The Impact of the Model-Driven Approach to Software Engineering on Software Engineering Education

Abdelwahab Hamou-Lhadj
Concordia University
Montréal, Québec, Canada
abdelw@ece.concordia.ca

Abdelouahed Gherbi
Concordia University
Montréal, Québec, Canada
gherbi@ece.concordia.ca

Jagadeesh Nandigam
Grand Valley State University
Allendale, MI, USA
nandigaj@gvsu.edu

Abstract

As businesses rely on software solutions to preserve their position in a highly competitive market, the need for reliable and robust software systems is vital. Lately, there has been a significant interest in building software using models as their main artifacts. Unlike traditional development techniques which tend to be code-centric, model driven approaches, such as the Model Driven Architecture (MDA) standard, stress the usage of models at all levels of the software development life-cycle. The MDA, which is supported by the Object Management Group (OMG), is based on a comprehensive set of standards including MOF, UML, and OCL, to mention a few. This paradigm shift in software engineering has impacted not only the way software is built but also the way software engineering is being taught - The standards introduce a significant body of knowledge that should be integrated in a software engineering curriculum. In this paper, we discuss the impact of the model-driven software engineering approach on software education. This discussion is based on an experience teaching a graduate course on model-driven software engineering at Concordia University.

Keywords: Model-driven software engineering, software engineering education, model-driven architecture (MDA).

1. Introduction

Perhaps, the most widely accepted definition of software engineering is the one given in IEEE Std 610.12-1990, which defines software engineering as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" [1]. As a result, teaching software engineering has always focused on software processes, specification, design, development, verification, validation and management [2]. Well established concepts and techniques such as object-oriented concepts, design patterns, service-oriented architecture, etc. have emerged over the years to support these activities. The model-driven approach to software engineering is perhaps one of the most recent techniques in software engineering, where models are first class artifacts, compared to code-centric approaches, which has dominated the software engineering arena for decades.

This paradigm shift, however, has not only impacted the way software systems are engineered but also the way software engineering is being taught. Indeed, teaching the model-driven approach to software engineering calls for a whole new set of techniques and concepts.

The objective of this paper is twofold:

- Present and discuss the concepts and enabling technologies of the model-driven approach to software engineering (based on MDA standards) which should be covered in a course on the topic.

- Report on our experience[1] teaching these concepts to graduate students. We also discuss the challenges encountered by the students when learning these concepts as well as their concerns with respect to how much they should know and the steepness of the learning curve.

The remaining part of this paper is organized as follows: In Section 2, we present the concepts that are involved in teaching model-driven software engineering based on the MDA standards. We discuss the experience teaching a course on model-driven software engineering in Section 4. In Section 5, we present a brief overview of the literature on software

[1] To be more precise, this is based on the first author's experience teaching a graduate course on the model-driven approach to software engineering at Concordia University.

education with a focus on model-driven software engineering. We conclude the paper in Section 6.

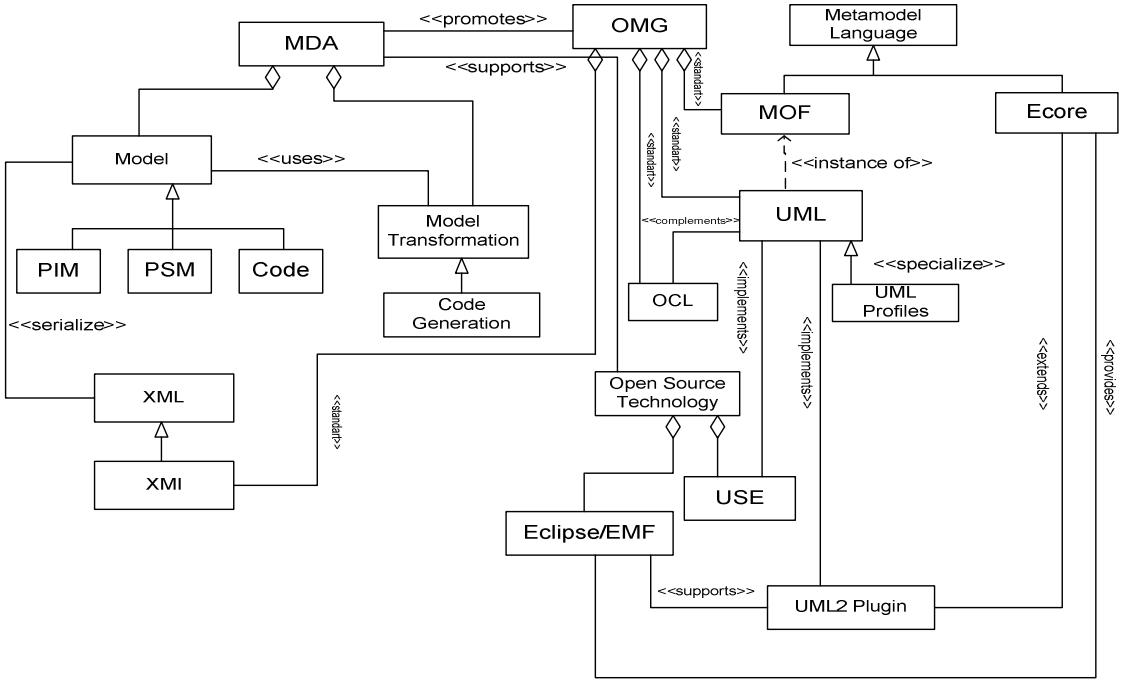

Figure 1. A Metamodel of the concepts involved in teaching MDA

2. Model-Driven Software Engineering Concepts

Figure 1 depicts the main concepts that are involved in MDA. We discuss each of these concepts in the following subsections.

2.1. Model-Driven Architecture (MDA)

The main idea behind the model-driven approach to software engineering is that software can be built starting with an abstract representation, which is successively refined until an implementation is reached. MDA, which is perhaps one of the most comprehensive approaches to model-driven software engineering, promotes the separation of design of the software system from its implementation. This separation ensures that the system be deployed on various platforms without having to change the design. The key concepts in MDA are, as illustrated in Figure 2:

- A Platform Independent Model (PIM): PIM is an abstract specification of the software business logic. The PIM is often expressed using UML or a particular domain specific language, which is often captured as a UML profile. Building a UML profile requires an extensive understanding of the UML metamodel as well as a good knowledge of MOF (Meta-Object Facility).

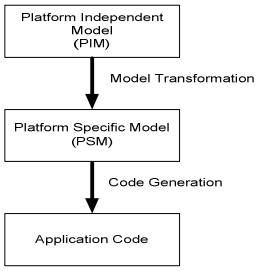

Figure 2. MDA Key Concepts

- A Platform Specific Model (PSM): PSM is a more detailed model than PIM, which encompasses details of the implementation platform. Code is generated automatically from a PSM.
- A model transformation is a mapping between models through transformation rules. A mapping from a PIM to a specific PSM leads to a system that runs on the platform described by this PSM. Transformation rules are expressed in a transformation language such as the OMG's standard model transformation language, QVT

(Query-View-Transformation) [18], or other several other languages have been used, including OCL, XSLT, MTrans, ATL.

2.2. Meta Object Facility (MOF)

MOF is a model that defines UML metamodel. As such, it is often referred to as a metametamodel. MOF is designed as a four-layered architecture (see Figure 3). The M3 layer is the language used by MOF to build lower level metamodels. UML metamodel is defined in Layer M2. An instance of UML metamodels (these are the domain models) are represented in layer M1, where instances of these models are defined in Layer M0. One of the key advantages of having MOF is the ability to exchange domain models that are created using MOF-compliant languages (such as UML).

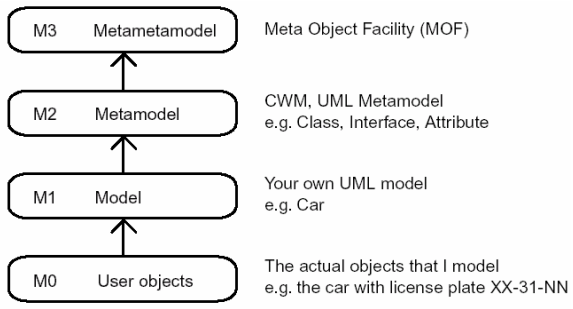

Figure 3. Four-layered architecture (from [4])

It is important to understand MOF in a model-driven software engineering course for a variety of reasons:

- MDA requires that any modeling language used to model PIMs and PSMs must be compliant with MOF to enable model transformations and exchange.

- UML semantics may need to be extended to capture specific concepts that exist in some application domains. For example, real-time systems require handling concepts such as time, sensors, actuators, etc., which are not readily supported by UML. Extending UML semantics requires not only an understanding of UML metamodel but also MOF.

2.3. Unified Modeling Language (UML)

UML with its set of diagrams (e.g., class diagrams, state diagrams, etc.) has long been taught in many software engineering courses. However, from the MDA perspective, teaching UML takes students beyond the diagrams to exploring its metamodel. This is necessary to address its extension through UML profiles as discussed earlier.

UML profiles are built using UML extension mechanisms such as stereotypes, tagged values, and constraints. These extension mechanisms (as well as their metamodel) must be an integral part of a course on MDA concepts.

A UML profile is a special version of UML tailored to the specifics of a particular domain. There are many UML profiles that have been adopted as OMG standards including the UML profile for real-time systems [5], the UML profile for system engineering (SysUML) [19], etc. The general approach used to define a UML profile comprises two steps: the definition of a domain model (this can be considered as the metamodel of the profile) and the mapping of the domain model to UML [6]. Discussing UML profiles and how they can be enforces the fact that any application domain can be modeled through UML or a UML profile. This can help students realize the power of modeling and the applicable modeling techniques.

2.4. Object Constraint Language (OCL)

The Object Constraint Language is declarative, side-effect free language, which was initially developed by IBM and then evolved into an OMG standard [9]. This language is now an integral part of the UML modeling language. OCL is used to specify constraints an UML models to make them more precise. It is also used to specify well-formedness rules for a metamodel, and validate particular instances of this metamodel against it.

2.5. Executable UML

Executable UML (xUML) is a UML profile that contains a subset of UML model elements (with changes to their semantics) that can be used to build PIMs from which code can be generated automatically [16]. xUML is based on two main UML diagrams, namely, the class and state diagrams. These diagrams have been refined to remove semantically weak elements. The purpose for doing this is to leave only these concepts that are implementable.

One of the key needs of xUML is for an action language, to express various types of actions such as class operations, constraints, bridge functions, etc., which complies with UML action semantics specifications [7]. There currently exist several action

languages including Object Action Language (OAL) [16], Action Specification Language (ASL) [17], That Action Language (TALL) [16], etc, however, none have yet gained wide acceptance. This is partly attributable to the fact they use syntactic constructs that are unfamiliar to many programmers (e.g., those from a Java background) making them hard to learn.

2.6. Open Source Support for MDA

Several open source initiatives support the model driven software engineering. The products of these initiatives represent the main components of a course lab that can be used for course projects.

Teaching UML and OCL can be practically illustrated and experimented using the open sour UML-based Specification Environment USE tool [10]. In addition several available tools can be used to illustrate modeling different aspects of software engineering.

The Eclipse Foundation supports MDA through the Eclipse Modeling Framework (EMF) [14]. This is a modeling and code generation facility. The metamodel of this framework is Ecore which is aligned with MOF [15]. It is worth mentioning that some industrial products such MagicDraw and Rational Software Architect (RSA) have specific editions for education. These are powerful tools that enable using full features such as the implementation of UML profiles.

3. Experience Report on Teaching a Course on MDA

In this section, we report on our experience teaching a graduate course on model-driven software engineering to two sections. The total number of student is 52. Before taking this course, the students must pass other software engineering courses that cover, at a minimum, basic object-oriented concepts, UML diagrams, software design methodologies, and a course on general concepts in software engineering, which discusses processes, maintenance, project management, and testing.

A course section is organized as a three-hour lecture a week during 13 weeks. Since it is a graduate course there is no specific lab component. The material covered in this course includes:

- Model driven architecture
- Platform-independent modelling with domains
- Specifying constraints with OCL
- Overview of UML Metamodel and MOF
- UML Profiling
- Executable UML
- Action Semantics Specification
- Automatic System Generation
- Modeling Patterns
- Case studies (all along the course)

The textbooks used are [4, 16]. These books were selected after reviewing many books about MDA. Some additional references such as the book [17] were posted on the course website as complimentary texts.

The course has a group project component where the students need to design a software system the model-driven software approach. At the end of the class, the students present their project to the class along with the challenges encountered during the design the system. They were also encouraged to have a slide or two where they can summarize their learning experience of the course concepts throughout the semester.

In what follows, we summarize the main concepts that the students had difficulties with. We acknowledge that the discussion with the students was informal and a more controlled experiment is needed. We do, however, recognize that the experience gained from these discussions can pinpoint the bottlenecks with teaching these new technologies. We therefore share this experience with the reader. The challenges and concerns raised by the students are presented in what follows:

The multitude and the complexity of the concepts:

Many students expressed some concerns with respect to the number of the new concepts introduced by MDA. When asked to elaborate on this. Some students pointed out to the number of new languages and related constructs that need to be learned including UML, OCL, QVT, and OAL (the action language used in the course). Students, in particular, had difficulties mastering OCL. This was also shown in their projects where, except for simple constraints, OCL was almost absent from the domain model. They also found OAL's syntax easy to understand but they would have preferred something similar to what they are already familiar with such as C++ and Java-like syntax. Most students understood the value of having something like QVT but again they had concerns with the learning curve that QVT will require.
In addition, the students seemed to have trouble grasp the whole concept of metamodeling. MOF and UML profiling seemed to be high-level concepts to them. They wanted to see a more practical example on how

profiles are created or how MOF elements are manipulated, etc.

Inability to reuse previous knowledge:

One of MDA shortcoming is that it does not reuse previous experience gained from code-centric approaches. For example, models require diagrammatic representations instead of text. However, most students had good experience using text-based editors. Moving towards diagrams would mean to them that they needed to learn new environments, or even more than that, new habits and way of doing things. Some students (programming gurus) would have liked to be able to switch between diagrams and text – A textual representation of UML. Although there is some work in the literature on textual UML, it has yet to find its place in existing tools.

The usage of subject-matter partitioning instead of reusing functional partitioning is another case where students have to change the way they think about system partitioning, and adapt to the new paradigm instead of reusing what they knew before.

Lack of efficient tools:

The tooling part was not covered during lectures. The students, however, were encouraged to use the Eclipse Modeling Framework (EMF) or Rational Software Architect (RSA). Both tools allow handling models very efficiently. However, they suffer from many limitations, among which the most important is their inability to support OAL as well as their inability to translate OCL (even simple ones) into Java (the target environment).

Lack of a good textbook:

Many students find the textbooks interesting but they wished to see a textbook that contains more practical examples. In addition, there is no single textbook that covers all concepts. MOF, UML metamodels and profiling are concepts missing from the proposed textbooks. In fact, these topics are barely covered in any other UML books.

4. Related Work

The importance of the education of software engineering has long been recognized. With the emergence of model-driven approach in general and the OMG's MDA in particular, new courses are now given in different academic venues. There is however very limited published work that report on the feedback from these experiences regarding the impact of the different concepts on the teaching experience.

In [11], the author reports on the decision on standardizing the usage of UML of the computer science programs at the U.S. Military Academy at West Point. In [12], the author discusses the issues stemming from the teaching of UML-based modeling technologies to software engineering students. However, the common observation is that these papers do not address the overall impact of the MDA on teaching software engineering but are mostly limited to the UML modeling language. With regards to the needs of teaching MDA in terms of tool support, the authors in [13] give an overview on a variety available tools that can be used.

5. Conclusions

The OMG's MDA software development approach is very attractive. The basic idea of MDA is a shift of the development focus from the code to the model. Moreover, the OMG supports the MDA with a suite of standards. These include the Meta Object Facility (MOF), the Unified Modeling Language (UML), the Object Constraint Language (OCL), the XML Metadata Interchange (XMI), etc. These standards introduce a significant body of knowledge that should be integrated in a software engineering course. In addition, the MDA introduces a shift from code-centric to model-centric. Such changes have an impact on software engineering education in general. In this paper we have introduced succinctly the body of knowledge implied by the MDA and its associated standards and discussed the impact of MDA on teaching software engineering in general based on an experience in teaching a graduate course on the topic.

6. References

[1] IEEE Standard Glossary of Software Engineering Terminology IEEE Std 610.12-1990

[2] I. Sommervile. Software Engineering. Addison-Wesley, 2007.

[3] OMG Object Management Group http://www.omg.org

[4] S. J. Mellor, K. Scott, A. Uhl, D. Weise, "MDA Distilled", Addison-Wesley, 2004.

[5] A. Gherbi and F. Khendek, "UML Profile for Real-Time Systems and their Applications", *Journal of Object Technology*, 5(4):149–169, 2006.

[6] B. Selic, "A Systematic Approach to Domain-Specific Language Design Using UML", *In Proc. of 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing,* pp. 2–9, 2007.

[7] OMG Unified Modeling Language: Superstructure. version 2.0 formal/05-07-04, August 2005.

[8] OMG. Meta Object Facility (MOF) Core Specification. OMG Available Specification Version 2.0 formal/06-01-01, January 2006.

[9] OMG. Object Constraint Language. OMG Available Specification Version 2.0 formal/06-05-01, May 2006.

[10] M. Gogolla, F. Büttner, M. Richters, "USE: A UML-based specification environment for validating UML and OCL", *Science of Computer Programming, 69(1-3)*, pp-27-34, 2007.

[11] A. S. Ruocco, "Experiences in Threading UML Throughout a Computer Science Program", *IEEE Transactions on Software Education, 46(2),* 2003.

[12] P. Ciancarini, "On the education of future software engineers", *In Proc. of the International Conference on Software Engineering,* pp. 649-650, 2005.

[13] K. Alfert, J. Pleumann, J. Schröder, "Software Engineering Education Needs Adequate Modeling Tools", *In Proc. of the 17th Conference on Software Engineering Education and Training,* pp. 72-77, 2004.

[14] F. Budinsky, D. Steinberg, E. Merks, R.Ellersick, and T. J. Grose. Eclipse Modeling Framework : a Developer's Guide. The Eclipse Series, Addison-Wesley, 2004.

[15] M. Mohamed, M. Romdhani and K. Ghedira, "MOF-EMF Alignment", *In Proc. of the 3rd International Conference on Autonomic and Autonomous Systems*, 2007.

[16] S. J. Mellor, M. J. Balcer, S. Mellor, M. Balcer,. Executable UML: A Foundation for Model Driven Architecture. Addison-Wesley Professional, 2002.

[17] C. Raistrick, P. Francis, J. Wright, C. Carter, I. Wilkie. Model Driven Architecture with Executable UML. Cambridge University Press, 2004.

[18] OMG Query-View-Transformation Specification: URL: www.omg.org/docs/ptc/05-11-01.pdf.

[19] UML Profile for System Engineering SysML, URL: www.omgsysml.org/