

Dynamic architectures management for cooperative applications adaptability

Ismael Bouassida Rodriguez

LAAS-CNRS, Université de Toulouse; 7, av. du Colonel Roche, F-31077 Toulouse

bouassida@laas.fr

Abstract

Handling context-aware dynamically adaptable architectures contributes to the design of self-configuring software systems. Dealing with such a problem for communicating system is even more challenging since adaptation should address simultaneously the different communication levels. This is necessary for handling both unpredictable changes in the low level constraints and predictable evolutions in the high level requirements. In this paper, we address this problem by providing a model-based, rule-oriented approach that supports the adaptation process. An architecture may represent the different possible service compositions and the associated architectural configurations. We consider the multi-level models of the communicating system architecture. We consider the intra-level architecture transformations as the elementary adaptation actions. We handle consistently the related inter-level adaptation actions by considering additional architectural relationships viewing the lower level architecture as a refinement or a mapping of the upper level. We provide the algorithm characterizing the multi-level architecture-based adaptation process. We then develop a rule-oriented implementation using graph grammar and handling architectural transformations as graph rewriting rules.

1. Introduction

Designing and implementing self-adaptive communicating systems is a complex task, which may be addressed via model-based design approaches associated with automated management techniques for dynamic architectural adaptability. In self-adaptable applications, components are created and connected, or removed and disconnected during the execution. The architectural changes respond to constraints of the communication and resources capacities variations. Providing solutions for distributed software systems supporting group communication requires managing dynamically evolving group membership and dynamically connecting deployment nodes. For a number of group

communication-based applications, deciding such a reconfiguration for one or more levels of interaction and distribution will depend on the situation of a run-time changing context. Providing generic solutions for automated self-reconfiguration in group communication support systems can be guided by rule-based reconfiguration policies. This is the approach we adopt in this paper. In order to guarantee the architecture updates correctness we use formal techniques. In particular, graphs represent a powerful expressive mean to specify respectively static and dynamic architectures aspects. Our solution is based on graph grammar theories. It handles architectural refinement as graphs where vertices are assimilated to software services and components. It provides the graph transformation rules allowing to handle the deployment architecture changes at run-time.

2. Model-based approach for adaptability management

Managing self-adaptability for optimizing QoS in group communication activities requires considering several kinds of evolving requirements and changing constraints, and leads to architectural adaptations at different levels of the communication stack. This raises a coordination problem which, if not properly addressed, may conduct to inefficient or even inconsistent solutions. For instance, adapting the architecture to maintain QoS under a given constraint may be handled at different levels considering a tradeoff between the different requirements and the different constraints.

Managing architectural adaptations requires defining and modelling abstraction levels dedicated to specific parts of the whole adaptation (Figure 1). Distinguishing these different abstraction levels allows designers and developers to respectively master the specification and the implementation of adaptation rules. For a given deployment configuration $A_{n,1}$ at level n , multiple deployment configurations $(A_{n-1,1}, \dots, A_{n-1,p})$ may be implemented at level $n - 1$. Adapting the architecture to constraint changes at level $n - 1$ by switching among these multiple deployment configurations allows maintaining unchanged the n -level deployment configuration. Moreover, when adaptation requires changes

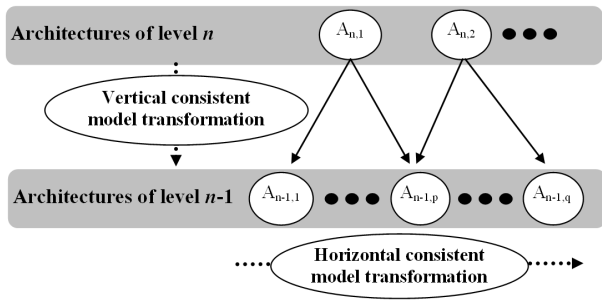


Figure 1. Abstraction levels

at level n , this may need no changes at level $n - 1$ if initial and new deployment configurations of level n (e.g. changes from $A_{n,1}$ to $A_{n,2}$) have common implementations (e.g. $A_{n-1,p}$) at level $n - 1$.

We present an approach that formalize the refinement and the reconfiguration process. We define also functions that allow to select the suitable deployment configurations at each step of the reconfiguration process.

The adaptation approach begins by selecting and refining the initial deployment configuration. The procedure $Refine()$ formalizes these two actions.

After the refinement of the initial deployment configuration, the adaptability approach selects the optimal deployment configuration at the level $n - 1$ that implements the initial deployment configuration at the level n . The procedure $Weight_Selection()$ formalizes this step. The selection is based on minimizing the *weight* that is a generic function independent of the context and the *cost* that is a generic function independent of the context.

After this first selection, the adaptability approach waits for a reconfiguration event e . When a reconfiguration event occurs, this triggers a reconfiguration procedure $Reconfigure()$. This procedure reconfigures the current deployment configuration and generates a new deployment configuration at level n .

After the reconfiguration of the initial deployment configuration, the adaptability approach selects the optimal deployment configuration at the level $n - 1$ that implements the deployment configuration at level n obtained through $Refine()$. The procedure $Distance_Selection()$ minimizes the distance between two deployment configurations at level $n - 1$ both implementing the correspondent deployment configurations at the level n and the cost according to the context.

The adaptability approach loops and waits for an other reconfiguration event that will triggers a new reconfiguration at level n .

we can summarize our approach by a set of steps. We select the initial deployment configuration of level n that

we call $A_{n,0}$. Then, we identify the correspondent deployment configuration at the level $n - 1$ using the function $Refine()$. After that, according the context values (e.g the available power, the available memory), we select the suitable deployment configuration (level $n - 1$) using the $Weight_selection()$. We wait for a reconfiguration event. When we detect one event, we reconfigure the current deployment configuration according to the event that generates a new deployment configuration (level n) using the function $Reconfigure()$. After this step, we identify the correspondent deployment configuration at the level $n - 1$ using the function $Refine()$. According the context values, we select the suitable configuration (level $n - 1$) using the function $Distance_selection()$. after that we wait for an other reconfiguration event that will trigger the reconfiguration again.

In our approach, we provide graph grammars to implement architecture reconfiguration at the service level (n) and architecture refinement (e.g for mapping S-level on to M-level). The refinement graph grammar GG_{Refine} , for a given configuration $A_{n,i} \in (A_{n,1}, \dots, A_{n,p})$ at the service level, produces the set of configurations $(A_{n-1,1}, \dots, A_{n-1,p})$ that can implement at the middle-ware level ($n - 1$). We present also a graph grammar $P_{Reconfigure}$ that allows transforming the architecture for handling the evolving of communication requirements.

3. Conclusion

In this paper, we have presented a multi-level architectural reconfiguration approach for implementing context-aware adaptation procedures. We have shown how describing deployment architectures as graphs and using graph grammars allows a rule-based management model to be elaborated. The rules handle both transforming a given architecture within the same level and architectural mappings between different levels. Using such a rule-based approach allows correct architectural reconfigurations to be characterized and used either offline to help implementing the decision process, or online to handle the architectural adaptation.

4. Publications

I. Bouassida, K. Drira, C. Chassot, and M. Jmaiel. Context-aware adaptation approach for group communication support applications with dynamic architecture. *System and Information Sciences Notes*, 2(1), 2007.