Language and Framework Requirements for Adaptation Models

Thomas Vogel and Holger Giese

Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
{thomas.vogel|holger.giese}@hpi.uni-potsdam.de

Abstract. Approaches to self-adaptive software systems use models at runtime to leverage benefits of model-driven engineering (MDE) for providing views on running systems and for engineering feedback loops. Most of these approaches focus on causally connecting runtime models and running systems, and just apply typical MDE techniques, like model transformation, or well-known techniques, like event-condition-action rules, from other fields than MDE to realize a feedback loop. However, elaborating requirements for feedback loop activities for the specific case of runtime models is rather neglected.
Therefore, we investigate requirements for *Adaptation Models* that specify the analysis, decision-making, and planning of adaptation as part of a feedback loop. In particular, we consider requirements for a modeling language of adaptation models, and for a framework as the execution environment of adaptation models. Moreover, we discuss patterns for using adaptation models within the feedback loop regarding the structuring of loop activities. The patterns and the requirements for adaptation models influence each other, which impacts the design of the feedback loop.

1 Introduction

Self-adaptation capabilities are often required for modern software systems to dynamically change the configuration in response to changing environments or goals [5]. *Models@run.time* are a promising approach for self-adaptive software systems since models may provide appropriate abstractions of a running system and its environment, and benefits of model-driven engineering (MDE) are leveraged to the runtime phases of software systems [3].

Most models@run.time efforts to self-adaptive software systems focus on causally connecting models to running systems, and just apply typical or well-known techniques from MDE or other fields on top of these models. These techniques are used for engineering a feedback loop that controls self-adaptation by means of *monitoring* and *analyzing* the running system and its environment, and the *planning* and *execution* of changes to the running system [13].

For example, the causal connection has been a topic for discussions at the last two workshops on models@run.time [1, 2], or the work of [17] particularly addresses the causal connection, and it just applies MDE techniques, like model transformation, on top to show their technical feasibility. We proposed an approach to use incremental model synchronization techniques to maintain mul-

tiple, causally connected runtime models at different abstraction levels, and thereby, we support the monitoring and the execution of adaptations [18, 19].

While causal connections provide basic support for monitoring and for executing changes, they do not cover the analysis and planning steps of a feedback loop, which decide *if* and *how* the system should be adapted. For these steps, techniques originating from other fields than MDE are used. Most approaches [4, 7, 8, 11, 12, 14] employ rule-based mechanisms in some form of event-condition-action rules that exactly specify when and how adaptation should be performed, and thus, the designated target configuration is predefined. In contrast, search-based techniques just prescribe goals that the system should achieve. Triggered by conditions or events and guided by utility functions they try to find the best or at least a suitable target configuration fulfilling these goals [10, 15].

All these approaches focus on applying such decision-making techniques for the analysis and planning steps, but they do not systematically investigate the requirements for such techniques in conjunction with models@run.time. Eliciting these requirements might help in engineering new or tailored decision-making techniques for the special case of models@run.time approaches to self-adaptive systems. Therefore, we elaborate requirements for such techniques by taking an MDE perspective. The techniques should be specified by models, which we named *Adaptation Models* in an attempt to categorize runtime models [20]. However, the categorization does not cover any requirements for runtime models.

In this paper, we discuss requirements for adaptation models, and in particular requirements for languages to create such models and for frameworks that employ and execute such models within a feedback loop. By language we mean a broad view on metamodels, constraints, and model operations, which are all used to create and apply adaptation models. Moreover, we discuss patterns for using adaptation models within the feedback loop. The patterns and the requirements for adaptation models influence each other, which impacts the design of the feedback loop by providing alternatives for structuring loop activities.

The rest of the paper is structured as follows. Section 2 discusses related work, and Section 3 sketches the role of adaptation models in self-adaptive systems. Section 4 discusses the requirements for adaptation models, while Section 5 presents different patterns of employing adaptation models within a feedback loop. Finally, the paper concludes and outlines future work in Section 6.

2 Related Work

As already mentioned in the previous section, most models@run.time approaches to self-adaptive software systems focus on applying techniques for decision-making and do not systematically elaborate on their requirements [4, 7–12, 14, 15]. A few approaches merely consider the requirement of performance and efficiency for their adaptation mechanisms to show and evaluate the applicability at runtime [10, 11, 15]. Likewise, several decision-making mechanisms are presented in [16] that primarily discusses their specifics for mobile applications in ubiquitous computing environments by means of performance and scalability regarding the size of the managed system and its configuration space. In general,

rule-based mechanisms are considered as efficient since they exactly prescribe the whole adaptation, while for search-based approaches performance is critical and often improved by applying heuristics or by reducing the configuration space.

This is also recognized by [9] that attests efficiency and support for early validation as benefits for rule-based approaches. However, they suffer from scalability issues regarding the management and validation of large sets of rules. In contrast, search-based approaches may cope with these scalability issues, but they are not as efficient as rule-based approaches and they provide less support for validation. As a consequence, a combination of rule-based and search-based techniques is proposed in [9] to balance their benefits and drawbacks.

To sum up, if requirements or characteristics of decision-making techniques are discussed, these discussions are limited to performance, scalability, and support for validation, and they are not done systematically. One exception is the work of Cheng [6] who discusses requirements for a self-adaptation language that is focused on specifying typical system administration tasks. However, the requirements do not generally consider self-adaptive software systems and they do not address specifics of models at runtime. Nevertheless, some of the requirements that are described in this paper are derived from this work.

3 Adaptation Models

Before discussing requirements for adaptation models, we sketch the role of these models based on a conceptual view on a feedback loop as depicted in Figure 1.

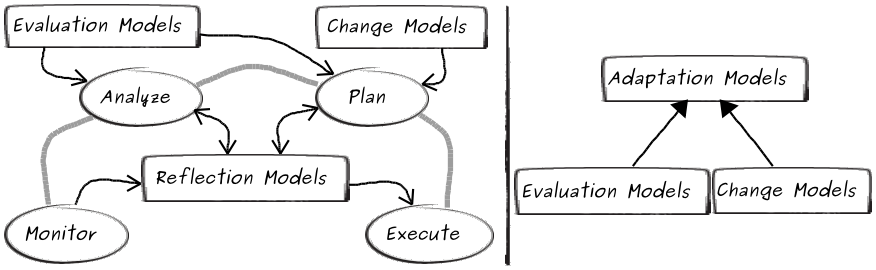

Fig. 1. Feedback Loop and Runtime Models (cf. [20])

The steps of monitoring and analyzing the system and its environment, and the planning and execution of changes are derived from the autonomic computing element [13], while we discussed the different models and a usage scenario of models in the loop in [20]. *Reflection Models* describe the running system and its environment and they are causally connected to the system. According to observations of the system and environment, the monitor updates the reflection models. Reasoning on these models is done by the analyze step to decide whether the system fulfills its goals or not, and thus, whether adaptation is required or not. The reasoning is specified by *Evaluation Models*, which can be constraints that are checked on reflection models. If adaptation is required, the planning step devises a plan defining how the system should be adapted, which is guided

by *Change Models* to explore the system's variability or configuration space. Deciding on the designated target configuration is guided by evaluation models to analyze different adaptation options, and the selected option is applied on reflection models. Finally, the execute step involved in the causal connection performs the adaptations on the running system to move it to the target configuration.

By *Adaptation Models* we generally consider evaluation and change models regardless of the concrete rule-based or search-based techniques that are employed for the analysis and planning steps, and thus, for the decision-making. This view on adaptation models is similar to [14], which just presents one adaptation model for the specific approach, but no general discussion of such models.

4 Requirements for Adaptation Models

In this section we describe requirements for adaptation models to be used in self-adaptive software systems to analyze and decide on adaptation needs, and to plan and decide on how to adapt the running system. We assume that the self-adaptive system employs runtime models, which influences the requirements for adaptation models. At first, we discuss requirements for a modeling language that is used to create adaptation models. Then, we elaborate the requirements for a framework as the execution environment for adaptation models. Being in the early requirements phase, we take a broad MDE view on the notion of languages as combinations of metamodels, constraints, and model operations, which are all used to create and apply adaptation models.

Likewise to the common understanding that requirements for real-world applications cannot be completely and definitely specified at the beginning of a software project, we think that the same is true for the requirements discussed here. It is likely that some of these requirements may change, become irrelevant, or new ones emerge when engineering concrete adaptation models for a specific self-adaptive system and domain. Thus, we do not claim that the requirements are complete and finalized with respect to their enumeration and definitions.

4.1 Language Requirements for Adaptation Models

Language requirements (LR) for adaptation models can be divided into functional and non-functional ones. Functional requirements target the concepts that are either part of adaptation models or that are referenced by adaptation models. These concepts are needed for the analysis, decision-making, and planning. Thus, functional requirements determine the expressiveness of the language. In contrast, non-functional language requirements determine the quality of adaptation models. At first functional, then non-functional requirements are discussed.

Functional Language Requirements

LR-1 ***Functional Specification/Goals***: Enabling a self-adaptive system to continuously provide the desired functionality to users or other systems, adaptation models have to know about the current functional specification or goals of the system. The functional specification or goals define *what* the system should

do, and this information needs to be available in an operationalized form to relate it with the actual behavior of the running system. This is the foundation for adapting the functional behavior of the system.

LR-2 ***Quality Dimensions***: While LR-1 considers *what* the system should do, quality dimensions address *how* the system should provide the functionality in terms of quality of service (QoS). To support QoS-aware adaptations, quality dimensions, like performance or security, must be characterized by adaptation models (cf. [6]).

LR-3 ***Preferences***: Since multiple quality dimensions (LR-2) may be relevant for the managed system, preferences across the dimensions must be expressed to trade-off and balance competing qualities (cf. [6]). Likewise, preferences for goals (LR-1) are necessary if several valid behavioral alternatives are feasible and not distinguished by the quality dimensions.

Thus, the language for adaptation models must incorporate the concepts of goals (LR-1), quality dimensions (LR-2), and preferences (LR-3) in an operationalized form, such that they can be referenced or described and automatically processed by concrete adaptation models. This operationalized form should be derived from the requirements of the self-adaptive system. Goals, quality dimensions, and preferences serve as references for the running system as they state what the system should do and how it should be.

LR-4 ***Access to Reflection Models***: Adaptation models must reference and access reflection models to obtain information about the current situation of the running system and its environment for analysis, and to change the reflection models to effect adaptations. Thus, a language for adaptation models must be based on the languages of reflection models.

LR-5 ***Events***: Adaptation models should reference information from events emitted by the monitor step when updating the reflection models due to runtime phenomena of the system. Besides serving as a trigger for starting the decision-making process, events support locating the phenomena in the system and reflection models (LR-4). Thus, evaluating the system and its environment (LR-6) may start right from the point in the reflection models where the phenomena have occurred. Events provided by the monitor step and signaling changes in the running system support reactive adaptation, while the decision-making process for proactive adaptations can be triggered periodically.

LR-6 ***Evaluation Conditions***: A language for adaptation models must support the specification of conditions to evaluate the running system and its environment (cf. [6]). These conditions relate the goals (LR-1), quality dimensions (LR-2), and preferences (LR-3) to the actual running system represented by reflection models (LR-4). Therefore, conditions may refer to events notifying about runtime phenomena (LR-5) as a starting point for evaluation, and they should be able to capture complex structural patterns for evaluating the software architecture of the running system.

LR-7 ***Evaluation Results***: Adaptation models must capture the results of computing the evaluation conditions (LR-6), because these results identify and decide on adaptation needs especially when the conditions are not met by the

system. Adaptation models may annotate and reference the evaluation results in reflection models (LR-4) to locate adaptation needs in the running system.

LR-8 ***Adaptation Options***: Adaptation models must capture the variability of the system to know the options for adaptation. These options define the configuration space for the system and how reflection models (LR-4) can be modified to adapt the running system.

LR-9 ***Adaptation Conditions***: Adaptation models must consider adaptation conditions since not all adaptation options (LR-8) are feasible in every situation. Thus, conditions should constrain all adaptation options to applicable ones for certain situations (cf. [6]). To characterize a situation for an adaptation option, conditions should refer to reflection models (LR-4), events (LR-5), evaluation results (LR-7), or other adaptation options. Likewise to such pre-conditions for adaptation options, post-conditions and invariants should be considered.

LR-10 ***Adaptation Costs and Benefits***: Adaptation models should characterize costs and benefits of adaptation options (LR-8) as a basis to select among several possible options in certain situation (cf. [6]). Costs should indicate that adaptations are not for free, and benefits should describe the expected effects of options on the goals (LR-1) and quality dimensions (LR-2) of the system. By relating costs and benefits to the preferences of the system (LR-3), suitable adaptation options should be selected and applied on the reflection models.

LR-11 ***History of Decisions***: Adaptation models should capture history of decisions, like evaluation results (LR-7) or applied adaptation options (LR-8) to enable learning mechanisms for improving future decisions.

Non-functional Language Requirements

LR-12 ***Modularity, Abstractions and Scalability***: An adaptation model should be a composition of several submodels rather than a monolithic model to cover all concepts for decision-making. For example, evaluation conditions (LR-6) and adaptation options (LR-8) need to be part of the same submodel, and even different adaptation options can be specified in different submodels. Thus, the language should support modular adaptation models. Moreover, the language should enable the modeling at different abstraction levels for two reasons. First, the level depends on the abstraction levels of the employed reflection models (LR-4), and second, lower level adaptation model concepts should be encapsulated and lifted to appropriate higher levels. For example, several simple adaptation options (LR-8) should be composable to complex adaptation options. Language support for modularity and different abstractions promote scalability of adaptation models.

LR-13 ***Side Effects***: The language should clearly distinguish between concepts that cause side effects on the running system and those that do not. For example, computing an evaluation condition (LR-6) should not affect the running system, while applying an adaptation option (LR-8) finally should. Making the concepts causing side effects explicit is relevant for consistency issues (FR-1).

LR-14 ***Parameters***: The language should provide constructs to parameterize adaptation models. Parameters can be used to adjust adaptation models at runtime, like changing the preferences (LR-3) according to varying user needs.

LR-15 ***Formality***: The language should have a degree of formality that enables online and offline validation or verification of adaptation models, e.g., to detect conflicts or thrashing effects in the adaptation mechanisms.

LR-16 ***Reusability***: The core concepts of the language for adaptation models should be independent of the languages used for reflection models in an approach. This leverages the reusability of the language and adaptation models.

LR-17 ***Ease of Use***: The design of the language should consider its ease of use, because adaptation models are created by software engineers. This influences, among others, the modeling paradigm, the notation, and the tool support. Preferably the language should be based on a declarative modeling paradigm, which is often more convenient and less error-prone than an imperative one. Imperative constructs should be deliberately used in the language. Likewise, appropriate notations and tools are required to support an engineer in creating, validating or verifying adaptation models.

4.2 Framework Requirements for Adaptation Models

In the following we describe framework requirements (FR) for adaptation models. By framework we consider the execution environment of adaptation models, which determines how adaptation models are employed and executed in the feedback loop. Thus, only requirements specific for such a framework are discussed. Typical non-functional requirements for software systems, like reliability or security, are also relevant for adaptation mechanisms, but they are left here.

FR-1 ***Consistency***: The execution or application of adaptation models should preserve the consistency of reflection models and thus, the consistency of the running system. For example, when adapting a causally connected reflection model, the corresponding set of model changes should be performed atomically and correctly. Thus, the framework should evaluate the invariants, pre- and post-conditions (LR-9) for adaptation options (LR-8) at the model level, before adaptations are executed to the running system.

FR-2 ***Incrementality***: The framework should leverage incremental techniques to apply or execute adaptation models to promote efficiency. For example, events (LR-5) or evaluation results (LR-7) annotated to reflection models should be used to directly locate starting points for evaluation or adaptation planning, respectively. Or, adaptation options (LR-8) should be incrementally applied on original reflection models rather than on copies. Incrementality could avoid costly operations, like copying or searching potentially large models.

FR-3 ***Reversibility***: Supporting incremental operations on models (FR-2), the framework should provide the ability to incrementally reverse performed operations. For example, the configuration space has to be explored for adaptation planning by creating a path of adaptation options (LR-8) applied on reflection models. Finding a suitable path might require to turn around and to try alternative directions without completely rejecting the whole path. Thus, *do* and *undo* of operations leverages, among others, incremental planning of adaptation.

FR-4 ***Priorities***: The framework should utilize priorities to organize modular adaptation models (LR-12) to efficiently and easily identify first entry points

for executing or applying adaptation models. For example, priorities can be assigned to different evaluation conditions (LR-6) based on their criticality, and the framework should check the conditions in decreasing order of their criticality.

FR-5 ***Time Scales***: The framework should simultaneously support different time scales of analysis and adaptation planning. For example, in known and mission-critical situations quick and precisely specified reactions might be necessary (cf. rule-based techniques), while in other situations comprehensive and sophisticated reasoning and planning are feasible (cf. search-based techniques).

FR-6 ***Flexibility***: The framework should be flexible by allowing adaptation models to be added, removed and modified at runtime. This supports including learning effects, and it considers the fact that all conceivable adaptation scenarios could not be anticipated at development-time. Moreover, it is a prerequisite of hierarchical control where the adaptation mechanisms as specified by adaptation models are managed by another, higher level control loop [13, 20].

Using these language and framework requirements for adaptation models, we investigate their dependencies on different patterns or designs of feedback loops.

5 Feedback Loop Patterns for Adaptation Models

In the following we discuss feedback loop patterns for adaptation models and how the functional language requirements (cf. Section 4.1) map to these patterns, while considering the framework requirements (cf. Section 4.2). The non-functional language requirements are not further addressed here because they are primarily relevant for designing a language for adaptation models and not for actually applying such models. The patterns differ in the coupling of the analysis and planning steps of a feedback loop, which influences the requirements for adaptation models. Moreover, the adaptation model requirements likely impact the patterns and designs of the loop. Thus, this section provides a preliminary basis for investigating dependencies between requirements and loop patterns.

5.1 Analysis and Planning – Decoupled

The first pattern of the feedback loop depicted in Figure 2 decouples the analysis and planning steps as originally proposed (cf. Section 3). The figure highlights functional language requirements (LR) at points where the concepts of the corresponding requirements are relevant. This does not mean that adaptation models must cover all these points, but they must know about the concepts.

In response to events notifying about changes in the running system or environment, the monitor updates the reflection models and annotates the events (LR-5) to these models. The analyze step uses these events to locate the changes in the reflection models and to start reasoning at these locations. Reasoning is specified by *evaluation models* defining evaluation conditions (LR-6) that relate the goals (LR-1), qualities (LR-2), and preferences (LR-3) to the characteristics of the running system. These characteristics are obtained by accessing reflection models (LR-4). Analysis is performed by evaluating the conditions and probably enhanced by consulting past analyses (LR-11). This produces analysis results (LR-7) that are annotated to the reflection models to indicate adaptation needs.

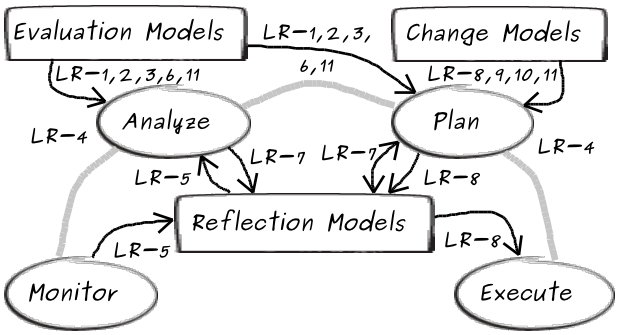

Fig. 2. Decoupled Analysis and Planning Steps

The planning step uses these results (LR-7) attached to reflection models (LR-4) to devise a plan for adaptation. Planning is based on *change models* specifying adaptation options (LR-8) and their conditions (LR-9), costs and benefits (LR-10). This information and probably plans devised in the past (LR-11) are used to find suitable adaptation options to create potential target configurations by applying these options on reflection models. These reflection models prescribing alternative target configurations are analyzed by applying evaluation models to select the best configuration among them. In contrast to the analyze step that uses evaluation models to reason about the current configuration (descriptive reflection models), the planning step uses them to analyze potential target configurations (prescriptive reflection models). Finally, the selected adaptation options (LR-8) are effected to the running system by the execute step.

This pattern is similar to the generic idea of search-based approaches, since planning is done by exploring adaptation options (LR-8, 9, 10) that are evaluated (LR-6, 7, 11) for their fitness for the preferenced system goals (LR-1, 2, 3) based on the current situations of the system and environment (LR-4). Explicitly covering all language requirements for adaptation models, this pattern rather targets comprehensive and sophisticated analysis and planning steps working at longer time scales (FR-5), while efficiency concerns could be tackled by incrementality.

This pattern leverages incrementality (FR-2) since the coordination between different steps of the loop is based on events, analysis results, and applied adaptation options, which directly point to location in reflection models for starting analysis, planning, or executing changes. Moreover, analysis and planning steps may incrementally interleave. Based on first analysis results that are produced by evaluation conditions with highest priorities (FR-4), a planning process might start before the whole system and environment have been completely analyzed. However, incrementality requires the reversibility of performed operations (FR-3) to ensure consistency of reflection models (FR-1), e.g., when alternative adaptation options are tested online on reflection models and finally discarded.

In our categorization of runtime models, we distinguished two kinds of adaptation models based on the feedback loop steps: evaluation models for the analyze step, and change models for the planning step [20]. This distinction is backed by the different language requirements each of these kinds of models are addressing.

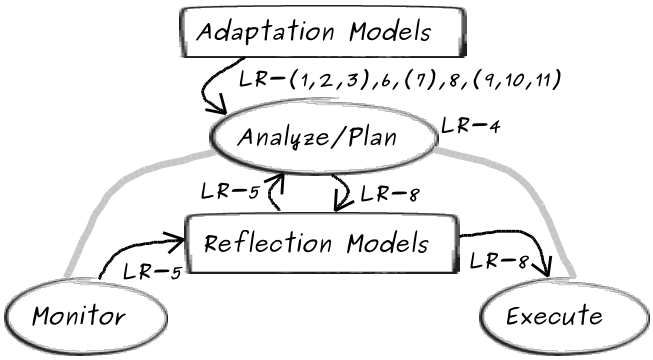

Fig. 3. Coupled Analysis and Planning Steps

5.2 Analysis and Planning – Coupled

In contrast to decoupling the analyze and planning steps, they can be closely integrated into one step, which is sketched in Figure 3. Based on events (LR-5) the integrated analyze/plan step computes evaluation conditions (LR-6) that are directly mapped to adaptation options (LR-8). If a condition is met, the corresponding adaptation options are applied on the reflection models and finally executed to the running system. Access to reflection models (LR-4) is realized by the analyze/plan step as a link between adaptation and reflection models.

In Figure 3, the language requirements written in brackets are not explicitly covered by adaptation models, because this pattern precisely specifies the adaptation mechanism by directly relating evaluation conditions to the application of adaptation options. Thus, this relation or mapping implicitly covers some of the language requirements listed in brackets. For example, it is assumed that the applied adaptation options modify the running system's configuration in a way that fulfills the desired goals, qualities and preferences (LR-1, 2, 3).

Considering the events and the mapping of evaluation conditions to adaptation options, this pattern is similar to rule-based approaches using event-conditions-action rules. Likewise to such rules covering the whole decision-making process, and due to the integration of analysis and planning into one step, the clear distinction between evaluation and change models is blurred. Therefore, both kinds of models are combined to adaptation models in Figure 3.

Thus, this pattern targets adaptation mechanisms requiring quick reactions to runtime phenomena by enabling adaptation at rather short time scales (FR-5). Moreover, efficiency is improved by incrementality (FR-2) and priorities (FR-4). The steps may incrementally coordinate each other through locating events and applied adaptation options in reflection models in order to incrementally evaluate conditions and execute adaptation options to the running system. Priorities may be used to order evaluation conditions for quickly identifying critical situations that need urgent reactions, while conditions for non-critical situations can be evaluated without strict time constraints.

The framework requirement of consistency (FR-1) is not explicitly covered, since it is assumed that the mapping of condition to adaptation options preserves

consistency by design of such rule-based mechanisms. Since these mechanisms strictly prescribe the adaptation, there need not to be any options left that have to be decided at runtime. This reduces the need for reversible operations (FR-3).

5.3 Discussion

Regarding the two different feedback loop patterns and their effects on adaptation models, we can make two observations. First, it might be necessary to combine both patterns in a self-adaptive system if simultaneous support for different time scales (FR-5) is required, or if the nature of a self-adaptive system requires both flavors of rule-based and search-based decision-making mechanisms. Second, we think that these two patterns span a range of several other patterns. By explicitly covering more and more language requirements, the adaptation models and thus, the adaptation mechanisms get more elaborate, and we may move stepwise from the coupled pattern (cf. Section 5.2) toward the decoupled one (cf. Section 5.1). Which pattern and adaptation models suit best depends on the concrete self-adaptive system, especially on the system's domain requirements.

Finally, the requirement of flexibility (FR-6) has not been discussed for the two patterns. However, it is relevant for both of them since it is usually not possible to anticipate all adaptation scenarios at development-time. Thus, changing adaptation models at runtime is required to adjust the adaptation mechanisms.

6 Conclusion and Future Work

In this paper we have elaborated the requirements for adaptation models that specify the decision-making process in self-adaptive software systems using models@run.time. In particular, requirements for a modeling language incorporating metamodels, constraints, and model operations for creating and applying adaptation models have been discussed, as well as requirements for a framework that executes adaptation models. Moreover, we discussed patterns of a self-adaptive system's feedback loop with respect to the requirements for adaptation models.

As future work, we plan to analyze existing approaches to self-adaptation regarding their fitness to the requirements presented in this paper. This analysis is challenging since it requires in-depth descriptions of approaches, which are often not available. However, it would give us feedback on the relevance and completeness of these requirements, and it may identify further need for research on adaptation models. Moreover, we want to engineer a language and framework for adaptation models, which are suitable for our approach [18, 19]. A particular challenge is to engineer a single language that fulfills most of the requirements presented in this paper, and it likely will be required to integrate several languages into the framework. However, having profound knowledge about the requirements is a promising start to systematically engineer adaptation models for self-adaptive software systems based on models@run.time techniques.

References

1. Bencomo, N., Blair, G., Fleurey, F., Jeanneret, C.: Summary of the 5th International Workshop on Models@run.time. In: Dingel, J., Solberg, A. (eds.) MODELS'10 Workshops, LNCS, vol. 6627, pp. 204–208. Springer (2011)

2. Bencomo, N., Blair, G., France, R., Munoz, F., Jeanneret, C.: 4th International Workshop on Models@run.time. In: Ghosh, S. (ed.) MODELS'09 Workshops, LNCS, vol. 6002, pp. 119–123. Springer (2010)
3. Blair, G., Bencomo, N., France, R.B.: Models@run.time: Guest Editors' Introduction. Computer 42(10), 22–27 (2009)
4. Chauvel, F., Barais, O.: Modelling Adaptation Policies for Self-Adaptive Component Architectures. In: M-ADAPT'07. pp. 61–68 (2007)
5. Cheng, B.H., Lemos, R., Giese, H., Inverardi, P., Magee, J. et al.: Software Engineering for Self-Adaptive Systems: A Research Roadmap. In: Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer (2009)
6. Cheng, S.W.: Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, USA (2008)
7. Dubus, J., Merle, P.: Applying OMG D&C Specification and ECA Rules for Autonomous Distributed Component-based Systems. In: Models@run.time'06 (2006)
8. Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., Jézéquel, J.M.: Modeling and Validating Dynamic Adaptation. In: Chaudron, M. (ed.) MODELS'08 Workshops, LNCS, vol. 5421, pp. 97–108. Springer (2009)
9. Fleurey, F., Solberg, A.: A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems. In: Schürr, A., Selic, B. (eds.) MODELS'09. LNCS, vol. 5795, pp. 606–621. Springer (2009)
10. Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E.: Using Architecture Models for Runtime Adaptability. Software 23(2), 62–70 (2006)
11. Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. Computer 37(10), 46–54 (2004)
12. Georgas, J.C., Hoek, A., Taylor, R.N.: Using Architectural Models to Manage and Visualize Runtime Adaptation. Computer 42(10), 52–60 (2009)
13. Kephart, J.O., Chess, D.: The Vision of Autonomic Computing. Computer 36(1), 41–50 (2003)
14. Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J.M., Solberg, A., Dehlen, V., Blair, G.: An Aspect-Oriented and Model-Driven Approach for Managing Dynamic Variability. In: Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M. (eds.) MODELS'08. LNCS, vol. 5301, pp. 782–796. Springer (2008)
15. Ramirez, A.J., Cheng, B.H.: Evolving Models at Run Time to Address Functional and Non-Functional Adaptation Requirements. In: Models@run.time'09. CEUR-WS.org, vol. 509, pp. 31–40 (2009)
16. Rouvoy, R.: Requirements of mechanisms and planning algorithms for self-adaptation. Deliverable D1.1 of MUSIC (EU-FP6 project) (2007)
17. Song, H., Huang, G., Chauvel, F., Sun, Y.: Applying MDE Tools at Runtime: Experiments upon Runtime Models. In: Models@run.time'10. CEUR-WS.org, vol. 641, pp. 25–36 (2010)
18. Vogel, T., Giese, H.: Adaptation and Abstract Runtime Models. In: SEAMS'10. pp. 39–48. ACM (2010)
19. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Incremental Model Synchronization for Efficient Run-Time Monitoring. In: Ghosh, S. (ed.) MODELS'09 Workshops, LNCS, vol. 6002, pp. 124–139. Springer (2010)
20. Vogel, T., Seibel, A., Giese, H.: The Role of Models and Megamodels at Runtime. In: Dingel, J., Solberg, A. (eds.) MODELS'10 Workshops, LNCS, vol. 6627, pp. 224–238. Springer (2011)