Assuring System Goals under Uncertainty with Active Formal Models of Self-Adaptation

M. Usman Iftikhar and Danny Weyns
Linnaeus University, Växjö, Sweden
{usman.iftikhar, danny.weyns}@lnu.se

ABSTRACT

Designing software systems with uncertainties, such as incomplete knowledge about changing system goals, is challenging. One approach to handle uncertainties is self-adaptation, where a system consists of a managed system and a managing system that realizes a feedback loop. The promise of self-adaptation is to enable a system to adapt itself realizing the system goals, regarding uncertainties. To realize this promise it is critical to provide assurances for the self-adaptive behaviors. Several approaches have been proposed that exploit formal methods to provide these assurances. However, an integrated approach that combines: (1) seamless integration of offline and online verification (to deal with inherent limitations of verification), with (2) support for runtime evolution of the system (to deal with new or changing goals) is lacking. In this paper, we outline a new approach named Active FORmal Models of Self-adaptation (ActivFORMS) that aims to deal with these challenges. In ActivFORMS, the formal models of the managing system are directly deployed and executed to realize self-adaptation, guaranteeing the verified properties. Having the formal models readily available at runtime paves the way for: (1) incremental verification during system execution, and (2) runtime evolution of the self-adaptive system. Experiences with a robotic system show promising results.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*formal methods*

General Terms

Theory, design

Keywords

Self-adaptive systems, formal models at runtime, verification

1. INTRODUCTION

Software engineers have to deal increasingly with uncertainties at design time. Uncertainties can result from lack of detailed knowledge about availability of resources, system goals that may change but need to be dealt with at runtime, etc. Dealing with such uncertainties and guaranteeing the required system goals requires rethinking of the way software systems have to be engineered.

One promising approach to handle uncertainties is self-adaptation [3]. Self-adaptive systems are software systems that can adapt autonomously at runtime to deal with uncertainties. Our particular focus is on architecture-based self-adaptation [8] where the self-adaptive system consists of two parts, i.e., a managed system that realizes the domain logic and a managing system that realizes a feedback loop that monitors and adapts the managed system to realize particular system goals (e.g., self-heal when a fault occurs, self-optimize when operating conditions change, self-reconfigure when a goal changes). A common approach to realize the feedback loop is by means of a *MAPE-K* loop [7]. *Monitor* monitors the managed system and environment through probes, and saves data in the *Knowledge*. *Analyze* performs data analysis to check whether an adaptation is required. If so, it will trigger *Plan* that will compose a work flow of actions that are then executed through effectors by *Execute*. The promise of self-adaptation is to enable a software system to adapt itself at runtime realizing the system goals, regarding uncertainties. To realize this promise it is critical to provide assurances for the self-adaptive behaviors. One prominent approach to provide such assurances is using formal methods that enable rigorous specification and verification of the behaviors of self-adaptive software systems.

2. STATE OF THE ART

We highlight a number of representative approaches that use formal methods to provide assurances in adaptive systems. [4] uses a probabilistic model to represent the possible execution flows of a system. This model is updated at runtime to acquire knowledge about the system behavior that was not available at design time. The probabilistic model can be used by a feedback loop to adapt the system dynamically. In [1], requirements are automatically analyzed to enforce optimal configurations of a service based system by adapting service selection and resource allocation. The approach employs a MAPE-K loop where the Knowledge models are processed by a series of tools that realize the MAPE functions. [5] uses a Markov model that specifies the probability distribution of utilities of the different execution paths of the system. The model is executed by an interpreter that drives system execution to guarantee the highest utility for a set of quality properties.

These approaches demonstrate the potential of formal methods for assuring goals for adaptive systems. However, an integrated approach that combines: (1) seamless integration of offline and online verification (to deal with inherent limitations of verification, i.e., the state-space problem), with (2) support for runtime evolution of the system (to deal with new or changing goals) is lacking. The first observation was also made in [2], where the authors state

ICSE Companion'14, May 31 – June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2768-8/14/05...$15.00
http://dx.doi.org/10.1145/2591062.2591137

that continual re-verification at runtime is a key requirement for self-adaptive systems. With runtime evolution, we refer to support for changes that go beyond uncertainties that require only updates of *parameters*, e.g., the likelihood of a failure or changing availability of a service (for which probabilistic models are well suited). Support for runtime evolution enables dealing with *structural* uncertainties and unanticipated changes. This typically requires dynamic updates of the managing system, and possibly updates of the managed system. Support for such changes adds another dimension of complexity to the engineering problem of self-adaptation.

3. ACTIVFORMS

ActivFORMS combines seamless integration of offline and online verification with support for runtime evolution. The basic idea of ActivFORMS is simple: design formal models of the self-adaptive system,[1] verify the required quality properties, implement and add probes and effectors to the managed system and possibly the environment, deploy the formal models, start the execution of the implemented managed system and start the execution of the model of the managing system. This approach assures that adaptations are being performed according to the formally verified models. Furthermore, having the formal models readily available at runtime enables incremental verification during system execution, and supports runtime evolution of the self-adaptive system. In our research, we use timed automata as modeling language and express required properties in timed computation tree logic. However, ActivFORMS is not restricted to these formal languages.

Fig. 1 shows an overview of ActivFORMS.[2] The managed system realizes the domain logic for users. ActivFORMS comprises two primary modules: the active model and goal management.

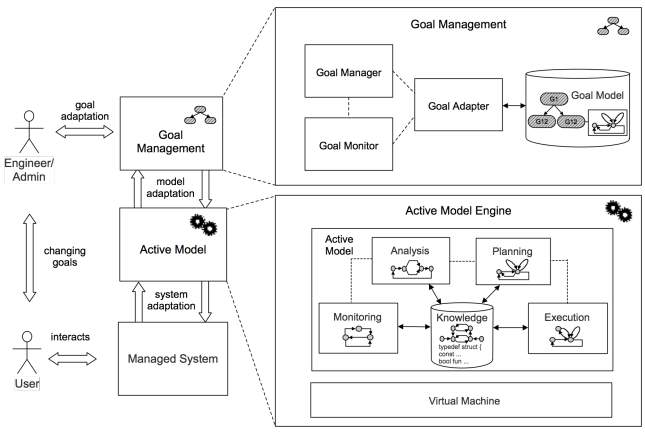

Figure 1: Primary modules of ActivFORMS at runtime

The active model consists of the formally verified model of the managing system that realizes a MAPE-K feedback loop and a virtual machine that can execute the formal model, hence active model.

Concretely, the virtual machine can perform the following functions: loading and executing a formal model, sending/receiving signals with data to/from probes and effectors connected to the managed system and possibly the environment, detecting goal violations, and updating the running model when requested, which is crucial to handle unanticipated changes.

Goal management deals with adaption issues that cannot be handled by the active model. To that end, goal management consists of four key parts (see Fig. 1): goal model, goal monitor, goal adapter, and goal manager. The goal model represents the adaptation goals with associated adaptation models. The goal monitor monitors the status of the goals with the virtual machine. In the first release of ActivFORMS support for detection of goal violations is limited to *boolean expressions* of the current state of the active model. When the goal adapter is signaled by the goal monitor about a change of goals, it consults the goal model and search for a matching adaptation model that satisfies the changing situation. If a model is found the goal adapter starts updating the model with the virtual machine. The goal manager offers support for three primary functions: inspecting the active model and its ongoing execution, monitoring goals, and updating the goal model. These functions allow an engineer to change existing goals or add new goals and associated adaptation models to deal with new requirements.

We have applied ActivFORMS to various adaptation scenarios in a simple multi-robotic system. This experience shows promising results. The interested reader find more information about this case study via the ActivFORMS website.

4. CHALLENGES AND FUTURE WORK

The first key challenge for our future work is to enhance ActivFORMS with support for incremental verification at runtime. Developing support for efficient incremental verification at runtime is a challenging task. We will study how offline verification needs to be combined with online verification, and how runtime verification can be realized efficiently. One starting point for our study is [6]. A second long-term challenge is to study support for co-evolution of the managed and managing system. Supporting such co-evolution requires synchronization between updates of the active model of the managed system, the running managing system, and the models of the managing system and its execution environment. This study will require the integration of knowledge from the field of dynamic evolution of software systems (live updates) with the field of self-adaptation. Finally, a third challenge is to study how ActivFORMS can be extended with support for engineers to design self-adaptive systems without direct exposure to the formal specification. An inspiring approach is proposed in [5].

5. REFERENCES

[1] R. Calinescu et al. Dynamic qos management and optimization in service-based systems. *TSE*, 37(3), 2011.

[2] R. Calinescu et al. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9), 2012.

[3] B. Cheng et al. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems, LNCS vol. 5525*. Springer, 2009.

[4] I. Epifani et al. Model evolution by run-time parameter adaptation. In *ICSE*, 2009.

[5] C. Ghezzi et al. Managing non-functional uncertainty via model-driven adaptivity. In *ICSE*, 2013.

[6] K. Johnson et al. An incremental verification framework for component-based software systems. CBSE, 2013.

[7] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1), 2003.

[8] D. Weyns et al. FORMS: Unifying Reference Model for Formal Specification of Distributed Self-Adaptive Systems. *ACM TAAS*, 7(1), 2012.

[1]The formal model of the managed system needs to conform to the implementation, which can be tested e.g., using model-based testing (www.geocities.com/model_based_testing/).

[2]http://homepage.lnu.se/staff/daweaa/ActivFORMS.htm