The Role of Models in Self-adaptive and Self-healing Systems

Jens Happe, Heiko Koziolek, Umesh Bellur, Holger Giese, Wilhelm Hasselbring, Robert Laddaga, Tiziana Margaria, Josu Martinez, Christian Müller-Schloer, and Roland Reichle

FZI Forschungszentrum Informatik jhappe@fzi.de
ABB AG, Forschungszentrum Deutschland heiko.koziolek@de.abb.com
Indian Institute of Technology Bombay umesh.bellur@iitb.ac.in
Hasso-Plattner-Institut - Potsdam holger.giese@hpi.uni-potsdam.de
Universität Kiel wha@informatik.uni-kiel.de
BBN Technologies - Cambridge MA rladdaga@.csail.mit.edu
Universität Potsdam margaria@cs.uni-potsdam.de
University College - Dublin josu.martinez@ucd.ie
Leibniz-Universität Hannover cms@sra.uni-hannover.de
Universität Kassel reichle@vs.uni-kassel.de

Abstract. Self-healing and self-adaptive systems dynamically react on changes in the environment. They enable software systems to adjust to new conditions and work optimally even in unstable environments. However, such systems have to cope with an ever increasing complexity and size of software systems. In order to handle such systems, models are an efficient means for analysis, control, and documentation. Furthermore, hierarchically structured models can make self-healing and self-adaptation manageable. In this report, we discuss several questions that address the role of models in self-healing and self-adaptive systems. We outline today's challenges and present different viewpoints on the application and benefit of models.

1 Introduction

Self-healing and self-adaptive systems dynamically alter their behaviour at run-time to react on changing conditions of the surrounding environment (physical or computational). These systems have to deal with an increasing complexity of the systems that are managed. In order to achieve correct, reliable, and efficient adaptations, models are an essential means to describe, analyse, and predict the behaviour of self-adaptive systems.

Models can support self-healing and self-adaptive systems during their whole lifecycle. In early development phases, they enable the design, validation, and documentation of such systems. At runtime, they can assist control systems and decision making. In self-adaptive systems, various kinds of models can be used for very different purposes. For example, queueing network models are used for software performance analyses; software architecture models, such as UML, are used for system design or re-

Dagstuhl Seminar Proceedings 09201
Combinatorial Scientific Computing
http://drops.dagstuhl.de/opus/volltexte/2009/2100

configuration at runtime; Petri nets are applied to verify the correctness of dynamic systems [4]. The importance of models for self-adaptive systems has already been recognised by several researchers [2, 3].

In this report, we discusses questions that address the role of models in self-healing and self-adaptive systems. This report is based on the discussion of the working group "Models and Learning" of Dagstuhl Seminar 9201 "Self-Healing and Self-Adaptive Systems" [1]. As such, the report is meant to foster discussions and future research. For this purpose, we address questions and challenges of using models in self-healing and self-adaptive system. For each question, we propose different perspectives. In the following, we first give an introductory example of a self-adaptive system. Then, we continue with a discussion of questions related to the usage of models in self-adapted and self-healing systems.

2 The Role of Models in Self-Healing and Self-Adaptive Systems: An Example

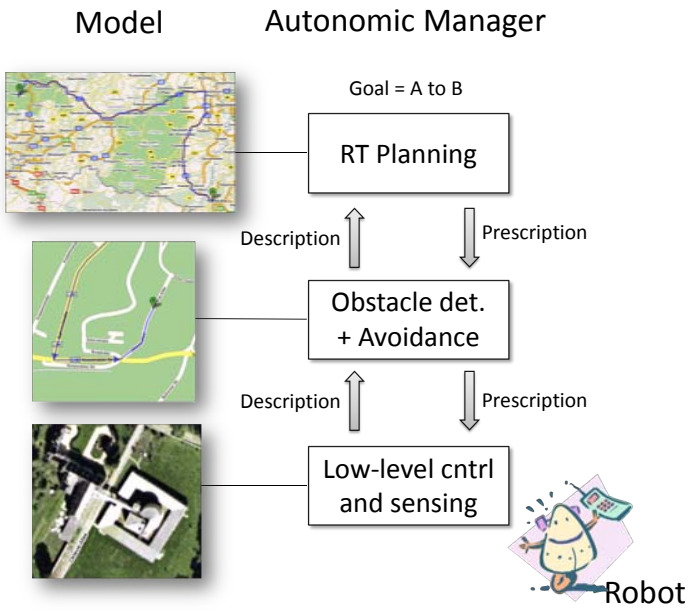

Fig. 1. Example: Autonomic manager of a robot.

Figure 1 illustrates the role of models for self-adaptive and self-healing systems using the example of a robot's autonomic manager. In this scenario, models are used on different levels of abstraction to control and manage the robot. On the highest level, route planning computes a path from location `A` to location `B`. It does not consider any

details about the terrain or potential obstacles. The route determined on this level defines the constraints and objective for further low-level planning. In this sense, the model representing the route is said to be *prescriptive* with respect to the lower level models (the obstacle detection and avoidance in Figure 1). The information that is provided by the middle level (e.g., a specific obstacle) to the higher level control describes the actual situation of the robot. For example, an obstacle that cannot be passed. In this case, new route planning must be initiated. Information that is passed from a lower level to higher levels is (at least in our example) based on observations of the real system and, thus, called *descriptive*. The obstacle detection and avoidance determines constraints and goals for the lowest level of our example, the low level control and sensing. On this level, control actions are determined based on the information provided by the higher levels. The data of the robot's sensors (its position and detected obstacles) is passed to the middle level control. The model of the actual situation is used to react on the situation and to update models of the higher layers.

This simple example illustrates most of the challenges with respect to models that occur in self-adaptive and self-healing systems. Models can take different roles depending on the entity that created the model, the information the model is based on, and the usage of the model. Different layers must interact using models as representatives of the real world; different models represent the same real world entity on different levels of abstraction and, thus, have to be consistent (or at least consistent enough). Within the same system, models are used for very different purposes. For example, models are used for planning, (e.g., finding a route from `A` to `B`) or acting (low-level control). Both models contain related information presented in entirely different ways. Furthermore, if models describe a system which changes over time, they have to reflect the system's evolution and they have to be able to deal with inconsistencies.

3 Challenges with respect to the usage of models

Models play can ease the development and analysis of self-adaptive and self-healing systems at design-time and support their management at runtime. They are abstractions of real world entities that allow humans and computers to reason on the structure and behaviour of a system. They are (or should be) designed according to a specific goal or purpose that contributes to the overall execution of a system. In the following, we discuss some of the questions that arise with respect to models for self-adaptive and self-healing systems.

What is modelled? The example in Figure 1 already points out information assets that can be captured in terms of models. In general, models describe the structure and behaviour of the system that is to be managed. Additionally, goals, objectives or constraints can be reflected. For self-adaptive and self-healing systems, the managed system is, in many cases, a software/hardware system whose architecture can be represented in terms of models. The software/hardware architecture contains representatives of objects, components, connections, behaviour, users, configurations, and any form of dependencies. In theory, there is no limit what can be modelled and managed by a self-adaptive or self-healing system.

What is the use/purpose of the model? Models follow a distinct purpose. For our example in Figure 1, we have models for route panning, models for obstacles and their avoidance, as well as models for low level control and actions. All these models are related to each other. They follow an overall goal: getting the robot from `A` to `B`. However, each of these models has its own specific purpose, such as describing a route from `A` to `B` or describing a specific control action that can be executed by the robot. The specific purpose of a model determines the information that the model must contain as well as the information it must neglect. For example, the model for low level control contains very specific information about the currency and duration necessary to turn the left wheel of the robot by 90° but neglects any information about the overall route. With respect to self-adaptive and self-healing systems, the purpose of a model can be very different. It ranges from prediction of quality attributes (such as performance and reliability) over documentation and explanation to specification.

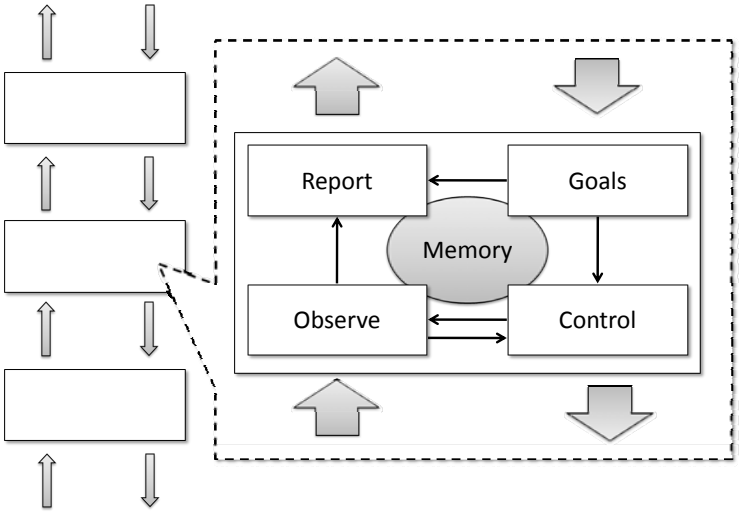

Fig. 2. Internal control structure of models.

How do the models contribute to the operation of the system? Models contribute in different ways to the management of self-adaptive and self-healing systems. Figure 2 shows how each level on a hierarchically structured self-adaptive or self-healing system (such as the one in Figure 1) influences the system's overall behaviour. The figure shows the system focussing on models and data flow. The typical control cycle [2] (consisting of the activities: act, collect, analyse, and decide) is encapsulated in the more general control action.

In Figure 2, control is the central element that determines the goals for the lower layer and reacts on observations. It is driven by goals that are defined by an upper layer (prescriptive models). Observation is configured and steered by control. It collects data

from the current and the lower layer(s). Reporting transmits aggregated information of this layer to the upper layers (descriptive models). In Figure 2, memory is attached to all areas. It contains models that reflect the current state of the system. These models can be updated according to the observations and can be used for control.

How are models used in the software life-cycle? Depending on the purpose of a model, it will be used at different stages of the software life-cycle. In general, we roughly distinguish design-time models and runtime models. *Design-time models* are used prior to system execution in order to document, verify, or analyse the system under study. These models can be seen as specifications that constrain the system development and decide on degrees of freedom. In this sense, design-time models are prescriptive. By contrast, *runtime models* are used during the execution of the system for adaptation and control.

However, the distinction is not as strict as it might seem on the first sight. Runtime models are often based on design-time models. For example, they are used as starting point for runtime models. Such models evolve based on observations during system execution. However, in a more extreme case, descriptive runtime models are solely derived from observations. Furthermore, design-time models can be used as reference/prescriptive model at runtime. In such a scenario, design-time models serve as a specification against which the runtime models can be checked.

Considering the tight coupling of both types of models, the distinction becomes artificial. However, a large part of the confusion arises from the fact that the terms ”design-time” and ”runtime” can refer to the construction and the usage of a model. A model that is constructed at design-time can be used at design-time for an a priori analysis or at runtime for control and adaptation. The model might evolve during runtime. The result might be used again during a redesign to analyse a larger change of the system. Especially for long-living software systems, the borders between design-time and runtime are blurred.

How to derive models? Models for self-healing and self adaptation can be derived in nearly arbitrarily many ways. In the following, we focus on the distinction between prescriptive and descriptive models. Prescriptive models state how the system under study should be structured and behave. They define the target state or the intention of the system. Descriptive models, by contrast, capture the actual state of the system under study. They state how things really are based on observations or analyses of the system under study. Descriptive models can be derived based on dynamic analyses, static analyses or a combination of these. In the case of static analysis, the artefacts of the system are analysed. For example, the source code can be used to reconstruct the static and dynamic architecture of a system. Re-engineering tools can extract dependencies between different parts of the software system (methods, classes, or components) and roughly reconstruct the system’s behaviour. In the case of dynamic analysis, the system is observed at runtime. The observations can be used to extract, for example, call sequences, timing information as well as reliability data. While these observations provide additional information to a static analysis, they are always limited to a specific number of traces and, thus, depend on the selected input data.

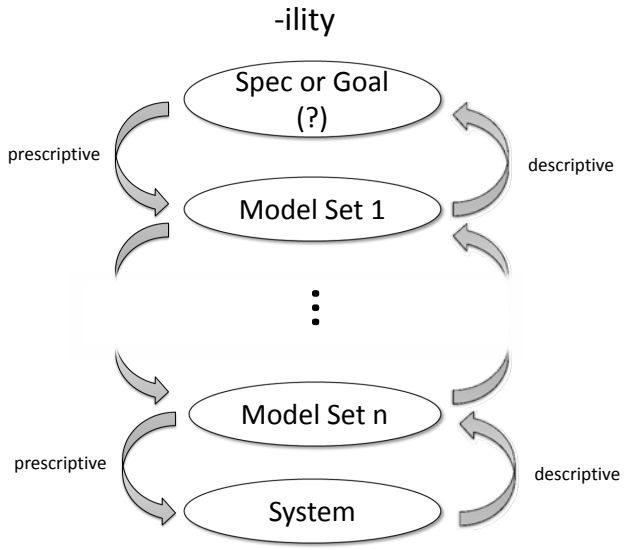

Fig. 3. Prescriptive versus descriptive model interpretation.

Figure 3 shows the relation of prescriptive and descriptive models in a self-adaptive system. Here, "-ility"' stands for any quality attribute, such as reliability, availability, performability, and performance. Whether a model is prescriptive or descriptive depends on its use. The same model can be interpreted both ways. Figure 3 shows a hierarchy of control systems where each interacts with its upper and lower layers. In the hierarchy, models passed to a lower layer are interpreted as prescriptions. They state how things should be organised on that level. By contrast, models passed to the higher layer are descriptive. They states how things actually are.

How are models kept updated in a dynamic environment? In a dynamic environment where self-adaptive and self-healing mechanisms are applied, models have to be synchronised with the environment which they represent. They need to be able to track changes in the environment and reflect them accordingly. However, inconsistencies of models and real systems are likely and self-adaptation and self-healing mechanisms need to be able to deal with them. The dynamic changes of the environment challenge the strict distinction between runtime and design-time models. Especially for such systems, both types of models are strongly interleaved.

How to deal with heterogeneity? In order to manage self-adaptive and self-healing systems, various models are used at different levels of abstraction. In the example in Figure 1, several models are used for different planning steps. All of these models depend on each other and share common information. Furthermore, all of them serve a common purpose. However, the relation between the models is not obvious. To clarify their dependencies, we would have to define relations between the models of different

layers. However, this may not be possible in all cases. For example, the low level control may not contain any information that represents obstacles. The obstacle detection and avoidance may not contain any information about turning the robot's wheels. Different models of the same system do not necessarily have to fit together.

4 General Issues

In the previous section, we discussed questions related to models in the context of self-adaptive and self-healing systems. In the following, we address some more general questions that can be asked when talking about models.

How to assess models? There are hundreds of ways to assess the quality or usefulness of a model. Which one is appropriate in a specific scenario strongly depends on the purpose of the model. For example, if the model is meant for performance prediction, the accuracy of the prediction results can be used as an indicator for the quality of the model. However, the assessment of a model may not only depend on single metric (such as prediction accuracy). Other aspects can also play a major role. Examples are the complexity of the model for analyses or the effort necessary to describe the system under study. In practice, the best model is often the simplest model that fulfils its purpose good enough. These models must avoid any unnecessary complexity.

What kinds of models do we have? Models can occur in many flavours. There structure depends on their purpose and the related analysis methods. For example, models can be continuous or discrete, (finite)state, (non-)deterministic, memory(less), or hybrid. The list is certainly not exhaustive. The properties of models have direct implications for their analysis and solution methods.

How are models solved/processed? Depending on the type of model, different solution methods are available, such as execution, interpretation, simulation, or analysis. Furthermore, transformations can map models to textual representations (such as source code) or to other models. Transformations allow the exploitation of existing solution methods for newly introduced models.

5 Conclusions

In this report, we discussed several questions addressing the role of models in self-adaptive and self-healing systems. The example of a robot's autonomic management illustrated the role of different models used at different levels of abstraction. The usage of multiple models leads to various challenges and different ways of interpretation. We discussed general questions related to models (such as "How are models solved/processed?") as well as questions specific for self-healing and self-adaptive systems (such as "How are models used in the software life-cycle?"). We focussed on the the discussion of different perspectives. This report is meant to foster the discussion on the role of models in self-adaptive and self-healing systems and, thus, does not provide any final answers.

References

1. Self-healing and self-adaptive systems. http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=09201, Mai 2009.
2. B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee. Software engineering for self-adaptive systems: A research road map. In *Dagstuhl Seminar Proceedings*, volume 8031. Springer, 2008.
3. Jeff Kramer and Jeff Magee. Self-managed systems: an architectural challenge. In *FOSE '07: 2007 Future of Software Engineering*, pages 259–268, Washington, DC, USA, 2007. IEEE Computer Society.
4. Ji Zhang and Betty H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 371–380, New York, NY, USA, 2006. ACM.