Towards Seamless Adaptation: An Agent-Oriented Approach

Nauman A. Qureshi and Anna Perini
Software Engineering Research Unit
Fondazione Bruno Kessler - IRST
Via Sommarive, 18, 38100 Trento, Italy
{qureshi, perini}@fbk.eu

Abstract

Self-adaptive systems are able to autonomously adapt to changing user requirements and resource variability at run-time, therefore addressing the problem of software complexity and maintenance. Ongoing research on development methodologies for Self-adaptive software points out a necessity for supporting, requirements-time, design-time and run-time, adaptivity. The main aim of our work is to devise a development process that enables seamless adaptation of software systems by exploiting requirements-, design- and run-time adaptation.

1. Introduction

Today software systems complexity is mainly driven by the dynamism of user needs and by the heterogeneity in the operating environment, making software maintenance a complicated task. This has motivated the proposal of the autonomic computing paradigm inspired by biological systems, which employs specialized adaptation and evolution strategies.

Software systems, which address such problems and are able to autonomously adapt to changing user requirements and resource variability at run-time are called *Self-adaptive* systems. Many different definitions of *Self-adaptive* systems have been proposed so far [1, 4]. In this work, we adopt the following definition for *Self-adaptive* systems: *Self-adaptive system is a system, which can modify its behaviour at run-time, steered by its "perception" of current user's requirements and operating environment changes, and by reusing requirements and design artefacts*.

Ongoing research on development methodologies for *Self-adaptive* software points out a necessity for supporting, requirements-time, design-time and run-time adaptivity. Several interesting work have been realized, which leverage Architecture Based (Middleware) [2, 4], Service-Oriented Architectures (SOA) [3], Multi-Agent Systems (MAS) [7], Requirement Engineering [1] and Goal-oriented Requirement Engineering [5] paradigms.

Our research develops in the context of the studies on software engineering methods and techniques to develop *Self-adaptive* software. The main aim of our work is to devise a development process that enables *seamless* adaptation of software systems that is the adaptation takes place without significant system performance compromise. Requirements-, design- and run-time adaptation mechanisms are considered. We move from an analysis of state-of-the-art approaches describing the proposed process view, revisiting and integrating ideas from an Agent-Oriented [6] and Architecture-based solution [2, 4], and sketch the resulting approach, which is more extensively described in detail and illustrated with an example in a technical report. [1]

The paper is organized as follows. Section 2 presents our view of the adaptation process, which goes from requirements to run-time. Section 3 provides an experience summary with discussion. Conclusion and future work is discussed in Section 4.

2. Self-Adaptive Systems: A Process View

State-of-the-art work on *Self-adaptive* systems development introduced the concepts of requirements-, design- and run-time adaptation, but typically focuses on one from among the three. In our approach we conceive a development process, called *3T*Process that considers all these three aspects with their mutual relationships as shown in Fig. 1. We believe that *seamless* adaptation should built on top of these mutual dependent elements using an agent-oriented approach.

Requirement-Time: Goal models are the key artefact at this stage. They are used to capture the variability in user preferences and resource needs, and to represent alternative ways to meet those needs, providing input for a high-

[1]N. A. Qureshi and A. Perini. Towards seamless adaptation:An agent-oriented approach. Technical report, (FBK-IRST), 2008

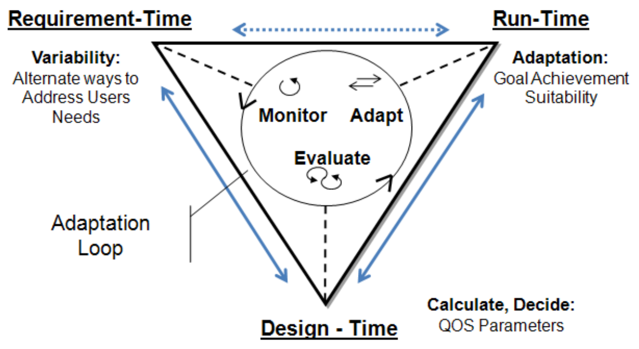

Figure 1. *3T***Process View**

variability design. Quality of Service (QoS) parameter values and conditions which may affect goal achievement are identified at this stage and specified (parameterized goals) for monitoring purposes.

Design-Time: Focus at this time is on defining a QoS models including evaluation policies that can be action-, goal-, utility-based. Different architecture configurations (compositions) with respect to low-level components and agents are design together with the specific mapping to their implementation platform (architecture model). This architecture model is linked to the QoS model that will drive the architecture selection at run-time.

Run-Time: Main artefact here is environment monitoring data. During the operational course of the system, real-time observed QoS values are re-evaluated along with user preferences to generate code (adaptation scripts), here the aim is to make the system to adapt *seamlessly* to the new situations by enforcing the generated code without significant performance compromise.

Adaptation-loop: It exploits artefacts of each edge and the automation is managed by employing agents. Context data, together with user input (if any) are classified according to the requirements goal tree in order to verify alignment of system behaviour to current users goals. The design-time QoS model, which drives the configuration selection, is instantiated with respect to the data collected during monitoring. At run-time, the process of monitoring and evaluation is revisited by finally generating a code artefact, which selects a configuration variant, making the system to adapt.

3. An Experience Summary

In order to illustrate our intended approach, we have adopted and simplified an example i.e. Personal Media Server used for video streaming [4]. In this example we develop a goal model and analyze how agents exploit them by providing an externalized adaptation mechanism. With further refinement, using [6] approach, we will see how we can achieve high-variability design. In this scenario, we consider different strategies, which motivates the need for having an externalized adaptation mechanism.

We instantiate our process view by integrating methods and ideas from existing requirements driven approaches [5] and architecture-based approaches [2, 4]. In particular, we use the TROPOS methodology [6], to derive a Late Requirements and Architectural Design model in which we have decomposed the system in terms of agents. Following are the modelling steps, according to our proposed process view. **[Step 1]**: Defining the Goals, **[Step 2]**: Analyzing AND / OR Decomposition, **[Step 3]**: Impact of Contributions on Soft Goals, **[Step 4]**: Parameterizing Goals, **[Step 5]**: Deciding Alternate Architecture, **[Step 6]**: Apply Chosen Architecture. Details of our modelling experience and approach followed can be found in a technical report. [1]

4. Conclusions

In this paper, We discussed how the agent paradigm provides effective concepts to represent users' goals and preferences and support an externalizing adaptation mechanism, providing, overall seamless adaptation. In particular, we focused on the development process for *Self-adaptive* systems, an aspect that has been only partially addressed by previous work. In our approach, the development process should consider requirements-, design and run-time adaptation as complementary views of a *Self-adaptive* system.

References

[1] D. M. Berry, B. H. Cheng, and J. Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *REFSQ05*, pages 95–100, 2005.

[2] S.-W. Cheng, A.-C. Huang, D. Garlan, B. R. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. In *ICAC*, pages 276–277. IEEE Computer Society, 2004.

[3] G. Denaro, M. Pezzè, and D. Tosi. Designing self-adaptive service-oriented applications. In *ICAC*, page 16. IEEE Computer Society, 2007.

[4] E. Gjørven, F. Eliassen, K. Lund, V. S. W. Eide, and R. Staehli. Self-adaptive systems: A middleware managed approach. In A. Keller and J.-P. Martin-Flatin, editors, *Self-Man*, volume 3996 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2006.

[5] S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, and S. M. Easterbrook. Configuring common personal software: a requirements-driven approach. In *RE*, pages 9–18. IEEE Computer Society, 2005.

[6] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. High variability design for software agents: Extending tropos. *TAAS*, 2(4), 2007.

[7] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White. A multi-agent systems approach to autonomic computing. In *AAMAS*, pages 464–471. IEEE Computer Society, 2004.