Preface

Computer systems became a settled thing in people’s life. Their application involves diverse aspects of our life from control of our family budget to solving serious scientific questions. Each domain of application or adoption of complex computer systems, a problem domain, has its own specific character. Complex computer systems, a solution domain, are intended for help in coping with this specificity and not for bringing additional hardship or constraints.

The specificity of the problem domain must be either adequately reflected in the computer system or adequately simplified, e.g. automated, optimized, modified, etc. Adequacy of the computer system to the application domain can only be provided by sufficient analysis and accurate modeling of the problem domain. Accurate means not only exact and complete, but also unambiguous for software developers who will take part in other stages of development.

Models and modeling have long history starting from times of the Ancient Near East and the Ancient Greek. People used to represent the surrounding world by using simple non-mathematical models. However, importance of mathematical models was also recognized and similar methods for building models were developed independently in China, India, and Persia. The closer to the modern times, the more complex become models and their building methods.

A model is a simplified representation of something that is real. In building models, the starting point is considering the real-world object, highlighting the necessary characteristics of this object and simplifying of unnecessary ones. Therefore, models describe only a part of the real-world object and are limited by their scope of application. An accurate model is a model that completely and unambiguously describes characteristics of the real-world object within the scope of application.

One of steps dedicated to increase accuracy of software development is appearance, or better to say, explicit formulation of ideas and principles of Model Driven Architecture (MDA) proposed by the Object Management Group (OMG). OMG MDA focuses developer’s attention on proper design of the solution domain by using models. Undoubtedly, models as a way of reflection of the solution domain have also existed before MDA. These models were formal and informal. Formal models have been used and are used at the present mostly for process simulations. However, because of their complexity, a use of formal models is relatively narrow in the beginning of software development. In turn, informal models have wider application because of their simplicity, understandability and fast trainability. An additional factor of popularity of informal models was and stays a way of using models in software development. Models are not used as basic documentation, but as additional illustration of the solution domain, i.e., what is planned to implement. Code is that was and remains the basic documentation.

Software development uses different techniques for identification and specification of characteristics (specific or not) and requirements for a planned system. Most of them are primarily aimed at solution
domain analysis, while the problem domain is regarded almost as a black box describing a number of aspects of the system. Formal mapping of knowledge about the problem domain into the software development process is a challenge until now.

The above described situation raises a few issues. The first one, which is the most important in our opinion, is a gap between the solution domain – a software system – and the real world. What domain is to be modeled at first: the domain of today’s reality (“as is” or the problem domain) or the domain of customer’s expected reality (“to be” or the solution domain)? If we develop software that must be used for some purposes in the real world, we must know how it will affect this world. It has especially critical importance for mechatronic, embedded and high risk systems, where failure costs could be human lives. Analysis of the problem domain is also important for complex business systems, wherein failures could lead to huge financial losses. The second issue is separation of concerns and unambiguous representation of the domains in specifications. We believe that formal mathematical methods could reduce inaccuracies and ambiguities of specifications.

Good news is that OMG MDA serves as a stimulus for changing the state of affairs. At the present, MDA puts the main focus on architectural separation of concerns in specifications, domain design, model transformation, and code generation activities. OMG MDA declares three independent but related viewpoints and corresponding models:

- Computation Independent Viewpoint and Computation Independent Model (CIM);
- Platform Independent Viewpoint and Platform Independent Model (PIM);
- Platform Specific Viewpoint and Platform Specific Model (PSM).

Models transformations are used for refinement of models. The transformation process can be performed manually, using profiles, using patterns and mappings and automatically. A mapping is a specification, including rules, output and input models, and other information, for transforming an input model in order to produce an output model. Technical choices and quality requirements can also be used as an input for transformation patterns. Models and mappings can be reused. MDA defines the following types of the mapping:

- A model type mapping, when it is specified between types described in a PIM language and a PSM language;
- A model instance mapping, where PIM elements, which are to be transformed, have to be identified and transformed in a particular way (marks can be used for identification of elements; marks may specify quality of service requirements on the implementation);
- A combined type and instance mappings;
- Templates, which are parameterized models that specify particular kinds of transformations;
- A mapping that is specified using a language to describe transformation of one model to another.

In our opinion, MDA and MDE partially address the second issue, namely, the separation of concerns. The PIM and PSM are related to customer’s expected reality, the solution domain. The theoretical foundation of the CIM is quite unclear. Considering its definition, we can assume that it relates not only to customer’s expected reality, but also to the domain of today’s reality (the problem domain).

Domain analysis is not less important activity than design. Thus, Model-Driven Engineering (MDE) uses domain specific modeling languages (DSMLs) for domain analysis and modeling. DSMLs help
in coping with domain specificity if universal modeling languages could not help. However, the issue is that models, which reflect the problem domain, usually are informal. This fact complicates application of transformation mechanisms from the beginning of software development. In order to implement transformation models must be at least formal in MDA sense, i.e. understandable by a computer, and at most they must have a formal theoretical base.

We believe that the use of formal models and approaches at the very beginning of the MDA software development life cycle addresses the both previously mentioned issues keeping and improving mechanisms of MDA. Formal mathematical methods could reduce inaccuracies and ambiguities of specifications. However, in contrast with informal methods, they usually require additional efforts in study and use. Therefore most developers of modeling languages try to avoid pure usage of those methods and suggest practice of so called semi-formal modeling languages that are formally described, but do not have any formal mathematical foundations. This book is dedicated to shift the way of developers’ thinking, and to show that mathematics could be “light-weight”. This means that mathematics is able to “hidden” complexity of its own mechanisms from a user, while keeping their efficiency.

AUDIENCE

The book is primarily intended for three audiences:

1. Software developers with some practical experience in development of information and software systems, who wish to acquire knowledge in applying Model Driven Architecture and Model Driven Engineering for systems such as software product lines, distributed systems, web application portals, and so on;
2. Professionals and researchers who interested in improvement of productivity of software development and quality of software products;
3. And could also be useful for computer science and information technology students who want to know more about the most advanced methods and approaches in software development.

Some prerequisite knowledge is desirable in order to benefit from the book:

1. Knowledge of the fundamental concepts of requirements engineering and object-oriented technology, such as business rules, classes, relationships, etc.;
2. Knowledge of the fundamental principles of Model Driven Architecture (and related languages such as a Unified Modeling Language) and Model Driven Engineering.

CONTENTS OF THE BOOK

The aim of this book is to give an overview of the current achievements in model-driven software development and to highlight those important things, which require deep research and efficient solutions. The necessity of efficient solutions that apply formal theories appeared from the longstanding crisis in software engineering. Just unification of standards and use of models without proper domain analysis will be not enough to get out of the crisis, while the very beginning of software development lacks ap-
lication of formal (mathematical and system) theories. Application of Model Driven Architecture will be ministerial to wide understanding of this fact. Section 1 of the book, *Theory-Driven Holistic Domain Modeling and Analysis in the Context of MDA*, comprises several discussions on application of formal theories and mechanisms for domain modeling and analysis within Model Driven Architecture:

- **Chapter 1, Is Modeling a Treatment for the Weakness of Software Engineering?** Crisis in software engineering still is in progress. Software development was and is rather art than engineering discipline. Osis and Asnina consider causes of weakness of software development and possible ways of improvement suggested by software community. They believe that Model Driven Architecture, which put main focus on models and modeling, can become a skeleton of software engineering, but it is not enough. In order to become software engineering in its actual sense, software development requires, first, a solid theory that could also be presented in a form of “light mathematics”; and second, standardization and critical assessment of development techniques.

- **Chapter 2, Topological Modeling for Model-Driven Domain Analysis and Software Development: Functions and Architectures.** Architectural separation of concerns and model transformations provided by Model Driven Architecture is an important step towards formalization of software development, i.e., towards software engineering. However, true software engineering requires engineering models and techniques, which have solid theoretical foundations. Osis and Asnina discuss the theoretical basis of the topological functioning model and topological modeling that can be a part of those foundations. Construction of the topological functioning model promotes careful domain analysis and formal specification of the domain. In turn, this formal model can serve as a source for further activities of model-driven software development.

- **Chapter 3, Topological Functioning Model as a CIM-Business Model.** Asnina and Osis discuss the first viewpoint, computation independent, and the corresponding model proposed by the OMG in Model Driven Architecture. Analysis of scientific publications shows that this model may include three sub-models conventionally named CIM-Knowledge Model, CIM-Business Model, and CIM-Business Requirements for the System. The first sub-model reflects a problem domain. The last one reflects a solution domain. The CIM-Business Model is a bridge between the problem and the solution. This chapter demonstrates properties of the topological functioning model as such a bridge – a formal CIM-Business Model.

- **Chapter 4, Derivation of Use Cases from the Topological Computation Independent Business Model.** Osis and Asnina demonstrate derivation of use cases from the formal domain description, Topological Functioning Model. Theoretical foundations of this model ground on topology and system theory help in avoiding several limitations of use cases while keeping their simplicity, e.g. strict orientation on a solution, informal identification, conflicts and duplicates among use cases. Topological and functional properties of the topological functioning model allow establishing formal conformance between the solution domain model expressed by use cases and the problem domain model, thus increasing adequacy of the solution. In its turn, goal-based decomposition of the topological functioning model deals with conflicts and duplicates of use cases.

- **Chapter 5, A Multidimensional Approach for Concurrent Model-Driven Automation Engineering.** Rose et al. introduce a conceptual framework for the customization and integration of domain-specific modeling languages and tools in automation engineering, Concurrent Model-Driven Automation Engineering (CMDAE). CMDAE developed by Siemens AG and Technical University of Darmstadt relies on a multidimensional framework for model classification and manipulation purposes.
CMDAE puts the main focus on the exchange and alignment of models needed for hardware and software development of automation devices and on the presentation of some extensions of the underlying graph grammar formalism motivated by its application to a real-world scenario. Application of this formal framework is demonstrated by the running example of High-Bay Warehouse Automation.

Section 2, *Model-Driven Engineering and Model Driven Architecture*, contains chapters, which describe application of Model-Driven Engineering and Model Driven Architecture for solving particular challenges in software system development. Chapters consider application of domain-specific techniques for cloud computing and distributed quality of services; and MDA for distributed real-time and embedded systems, system, and design of executable UML components, e-services and database systems. The proposed applications allow heightening quality of software development processes and products:

- **Chapter 6**, *Model-Driven Configuration of Distributed Real-time and Embedded Systems*. Dougherty, White and Schmidt suggest applying a paradigm of Model-Driven Architecture for ensuring that real-time quality-of-service (QoS) and resource constraints in distributed real-time and embedded systems (DRE) are satisfied. This chapter presents MDA techniques and tools that simplify and automate the configuration of COTS components for DRE systems. The authors provide an incremental methodology based on MDA principles for constructing modeling tools that can be used to create model instances of potential DRE system configurations.

- **Chapter 7**, *Model-Driven Automated Error Recovery in Cloud Computing*. Sun et al. represent application of a Model-Driven Engineering paradigm, models and Domain Specific Modeling Languages (DSML), to automate error detection and recovery in clouds. The suggested approach is based on a run-time model specified by DSML that monitors and manages the running nodes in a cloud. Editing the models instead of using the traditional command-line interface facilitates administering changes in clouds. Besides that, the proposed approach allows semi-automation of cloud recovery - manually created recovery patterns can be used by automation infrastructure, which automatically replays the same recovery actions when an identical error state is encountered in the future.

- **Chapter 8**, *Productivity Analysis of the Distributed QoS Modeling Language*. Hoffert, Schmidt and Gokhale discuss quantitative benefits of Distributed Quality-of-Services Modeling Language used within Model-Driven Engineering for designing valid QoS policy configurations and transforming the configurations into correct-by-construction implementations. The performed productivity analysis of DQML shows significant productivity gains compared with common alternatives, such as manual development using third-generation programming languages. The Data Distribution Service Benchmarking Environment tool suite is used as a case study to highlight the challenges of developing correct and valid QoS configurations, as well as to analyze the productivity benefits of DQML.

- **Chapter 9**, *Domain-Driven Reuse of Software Design Models*. Kalnins et al. present an approach that combines facilities of model-driven software development and software reuse. The development process is illustrated starting from requirements specification in semiformal language. Then, the authors demonstrate composing of a platform independent model from the specified requirements by using transformations. Transformations are also applied to the platform independent model in order to get the software case, platform specific models and code. The suggested way of require-
ments specification supports discovering of similarities in requirements, which further can be used for reuse.

- Chapter 10, *Quality-Driven Database System Development*. Dubielewicz et al. discuss integration of database system development based on Model Driven Architecture with quality models. The authors suggest using Quality-driven MDA framework that combines MDA software development and quality specification and assessment processes. The suggested framework demonstrates assessment of database system models by using external and internal quality models during the development process. The result is controlled quality of the final software product.

- Chapter 11, *Exploring Business Value Models for E-Service Design*. Zdravkovic and Ilayperuma discuss challenges in efficient application of inter-organizational information systems where necessity to keep aligned business and ICT models exists. The authors suggest an MDA-based approach for design of Web services and their coordination. This approach focuses on a value-explorative analysis and modeling of business services starting from the computation independent model (CIM). Well-defined mappings and UML2 are used for transformations from computation independent to platform independent model.

- Chapter 12, *An MDA Approach for Developing Executable UML Components*. Motogna, Pârv and Lazăr discuss a use of main principles of model-driven development for applications that consist of service-oriented components. The authors suggest a framework for Software Component Definition, Validation, and Composition, ComDeValCo. It consists of modeling (action) language, a component repository and a toolset, and supports separation of business logic and non-functional requirements. This framework can be used for developing executable service-oriented component models and in many software engineering courses as an example of applying model-driven principles in the software development process.

Section 3, *Modeling of Product Lines and Patterns*, is dedicated to the development of Software Product Lines (SPLs) by using principles of Model Driven Engineering and MDA. In particular, chapter authors consider model-driven impact analysis, deployment, configuration, testing, and requirements specification of SPLs. Besides that, classification of patterns according to OMG metalevels and MDA principle of architectural separation of concerns is proposed. Chapters that summarize authors’ experience are the following:

- Chapter 13, *Model-Driven Impact Analysis of Software Product Lines*. Cho et al. discuss how model-driven engineering can help software developers to deal with difficulties in understanding the consequences of changes in software product lines. The approach uses domain-specific modeling techniques to automate establishment of traceability relations, and impact analysis to understand the effect of changes on a product line. The domain-specific modeling makes it possible to lift the abstraction and representation of a software change such that it can be analyzed and separated from specific platform and environment details that would exist at the implementation level. The authors demonstrate the application of model-driven impact analysis by a case study in the domain of Mobile Media.

- Chapter 14, *Systematic Use of Software Development Patterns through a Multilevel and Multistage Classification*. Azevedo et al. propose a systematic approach for identification of patterns that should be used in each certain stage of software development. This approach suggests multilevel and multistage pattern classification that is based on the software development process. The authors
believe that leveraging patterns to metalevel M2 helps to turn the decisions on their application more objective as well as to reduce the misinterpretation of patterns from catalogues and the corruption of patterns during the pattern adaptation process. The pattern classification by type is illustrated by pattern examples.

- Chapter 15, Reducing Enterprise Product Line Architecture Deployment and Testing Costs via Model-Driven Deployment, Configuration, and Testing. White and Dougherty discuss evaluation of the reliability of product-line architectures by using a technique supported by Model Driven Architecture. The large number of variations in product-line architectures is a challenge for developers, because only working configurations should be used. The MDA-based technique allows automation of much of the deployment, configuration, and testing of product-line architectures. The suggested approach is validated using a distributed constraint optimization system case study.

- Chapter 16, Applying UML Extensions in Modeling Software Product Line Architecture of a Distribution Services Platform. Dobrica and Ovaska present a use of the built-in extension mechanisms of UML, namely, stereotypes, constraints and tagged values, for specification variations in software product line architectures for middleware services. The authors use two abstraction levels of design, conceptual and concrete, and three architectural views, structural, behavioral and deployment. The conceptual level means delayed design decisions concerning, e.g. technologies to be selected or details in functionality, whereas the concrete level illustrates realization of conceptual architecture. The proposed UML profile together with the design method would allow extensive and systematic using and maintaining software product line architectures in the framework of Model Driven Architecture.

- Chapter 17, Model-Driven Requirements Specification for Software Product Lines. Alférez et al. discuss contribution of Model-Driven Development for requirements engineering in software product lines. The authors provide an overview of different approaches for specifying requirements models and composing models for specific products of a software product line and introduce a classification of them. One of the challenges in software product line modeling is mixing variability data into requirements specifications. The authors focus their research on exploiting a model-driven requirements specification approach, i.e. use case scenario-based techniques and feature models to model variability information in combination with specially tailored composition rules for requirements models provided by the Variability Modeling Language for Requirements (VML4RE).

Section 4, Surveys, comprises two interesting, wide surveys. The first one discusses domain modeling approaches in the field of information system engineering. This topic is very interesting because it shows fields that MDA covers at the present and should cover in future. The second one gives an overview of techniques for performance evaluation of Web application portals and analyzes advantages of the proposed model-driven approach. They are the following:

- Chapter 18, Domain Modeling Approaches in IS Engineering. Kirikova provides an overview of domain modeling approaches and techniques used in Information System (IS) Engineering mainly in requirements engineering and enterprise modeling. This overview is interesting for users of Model-Driven Architecture (MDA), because IS Engineering has applied many different domain models long before the appearance of Model-Driven Architecture. The author demonstrates a place of MDA in the framework of cognitive engineering. Besides that the author discusses tool support for construction, maintenance and change management of the considered domain models.
• Chapter 19, Model-Driven Performance Evaluation of Web Application Portals. Roy and Schmidt give an overview of performance evaluation techniques. Basing on this overview, they discuss integration of Model Driven Architecture methods with performance analysis methods of Web application portals. The authors use this model-driven performance analysis and system management throughout Web application portal development lifecycles. Model-driven techniques and tools can predict web application portal performance accurately and help close the gap between domain-oriented performance problems and conventional performance modeling technologies. The proposed strategy should simplify the analysis of web application performance.

The first part of the book is special. It demonstrates application of the “light mathematics” for formalization of problem domain modeling, and hence, for advancement of software development. The book shows that application of formal models and methods as well as knowledge of system development accumulated in requirements engineering and enterprise modeling (as mentioned in the last part of the book) are ways of increasing quality of problem domain analysis and modeling in the field of MDE, in general, and MDA, in particular.

The rest of chapters of the book are dedicated to different applications of MDA and MDE in software development for increasing quality of software products (e.g. web applications, database systems, software product lines) and processes (impact analysis, performance analysis, QoS analysis). These applications demonstrate MDA and MDE power in implementing architectural separation of concerns.

Janis Osis and Erika Asnina
Riga, Latvia,
November, 2010