Runtime Adaptation of Component Based Systems

Sihem Loukil, Slim Kallel, and Mohamed Jmaiel

ReDCAD Laboratory, University of Sfax, Tunisia
sihem.loukil@redcad.org,
slim.kallel@fsegs.rnu.tn, mohamed.jmaiel@enis.rnu.tn

Abstract. The need for continuously available software systems and their ability to support runtime adaptation is increasingly considered as one key issue in the software development. In particular, the software architecture of dynamically adaptive component based systems must continuously adapt to varying environmental conditions and user requirements. Therefore, they propose a wide range of possible adaptations that can not all be foreseen at design time. In this context, we propose to combine the Architecture Description Languages and the Aspect-Oriented Software Development which allow to make the adaptation process easier to design, understand and possible to validate.

1 Introduction

Software architecture modeling using Architecture Description Languages (ADLs) is becoming increasingly popular in the early phases of system development. Such languages facilitate the construction of high-level models in which systems are described as compositions of components. They play an important role in developing software systems deployed in large number of domains (companies, banks, air-ports, etc). Such systems must be always available and continuously adapt to varying environmental conditions and user requirements even at runtime. Hence, they should be modified/maintained during their execution, for example to include new functionalities, without being obliged to stop the system. This dynamic reconfiguration to maintain the system available presents a tedious task. In fact, not all possible reconfigurations that will be applied to the system can be foreseen at the time it is initially built and deployed. Therefore, the system must be flexible to support new needs that may appear during execution.

Very recently, several approaches like [1] are proposed to synchronize high level models with the running system. Such approaches focus on managing the variability of the dynamically adaptive systems by building a causal connection between abstract design models and the running system. In such approaches, an application is modeled using a base model and a set of variant models (aspects that encapsulate the variation points) in order to manage the variability of the adaptive application. Hence, an adaptation model is established to specify which variants should be selected according to the adaptation rules and the current

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 284–288, 2013.

context of the executing system. The main problem of such approaches is the use of more than one formalism to represent the design model (model representing at a high level of abstraction the architecture of the initial system before its deployment) and the execution model (model representing the abstraction of the architecture of a system at runtime). This forces the use of transformation languages and/or comparison models which increases the risk errors as it can give place to problems related to the synchronization of the different models, or to the propagation of changes among the different views. Moreover, to support unforeseen adaptations at design time in such approaches, the designer should manually create the modified model from scratch.

To tackle these issues, we propose an approach to manage the runtime adaptation of component based systems that uses one formalism to represent the design model as well as the execution model. This approach supports the unforeseen adaptations at design time by manually editing the parts in question using a graphical editor without being obliged to design the modified model from scratch.

In this context, we aim at combining the Architecture Description Languages (ADLs) and the Aspect-Oriented Software Development (AOSD) [2] paradigm which allow to make the adaptation process easier to design, understand and possible to validate.

For this purpose, we selected the Architecture Analysis & Design Language (AADL) [3], as an ADL, for specifying the architecture of dynamically adaptive component-based systems. This language uses the same formalism to represent the runtime model and the design model. Moreover, a previous publication [4] has provided a general overview of AO4AADL language, an Aspect-oriented extension for AADL. This aspect-oriented modeling language extended AADL with aspect-oriented concepts to design the crosscutting concerns related to the non-functional and technical properties. It is used in this work to allow designer monitoring the running system and performing the corresponding adaptation.

Our approach supports two types of runtime adaptations. First, the runtime adaptations resulting from a change in the execution context of the running system. Second, the ones resulting from the apparition of new user requirements that requires the manual intervention of the designer on the architectural specification of the system. In both cases, the adaptation actions are performed first on the model representing the architecture of the system. Applying the adaptation actions at the model level before applying them to the running system has the advantage that we can test their effect when applied as a whole without actually changing the system. Thereby, it is always possible to jump back to the state before starting to apply the adaptation actions in case an error is detected saving costly executions of roll-back operations on the system.

The remainder of this paper is organized as follows. Section 2 presents an overview of the proposed approach. In Section 3, we briefly present the monitoring module of the adaptation process. Section 4 details the related work. Finally, Section 5 concludes this paper and presents future work.

2 Architectural Reconfiguration of Component-Based Systems

Figure 1 shows the architecture of the proposed approach for managing component-based systems at runtime. This architecture comprises four levels: declarative level, instance level, runtime level and adaptation level.

At the declarative level, the designer defines the types of the components that can be used in the specification of the system. At the instance level, he specifies the base model that contains instances of the declared types as well as the connections between them. At this level, architectural aspects can be integrated to specify crosscutting concerns. The runtime level contains the runtime machinery that supports the execution of the system. The executed code is generated from the instance model and eventually from the defined architectural aspects. Finally, the adaptation level, which represents our main contribution, is responsible for the management of runtime adaptations that may result from a change in the execution context or the apparition of new user requirements at runtime. Then, these adaptations can affect either the running system (change in the execution context) or the instance model (apparition of new user requirements).

For the adaptations resulting from changes in the execution context, we use the aspect-oriented technique to define a set of architectural aspects that are intended to intercept the execution context variables and perform the corresponding reconfigurations. For the other type of adaptations, we employ the Hook methods technique which is intended to capture any manual intervention of the designer on the architectural model of the system.

In both cases, the effect of the adaptation actions to perform should be checked before committing them to the running system. For this purpose, they are applied first on the model representing the architecture of the system and the obtained configuration is validated through a set of architectural constraints defined at the declarative level. If the new configuration is valid, the adaptation actions are applied on the running system. Otherwise, it is simply discarded which allows to save costly executions of roll-back operations on the system.

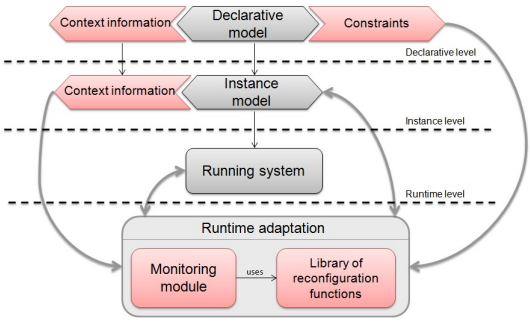

Fig. 1. The proposed architecture

As shown in figure 1, the runtime adaptation is composed of a main module called the monitoring module. This module checks if any change is occurred at the runtime level resulting from a change in the execution context or at the instance level. For this purpose, we employ an extended version of the declarative level that supports the specification of the context information. The context information and the adaptation rules are specified in a set of architectural aspects. To perform the adaptation actions, this module uses a library of reconfiguration functions that are included into the Ocarina tool [5].

At the declarative and instance level we use an extended version of AADL with aspect concepts described in AO4AADL language [4]. The choice of AADL was driven by many reasons. First, AADL is a standard and the resulting architecture enables simulation and analysis of architectural characteristics using precise execution and communication semantics. Second, AADL introduces two extension mechanisms (properties and annexes) which make the language much easier to extend. Moreover, it allows specifying architectural aspects using the AO4AADL extension. To achieve the step of designing the system, we developed our own graphical editor that integrates both AADL and AO4AADL concepts in order to make the job of the designer easier. The declarative model and the instance model are located at the server machine in the distributed application.

At the runtime level we use the RTSJ (Real Time Specification for Java) platform which allows executing RTSJ code generated from the AADL specification using Ocarina tool suite and easily weaving the AspectJ aspects generated from the AO4AADL ones using our AspectJ generator presented in [4].

3 Monitoring Module

The monitoring module is composed of two types of monitors: the running system monitor which looks at the changes that may be occurred at the runtime level and the instance model monitor which checks for changes at model level.

The running system monitor checks the changes in the context information through a set of AspectJ aspects woven into the code of the application. These aspects are generated from the AO4AADL aspects defined at the declarative level using our Aspect generator developed in the Ocarina tool suite [5]. The pointcut of such architectural aspect intercepts the execution of a port of a component or a parameter of a subprogram through which the information on the intercepted context variable is transferred. The advice code is fulfilled by the designer to specify the corresponding adaptation rules.

The instance model monitor is intended to check for changes on the instance model performed manually by the designer while the system is running. It is composed of a library of functions called Hook methods. These Hooks allow the designer to modify the functionalities of his software by realizing customized actions at well-defined times by inserting entry points to a list of actions. These methods include listeners to capture the evolution of the instance model.

4 Related Work

There are various points of view on how to reconfigure a system at runtime. Although this technique is recently introduced, several researchers have deepened their work in this area.

Similarly to our work, some approaches like [1] are based on aspect-oriented programming and model-oriented techniques to monitor and adapt application by building a causal connection between design models and the running system. Unlike our approach, the designer should manually create the modified model from scratch to support unforeseen adaptations. The authors present in [6] model-based traces as runtime models and traces analysis methods. However, the syntax and semantics of various types of the model-based traces in this work are not formally defined. Some other existing approaches show how runtime models can be derived efficiently from the specification, and how they support the designer in considering the execution of the application in the same formalism as the specification [7]. Unlike our approach, designers are required to consider the execution model when specifying any runtime adaptation, forcing them to understand the different formalisms of both the execution and the specification models.

5 Conclusion and Future Work

We have proposed an approach to manage the runtime adaption of component-based systems. Architectural aspects described in AO4AADL are used to catch the adaptations resulting from context information changes. These adaptations are foreseen at design time. For the unforeseen adaptations, we propose to manually act on the model to perform the adaptation.

In future work, we plan to extend our approach to support the detection of potential conflicts between runtime adaptation in a distributed system.

References

1. Morin, B., Barais, O., Jezequel, J.M., Fleurey, F., Solberg, A.: Models@ run.time to support dynamic adaptation. Computer 42, 44–51 (2009)
2. Filman, R.E., Elrad, T., Clarke, S., Akşit, M. (eds.): Aspect-Oriented Software Development. Addison-Wesley, Boston (2005)
3. SAE: Architecture Analysis & Design Language (2004), `http://www.sae.org`
4. Loukil, S., Kallel, S., Zalila, B., Jmaiel, M.: Toward an Aspect Oriented ADL for Embedded Systems. In: Babar, M.A., Gorton, I. (eds.) ECSA 2010. LNCS, vol. 6285, pp. 489–492. Springer, Heidelberg (2010)
5. Vergnaud, T., Zalila, B., Hugues, J.: Ocarina: a Compiler for the AADL. Technical report, Telecom Paristech - France (2006)
6. Maoz, S.: Using model-based traces as runtime models. Computer 42, 28–36 (2009)
7. Saudrais, S., Staikopoulos, A., Clarke, S.: Using specification models for runtime adaptations. In: International Workshop on Models@RunTime (2009)