. All elements are in the component library. The distribution of the elements is determined by the framework and based on rules. The functionality of the system is captured in the managed resources. These consist of BASE services, which are managed through the adaptation logic elements. Figure 2 shows the interplay of the adaptation logic and the managed resources.

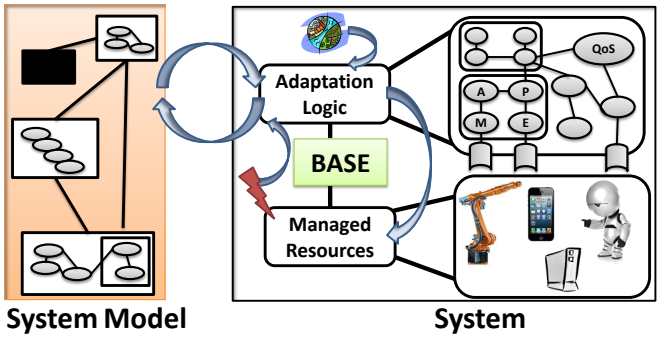

Fig. 2: Connection of Adaptation Logic and Managed Resources in the System

The system architecture is based on the service-oriented architecture (SOA) principle. Communication between the different system artifacts is facilitated by the BASE middleware [1]. 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. Beside the location of services (no difference in the usage of local or distributed services), BASE is able to abstract the communication in a way that different transport mediums like W-LAN, Ethernet or Bluetooth are supported and the middleware decides autonomously on the suitable one [1]. Each element of the component library, as well as additional functionalities, are modeled as BASE services. With the BASE middleware, the system is modeled as a connection of services and compositional adaptation as exchange of services. In Figure 2 circles represent BASE services, connections between circles represent data exchange via a network connection. The exchange is managed by the BASE middleware. Additionally, sensors are represented by the rectangles with the half-cycled sides top and down.

The creation and adaptation of the system is done in accordance with 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), which can activate the reorganization of system components. The reorganization is determined by the adaptation logic and triggered by environment changes or corruption of system components as shown in Figure 2. Through this reorganization, compositional adaptation is achieved which enables adaptation in changing environments.

Finally, the engineering framework will provide tools for engineering SAS. With these tools and the library of reusable components, models for capturing requirements, goals, and constraints can be created and transformed during run-time into an infrastructure of services.

A specific degree of decentralization results from the heterogeneity of devices. So special attention is laid on decentralized vs. centralized control structures. Modeling an explicit control structure is an important issue in engineering SAS [2]. Additionally, how to implement and maintain the self-* properties according to the degree of decentralization is considered by the framework. The issues mentioned influence the creation of the adaptation logic.

VI. Challenges and Future Work

For achieving the goal, various research challenges have to be tackled:

- Analyzing existing approaches regarding their strengths, weaknesses and similarities.
- Development of a design model language which captures requirements, goals and constraints. Furthermore, a language for developing system models is needed.
- Implementation of a transformer for transforming the design model into a run-time system model and system configuration.
- Derive a catalog with common reusable components for SAS. Special focus will be on control mechanisms for the adaptation and self-* properties.
- Propose a reference architecture for SAS.
- Introduce a software engineering approach for constructing SAS based on the reference architecture and the catalog of reusable components. Therefore, software engineering processes must be tailored to SAS and a new requirements engineering approach for requirements capturing during design as well as run-time must be developed.
- Evaluation of the approach: The systematic evaluation of the framework must be ensured with a methodology.

Right now, a prototype for modeling systems with the component library is under development. Next to a catalog of components for SAS, the components will be implemented as BASE services using Java. Additionally, a modeling syntax based on UML will be designed and a transformer for initializing the system based on the system model will be implemented. Typical components are elements for controlling the adaptation to environment changes. Therefore, the MAPE-K cycle will be implemented and the components for Monitoring, Analyzing, Planning and Executing as well as a Knowledge component will be implemented as BASE services. Further components are sensor interfaces, sensor fusion components (for supporting the monitoring component) or a QoS manager for controlling non-functional requirements.

Furthermore, an evaluation of SAS use cases and scenarios will be done. In this evaluation, factors that induce a specific

degree of (de)centralization will be analyzed and patterns for decentralization will be implemented for the framework. Using these factors, the framework can automatically determine the need of decentralization based on the design model and implement it accordingly in the adaptation logic.

One of the challenges is the modeling of requirements, goals and constraints in an appropriate modeling language. Due to the dynamics in the environment of SAS and the resulting uncertainty of the environment during design time, the requirements engineering of SAS needs a special handling [2]. Therefore a suitable requirements engineering process must be constructed.

Afterwards, a component for transforming the design model into a run-time model and its system configuration must be implemented. Special focus will be on decentralization, control loop aspects, and the degree of user integration. The evaluation of the framework must be ensured. Therefore, a methodology will be developed or an existing one will be adjusted. Due to the requirements that the framework should be as generic as possible, it sounds reasonable that the evaluation should be done in different scenarios. As such scenarios, traffic management, smart logistics and production facilities, or software for robots can be conceivable.

If feasible, existing approaches and components will be integrated into the framework. Especially for the model-driven part of the framework, different solutions are present that could be integrated (e.g. [4], [7], [13]). So the focus of the work will be on the requirements specifications, software engineering processes, and the composition of the resulting system with reusable components.

Remarks

This work is supervised by Prof. Dr. Christian Becker, University of Mannheim, Germany. The FESAS project's homepage is available at:

`http://fesas.bwl.uni-mannheim.de`

REFERENCES

[1] BECKER, C., SCHIELE, G., GUBBELS, H., AND ROTHERMEL, K. BASE - a Micro-broker-based Middleware for Pervasive Computing. In *IEEE International Conference on Pervasive Computing and Communications* (2003), IEEE, pp. 443–451.

[2] CHENG, B. H. ET AL. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems, LNCS 5525*. Springer, 2009, pp. 1–26.

[3] DE LEMOS, R. ET AL. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II, LNCS 7475*. Springer, 2013, pp. 1–32.

[4] DI NITTO, E., GHEZZI, C., METZGER, A., PAPAZOGLOU, M., AND POHL, K. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering 15*, 3-4 (2008), 313–341.

[5] FLOCH, J., HALLSTEINSEN, S., STAV, E., ELIASSEN, F., LUND, K., AND GJORVEN, E. Using architecture models for runtime adaptability. *IEEE Software 23*, 2 (2006), 62–70.

[6] GARLAN, D., CHENG, S.-W., HUANG, A.-C., SCHMERL, B., AND STEENKISTE, P. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer 37*, 10 (2004), 46–54.

[7] GEIHS, K., REICHLE, R., WAGNER, M., AND KHAN, M. U. Modeling of Context-Aware Self-Adaptive Applications in Ubiquitous and Service-Oriented Environments. In *Software Engineering for Self-Adaptive Systems, LNCS 5525*. Springer, 2009, pp. 146–163.

[8] HALLSTEINSEN, S., GEIHS, K., PASPALLIS, N., ELIASSEN, F., HORN, G., LORENZO, J., MAMELLI, A., AND PAPADOPOULOS, G. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *Journal of Systems and Software 85*, 12 (2012), 2840–2859.

[9] HANDTE, M., SCHIELE, G., MATJUNTKE, V., BECKER, C., AND MARRÓN, P. J. 3PC: System Support for Adaptive Peer-to-Peer Pervasive Computing. *ACM Transactions on Autonomous and Adaptive Systems 7*, 1 (2012), Article 10.

[10] KEPHART, J. O., AND CHESS, D. M. The Vision of Autonomic Computing. *Computer 36*, 1 (2003), 41–50.

[11] KRAMER, J., AND MAGEE, J. Self-Managed Systems: an Architectural Challenge. In *ICSE Workshop on the Future of Software Engineering* (2007), pp. 259–268.

[12] MCKINLEY, P., SADJADI, S., KASTEN, E., AND CHENG, B. Composing adaptive software. *Computer 37*, 7 (2004), 56–64.

[13] MENASCE, D., GOMAA, H., MALEK, S., AND SOUSA, J. SASSY: A Framework for Self-Architecting Service-Oriented Systems. *IEEE Software 28*, 6 (2011), 78–85.

[14] MORIN, B., BARAIS, O., JEZEQUEL, J.-M., FLEUREY, F., AND SOLBERG, A. Models@Run.time to Support Dynamic Adaptation. *Computer 42*, 10 (2009), 44–51.

[15] MÜLLER-SCHLOER, C., SCHMECK, H., AND UNGERER, T., Eds. *Organic Computing - A Paradigm Shift for Complex Systems*, 1st ed. Springer, 2011.

[16] OREIZY, P., GORLICK, M., TAYLOR, R., HEIMHIGNER, D., JOHNSON, G., MEDVIDOVIC, N., QUILICI, A., ROSENBLUM, D., AND WOLF, A. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems 14*, 3 (1999), 54–62.

[17] SALEHIE, M., AND TAHVILDARI, L. Self-Adaptive Software: Landscape & Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems 4*, 2 (2009), Article 14.

[18] SCHMECK, H., MÜLLER-SCHLOER, C., CAKAR, E., MNIF, M., AND RICHTER, U. Adaptivity and Self-organisation in Organic Computing Systems. In *Organic Computing - A Paradigm Shift for Complex Systems*, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds., 1st ed. Springer, 2011, pp. 5–37.

[19] SEEBACH, H., NAFZ, F., STEGHOFER, J.-P., AND REIF, W. A Software Engineering Guideline for Self-Organizing Resource-Flow Systems. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems* (2010), IEEE, pp. 194–203.

[20] VOGEL, T., AND GIESE, H. Adaptation and Abstract Runtime Models. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* (2010), ACM Press, pp. 39–48.

[21] WEYNS, D., MALEK, S., AND ANDERSSON, J. FORMS: a FOrmal Reference Model for Self-adaptation. In *International Conference on Autonomic computing* (2010), ACM Press, pp. 205–214.