Requirements and Architectures for Adaptive Systems

João Pimentel[1,2], Jaelson Castro[1], Emanuel Santos[1], Monique Soares[1], Jessyka Vilela[1], and Gabriela Guedes[1]

[1] Centro de Informática, Univ. Federal de Pernambuco (UFPE), Recife, Brazil
{jhcp,jbc,ebs,mcs4,jffv,ggs}@cin.ufpe.br
[2] Department of Information Eng. and Computer Science, University of Trento, Italy

Abstract. The growing interest in developing adaptive systems has led to numerous proposals for approaches aimed at supporting the development of such systems. Some approaches define adaptation mechanisms in terms of architectural designs, consisting of concepts such as components, connectors and states. Other approaches are requirements-based, thus concerned with goals, tasks, contexts and preferences as concepts in terms of which adaptation is defined. By considering only a partial view of software systems (either the problem space or the solution space), such proposals are limited in specifying the adaptive behavior of a software system. In this paper we present ongoing work towards deriving architectural models in order to support the design and runtime execution of software adaptation both at a requirements and architectural level.

Keywords: adaptive systems, architectural design, adaptation control mechanisms, requirements

1 Introduction

In [1] the authors, by conducting a comparative study, concluded that requirement and architecture based approaches for software adaptation share common elements, such as the use of feedback loops and of external control mechanisms. However, there are differences that reveal complementary advantages and disadvantages of the two approaches.

On the one hand the requirement-based approaches capture and model the objectives of the system, but they lack awareness about the capabilities and the limitations of the proposed solution. On the other hand, architectural models provide guidance for the deployment of the monitoring mechanisms and the effectors that apply the adaptation process on the target system. The objectives of the system, however, are not modeled making it difficult to handle changes at the requirements level.

A third dimension to this combination of models is related to the system behavior. It is often the case that a system fails because the execution plan of its tasks was not appropriate for the holding conditions. Therefore, we propose the derivation of statecharts from goal models, completing this variability puzzle that captures all the aspects of the software system.

2 Objectives of the research

2.1 Baseline

The baseline for this ongoing work is the *Zanshin* approach for the design and development of adaptive systems [2–4] which, in its turn, is based on Goal-Oriented Requirements Engineering (GORE) [5]. Focussed on the feedback loop for adaptation, *Zanshin* augments goal models with the requirements for the monitor and adapt phases of such loops.

To illustrate *Zanshin*, Fig. 1 shows a goal model specifying the requirements for an adaptive Meeting Scheduler system. Traditional $i^\star$ elements (goals, soft-goals, tasks) appear alongside domain assumptions (rectangles) and quality constraints (round-cornered rectangles), which are necessary for our adaptation purposes. Also, lines connecting different elements represent refinement/operationalization relations, following obvious AND/OR Boolean semantics for goal satisfaction. Finally, small circles and diamonds are elements introduced by *Zanshin*, namely *Awareness Requirements* (*AwReqs*) and *Control Variables*.

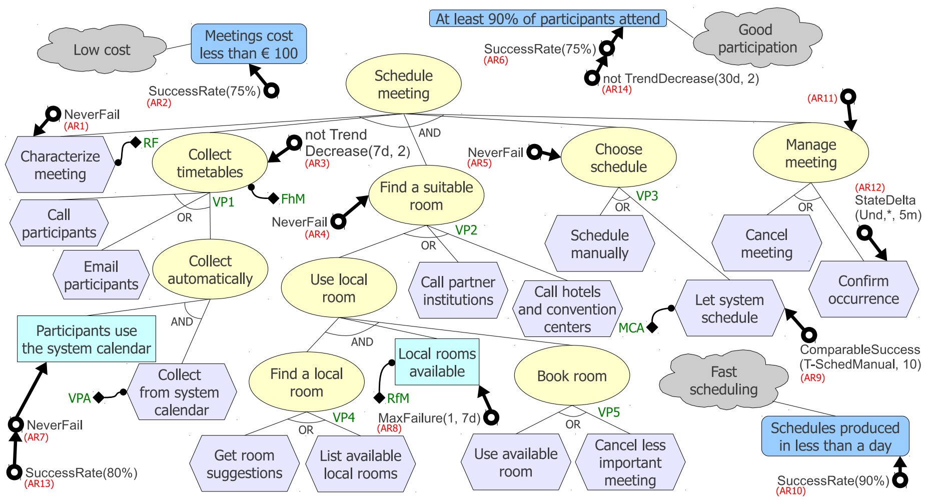

Fig. 1. Goal-based requirements specification for a Meeting Scheduler.

In the first step of the approach, *AwReqs* are elicited as requirements for the monitoring component of the feedback loop. *AwReqs* talk about the states assumed by other requirements—such as their success or failure—at runtime [2], representing, thus, situations in which the stakeholders would like the system to adapt. For example, *AR1* states that task *Characterize meeting* should never fail, whereas *AR2* indicates that the quality constraint that operationalizes softgoal *Low cost* should succeed 75% of the time.

The second step is called *System Identification* [3] and aims at identifying system parameters that can be changed at runtime and representing, in a qualitative way, how such changes can affect indicators of requirements convergence. We use *AwReqs* as indicators and consider two kinds of parameters: OR-refinements are called *Variation Points* (e.g., how to *Collect timetables* in *VP1*), whereas *Control Variables* abstract OR-refinements which are impossible or infeasible to be represented in the model (*FhM*, from how many people to collect timetables). The relation between parameters and indicators is represented by differential relations, e.g., $\Delta\,(AR3/FhM) < 0$, which reads "increasing (resp. decreasing) *FhM* contributes negatively (resp. positively) to *AwReq AR3*".

The third and last step concerns the requirements for the adaptation component of the feedback loop, represented by *Evolution Requirements* (*EvoReqs*) [4]. *EvoReqs* specify what the system should do to adapt in one of two ways: stakeholders can either give specific instructions or they can choose to ask the system to analyze the different parameters which have an impact on the failing indicator and change one (or more) of them accordingly. A framework[1], also called *Zanshin*, implements the generic features of a feedback loop in order to provide adaptation to base systems according to their requirements models. See, e.g., [6].

2.2 Objectives

Our research interest is to take further the baseline that we described above by combining requirement models with the software architecture. Towards this direction we propose a guiding methodology to derive architectural models from requirements. This effort would result in a more suitable specification for an adaptive software system, since the entire spectrum of its operational variability would be represented. Therefore, the system would have the maximum variety of alternatives when it has to deal with failures or with environmental changes. The last part of this work would be to advance the adaptation process implemented by *Zanshin* by making it quantitative, in order to acquire higher precision. Moreover, the framework will be extended to be able to deal with multiple failures, exploiting techniques inspired from Control Theory.

3 Scientific contributions

3.1 Architectural derivation

Architectural derivation is concerned with the generation of architectural models, which can include: components & connectors models for describing the system structure; statecharts for describing the system behavior; feature model for expressing the variability of the system; and so on. These different models are complementary, each one capturing a particular view of the system being designed. Thus, different approaches are required in order to derive these different models.

[1] See `https://github.com/sefms-disi-unitn/Zanshin/wiki`.

In previous work [7] [8], we proposed a set of methods to derive the aforementioned models from goal models. The key of that proposal was to derive the models in such a way as to preserve the variability expressed in the input model. However, when considering architectural derivation and its design decisions for the particular case of adaptive systems, there are three new concerns that arise:

a. *Additional variability* — there may be different alternatives to accomplish a given task. For instance, different algorithms can be used to schedule a meeting automaticallly, each with its different benefits and drawbacks. The alternatives identified during architectural derivation will expand the space of adaptation possibilities.
b. *Additional control elements* — besides referring to requirements concepts, *Zanshin* elements (such as *AwReqs* and *Control Variables*) may also refer to and influence architectural concerns. For instance, the time interval for a timed transitition could be defined as a *Control Variable*, rather than as a pre-defined, static interval.
c. *Additional features to support adaptation* — the support of self-adaptation may require the inclusion of new features in the system. This is the case, for instance, when the system requires some kind of instrumentation in order to monitor the satisfaction of *AwReqs*.

In [9] we handled the identification of additional features, considering the monitoring capabilities required to monitor the context. There, we were concerned with the derivation of components & connectors models. An approach for eliciting future requirements, which can be used to identify additional variability (both at requirements and architectural level) was presented in [10]. In [11] [12] we explore additional variability derived from different web services that are available.

Currently, we are working on including additional variability and additional control elements, while supporting the derivation of statecharts.

3.2 Derivation of statecharts

The process for deriving statecharts from goal models comprises 6 steps, depicted in Fig. 2. The first step, *Delegate tasks*, consists of assigning the tasks that will not be performed nor supported by the system proper – e.g., tasks that will be performed by an external actor (human or otherwise). Since these tasks are not carried out by the software system itself, they are not considered during the remainder of the process. In the next step, *Define basic flow*, the architect analyzes all refinements of the goal model and defines a flow expression for each. These flow expressions will be used in the next step, *Generate base statechart*, to create a skeleton of the statechart. Since these expressions are not as expressive as statecharts (although rich enough for defining the basic flow), and can be included as annotations in a goal model, they are a useful intermediate abstraction between goal models and statecharts.

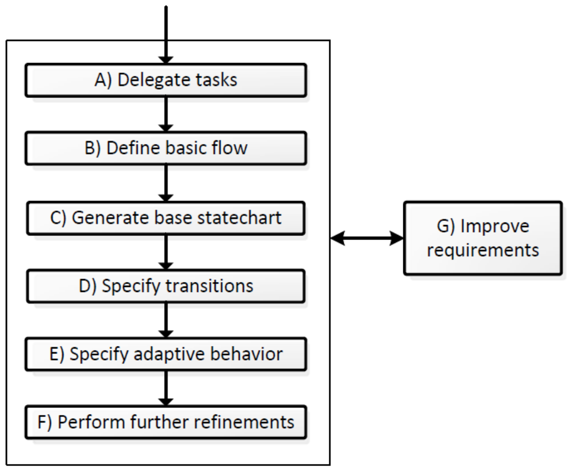
A) Delegate tasks
B) Define basic flow
C) Generate base statechart
D) Specify transitions
E) Specify adaptive behavior
F) Perform further refinements
G) Improve requirements

Fig. 2. Process for statechart derivation.

In the remaining steps the statechart will be refined, as follows. First, during *Specify transitions* the architect defines events and conditions of the derived transitions. Then, the statechart is enriched to describe the system's adaptivity behavior, including the interaction with an external component that provides adaptation-related functionality. This takes place during *Specify adaptive behavior*. As a last step, *Perform further refinements* allows the architect to expand the model in order to include technical details and other concerns that may not have been handled earlier, by exploiting the concept of sub-states.

4 Conclusions and Future Work

In this paper we present ongoing work towards improving the support for the development of adaptive software systems. The combination of requirements and architectural models will provide a richer space of adaptation specification. The proposed derivation methodologies will facilitate the creation of adaptive systems making use of the *Zanshin* framework.

A prototype tool for the derivation of statecharts, as presented in Section 3.1, is currently under development[2].

Acknowledgments. This work has been supported by the ERC advanced grant 267856 "Lucretius: Foundations for Software Evolution" and by Brazilian institutions CAPES and CNPq.

[2] Available at `https://github.com/jhcp/GoalArch`.

References

1. Angelopoulos, K., Souza, V.E.S., Pimentel, J.: Requirements and Architectural Approaches to Adaptive Software Systems: A Comparative Study. In: Proc. of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (to appear). (2013)
2. Souza, V.E.S., Lapouchnian, A., Robinson, W.N., Mylopoulos, J.: Awareness Requirements. In Lemos, R., Giese, H., Müller, H.A., Shaw, M., eds.: Software Engineering for Self-Adaptive Systems II. Volume 7475 of Lecture Notes in Computer Science. Springer (2013) 133–161
3. Souza, V.E.S., Lapouchnian, A., Mylopoulos, J.: System Identification for Adaptive Software Systems: A Requirements Engineering Perspective. In: Conceptual Modeling ER 2011. (2011) 346–361
4. Souza, V.E.S., Lapouchnian, A., Angelopoulos, K., Mylopoulos, J.: Requirements-driven software evolution. Computer Science - Research and Development (2012) 1–19
5. Mylopoulos, J., Chung, L., Yu, E.S.K.: From Object-Oriented to Goal-Oriented Requirements Analysis. Communications of the ACM **42**(1) (1999) 31–37
6. Tallabaci, G., Souza, V.E.S.: Engineering Adaptation with Zanshin: an Experience Report. In: Proc. of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (to appear). (2013)
7. Yu, Y., do Prado Leite, J.C.S., Lapouchnian, A., Mylopoulos, J.: Configuring features with stakeholder goals. In: Proceedings of the 2008 ACM symposium on Applied computing - SAC '08, ACM Press (2008) 645–649
8. Yu, Y., Lapouchnian, A., Liaskos, S., Mylopoulos, J., Leite, J.C.S.P.: From Goals to High-Variability Software Design. In: Foundations of Intelligent Systems. Volume 4994/2008. (2008) 1–16
9. Pimentel, J., Lucena, M., Castro, J., Silva, C., Santos, E., Alencar, F.: Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach. Requirements Engineering **17**(4) (June 2012) 259–281
10. Pimentel, J., Castro, J., Perrelli, H., Santos, E., Franch, X.: Towards anticipating requirements changes through studies of the future. In: 5th International Conference on Research Challenges in Information Science, IEEE (May 2011) 1–11
11. Pimentel, J., Castro, J., Santos, E., Finkelstein, A.: Towards Requirements and Architecture Co-evolution. In: Advanced Information Systems Engineering Workshops. (2012) 159–170
12. Franch, X., Grunbacher, P., Oriol, M., Burgstaller, B., Dhungana, D., Lopez, L., Marco, J., Pimentel, J.: Goal-Driven Adaptation of Service-Based Systems from Runtime Monitoring Data. In: 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops, IEEE (July 2011) 458–463