The Journal of Systems and Software 85 (2012) 2720–2737

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Achieving dynamic adaptation via management and interpretation of runtime models☆

Mehdi Amoui[a,*], Mahdi Derakhshanmanesh[b], Jürgen Ebert[b], Ladan Tahvildari[a]

[a] University of Waterloo, 200 University Ave West, Waterloo, Ontario, Canada
[b] University of Koblenz-Landau, Universitätsstr. 1, 56070 Koblenz, Germany

ARTICLE INFO

Article history:
Received 1 May 2011
Received in revised form 23 April 2012
Accepted 8 May 2012
Available online 16 May 2012

Keywords:
Adaptation framework
Runtime adaptivity
Self-adaptive software
Model transformation
Models at runtime

ABSTRACT

In this article, we present a generic model-centric approach for realizing fine-grained dynamic adaptation in software systems by managing and interpreting graph-based models of software at runtime. We implemented this approach as the *Graph-based Runtime Adaptation Framework* (GRAF), which is particularly tailored to facilitate and simplify the process of evolving and adapting current software towards runtime adaptivity. As a proof of concept, we present case study results that show how to achieve runtime adaptivity with GRAF and sketch the framework's capabilities for facilitating the evolution of real-world applications towards self-adaptive software. The case studies also provide some details of the GRAF implementation and examine the usability and performance of the approach.

1. Introduction

Self-adaptive software (SAS) is a software system that is able to modify its own behavior in response to changes in its operating environment (Oreizy et al., 1999). One of the main advantages of SAS is its ability to manage the complexity that stems from highly dynamic and nondeterministic operating environments. However, in practice, implementing self-adaptive and autonomic behavior can lead to an increase in overall system complexity (e.g., due to a complex *adaptation logic* that needs to be embedded into the system), as well as subsequent future maintenance costs.

In software engineering, modeling is as an effective strategy for managing system complexity (Ludewig, 2003) (e.g., by eliminating or hiding unnecessary details). To tackle the complexity issue in the context of SAS, the construction of precise and accurate models of software that can support various aspects of adaptation is essential (Cheng et al., 2009; Andersson et al., 2009). However, modeling in the SAS domain is a hard task, mainly because of the highly dynamic nature of SAS systems and their operating environments. Difficulties arise especially in the area of *model-centric runtime adaptation*, where models need to be managed at runtime (Garlan and Schmerl, 2004; Morin et al., 2009).

In order to effectively use models at runtime, we propose a model-centric architecture for SAS, in which adaptation is achieved through managing and interpreting a dynamic *runtime model* of software, instead of directly changing the software system. This approach explicitly separates software and its common business logic from the adaptivity-related concerns such as runtime models and adaptation logic. We realize our approach as the *Graph-based Runtime Adaptation Framework* (*GRAF*) (Derakhshanmanesh et al., 2011), which utilizes TGraphs (Ebert et al., 2008) and its accompanying technologies, as the enabling technology, for modeling and manipulating runtime models.

When designing GRAF, our primary goal was to show that runtime adaptivity can be achieved by applying techniques that are common and well-established in the model-driven domain. Our focus was set on the use of software models at runtime in the technical context of a *reflective architecture* (Buschmann et al., 1996) that can support the evolution of existing software towards SAS. Hence, the applied set of operations on runtime models consists of (i) *querying*, i.e., gathering change information from the meta-layer, (ii) *transforming*, i.e., adapting the meta-layer, and (iii) *interpreting* as a means of reflecting changes from the meta-layer to the executing software parts in the underlying base-layer.

Our proposed model-centric approach is supported by a number of case studies that serve as a proof of concept, show by example, how to achieve runtime adaptivity with GRAF, and sketch the framework's capabilities for facilitating the evolution of real-world

☆ This article is a revised and expanded version of a paper accepted at the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), Honolulu, Hawaii, May 2011.

* Corresponding author.
E-mail addresses: mamouika@uwaterloo.ca (M. Amoui), manesh@uni-koblenz.de (M. Derakhshanmanesh), ebert@uni-koblenz.de (J. Ebert), ltahvild@uwaterloo.ca (L. Tahvildari).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.05.033

applications towards a self-adaptive software system. The case studies support understanding the benefits and drawbacks of *fine-grained adaptation* using GRAF, i.e., adaptation at the level of fields and methods, and present our proposed non-intrusive approach for preparing the required adaptable software and its runtime model.

This article is organized as follows: Section 2 elaborates on the concept of *models at runtime* in SAS with an emphasis on the design considerations and motivating factors that drive this work. Section 3 gives a comprehensive overview of GRAF's model-centric architecture together with its runtime behavior. Section 4 gives an overview of the technologies that are involved in the framework's implementation. Section 5 puts GRAF in a migration context and illustrates how to prepare adaptable software by evolving non-adaptive software. Section 6 presents the case studies of applying GRAF to two real-world application domains: computer gaming and Internet telephony. Section 7 covers the related work. Finally, we conclude this paper in Section 8 and outline possible future research directions.

2. Design considerations

There is a multitude of approaches to achieve runtime adaptivity by incorporating runtime models (Garlan and Schmerl, 2004; Morin et al., 2009; Vogel and Giese, 2010; Blair et al., 2009). To make our design considerations more clear, we initially present an introduction to model-centric runtime adaptivity in the context of computational reflection (Maes, 1987). Thereupon, we give an overview on the influences from different areas that had an impact on our work and especially on the framework's design.

2.1. Model-centric runtime adaptivity

SAS must be *self-aware*, i.e., it must be able to access state information regarding (a subset of) its own structural and behavioral elements. This is essential, because changes in the software's operating environment as well as in the state of software itself need to be observed by the adaptation manager.

Computational reflection, as defined by Maes (1987), is a common solution for achieving the self-awareness property (Weyns et al., 2010). Such a *reflective system* is connected to a representation of itself. If the system changes, its self-representation changes accordingly (*reification*). Vice versa, changes in the self-representation result in changes to the underlying software (*reflection*).

In a *reflective architecture* (Buschmann et al., 1996), the part of software that specifies the business logic and interacts with the application domain is called the *base-layer*. The base layer is *causally connected* (Morin et al., 2008) to a *meta-layer*, i.e., the self-representation of the system. This meta-layer can be realized in different ways, for instance, it can be implemented as an object model in an object-oriented programming language.

From our perspective on *model-centric* runtime adaptivity, the software system to be controlled at runtime (*adaptable software*) serves as a base-layer of a reflective architecture. Adaptable software is connected to a meta-layer, which is the actual *runtime model* acting as a mediator. Moreover, an *adaptation manager* controls the adaptable software by manipulating the runtime model instead of directly operating on the adaptable software. This structure is sketched in the context diagram illustrated in Fig. 1.

In Fig. 1, starting from the adaptable software, the state of the adaptable software is propagated to its runtime model (*reification*). These changes are then observed by the adaptation manager (*sensing*). Subsequently, the adaptation manager plans and selects adjusting actions (*controlling*) and adapts the runtime model accordingly (*effecting*). Finally, changes made to the runtime models are propagated back into the adaptable software (*reflection*) by

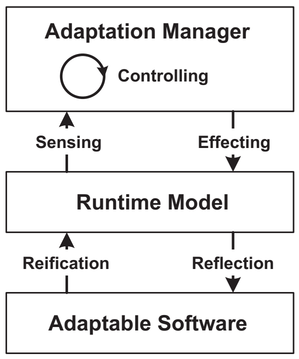

Fig. 1. Context diagram of model-centric SAS (Derakhshanmanesh et al., 2011).

either *parameter adaptation*, i.e., by changing variables values, or via *compositional adaptation*, i.e., by adjusting the software's structure (McKinley et al., 2004).

Given such a model-centric architecture, the structure and behavior of software can be changed by modifying the model only. As a result, software engineers can work with the models first, instead of modifying the adaptable software directly. This approach supports the implementation of adaptivity by modifying the intermediate layer automatically via a set of predefined transformations.

In the following, we describe the major concerns that we had in mind when designing GRAF, namely (i) change reflection via *model interpretation* and (ii) *low coupling* between the adaptable software to the framework. It is noteworthy that several other aspects, such as the *verification* and *synchronization* of models, are also essential to the success of models at runtime in the area of SAS, which are not tackled in this article.

2.2. Model interpretation for change reflection

Given that the meta-layer is available as a model, the process of *reflecting*, i.e., injecting changes from the meta-layer into the base layer, can be done in several ways and using different techniques. For instance, software components can be composed at runtime (Cheng et al., 2004; Vogel and Giese, 2010) or functionality can be adjusted using dynamic aspect weaving (Grace et al., 2008).

In GRAF, we take a different direction, because we intend to integrate the runtime model as deeply into the software as possible. The idea is to build generic software based on runtime models that describe the variable parts of the system. These explicit models are then *interpreted* at runtime using a generic and stable software core. In fact, GRAF is the first attempt for applying this approach in the area of SAS.

Besides achieving adaptation by adjusting program variable values, the main way of achieving adaptivity in this research is by redirecting the adaptable software's control flow to a model interpreter component at points where the need for adaptivity is expected. When such an *interpretation point* in the control flow is reached during the execution of the adaptable software, the model interpreter executes an associated behavior, as described in the runtime model. Therefore, transforming behavior descriptions that are stored in the runtime model results in adapted program behavior.

2.3. Low coupling to adaptation framework

Assume that some software already exists and shall be evolved towards an adaptive system. In the case that no adaptation was performed at runtime, the resulting SAS has to perform the same functionality as the non-migrated legacy system. An obvious strategy towards this goal is to make as little changes as possible to the