Chapter 16
Supporting Model-Driven Development:
Key Concepts and Support Approaches

Rita Suzana Pitangueira Maciel
Federal University of Bahia, Brazil

Ana Patrícia F. Magalhães Mascarenhas
Federal University of Bahia, Brazil

Ramon Araújo Gomes
Federal University of Bahia, Brazil

João Pedro D. B. de Queiroz
Federal University of Bahia, Brazil

ABSTRACT

The adoption of Model-Driven Development (MDD) is increasing and it is widely recognized as an important approach for building software systems. In addition to traditional development process models, an MDD process requires the selection of metamodels and mapping rules for the generation of the transformation chain which produces models and application code. However, existing support tools and transformation engines for MDD do not address different kinds of software process activities, such as application modeling and testing, to guide the developers. Furthermore, they do not enable process modeling nor the (semi) automated execution of activities during process enactment. MoDErNE (Model Driven Process-Centered Software Engineering Environment) uses process-centered software engineering environment concepts to improve MDD process specification and enactment by using a metamodeling foundation. This chapter presents model driven development concept issues and the MoDErNE approach and environment. MoDErNE aims to facilitate MDD process specification and enactment.

DOI: 10.4018/978-1-5225-3923-0.ch016

INTRODUCTION

Model Driven Development (MDD) is an approach that is primarily concerned with reducing the gap between problem and solution spaces. More specifically, the application of MDD relies on software implementation domains through the use of technologies that support systematic transformation of problem-level abstraction in software implementations (France & Rumpe, 2007). System models are not only used for system documentation, but they actually serve as a basis for the implementation phase. Each activity in the development process requires a number of input models that produce further models as output. This way the development of an application can be viewed as a set of transformations that lead to the final system. MDD has changed not only the way systems are built but also the way they are tested (Mussa et al., 2009). Model Driven Testing (MDT) (Blaker et al., 2007) is an approach based on MDD in which tests can be generated from development models in an automated way through the use of transformations. One of the most well-known initiatives in this scenario is Model-Driven Architecture (MDA) proposed by the Object Management Group (OMG) (OMG, 2003). MDA relies on several OMG standards to apply MDD concepts. As MDA is an MDD realization, in this text we use only the MDD acronym for software processes that use this approach, including those which follow OMG standards.

Unlike traditional development process models (Rational Unified Process (RUP), eXtreme Programming (XP), Open/UP, etc.), an MDD process requires the selection of metamodels and mapping rules for the generation of the transformation chain which produces models and application code. In this context, if modeling and transformation tasks are not properly performed, the desired final code will not be reached. Existing research in MDD practice has revealed the importance of software processes and suitable tools, concluding that they are crucial for the use of the MDD approach in industry (Hutchinson, Rouncefield, & Whittle, 2011).

The techniques to apply Model-Driven Engineering (MDE) correctly depend on tool support and integration in the software process project (Hutchinson, Rouncefield, & Whittle, 2011; Hutchinson et al., 2011). Many tools have been designed to support MDD. These environments usually have a specific focus on a transformation strategy or transformation engine in order to automatically generate models, codes and test cases from a variety of models. However, current MDD supporting tools are basically interested in defining and executing transformations which produce code and deployment artifacts from models (e.g. AndroMDA, BluAge and others) for a specific part of the software life cycle or for a specific domain. Indeed, other activities in a software process are usually not considered. They do not focus on the software process specification, neglecting support for the integration of different process specification activities into the software development process phases. On the other hand, tools for process modeling and specification (e.g. Eclipse Platform Foundation – EPF) lack integration to modeling tools and model transformation engines. This scenario does not help software engineers who want to use MDD as a main software development approach or to adapt existing software processes.

There have been attempts to integrate process design and enactment (Bispo et al., 2010). Environments called PSEEs (Process-Centered Software Engineering Environment) with different characteristics, features and contexts (Alves, Machado, & Ramalho, 2008; Baker et al., 2007) have been proposed. However, most of them have shortcomings regarding process enactment. Some of them are enactable but proprietary and use a non-standard Process Modeling Language (PML) (Magalhães et al., 2011). In some cases they have a restricted focus on the management view of a software development process alone (Gomes et al., 2011). PSEE concepts can be applied to support the enactment peculiarities of an