ABSTRACT
The emerging approach to tackle the increasing complexity of today’s software systems is the use of self-adaptation techniques. Modeling and implementing adaptivity features is a burdensome and error-prone task that potentially results in erroneous system models. As a consequence, quality analysis and assurance must be considered early in the development of self-adaptive systems.

We propose a quality assurance approach for self-adaptive systems in terms of an integrated modeling and analysis approach, which helps identifying errors in modeled self-adaptive systems early in the design process. We employ a modeling language for self-adaptive systems including adaptation rules and formally define their semantics. Given the language and its formal semantics, we formulate quality properties, such as fairness of the specified adaptation rule system. These quality properties are verified using a model checking approach.

Categories and Subject Descriptors
D.2.4 [Software Engineering]: Software/Program Verification—model checking

General Terms
Design, Verification, Languages

Keywords
Adaptive systems, design, analysis, quality assurance

1. INTRODUCTION
Software and software development continuously increase in complexity. As a consequence, single software aspects, such as performance and security, are treated separately forming dedicated disciplines. Another software aspect that recently increases in complexity is maintenance including adapting the software to new specifications or operating environments. Since the environment changes ever faster, maintenance of software both has to gain speed and also flexibility concerning an increasing range of possible operating environments. One approach to handle this complexity is automating maintenance such that the system configures itself and also adapts to changing environments. Such systems are referred to as self-adaptive systems [1].

Building self-adaptive systems means introducing extra functionality that enables the system to self-adapt to changing environments. For instance, sensors and effectors need to be considered and different environments have to be distinguished resulting in complex adaptation rule systems. In order to handle this increasing complexity, adaptation decisions are externalized in terms of adaptation rules which specify necessary actions to maintain the system. Large rule systems are hard to manage and errors easily creep in.

Therefore, a systematic and dedicated approach for modeling, and analyzing self-adaptive systems is required, analogously to other disciplines, such as performance analysis or security engineering. While there has been a lot of work in the area of modeling and implementing self-adaptive systems, quality assurance of self-adaptive systems did not catch lots of attention [1].

In our approach, we tackle that problem early in the development process. More precisely, we propose an approach to model self-adaptive systems including adaptation rules. Based on quality constraints, we analyze quality properties of the adaptation rules such as fairness or rule instability using model checking. This helps the modeler to assure the quality of his adaptive system at design time.

2. QUALITY ASSURANCE APPROACH

Figure 1 shows the machine model we use. The system model describes a system including the adaptation rules\(^1\). \(I\) is an instance of the system model and \(C\) is the corresponding context instance. We define context to be the interface to the environment, the system is situated in. As such, changes in the environment are reflected by the system’s context. If some change occurs in the context (e.g. a defect third-party

---
\(^1\)Note that rules are externalized in a rule system.
(service) or the system instance itself, adaptation rules apply and create a new instance $I'$ (e.g. with a different third-party service).

The figure's right-hand side depicts constraints that are specified to constrain the system model or its instances.

Global constraints are not application-specific, but express properties which every self-adaptive system must assure. An example is fairness or the absence of deadlocks when adaptation rules are applied. Application-specific constraints are defined over a concrete system model and assure application-specific properties, which are usually referred to as invariants and pre/post conditions. An example for application-specific constraints is that a webserver's availability must never be affected by some adaptation actions (requirement for recovery actions).

2.1 QUAASY Overview

Our approach to QUality Assurance for Adaptive SYstems (QUAASY) is depicted in Figure 2; it relies on the following prerequisites:

1. Syntax Definition
   (a) System and Context Model
   (b) Adaptation Rules (Adapt Cases)
2. Semantics Definition
   (a) DMM Rules for System and Context Model
   (b) DMM Rules for Adaptation Rules (Adapt Cases)
3. LTL formula for desired quality properties

In a nutshell, our approach works as follows: First, the syntaxes of the involved languages need to be defined by means of metamodels. We use metamodels describing our systems and their contexts as well as a metamodel describing our Adapt Cases [4] (i.e. adaptation rules) as depicted in the upper left in Figure 2. In a next step, we define the semantics of our languages by means of Dynamic Meta Modeling (DMM) [2], a rule-based semantics specification technique: A DMM specification together with an initial system model give rise to a labeled transition system (LTS) describing the complete system's semantics. The LTS has concrete model instances as states exposing concrete model objects and assigned attributes. The transitions are given by the applications of DMM rules. The transition system can then e.g. be analyzed using model checking techniques [3]. Finally, we formulate our quality properties as temporal logic formulas, which are then model checked on the computed LTS using the Groove tool set [5]. If one of our properties is violated by the system model under consideration, the model checker provides a counter example which can be used by the modeler to fix the system model.

Both Adapt Cases and the system & context modeling language are UML-based domain-specific languages developed in our group. They both are designed to be analyzed using the approach described above. The system & context modeling language allows the designer to specify how exactly the context might change. For instance, considering a rack-server system with a fan in its context, the fan’s range of change might consider the fan’s speed ranging from 1 to 5. This information is used to simulate context changes. The information provided by the Adapt Cases is used to monitor the system, analyze the gathered information, plan an appropriate adaptation action, and finally execute this adaptation action.

3. CONCLUSION

The overall objective of the presented approach is the quality assurance of self-adaptive system design. Therefore, languages are developed that allow the specification of system, context, rules, and quality constraints. The languages are formally supported using DMM as semantic specification language.

4. REFERENCES