Chapter IV

MDA Design Space and Project Planning

Boris Roussev
University of the Virgin Islands, USA

Abstract

The change to Model-Driven Architecture (MDA) with Executable UML (xUML) results in changes to the existing object-oriented development practices, techniques, and skills. To use a transformational approach to create objects with Finite State Machines (FSMs), communicating by exchanging signals, adopters of MDA and xUML have to acquire expertise in new areas such as domain modeling, concurrency, non-determinism, precise modeling with FSM, programming in model-manipulation action languages, writing OCL constraints, and clear separation of application from architecture. The much more complex xUML object model presents system analysts with a longer and steeper learning curve. In this chapter, we critically evaluate the opportunities, capabilities, limitations, and challenges of MDA based on xUML. The purpose of our analysis is to aid organizations, software developers, and software product managers in their transition to this new development paradigm, and to assist them in understanding how MDA and xUML change the software design space and project planning.
Introduction

With the conventional way of developing software, analysts create high-level system models independent of implementation detail. Then, designers adapt these models to the constraints imposed by the implementation platform and the non-functional system requirements. Finally, programmers write the system code from the design models. The software design activity is the act of elaborating informal analysis models by adding increasing amounts of detail (Kruchten, 2000). Analysis-level models are informal because they do not have executable semantics, and therefore cannot be formally checked.

With MDA (Mellor & Balcer, 2002), analysts create executable models in xUML, test these models, translate them to design models, and finally compile (translate) the executable models into source code for a particular platform. The MDA design activity is the act of buying, building, or adapting mapping functions and model compilers, and targeting a particular software platform (e.g., J2EE or .NET). We say that MDA is a translational approach to systems development because executable analysis-level models are translated to system code.

Standardized approaches to software development, such as Agile (Beck, 1999) and RUP (Kruchten, 2000), are weathered best practices applicable to wide ranges of software projects. These methods are not sufficient in constructing effective project plans and processes, reflecting the unique realities in software firms and markets. As useful and distilled as they are, best practices have two major shortfalls: 1) lack of predictive relationship between adopted process and product quality; and 2) inability to adapt to changing environments. The following example illustrates our point.

Consider the statement, “XP (an Agile methodology) without pair programming is not XP.” This is a dogmatic, but not theoretically sound statement. There is no sufficient data in support of the proposition that the immediate feedback cycle of pair programming is better or worse than, say, solo programming with team software inspections, with respect to quality or productivity. There is evidence that in many contexts, software inspections produce products of superior quality and/or increase productivity (ePanel, 2003).

To overcome the deficiencies of the standardized approaches to software development, Russ and McGregor (2005), see the previous chapter, propose a model which structures the software development landscape into three cornerstone concerns: technology, process, and strategy. Each concern is