Designing adaptive systems

João Pimentel, Jaelson Castro

Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
{jhcp, jbc}@cin.ufpe.br

Abstract. In this work, we investigate the interplay between requirements and architecture in the context of adaptive systems. Furthermore, we propose the Multi-Level Adaptation for Software Systems (MULAS) framework. It is centred on the iterative and incremental refinement of a goal model, towards the creation of a design goal model, which can be used at runtime to drive adaptation on a system that is properly instrumented. Moreover, the framework includes a tool-supported process for generating statechart behavioural models from a design goal model.

Keywords: Requirements-driven software adaptation, architecture-driven software adaptation, goal-oriented requirements models, model-driven development.

1 Introduction

Different approaches to support the development of self-adaptive systems have been proposed in the literature. However, those are often restricted to a single aspect of software development. For instance, the Zanshin framework [1] provides support for handling adaptation at the requirements level, enacting a monitoring-diagnosis-compensation cycle. With Zanshin, adaptation is specified in terms of stakeholders' goals, tasks, quality constraints, and other elements.

On the other hand, Rainbow [2] provides similar capabilities, but addressing architectural models. Thus, it is concerned with properties of systems' components and connectors, e.g., response time, number of servers and load balancing. The differences between requirements-based and architecture-based approaches are discussed in [3].

Requirements engineering and architectural design, while addressing the system specification at different abstraction levels, comprise intertwined activities [4]. The former focuses on the problem at hand, whereas the latter provides solutions for that problem.

Approaches that only support requirements-based or architecture-based adaptation thus, lack relevant elements of the adaptation space. For instance, architecture-based

approaches might ignore stakeholders' goals and preferences, while requirements-based ones may not address concerns related to the system implementation, such as algorithms and components.

Hence, the investigation of how to support seamless adaptation mechanisms across the different phases of software development seems to be a promising venue to improve the development of self-adaptive software systems. In this paper we provide an overview of a design process centred on an extended goal model, which incorporate elements aiming to support requirements-based and architectural-based adaptation. To illustrate, we adopt a meeting scheduler exemplar.

The remainder of this paper is organized as follows. In Section 2, we present our process to design adaptive systems, focusing on behavioural specification. Section 3 discusses the limitations of this work. Later, we present ongoing and future work in Section 4.

2 Design process

The proposed process, which is a part of the Multi-Level Adaptation for Software Systems (MULAS) framework, comprises eight steps (Fig. 1). The first five steps are related to the refinement of design goal models: *Identify design tasks, constraints and assumptions*; *Assign tasks*; *Define basic flows*; *Identify indicators, parameters and relations*; and *Specify adaptation strategies*. The other three steps are related to statecharts: *Generate base statechart*; *Specify transitions*; and *Include adaptation elements*. While these steps may be followed mostly sequentially, waterfall-like, in realistic settings it is expected that the architect will go back and forth, by introducing additional refinements to already refined elements.

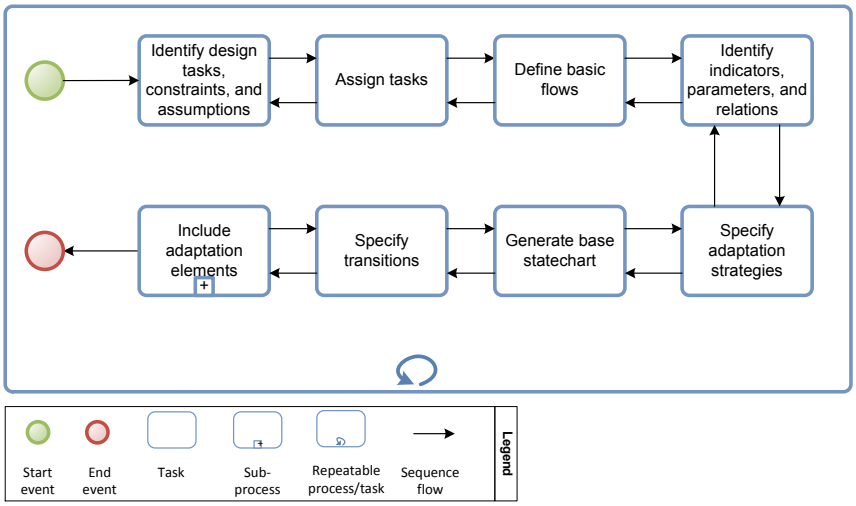

Fig. 1.MULAS design process

The first step, *Identify design tasks, constraints and assumptions*, supports the refinement of a goal model by including elements that are not initially required by stakeholders, but are relevant from the architectural point of view, expressed as design tasks, design constraints and design assumptions. The second step, *Assign tasks*, consists of assigning the responsibilities for the execution of tasks — e.g., tasks that will be performed by an external actor (human or otherwise). This assignment is helpful for defining the scope of the system.

In the next step, *Define basic flows*, the architect introduces possible flows for every sub-tree in the goal model. Roughly, these flows describe the order that the sub-elements are going to be fulfilled or executed, so that their parent element can be considered fulfilled or executed. These flows are expressed as alternative flow expressions, introduced as annotations to a goal model using a top-down, bottom-up, or middle-out strategy. These expressions are later used to automatically generate a statechart that represents the system's behaviour.

The next two steps are related to the adaptation capabilities of the system: *Identify indicators, parameters and relations* and *Specify adaptation strategies*. The former is related to the addition, in the design goal model, of some elements proposed by Zashin [1], in light of the design elements previously included in the first step. In the *Specify adaptation strategies* step it is considered how the system will react to failures — e.g., by retrying the execution of a task, or by changing the parameters described in the goal model.

The second part of the process is related to system behaviour. The first step, *Generate base statechart*, makes use of derivation patterns to automatically create a statechart from the flow expressions previously defined. Although flow expressions are a useful intermediate abstraction between goal models and statecharts, they are not as expressive as statecharts. Thus, in the next step, *Specify transitions*, the transitions of the statechart are refined with their events and conditions, which are identified by analyzing when any given transition should take place.

An example of a resulting (Design) Goal Model is shown in Fig. 2, which is an excerpt from a Meeting Scheduler system [10]. Besides the scheduling itself, the system supports the characterization of meetings, the gathering of timetables and the management of meetings, while satisfying the non-functional requirements of scalability and portability. The excerpt on Fig. 2 depicts the sub-tree of the *Define Schedule* goal, which is refined with the *Schedule Manually* and *Schedule Automatically* tasks. Both tasks must be supported by the system, thus it is an AND-refinement. The automatic scheduling can only be performed if the *Rooms Available* assumption is satisfied, since the system is not able to book additional rooms. Moreover, the *Schedule Automatically* task is refined with design elements – elements that result from design decisions, i.e., they are not mandated by customers or users.

The tasks defined during architectural design (the so called design tasks) in this example are: *Brute Force Algorithm*, *Heuristics-based Algorithm*, and *Select Date*. The first two tasks define algorithms that can be executed to perform the scheduling, while *Select Date* is a task that must be performed once the algorithms find a set of possible dates. Additionally, the automatic scheduling presents two design constraints: it must be performed in less than ten minutes and it must be implemented with web-services. In particular, the selected web-service must be available at least 90 % of the time.

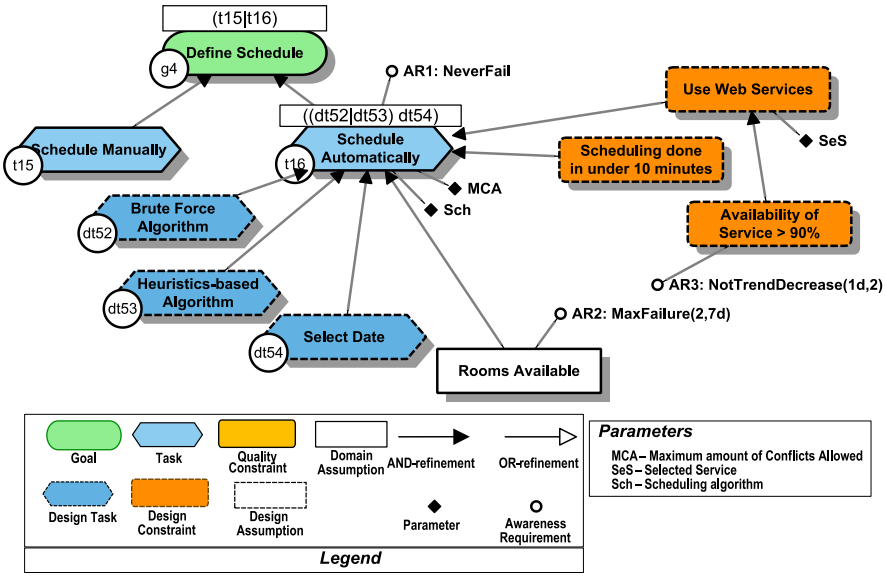

Fig. 2.Excerpt of the design goal model of the Meeting Scheduler system.

Besides goals, tasks, constraints and assumptions, the DGM also contains *flow expressions*, an extension of regular expressions that allow the definition of the execution flow of the system. Six constructs can be used in these expressions: alternative (vertical bar, |), optional (question mark, **?**), sequence (blank space,), repetition (star or plus symbol, * or +), parallelism (hyphen,–), and idle states (*iX*, where *X* is a natural number).

Each flow expression defines the behaviour for the element which it is on top of. For instance in Fig. 2, the *(t15|t16)* expression states that, when the system needs to *Define Schedule*, it may either perform *Schedule Manually (t15)* or *Schedule Automatically (t16)*. Moreover, for the execution of *Schedule Automatically*, the expression is *((dt52|dt53) dt54),* with this meaning: after performing either *Brute Force Algorithm (dt52)* or *Heuristics-based Algorithm (dt53)*, the system will perform the *Select Date* task *(dt54)*.

Lastly, the DGM defines what must be monitored during the system execution (the so called *awareness requirements*), and what can be modified in the system (the *parameters*). In our example, the AR1 awareness requirement, linked to the *Schedule Automatically* task, states that it must never fail. AR2, linked to the *Rooms Available* assumption, indicates that it should be false no more than twice a week (*Max Failure 2, 7d*). On the other hand, AR3 linked to the *Availability of Service* design constraint, defines that its success rate should not decrease for two days in a row (*NotTrendDecrease 1d, 2*).

As illustrated with the aforementioned example, the design goal model allows the integration of requirements and architectural concerns in a single model. Both requirements and architecture elements can be used to specify the system adaptation, with awareness requirements, parameters, relations, and adaptation strategies. In the next subsection we discuss some of the limitations of the proposed process and the design goal model.

3 Limitations

This MULAS design process, as well as the design goal model, presents a series of limitations, regarding the following aspects: expressiveness of the design goal model, heuristics for selecting optimal flows, tool support, and compositional adaptation.

Expressiveness of the design goal model – The design goal model proposed in this paper is based on goal model extension [1]. That extension includes awareness requirements and parameters, which are relevant as they correspond to the control theory concepts of *reference value* and *control input*. However, as a result of the focus on these control theory concepts, two other important concepts have been partially neglected: contribution links and context. The explicit use of contribution links and context annotations may improve the expressiveness of the design goal model. Nonetheless, it is necessary to balance this expressivity with the complexity of the proposed model.

Heuristics for selecting optimal flows – In the MULAS framework, we propose the use of flow expressions to define the possible flows of the system. However, we do not provide any guidance that helps the architect in the decision of which flow may be best in different contexts and scenarios. Further investigation is required in order to identify heuristics, patterns, or techniques to facilitate such decision.

Tool support – A supporting tool was developed specifically to support the MULAS framework. Even though this tool is functional, more effort is required in order to make the tool suitable for public use, related not only to actual development but also to the creation of user documentation, such as user guides or tutorials.

Compositional adaptation – Parameterized adaptation is adaptation related to the modification of variables. In contrast, compositional adaptation is related to modifying structural parts of the system. While we have conducted early endeavours on the latter [6][7] during this research, the MULAS framework is focused only on the former.

4 Ongoing and Future Work

This is an ongoing work, with early results presented in [9][10]. Its most recent results composed a doctoral thesis [8] which includes: detailed description of the MULAS framework; description of a support tool; case studies; experiments. We were able to use this framework for developing information systems, which were verified by means of simulation. Moreover, a mobile differential drive robot was designed and developed using the MULAS framework, providing satisfactory results.

Through an experiment with 15 requirements engineering students, we were able to obtain evidence in favour of the feasibility of the framework. Nonetheless, further experimentation is required in order to properly evaluate and evolve the proposal, specifically in the context of large industrial system.

Another interesting line of research is to adapt the MULAS framework for developing context-sensitive systems [11]. As future work, we intend to investigate the integration of a control theoretic approach (Zanshin) with a context-based one, aiming to expand the expressiveness of the proposal.

Acknowledgments

This work has been supported by the ERC advanced grant 267856 “Lucretius: Foundations for Software Evolution”, as well as by the following Brazilian institutions: FACEPE, CAPES and CNPq.

References

1. Souza, V.E.S.: Requirements-based software system adaptation, Ph.D. Thesis (2012).
2. Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer, 37, 2004, pp. 46–54.
3. Angelopoulos, K., Souza, V.E.S., Pimentel, J.: Requirements and Architectural Approaches to Adaptive Software Systems: A Comparative Study. 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2013, pp. 23–32.
4. Castro, J., Lucena, M., Silva, C., Alencar, F., Santos, E., Pimentel, J. Changing attitudes towards the generation of architectural models. Journal of Systems and Software, 85, 2012, pp. 463–479.
5. Yu, Y., Mylopoulos, J., Lapouchnian, A., Liaskos, S., Leite, J.C.S.P. From Stakeholder Goals to High-Variability Software Design. Technical report csrg-509, University of Toronto, 2005.
6. 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 Journal, 17-4, 2012, pp.259–281.
7. Dermeval, D., Soares, M., Alencar, F., Santos, E., Pimentel, J., Castro, J., Lucena, M., Silva, C., Souza, C.: Towards an i*-based Architecture Derivation Approach. Fifth International i* Workshop, 2011, pp. 66-71.
8. Pimentel, J.: Systematic Design of Adaptive Systems — A Control-Based Framework, Ph.D. thesis (2015). Available at http://www.cin.ufpe.br/~ler/supplement/istar2015/
9. Pimentel, J.; Angelopoulos, K.; Souza, V. E. S.; Mylopoulos, J.; Castro J. From Requirements to Architectures for Better Adaptive Software Systems. 6th International i* Workshop, 2013, pp. 91-96.
10. Pimentel, J. et al. From requirements to statecharts via design refinement. 29th Symposium on Applied Computing, 2014, pp. 995–1000.
11. Vilela, J.; Castro, J.; Pimentel; J.; Soares, M.; Lima, P.; Lucena, M. Deriving the behavior of context-sensitive systems from contextual goal models. 30th Symposium on Applied Computing, 2015.