Chapter 7
Software System Modernization: An MDA-Based Approach

Liliana Favre
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina & Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, Argentina

Liliana Martinez
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Claudia Pereira
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

ABSTRACT

System modernization requires the existence of technical frameworks for information integration and tool interoperation that allow managing new platforms technologies, design techniques, and processes. The Model Driven Architecture (MDA) is aligned with this requirement. It is an evolving conceptual architecture to achieve cohesive model-driven technology specifications. MDA distinguishes the following models: Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). The integration of classical reverse engineering techniques with the MDA initiative will play a crucial role in software system modernization. In light of these issues, this chapter describes a framework for MDA-based reverse engineering that integrates static and dynamic analysis, meta-modeling, and formal specification. The essential idea is to combine static and dynamic analysis to generate software models (PSM and PIM) from code, and to analyze the consistency of these transformations by using meta-modeling techniques and formal algebraic specification. The chapter shows how to use reverse engineering to create PSM and PIM (that are expressed in terms of UML models) from object oriented code. More specifically, the chapter emphasizes the bases of a reverse engineering approach and describes how to reverse engineer class diagram, state diagram, activity diagram, and use case diagram within the context of MDA initiative.

DOI: 10.4018/978-1-61350-438-3.ch007

INTRODUCTION

Reverse Engineering is the process of analyzing available software artifacts such as requirements, design, architectures, code or byte code, with the objective of extracting information and providing high-level views on the underlying system.

Reverse engineering is an integral part of the modernization of legacy systems whose aging can or will have a negative impact on the economy, finance and society. These systems include software, hardware, business processes and organizational strategies and politics. Many of them may be written for technology which is expensive to maintain and which may not be aligned with current organizational politics, however they resume key knowledge acquired over the life of an organization. Important business rules are embedded in the software and, may not be documented elsewhere and the way in which legacy systems operate is not explicit. There is a high risk to replace them because they are generally business-critical systems (Sommerville, 2004).

Reverse engineering techniques are used as a mean to design software systems by evolving existing software systems for the purpose of adapt them to new requirements or technologies. 20 years ago, they focused mainly on recovering high-level architectures or diagrams from procedural code to face up to problems such as comprehending data structures or databases or the Y2K problem. By the year 2000, many different kinds of slicing techniques were developed and several studies were carried out to compare them. Basically, the initial reverse engineering techniques were based on static analysis and the concept of abstract interpretation, which amounts the program computations using value descriptions or abstract values in place of actual computed values. Abstract interpretation allows obtaining information about run time behavior without actually having to run programs on all input data.

When the object oriented languages emerged, a growing demand for reengineering object oriented systems appeared on the stage. New approaches were developed to identify objects into legacy code (e.g. legacy code in COBOL) and translate this code into an object oriented language. Object oriented programs are essentially dynamic and present particular problems linked to polymorphism, late binding, abstract classes and dynamically typed languages. For example, some object oriented languages introduce concepts such as the reflection and the possibility of loading dynamically classes, although these mechanisms are powerful, they affect the effectiveness of reverse engineering techniques. During the time object oriented programming, the focus of software analysis moved from static analysis to dynamic analysis, more precisely static analysis was complemented with dynamic analysis (Fanta & Rajlich, 1998; Systa, 2000).

When the Unified Modeling Language (UML) comes into the world, a new problem was how to extract higher level views of the system expressed by different kind of UML diagrams. Relevant work for extracting UML diagrams (e.g. class diagram, state diagram, sequence diagram, object diagram, activity diagram and package diagram) from source code was developed (Tonella & Potrich, 2005).

Nowadays, software and system engineering industry evolves to manage new platform technologies, design techniques and processes and a lot of challenges still need to be done. New technical frameworks for information integration and tool interoperation such as the Model Driven Development (MDD) created the need to develop new analysis tools and specific techniques. MDD refers to a range of development approaches that are based on the use of software models as first class entities. The most well-known is the OMG standard Model Driven Architecture (MDA), i.e., MDA is a realization of MDD (MDA, 2005).

MDA can be viewed as an evolution of OMG (Object Management Group) standards to support model centric development increasing the degree of automation of processes such as source code