Preface

The software industry has evolved to tackle new approaches aligned with the Internet, object-orientation, distributed components and new platforms. However, the majority of the large information systems running today in many organizations were developed many years ago with technologies that are now obsolete. These old systems, known as legacy systems, include software, hardware, business processes and organizational strategies and policies. Many are still business-critical and their complete replacement is dangerous and their maintenance is increasingly expensive. The amount of code in legacy systems is immense; there are billions upon billions of lines of code in existence that must be maintained.

The demand for modernization of legacy systems created the need for new architectural frameworks for information integration and tool interoperation that allow managing new platform technologies, design techniques and processes. The Object Management Group (OMG) adopted the Model Driven Architecture (MDA) that is an evolving conceptual architecture aligned with this demand.

Beyond interoperability reasons, there are other benefits to using MDA such as improving productivity, process quality and maintenance costs. MDA itself is not a technology specification, but it represents an evolving plan to achieve cohesive model-driven technology specifications.

All artifacts, such as requirement specifications, architecture descriptions, design descriptions and code are regarded as models. MDA distinguishes at least the following ones:

- **Computation Independent Model (CIM):** a model that describes a system from the computation independent viewpoint that focuses on the environment of and the requirements for the system. In general, it is called domain model.
- **Platform Independent Model (PIM):** a model with a high level of abstraction that is independent of any implementation technology.
- **Platform Specific Model (PSM):** a tailored model to specify the system in terms of the implementation constructs available in one specific platform.
- **Implementation Specific Model (ISM):** a description (specification) of the system in source code.

The idea behind MDA is to manage the evolution from CIMs to PIMs and PSMs that can be used to generate executable components and applications. In MDA, it is crucial to define, manage, and maintain traces and relationships between different models and automatically transform them and produce code that is complete and executable.

We can distinguish three main transformations in MDA processes: refinements, anti-refinements and refactorings. A refinement is the process of building a more detailed specification that conforms to another that is more abstract. On the other hand, an anti-refinement is the process of extracting from a more detailed specification (or code) another, more abstract specification that is conformed by the
more detailed specification. Refactoring means changing a model, leaving its behavior unchanged, but enhancing some non-functionality quality factors such as simplicity, flexibility, understandability and performance.

The initial diffusion of MDA was focused on its relation with the Unified Modeling Language (UML) as modeling language. However, there are UML users who do not use MDA, and MDA users who use other modeling languages such as Domain Specific Languages (DSLs).

MDA requires the ability to understand different languages such as general purpose languages, domain specific languages, modeling languages or programming languages. An underlying concept of MDA for integrating such languages semantically in a unified and interoperable way is metamodeling.

The essence of MDA is the Meta-Object-Facility (MOF) metamodel that allows different kinds of artifacts from multiple vendors to be used together in the same project.

MOF (latest revision 2.0) defines a common way to capture all the diversity of modeling standards and interchange constructs. It provides a metadata management framework where models can be, for instance, exported from one application, imported into another, stored in a repository and then retrieved, transformed, and used to generate code. The MOF 2.0 Query, View, Transformation (QVT) metamodel addresses queries on models, views on metamodels and transformations of models.

With the emergence of MDA, new approaches should be developed in order to reverse engineering, both platform independent and platform specific models, from object oriented code.

This book is a contribution for the demand of system modernization. In particular, the objective of this book is to analyze the integration of MDA with reverse engineering techniques to control the evolution of systems towards object oriented code.

A central problem is how to correctly define metamodels and align them with MOF. Inconsistencies in a metamodel specification will affect models and their implementations. MOF-metamodels are expressed as a combination of UML, the Object Constraint Language (OCL) and natural language. MOF has no built-in semantics apart from the well-formedness rules in OCL and what can deduced from them. This form of specification does not make it possible to validate that specific metamodels like UML metamodel conform to MOF (in the sense of each metaclass of the metamodel conforms a MOF meta-metaclass). A combination of MOF metamodeling and formal specification can help us to address MDA. A formal specification allows us to produce a precise and analyzable software specification and clarifies the intended meaning of metamodels. It also helps to validate model transformations, and provides reference for implementations.

In light of this, the book proposes an integration of classical compiler techniques, metamodeling techniques and algebraic specification techniques to make a significant impact on the automation of MDA-based reverse engineering processes.

The proposed approach has two main advantages linked to automation and interoperability. On the one hand, our approach shows how to automatically generate formal specifications from MOF metamodels. Due to scalability problems, this is an essential requisite. On the other hand, it focuses on interoperability of formal languages.

Reverse engineering and software evolution are crucial and complex research domains in software engineering. This book intends to increase the consciousness of the advantages of defining MDA-based reverse engineering and software evolution processes. It emphasizes techniques that are the foundations of innovative MDA processes and inspires research to open new frontiers with the power of MDA Case tools.

To date, most model-driven development research emphasizes on “Software Language Engineering.” Perhaps in the coming years, the focus will be on “Software Engineering Processes.” This book intends to shorten the path to this goal by providing an overview of several techniques that can be adopted in MDA-based processes.
This book was written for a broad audience of researchers, advanced students, professionals and those people that have adopted reverse engineering practices or are about to invest in system modernization. It encourages software professionals to explore the use of MDA for innovative projects that involve reverse engineering efforts combined with software evolution. It can also be used in advanced undergraduate courses to teach reverse engineering as an integral part of software design processes.

We assume that readers have a general knowledge of object oriented modeling, in particular UML models. A self-contained discussion of the principles of reverse engineering in a novel context including topics such as MDA, OCL, MOF, UML metamodel and QVT is presented.

**ORGANIZATION AND STRUCTURE**

The book is divided into six sections:

- **Section 1: Basics**
- **Section 2: Formalization of MDA Processes**
- **Section 3: Techniques Underlying MDA-Based Reverse Engineering**
- **Section 4: Conclusions**
- **Section 5: Selected Readings**
- **Section 6: Appendices**

Section 1 includes a discussion of the fundamentals of reverse engineering and MDA. It also includes a description of the main OMG standards involved in MDA processes. It introduces the main concepts of MOF-based metamodeling techniques for specifying platforms, models and metamodel-based transformations. It includes three chapters:

- **Chapter 1:** Reverse Engineering and MDA: An Introduction
- **Chapter 2:** Model Driven Architecture (MDA)
- **Chapter 3:** MDA, Metamodeling and Transformation

Section 2 describes foundations for metamodeling. It shows how to specify a metamodel by using formal specifications and how to generate formal specifications in an automatic way. Also, this section show how different formalization styles can be integrated. It includes four chapters:

- **Chapter 4:** Formalization of MOF-Based Metamodels
- **Chapter 5:** MOF-Metamodels and Formal Languages
- **Chapter 6:** Mappings of MOF Metamodels and Algebraic Languages
- **Chapter 7:** Mappings of MOF Metamodels and Object-Oriented Languages

Section 3 is a central part of this book. It describes underlying techniques in MDA processes, in particular in reverse engineering and software evolution. It describes how to adapt crucial techniques such as design patterns, model refactoring and pattern recovery in a way that fits with MDA. It includes the description of a framework for reverse engineering object oriented code. It includes three chapters:

- **Chapter 8:** Software Evolution, MDA and Design Pattern Components
- **Chapter 9:** Evolution of Models and MDA-Based Refactoring
- **Chapter 10:** MDA-Based Object-Oriented Reverse Engineering
Section 4 summarizes the main contributions and includes strategic directions and challenges in MDA reverse engineering and software evolution. It includes two chapters:

- **Chapter 11:** Summing Up the Parts
- **Chapter 12:** Towards MDA Software Evolution

Finally, the book also includes appendixes and selected readings that provide complementary information about metamodels, platforms, languages and formalisms.

Next, we describe contents of the different sections.

**Section 1: Basics**

**Chapter 1: Reverse Engineering and MDA: An Introduction**

This chapter gives an overview of state-of-the-practices in reverse engineering techniques and motivates the interest that Model Driven Reverse Engineering has gained in different application areas related to the evolution of existing software.

**Chapter 2: Model Driven Architecture (MDA)**

Chapter 2 explains MDA and its main concepts such as model, metamodel and transformations. It introduces the main OMG standards related to MDA such as UML Infrastructure, UML Superstructure, MOF, QVT and XMI (XML Metadata Interchange). Besides, it includes a comparison of UML Profiles, metamodels and DSL (Domain Specific Languages).

**Chapter 3: MDA, Metamodeling and Transformation**

Chapter 3 explains the main MOF modeling concepts and MOF-based transformations. It also provides UML/OCL notation for specifying metamodels and transformations. It includes MOF-based metamodels for object oriented languages such as Java, C++ and Eiffel and examples of transformations.

**Section 2: Formalization of MOF-Based Processes**

**Chapter 4: Formalization of MOF-Based Metamodels**

This chapter proposes a combination of metamodeling and formal specification techniques to address MDA-based processes involved in software evolution. It introduces an MDA Infrastructure, a minimal subset of packages of OMG standards to formalize MDA process. It describes an algebraic language called NEREUS which is suited for specifying metamodels, and particularly the MDA Infrastructure. The chapter also includes several examples of specifications such as OCL Collection and the QVT Core package.

**Chapter 5: MOF-Metamodels and Formal Languages**

The chapter explains how to integrate MOF metamodels with formal specification. It describes a bridge between MOF and NEREUS that is supported by reusable schemes and a system of transformation
rules for translating OCL specifications into NEREUS. The chapter exemplifies the different steps of the transformation process.

**Chapter 6: Mappings of MOF Metamodels and Algebraic Languages**

The chapter analyzes mapping between MOF metamodels and traditional formal languages. In particular, it examines the relation between NEREUS and CASL (Common Algebraic Specification Language) as a common algebraic language. It proposes a transformation process that could be automated.

**Chapter 7: Mappings of MOF Metamodels and Object-Oriented Languages**

The chapter analyzes mapping between MOF metamodels and traditional object oriented languages. In particular, it examines the relation between NEREUS and the Eiffel language. It proposes a transformation process and a set of heuristics for integrating OCL specification and Eiffel contracts.

**Section 3: Techniques Underlying MDA-Based Reverse Engineering**

**Chapter 8: Software Evolution, MDA and Design Pattern Components**

This chapter describes how to define MDA-based reusable components. It defines a megamodel for defining MDA components at different abstraction levels (PIM, PSM and ISM). Considering the relevant role that design patterns have in software evolution, this chapter exemplifies MDA components for classical design patterns. Besides, it shows how to integrate design patterns components with MDA-based processes and also introduces formalization of metamodels and metamodel-based transformation. This chapter is used to exemplify the specification of refinements.

**Chapter 9: Evolution of Models and MDA-Based Refactoring**

The chapter analyzes MDA-based refactoring techniques. It explains an MDA framework for refactoring that is structured at three different levels of abstraction linked to models, metamodels and formal specification. The main contributions of this chapter are the definition of refactorings as metamodel-based transformations that are expressed as OCL contracts, a technique for identifying refactoring patterns and an algebraic formalization of refactorings. The chapter proposes a uniform treatment of refactoring at level of PIM, PSM and code.

**Chapter 10: MDA-Based Object-Oriented Reverse Engineering**

This chapter describes a reverse engineering approach that fits with MDA. It explains a framework to integrate different techniques that come from compiler theory, metamodeling and formal specification. It emphasizes the use of static and dynamic analysis for generating MDA models. The chapter also shows how MOF and QVT metamodels can be used to drive model recovery processes. It also describes how metamodels and transformations can be integrated with formal specifications in an interoperable way. The reverse engineering of class diagram and state diagram at PSM level from Java code is exemplified. This chapter is used to exemplify the specification of anti-refinements.
Section 4: Conclusions

Chapter 11: Summing Up the Parts

This chapter summarizes the main results described in the book and challenges in MDA reverse engineering.

Chapter 12: Towards MDA Software Evolution

This chapter discusses software evolution, challenges and strategic directions in the context of MDA.

Section 5: Selected Readings contains two previously published chapters. For Section 6: Appendices, contains four appendices. Appendix A: Platform Specific Metamodels and Language Metamodels; Appendix B: OCL and NEREUS: Type System; Appendix C: Transformation Rule System; and Appendix D: Design Pattern Metamodels.