# From Requirements to Architectures for Better Adaptive Software Systems

João Pimentel[1,2], Konstantinos Angelopoulos[2], Vítor E. Silva Souza[3], John Mylopoulos[2], and Jaelson Castro[1]

[1] Centro de Informática, Univ. Federal de Pernambuco (UFPE), Recife, Brazil
{jhcp,jbc}@ufpe.br

[2] Department of Information Eng. and Computer Science, University of Trento, Italy
{angelopoulos,jm}@disi.unitn.it

[3] Computer Science Dept., Federal University of Esprito Santo (Ufes), Vitória, Brazil
vitorsouza@inf.ufes.br

**Abstract.** The growing interest in developing adaptive systems has led to numerous proposals for approaches aimed at supporting their development. Some approaches define adaptation mechanisms in terms of architectural design, 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 problem- or a solution-oriented view, such proposals are limited in specifying adaptive behavior. In this paper we present ongoing work on supporting the design and runtime execution of adaptive software systems both at a requirements and architectural level, as wells as its challenges, ranging from architectural derivation from requirements to refined adaptation control mechanisms.

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

## 1 Introduction and Objectives

In [1], the authors conducted a comparative study, concluding that requirements- 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 also differences that reveal complementary advantages and disadvantages of the two approaches. On one hand, requirements-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 coded into the adaptation strategy, making it difficult to handle changes at the requirements level.

Based on the results of this study, the authors have embarked on a research project to better link requirements and architectural models by (a) developing techniques for deriving architectural models from requirements, and (b) extending existing techniques for designing adaptive software so that they exploit both requirements and architectural models.

## 2 Baseline

Our baseline is the *Zanshin* framework for the design of adaptive systems [2–4], which in turn is founded on Goal-Oriented Requirements Engineering (GORE) [5]. Adopting from Control Theory the concept of feedback loop for adaptation, *Zanshin* augments goal models with requirements for monitoring and adaptation of such loops.

To illustrate, Fig. 1 shows a goal model for an adaptive Automated Teller Machine (ATM). Traditional $i^\star$ elements (goals and tasks) are connected by refinement/operationalization relations, using AND/OR Boolean semantics for goal satisfaction. *Zanshin* introduces *Awareness Requirements* (*AwReqs*, represented by small circles) and *Control Variables* (rep. by diamonds).

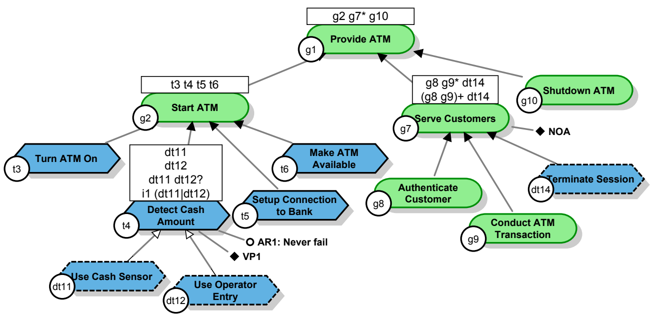

**Fig. 1.** Goal-based requirements specification for an ATM.

*AwReqs* are the requirements for the monitoring component of the feedback loop and impose requirements on the success/failure of other requirements by talking about the states assumed by other requirements at runtime [2]. As such, they represent situations in which the stakeholders would like the system to adapt. In Fig. 1, *AwReq AR1* states that task *Detect Cash Amount* should never fail.

*Control Variables* (CVs) are elicited during *System Identification* [3], alongside *Variation Points* (VPs) and qualitative relations between these two parameters (CVs and VPs) and indicators of requirements convergence, namely, *AwReqs*. Examples in Fig. 1 are the VP for *Detect Cash Amount* (two ways of satisfying it) and the CV *Number of Operators Available* (*NOA*). Differential relations, e.g., $\Delta(AR1/NOA) > 0$ indicate how changes in parameters affect indicators (in this case, increasing *NOA* increases the success of *AR1*).

Lastly, *Evolution Requirements* (*EvoReqs*) [4] specify when and how should other requirements change at runtime. For example, an *EvoReq* may be "If requirement $R$ fails three times in a row, replace it with requirement $R-$", where $R-$ is a weaker (i.e., easier to fulfil) requirement. Using these new classes of requirements, *Zanshin*[1] implements a feedback loop that supports adaptation of a base system.

## 3 Research Agenda

We are interested in taking further the baseline by combining requirement models with software architectures. Towards this end, we propose a systematic methodology for deriving architectural models from requirements. This research will allow us to design adaptive software systems that exploit combined goal and architectural models, thereby capturing allowable adaptations at both levels of abstraction. Therefore, the system would have the maximum variety of alternatives when it has to deal with failures or with environmental changes. The second part of this work involves extending *Zanshin* to exploit combined goal and architectural models, but also making it quantitative, in order to acquire higher precision. Moreover, *Zanshin* will be extended to deal with multiple failures, exploiting techniques inspired by Control Theory.

### 3.1 Architectural derivation

Architectural derivation is concerned with the generation of architectural models, which can include: (a) components & connectors models for describing the system structure; (b) statecharts for describing system behavior; and (c) feature model for expressing the variability of system configuration. These different models are complementary, each one capturing a particular view of the system being designed, thus requiring different derivation approaches.

In previous work [6] [7], we proposed 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 goal 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 and different technologies can be applied, 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 have an influence on architectural concerns. For instance, the time interval for a timed transition could be defined as a *Control Variable*, rather than as a pre-defined, static interval.

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

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

Currently, we are working on including additional variability and additional control elements, while supporting the derivation of statecharts. After all, statecharts capturing system behavior constitute the most important architectural view for adaptive systems.

The process for deriving statecharts from goal models comprises 7 steps. The first step, *Identify design tasks and constraints*, allows to refine the requirements model by including elements that are relevant from the architectural point of view. Next, *Assign tasks*, consists of assigning the tasks that will not be performed nor supported by the base software system — e.g., tasks that will be performed by an external actor (human or otherwise). In the next step, *Define basic flow*, the architect analyzes all refinements of the goal model and defines flow expressions that define their runtime behavior. These expressions, which allow to define flows with a notation akin to regular expressions, are used in the next step (*Generate base statechart*) to create a skeleton of the statechart. The statechart depicted in Fig. 2 was derived from the goal model in Fig. 1. For this statechart, we selected the third flow expression of *Detect Cash Amount (dt11 dt12?* — perform dt11, then optionally perform dt12) and the first flow expression of *Serve Customers (g8 g9* dt14* — perform g8, then perform g9 zero-or-more times, lastly perform dt14).

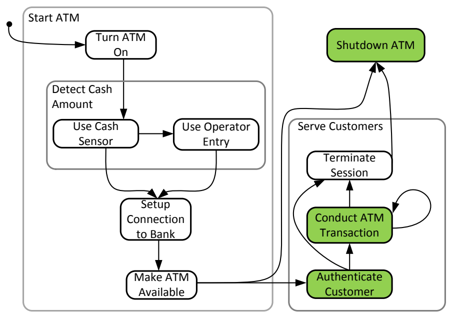

**Fig. 2.** Statechart for a partial behavior refinement of the ATM system

In the remaining steps the statechart skeleton is 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 adaptive 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 statechart concept of sub-states.

### 3.2 Advancing runtime software adaptation mechanisms

Section 2 sketched *Zanshin* and how it defines adaptation mechanisms based on parameters and indicators as well as qualitative relations among them. Given the subjective nature of requirements, it is important to support such qualitative mechanisms. However, in some scenarios, it is possible to identify quantitative relations instead of qualitative. This is especially true for architectural models, which are more tangible than requirements ones.

The use of quantitative relations will allow finer tuning of parameters when restoring failed indicators, which in turn can reduce critical overshooting and redundant oscillations. Methods such as regression analysis can then be used to extract quantitative information about the relation among parameters and indicators.

Moreover, such quantitative relations assist in solving the issue of multiple failing indicators. Usually, software systems involve conflicting requirements (e.g., cost and performance) that result in conflicting indicators (i.e., when one is failing the other is succeeding and vice versa). When several indicators fail it is hard to perform a trade-off analysis with a good degree of confidence using only qualitative information. Thus, the quantitative relations, combined with prioritized indicators, can be exploited in order to apply optimization techniques and identify the best possible adaptation.

## 4 Conclusions

We have presented ongoing work towards improving support for the development of adaptive software systems. On one hand, the combination of requirements and architectural models will provide a richer space of possible adaptations. The proposed derivation methodology will facilitate the creation of adaptive systems based on the *Zanshin* framework. On the other hand, the control mechanisms of the framework itself will be improved, by tackling the occurrence of multiple (and possibly conflicting) failures and using quantitative relations to increase the precision of adaptation mechanisms.

As mentioned in Section 2, a prototype implementation of the *Zanshin* framework is available. This implementation will be extended in order to support the enhancements proposed in Section 3.2. 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.

## References

1. K. Angelopoulos, V. E. S. Souza, and J. Pimentel, "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. V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos, "Awareness Requirements," in *Software Engineering for Self-Adaptive Systems II* (R. Lemos, H. Giese, H. A. Müller, and M. Shaw, eds.), vol. 7475 of *Lecture Notes in Computer Science*, pp. 133–161, Springer, 2013.
3. V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos, "System Identification for Adaptive Software Systems: A Requirements Engineering Perspective," in *Conceptual Modeling ER 2011*, pp. 346–361, 2011.
4. V. E. S. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos, "Requirements-driven software evolution," *Computer Science - Research and Development*, pp. 1–19, 2012.
5. J. Mylopoulos, L. Chung, and E. S. K. Yu, "From Object-Oriented to Goal-Oriented Requirements Analysis," *Communications of the ACM*, vol. 42, no. 1, pp. 31–37, 1999.
6. Y. Yu, J. C. S. do Prado Leite, A. Lapouchnian, and J. Mylopoulos, "Configuring features with stakeholder goals," in *Proceedings of the 2008 ACM symposium on Applied computing - SAC '08*, pp. 645–649, ACM Press, 2008.
7. Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J. C. S. P. Leite, "From Goals to High-Variability Software Design," in *Foundations of Intelligent Systems*, vol. 4994/2008, pp. 1–16, 2008.
8. J. Pimentel, M. Lucena, J. Castro, C. Silva, E. Santos, and F. Alencar, "Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach," *Requirements Engineering*, vol. 17, no. 4, pp. 259–281, 2012.
9. J. Pimentel, J. Castro, H. Perrelli, E. Santos, and X. Franch, "Towards anticipating requirements changes through studies of the future," in *5th International Conference on Research Challenges in Information Science*, pp. 1–11, IEEE, 2011.
10. J. Pimentel, J. Castro, E. Santos, and A. Finkelstein, "Towards Requirements and Architecture Co-evolution," in *Advanced Information Systems Engineering Workshops*, pp. 159–170, 2012.
11. X. Franch, P. Grunbacher, M. Oriol, B. Burgstaller, D. Dhungana, L. Lopez, J. Marco, and J. Pimentel, "Goal-Driven Adaptation of Service-Based Systems from Runtime Monitoring Data," in *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, pp. 458–463, IEEE, July 2011.

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