Chapter 12
An MDA Approach for Developing Executable UML Components

Simona Motogna
Babeş- Bolyai University, Romania

Bazil Pârv
Babeş- Bolyai University, Romania

Ioan Lazăr
Babeş- Bolyai University, Romania

ABSTRACT

Model-driven Architecture frameworks provide an approach for specifying systems independently of a particular platform and for transforming such system models for a particular platform. But development processes based on MDA are not widely used today because they are in general heavy-weight processes - in most situations they cannot deliver (incrementally) partial implementations to be executed immediately. Executable UML means a execution semantics for a subset of actions sufficient for computational completeness. This chapter introduces a foundational UML (fUML) based action language (AL) and describes its concrete syntax. AL is used to describe the operations for iCOMPONENT - the proposed solution for a platform-independent component model for dynamic execution environments. Moreover, a UML profile for modeling components is defined and applied, following agile principles, to the development of service-oriented components for dynamic execution environments. The intended use of the proposed approach is enterprise systems.

INTRODUCTION

Some service-oriented component models support the dynamic availability of components at runtime and offer the possibility to build dynamically adaptable applications. However, building service-oriented components is a complex task due to the complexity of service-oriented frameworks. In this context today frameworks try to simplify the component development by allowing developers to concentrate only on implementing the business

DOI: 10.4018/978-1-61692-874-2.ch012

logic of the component and then to configure declaratively the component deployment.

In this chapter, we describe our contribution to this domain that can be expressed in:

- A fUML based action language and its concrete syntax. AL is part of the ComDeValCo framework (Pârv, Lazăr & Motogna, 2008; Pârv, Lazăr, Motogna, Czibula & Lazăr, 2009) and we are using it to model the functionality of the components, to simulate the execution of the models and to test the models;
- A Platform-independent component model, iComponent with a corresponding UML profile for constructing components as UML models according to MDA;
- An agile MDA approach for constructing executable models for service oriented components;
- Mappings of iComponent to some existing service-oriented component frameworks.

BACKGROUND

Why

Component-based approaches lead to applications developed and deployed as a set of components. The main benefits of these approaches consist of loose coupling among the application components, third-party component selection, and increased component reuse. In traditional component-based approaches the set of components is statically configured; this means that the benefits outlined above typically extend only to the development portion of the software system life-cycle, not to the run-time portion (Escoffier & Hall, 2007).

Nowadays, there are component models and frameworks which allow components unavailable at the time of application construction to be integrated into the application later into its life-cycle, i.e. after the application has been installed (OSGi, 2007). Such a framework should offer a dynamic execution environment, providing: (i) *dynamic availability of components* - the ability to install, update, and remove components at run-time, and to manage their provided and required interfaces; (ii) *dynamic reconfiguration* - the ability to change the configuration properties of a running component; (iii) *dynamic composition* - the ability to compose components from other existing components at runtime.

Most frameworks that support dynamic availability of components use the general principles of service-oriented component models (Cervantes & Hall, 2004), merging the concepts of service-oriented computing (Papazoglou & Georgakopoulos, 2003) into a component model.

Typically, a service-oriented component approach to build an application includes the following steps: (1) *Decompose the application into a collection of interacting services*. The semantics of these services are described independently of each other, and of any implementation. The service specifications will provide a basis for substitutability. (2) *Define a set of components implementing the application services.* A component may provide and require zero or more services. (3) *Define composite components that guide the application execution*. These composite components are described in terms of service specifications, and the concrete implementations of services will be resolved at run-time.

One of the main ideas for simplifying the construction of components is to separate the business logic of a component from the non-functional requirements related to the container in which the component execution will be managed. In such a context, developers concentrate first on implementing the business logic of the component, and then they configure declaratively the component deployment. Another important aspect of component models and frameworks refers to the development approach. Approaches in which modeling is at the core of the development