On: 26 February 2013, At: 06:02
Publisher: Routledge
Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: Mortimer House, 37-41 Mortimer Street, London W1T 3JH, UK

Computer Science Education

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ncse20

Experiences in teaching a graduate course on model-driven software development

Bedir Tekinerdogan [a]

[a] Department of Computer Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey

Version of record first published: 29 Nov 2011.

To cite this article: Bedir Tekinerdogan (2011): Experiences in teaching a graduate course on model-driven software development, Computer Science Education, 21:4, 363-387

To link to this article: http://dx.doi.org/10.1080/08993408.2011.630129

Computer Science Education
Vol. 21, No. 4, December 2011, 363–387

Routledge
Taylor & Francis Group

Experiences in teaching a graduate course on model-driven software development

Bedir Tekinerdogan*

Department of Computer Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey

(*Received 5 May 2011; final version received 27 September 2011*)

Model-driven software development (MDSD) aims to support the development and evolution of software intensive systems using the basic concepts of model, metamodel, and model transformation. In parallel with the ongoing academic research, MDSD is more and more applied in industrial practices. After being accepted both by a broad community of researchers and the industry, it is now being introduced in university courses. This article describes the experiences of three years of teaching of the graduate course *Model-Driven Software Development* at Bilkent University in Turkey. The lessons learned can be useful for peer educators who teach or aim to teach a similar course.

Keywords: education; software engineering; model-driven software development; experience; graduate course

Introduction

In traditional non-model-driven software development, the link between the code and higher level design models is not formal but intentional. Required changes are usually addressed manually using the given modeling language. Because of the manual adaptation, the maintenance effort is not optimal and as such sooner or later the design models become inconsistent with the code since changes are, in practice, defined at the code level. One of the key motivations for introducing model-driven software development (MDSD) is the need to reduce the maintenance effort and as such support evolution (Briand, Labiche, & O'Sullivan, 2003; van Deursen, Visser, & Warmer, 2007). MDSD aims to achieve this goal through defining models and metamodels as first class abstractions, and providing automated support using model transformations (Bézivin, 2005). For a given change requirement, the code is not changed manually but automatically generated or regenerated, thereby

*Email: bedir@cs.bilkent.edu.tr

ISSN 0899-3408 print/ISSN 1744-5175 online

http://dx.doi.org/10.1080/08993408.2011.630129
http://www.tandfonline.com

substantially reducing maintenance effort. Further, because of the formal links between the models and the code, the evolution of artefacts in the model-driven development process is synchronized. The link between the code and models is formal. Research on MDSD is progressing to enhance the automated support for coping with changing requirements and as such for providing reuse, portability, interoperability, and maintenance.

Because of the promising benefits for development and evolution, MDSD is more and more applied in industrial projects. With the increasing maturation and a growing consensus on the core MDSD concepts, we can now speak of a separate MDSD community. Separate conferences such as Models (MODELS, 2011), SLE (SLE, 2011), ECMFA (ECMFA, 2011) are organized yearly, large international projects are being funded and special issues of journals have started to publish papers on MDSD. The increasing interest for MDSD also affects software engineering education. The need for separate courses on MDSD is growing and an increasing number of courses on MDSD are actually starting to be taught or are planned to be introduced into the curriculum.

This article describes the experiences in teaching the graduate course *Model-Driven Software Development* (Tekinerdogan, 2011) at Bilkent University in Ankara (Bilkent, 2011). Bilkent University, is the first private, nonprofit university in Turkey and is currently ranked number 112 worldwide and number 1 nationwide in the Times Higher Education World University Rankings (THEWUR, 2011) of the world's top universities for 2010/2011. The department of Computer Engineering offers BS, MS, and PhD degree programs.

In Turkey, the MDSD course has been introduced for the first time in 2008 and has been taught by the author for three years. This article covers the experiences of teaching the course over three years for graduate students. Several important lessons were learned from this course and we think that our experiences can be useful for peer educators who teach or aim to teach a similar course. Our experiences may help guide educators on:

- planning an MDSD course in a semester;
- dealing with an introductory course for which no suitable education material yet exists;
- adopting education forms that were used (presentations, project, demonstrations, and workshop organizations);
- setting up a course project on MDSD;
- defining the evaluation criteria for the course; and
- organizing a workshop within a course.

The remainder of the article is organized as follows. Section 2 provides the short background on MDSD. Section 3 describes the course topics

and the organization. Section 4 describes the course material selection. Section 5 describes the selection of pedagogical techniques that were used during the course. In Section 6, we report on the workshop organization that was part of the course. In Section 7, we present the results of the realization of the course. Finally, in Section 8, we provide our conclusions.

Model-driven software development

In this section, we briefly explain the background for MDSD. The concepts that we describe in this section are later used to support the presentation and selection of the course topics.

Motivation

In MDSD, models are not mere documentation but become "code" that is executable (Martin, 2002; Mellor, Scott, Uhl, & Weise, 2004) and that can be used to generate even more refined models or code. This is in contrast to model-based software development in which models are used as blueprints at the most (Stahl & Voelter, 2006). The language in which models are expressed is defined by *metamodels*. As such, a model is said to be an instance of a metamodel, or a model conforms to a metamodel. A metamodel by itself is also a model and conforms to a *meta-metamodel*, the language for defining metamodels. Given the different levels in which the models reside in model-driven development, models are usually organized in a four-layered architecture (Bezivin, 2005; Favre & NGuyen, 2004; Kleppe, Warmer, & Bast, 2003). The top (M3) level in this model is the meta-metamodel that defines the basic concepts from which specific metamodels are created at the meta (M2) level. Normal user models are regarded as residing at the M1 level, whereas real world concepts reside at level M0.

Metamodeling

As stated before, metamodels define the language for the models. A proper definition of metamodels is important to enable valid and sound models. The application of a systematic, disciplined, quantifiable approach to the development, use, and maintenance of these languages is usually called *software language engineering* (Kleppe, 2009) and this forms also an important part of MDSD.

Figure 1 shows the conceptual model for the relation of models and metamodels in MDSD. The starting point for metamodeling or software language engineering forms the *Domain*. A domain can itself consist of subdomains. A metamodel conforms to meta-metamodel, which usually conforms to itself.

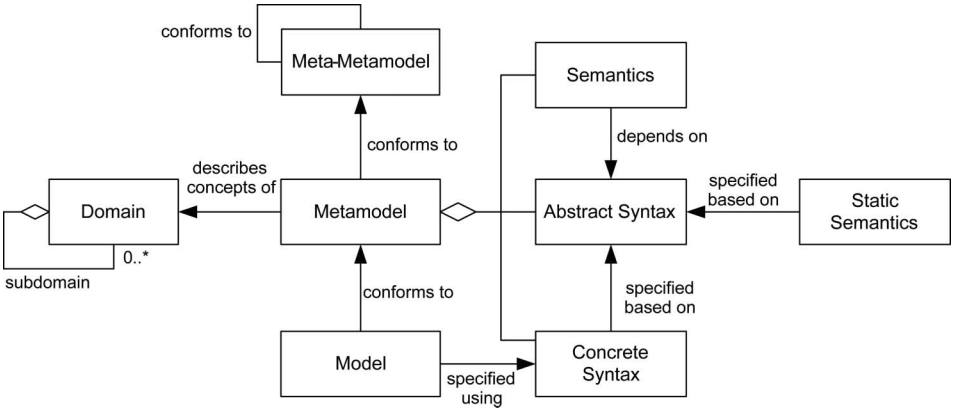

Figure 1. Conceptual model for model-driven software development – metamodeling (Stahl & Volter, 2006).

In both the software language engineering (Kleppe, 2009) and model-driven development domains (Stahl & Voelter, 2006), a metamodel should include the following elements:

- *Abstract Syntax*: describes the vocabulary of concepts provided by the language and how they may be combined to create models. It consists of a definition of the concepts and the relationships that exist between concepts.
- *Concrete Syntax*: defines the syntax, the notation that facilitates the presentation and construction of models or programs in the language. Typically two basic types of concrete syntax are used by languages: textual syntax and visual syntax. A textual syntax enables models to be described in a structured textual form. A visual syntax enables a model to be described in a diagrammatical form.
- *Static Semantics* (*well-formedness rules*): provides definitions of additional constraint rules on abstract syntax that are hard or impossible to express in standard syntactic formalisms of the abstract syntax.
- *Semantics*: provides the description of the meaning of the concepts and relation in the abstract syntax. Semantics can be defined in natural language or using other more formal specification languages.

Model transformations

Besides software language engineering, a second important part of MDSD are model transformations. A model transformation which is shown in Figure 2 takes a source model and transforms it into a target

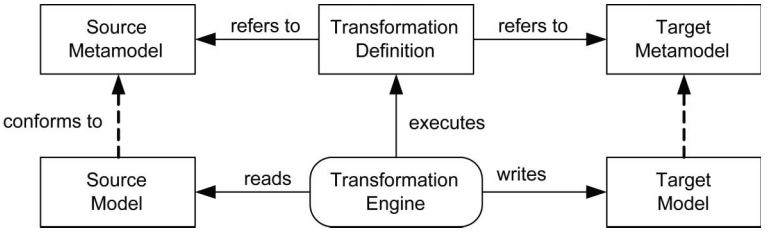

Figure 2. Conceptual model for model-driven software development – model transformations.

model by using predefined transformation definition. Both models conform to their respective metamodels. A transformation is defined with respect to the metamodels. The transformation definition is executed on concrete models by a transformation engine.

In general, a distinction is made between *model-to-model transformation* (*M2M*) and *model-to-text transformation* (*M2T*) (Bezivin, 2005; Kleppe, Warmer, & Bast, 2003). An M2M transforms a model to another model conforming to the same or a different metamodel. In an M2T, a model is directly transformed to text such as code or documentation.

Course topics and organization

The course *Model-Driven Software Development* was defined as a 7.5 ECTS graduate course similar to the other graduate courses in the department. ECTS (ECTS, 2009) is the credit system for higher education used in Europe, whereby one credit corresponds to 25 to 30 h of work.

The course planning consists of the basic activities: *Course Topic Identification, Course Material Selection, Pedagogic Form Selection, Course Realization*, and *Course Evaluation*. Since the course has been given for three years, now, we have followed the plan for three times. In the activity *Course Topic Identification*, the topics were selected based on the conceptual models as defined in the previous section and also as a result of the *Course Material Selection* activity. Figure 3 shows the topic tree for the course. As shown in this figure, we have decided that the course had to be organized around the following four basic topics:

Motivation for MDSD

This topic consists of the subtopics ‘‘conventional software development paradigms’’, ‘‘problems of these model-based development approaches’’, and ‘‘the possible solution approaches’’. A distinction is made between model-based development approaches and model-driven approaches. In the model-*based* approaches, models are basically used as blueprints,

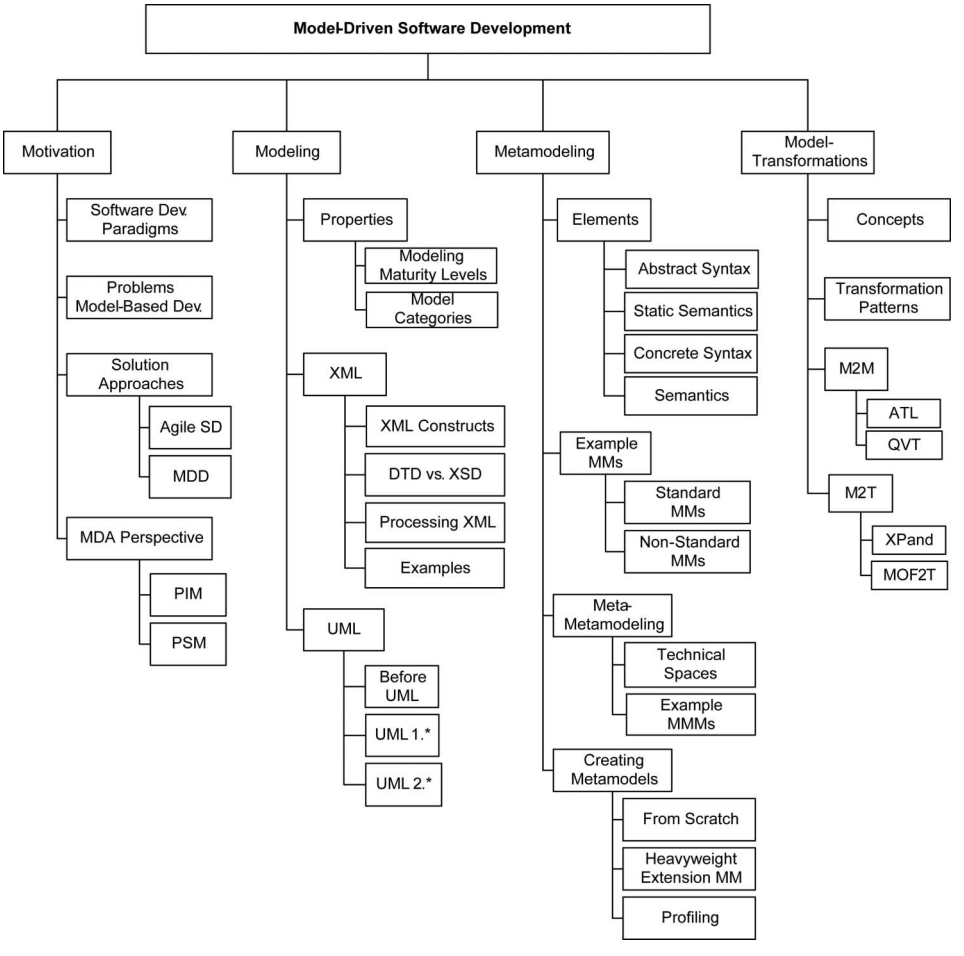

Figure 3. Topic tree for the MDSD course.

whereas in the model-*driven* approaches, models are executable units that can be interpreted by model transformation engines. This topic concerns the idea that the contemporary model-based approaches are not sufficient due to the informal link between code and models as discussed before. Two important tracks are identified that aim to solve this problem: agile software development (Martin, 2002) and MDSD. In agile software development, the focus is put on code and less on models and as such the required changes are defined usually directly in the code. On the other hand, as we have stated before, MDSD aims to define executable models that can be transformed automatically to provide more refined models or code. Whereas agile software development focuses on the principle, the ''code is the documentation'', MDSD adopts the philosophy ''the documentation is the code''.

Another perspective for motivating MDSD is based on the portability and interoperability concerns as defined in the Model-Driven Architecture (MDA) (Kleppe, Warmer, & Bast, 2003) approach. MDA distinguishes between so-called Platform Independent Models (PIM) and Platform Specific Models (PSM). The PIM is usually considered as a stable model, whereas the PSMs and the code are automatically generated.

Modeling in software engineering

Here the focus is on the notion of modeling as a concept, as well as the state-of-the-art and state-of-the-practice of modeling in software engineering. Two important modeling approaches in software engineering are selected, i.e. XML and UML. For both approaches, the history and motivation behind the approaches are given. For XML, the XML constructs, the notion of DTD and XSD, the XML processing techniques, and example XML schemas are selected as useful topics. For UML, the situation before UML, UML 1.*, and UML2.* are considered as important topics.

Metamodeling

This topic includes the sub-topics elements of metamodels, example metamodels, meta-metamodels, and the creation of metamodels. The elements of the metamodel topic include the subtopics: abstract syntax, static semantics, concrete syntax, and semantics. The topic, example metamodels, illustrates both existing standard metamodels and non-standard metamodels. For *Meta-Metamodeling*, the so-called technical spaces (Bézivin, 2006) concept and example meta-metamodels such as MOF, EMOF, CMOF, and ECORE are included (Bézivin, 2005; EMP, 2011; Stahl & Voelter, 2006). The technical space topic includes the examples of OMG/MDA, XML, and EBNF technical spaces.

Model transformations

The Model Transformations topic includes the sub-topics concepts of model transformations, model transformation patterns, model-to-model transformations (M2M), and model-to-text transformations (M2T). For explaining M2M, the tools ATL (Jouault et al., 2008) and QVT (QVT, 2011) are selected. For M2T, the languages XPand (Stahl & Voelter, 2006) and MOFM2T (MOFM2T, 2011) are selected.

Table 1 shows how the course topics were distributed throughout the 15 weeks of the semester. So far we have discussed the topic identification activity. In the following sections, we will elaborate on the subsequent four steps.

Table 1. Course content.

Part	Week	Course topic	Sub-topics
Motivation	1	General overview of course motivation for MDSD	- General guidelines for the course. - A first general motivation on MDSD
Modeling	2	Modeling–UML2	- History of UML 0.* to UML 2.* - Current UML 2.* architecture - UML 2.* models and diagrams
	3	Modeling–XML	- Motivation for markup languages and the particular role of XML. - DTD and XSD
Metamodeling and software language engineering	4	Metamodeling software language engineering	- Metamodeling concepts - Abstract syntax
	5	Metamodeling – static semantics	- Static semantics using OCL - Concrete syntax - Example metamodels
	6	Metamodeling	- Semantics of metamodels - Quality of metamodels - Tools for defining metamodels
	7	Metametamodeling project description	- Meta-metamodeling - Example meta-metamodels (MOF, EMOF, CMOF, ECORE) - Notion of Technical Space
	8	Creating metamodels	- Creating metamodels from scratch using MOF - UML profiles - Example UML profiles - EBNF grammars - Grammars vs. metamodeling
	9	Mid-term exam consultancy for projects	- Open book, open slides, laptop use
	10	Student project presentations Software language engineering	- Presentation of each student team - Demo of each case
Model transformations	11	Spring recess	
	12	Model-driven architecture (MDA) Model transformation concepts	- Motivation for MDA - Transformation patterns in MDA
	13	Model-to-model transformations	- Basic concepts of M2M - M2M using ATL - M2M using QVT

(*continued*)

Table 1. (*Continued*).

Part	Week	Course topic	Sub-topics
	14	Model-to-text transformations	- Basic concepts of M2T - M2T using XPand - M2T using MOF M2T
	15	Student project presentations Model transformations	- Presentation of each student team - Demo of each case
	16	Final exam	

Course material selection

In this section, we will describe the course material for the MDSD course. This includes the selection and evaluation of the textbook, additional research papers, and the tools to be used in the course.

Textbook evaluation

Conventional courses on software engineering usually have no difficulty finding the right textbooks, which include the important topics and suitable exercises. However, one of the problems with the introduction of relatively new topics into the curriculum is that suitable course material is hard to find. At the time of introducing the course, we could identify several textbooks on MDSD but the choice was limited. To make the most effective use of a textbook, we had to decide which textbooks were appropriate for the course needs. Likewise, the first step in evaluating the selected textbook was to identify the *learning goals* with which the textbooks should be aligned. The learning goals indicate what all students should know and be able to do. The primary learning goals were based on the conceptual models as defined in Figures 1 and 2 in Section 2. The learning goals are shown in Figure 4. All these learning goals relate to the MDSD domain, however; the learning goal ''write a report and workshop paper on an MDSD topic'' can be considered as a more general goal. First, we hesitated to present this teaching goal as a learning goal to the students. However, since the technical writing of a workshop paper for the graduate students is an important requirement from the department, we kept this in the list of learning goals to underline its importance (next to the other more domain-specific learning goals).

The judgment on whether the selected textbooks actually address these learning goals was based on both *content analysis* and *instructional analysis* (Kulm, Roseman, & Treistman, 1999). In the context of content analysis, we adopted the distinction between ''topic match'' and ''content match''. ''Topic match'' refers to the case in which the topic is slightly

- understand the notion of model-driven development vs. model-based development
- understand the motivation for model-driven software development
- know the concept of m etamodeling and the sub -concepts abstract syntax, static semantics, concrete syntax and semantics
- be able to develop and analyze metamodels
- know the concept of meta-metamodeling and the notion of technical spaces
- compare the solution approaches of the different technical spaces
- understand model transformation concepts and the key motivation for model transformations
- be able to implement model-to-model transformation definition in selected languages
- be able to implement model-to-text transformation definition in selected languages
- write a report and workshop paper on an MDSD topic
- get insight in the current set of tools in the MDSD community

Figure 4. Identified learning goals for the model-driven software development course.

addressed but not discussed in detail. ''Content match'' refers to the case in which the topic of the learning goal is addressed in detail. For the MDSD content profile, the coverage of each topic in the selected learning goal was rated on a 0–5 scale (no coverage to full coverage).

The instructional analysis of the textbooks evaluated the quality of instructional support for both the students and the instructor. The score for each instructional category was rated on a scale of 0–5 (*high potential for learning to take place* to no *instructional activity present*).

As it can be observed from our evaluation in Table 2, some of the textbooks got similar ratings. Actually, none of the textbooks that were available and that we selected were considered as being suitable and comprehensive enough. We required a book that covered the state-of-the art on MDSD techniques, and was not too specific in dealing with, for example, a single metamodeling or model-transformation language. Despite the lack of a textbook that covered all the learning goals, we decided to select a textbook that got the highest rating and that was somehow also based on existing tool support. For this reason, we chose

Table 2. List of selected textbooks and the result of content analysis (0: no coverage to 5: full coverage).

	Textbook	Motivation	Metamodeling	Model transformation
1.	M. Fowler, Domain-Specific Languages, Addison-Wesley, 2008	4	4	1
2.	D.S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, Wiley, 2003	5	3	2
3.	J. Greenfield, Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Wiley, 2004	4	3	3
4.	R.C. Gronback, Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit, Addison-Wesley Professional, 2009	2	2	2
5.	A. Kleppe, J. Warmer, W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley, 2003	3	3	3
6.	A. Kleppe, Software Language Engineering: Creating Domain-Specific Languages Using Metamodels, Addison-Wesley Professional, 2008	2	5	2
7.	S. Kelly, J.K. Tolvannen, Domain-Specific Modeling: Enabling Full Code Generation, Wiley, 2008	3	4	4
8.	Stephen J. Mellor, Marc J. Balcer, Executable UML: A Foundation for Model-Driven Architecture, 2002.	3	3	2
9.	S.J. Mellor, K. Scott, A. Uhl, Dirk Weise, MDA Distilled, 2004.	4	2	2
10.	T. Stahl and M. Voelter, Model-Driven Software Development: Technology, Engineering, Management, May 2006	3	5	4

the textbook of Stahl and Voelter (2006). Please note that the evaluation is also subjective both with respect to the learning goals that we have defined and our ratings. Each instructor/reviewer might come up with a slightly different evaluation. Also at the time of writing, several new textbooks appeared to have been published (in 2010 and 2011). What is important here is that the selection in each year is based on a systematic evaluation process.

Paper selection

To cover the missing topics and increase awareness on research in MDSD, we also added a list of research papers. These are shown in Table 3. We did not apply here a systematic evaluation process but the selection was based on our own background and the quality of the papers. All the papers were accessible to our students through the subscriptions of Bilkent University.

Tool selection

One of the important learning goals of the course was to explore the current state-of-the-art tools in the MDSD context. Knowledge about tools is particularly important for MDSD since it primarily focuses on automation of the development process. Tool support was needed for three different cases: (1) modeling in XML and UML, (2) metamodeling, and (3) model transformations. For the modeling part in XML and UML, students were allowed to use any tool. To support the modeling of XML and its processing, the students were directed to http://www.w3schools.com/ to model in XML. For the selection of the tool support for metamodeling and model transformations we adopted two important criteria. First, the adopted tools in the course should be

Table 3. Selected papers for the MDSD course in addition to the textbook.

General MDSD/introduction to MDSD

1. D.C. Schmidt. "Model-Driven Engineering". IEEE Computer, 39(2), February 2006.
2. E. Seidewitz. What Models Mean. IEEE Software 20(5), 26–32, 2003.
3. P.A. Muller, F. Fondement, B. Baudry. Modeling Modeling. MoDELS Conf., 2–16, 2009.
4. J. Bézivin. On the Unification Power of Models, 2005.
5. D.S. Frankel. Domain-Specific Modeling and Model Driven Architecture, 2004.

Modeling

6. UML 2. Specification, www.omg.org/spec/UML/2.0/.
7. XML Specification, www.w3.org/XML/.

Metamodeling

8. C. Atkinson and T. Kühne. Profiles in a Strict Metamodeling Framework, Science of Computer Programming, Vol. 44, Issue 1, July 2002.
9. A. van Deursen, P. Klint, J. Visser. Domain-Specific Languages: An Annotated Bibliography, 2000.

Meta-metamodeling and Technical Spaces

10. J. Bezivin and I. Kurtev. Model-based Technology Integration with the Technical Space Concept, 2005.

Model Transformations

11. K. Czarnecki & S. Helsen. Feature-based survey of model transformation approaches, IBM Systems Journal, Vol 45, No 3, 2006.
12. F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev. ATL: A model transformation tool, Science of Computer Programming 72, 31–39, 2008.

open source and free. Second, the adopted tools should be reliable and reflect the topics of the course. In alignment with these criteria we adopted the MDSD toolset as defined in the Eclipse Modeling Project (EMP, 2011). This project focuses on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modeling frameworks, tooling, and implementation of standards. The project provides tools for abstract syntax development, concrete syntax development, static semantics, M2M, and M2T. The students were free to choose any language defined in this Eclipse Modeling Project.

Pedagogic form selection

In this section, we describe the adopted pedagogic forms for the MDSD course. This includes Powerpoint presentations, in-class demonstrations, homework, project, and the decisions on assignment and grading.

Powerpoint presentations

The course was conducted two days a week, including one-hour and two-hour lecture sessions. Each lecture was presented by using Powerpoint slides. The Powerpoint slides were prepared based on the guidelines for multi-media presentations (Joyce, Weil, & Calhoun, 2003) and they were effectively used to explain the topics. During the course, active discussions were encouraged as much as possible. In general, the students appreciated the use of electronic presentations, especially when showing the example metamodels. The presented course slides were sent by e-mail to the students.

In-class demonstrations

The MDSD course is an advanced course in software development and requires a considerably conceptual effort from students in order to understand both the problems and the proposed solutions. To clarify the topics and to make it more concrete, we focused on presenting as many examples as possible. In addition to the examples presented, we arranged demonstrations of XML modeling, XSD and DTD modeling, Metamodeling using XText, Metamodeling using ECore, model transformations using ATL, model transformations using QVT, and M2T using XPand. All of these tools were defined in the Eclipse Modeling Project. The demonstrations were shown after the discussion on the theory of the corresponding MDSD topic was completed, and were accompanied with separate presentations explaining the cases of the demonstrations. The demonstrations were kept short (around 15 min) and were mainly used to

support and direct the students in the selection and usage of the tools. In addition to the plenary demonstrations, the students were also given the opportunity to redo the demonstration at home. In our experience, these demonstrations have not only been of great help for an improved understanding of the presented topics but also improved the enthusiasm for the course topics.

Homework

The students were given homework on the topics that were discussed in the corresponding week. The homework included reading and evaluation of selected papers, small assignments on MDSD, and the usage of MDSD tools. Each homework was planned so that it could be realized in 5–10 h. The assignments were briefly discussed in the lecture and, if needed, explained in more detail afterwards.

Project

The course included one project in which the basic MDSD concepts and techniques were practiced. The project could start in week eight after sufficient information about the MDSD topics was discussed, and the students got some experience with the MDSD tools during the homework assignments. The project's aim was to clarify the concepts that had been taught during the classes using teamwork. For this, the students formed groups of three, working together on a case that they had selected themselves. It was expected that each group had to work on the project every week for about 3–5 h. To control the progress and provide intermediate feedback, separate consultancy meetings (with instructor) of half-an-hour per group were planned.

Table 4 shows the number of students, the groups, and the selected cases for the three academic years. Most of these cases were existing cases or research topics of the group in which the students had to finalize their MSc assignments. As such the projects had also an indirect positive effect since the students directly applied the concepts in the real research setting. Another important aim of the project was to define a proper problem statement for the selected case from the perspective of MDSD. In this way, the students were directed towards exploring different motivations for MDSD and this further enhanced the understanding of the practical value of MDSD. As such we aimed to support not only the understanding of common domain-specific issues of MDSD but also the individual problem-solving skills as described in, for example, Deek, Turoff, and McHugh (1999).

The concrete aims of the projects were all different but we could also observe problem statement categories based on software language

Table 4. Selected projects by student groups in the MDSD course over three years.

2008–2009 (20 students)	2009–2010 (15 students)	2010–2011 (17 students)
MD-CHISIO: A model driven approach for graph visualization	A model-driven approach for automating architecture documentation process	Model-driven analysis of wireless networks
Model-driven development for procedural building modeling	Model-driven development approach for tactical battlefield management systems	A model-driven software architecture for mind-mapping
Model-driven flight deck display modeling	A model-driven approach to role playing games	Model driven development of car navigation system
Model-driven approach for emergency resource management	A model-driven approach for information retrieval systems	Model-driven integration of ARINC 653 compatible real time avionics system
Automatic generation of board games using MDSD	A model-driven approach to the planning domain	Model-driven approach for organizational structures
Model-driven development of command and control of unmanned systems		Model-driven approach for kinematic human body model generation
Model-driven development of discrete event simulation modeling		
Model-driven integrated circuit modeling and analysis		

engineering and model transformations. For the software language engineering part, the basic motivation was to initially provide a metamodel (DSL) for the selected domain to increase the understandability and/or to use it for model validation. In the final part of the project, the problem statement had to be sharpened including the motivation for M2M and M2T. The various aims from the model-transformation perspective were to enhance portability, supporting interoperability, and enhancing productivity. Detailed information about the projects can be found on the workshop web sites (TModels, 2009; TModels, 2010).

After defining the problem statement the students had to do a domain analysis on the selected case. This was needed to support the idea of domain-driven design and domain specific language engineering. Based on the domain model concepts derived from the domain analysis process, the students had to provide a language engineering solution using three different technical spaces, i.e. XML-based, grammar-based, and OMG metamodeling-based.

For the model transformation part both M2M and M2T were required. As stated in the previous sections, we chose to adopt the tools

as defined in the Eclipse Modeling Project. Several different alternative tools are provided in the Eclipse Modeling Project. We deliberately did not choose a particular tool but let the students explore the possible and suitable tools for their projects. In this way, we stimulated the analysis of multiple tools and gaining insight in the current state-of-the-art of MDSD tools. The deliverables of the project included a report (about 30–40 pages). The general outline of the report is shown in Figure 5.

We can state that the students were, in general, very positive about the project since it provided them the opportunity to experience a real MDSD process.

Assignment and grading

The final course grade was based on a set of written exams/assignments, which were used to evaluate students understanding of the course material. There were two homework assignments, two examinations, one mid-term, and one final exam. The overall grading scheme is given in Table 5.

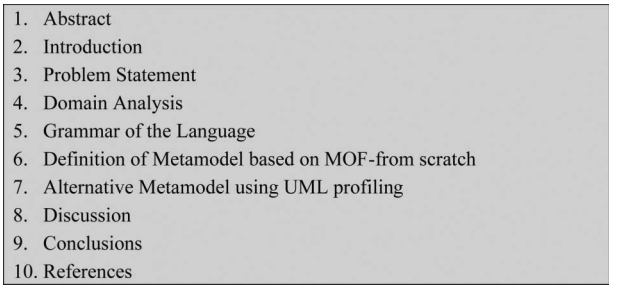

Figure 5. Outline of the report that was required as a result of the project deliverables.

Table 5. Examinations and grading.

Task	Description	Weight
Homework	Individual assignments	10%
Mid-term project (software language eng.)	Report (50%) Presentation (15%) Code (35%)	20%
Final project (model transformations)	Report/workshop paper (50%) (Workshop) Presentation (15%) Code (35%)	20%
Mid-term exam	Open book 120 min	25%
Final exam	Open book 120 min	25%

The homework assignments helped to assess the students' knowledge of particular topics and stimulated the students to explore the required MDSD tools on time. The mid-term and final exams were more comprehensive and focused on a broader understanding and application of the topics. The examinations were open book and open slides, and even open notebook, i.e. the students could use their notebooks during the exam. This is because our focus was not on memorization of the topics but rather on the ability to apply what was learned during the course. To prevent plagiarism (using the internet), the open notebook exams were held in a classroom in which no wireless internet access was possible.

The homework and examinations did not only help to evaluate and grade the students but also ensured indirectly that the course topics were studied on time. This was important considering the relative advanced nature of the course. Besides the examinations, which were evaluated individually, the students also got a group grade for the project and the related workshop (see next section). The project was graded based on the report (workshop paper), presentation, and code. Note that the project counts for a significant part of the grade, which emphasizes its importance.

Workshop organization

During the project, complex cases were selected from industry and ongoing projects at the university and these were analyzed and re-engineered as MDSD designs. Each student team implemented a metamodel, an M2M, and an M2T. The deliverables were provided as a report. To make these valuable practices public for a broad audience we decided to organize the First and Second Turkish MDSD (TModels) Workshop (TModels, 2009; TModels, 2010) for which we invited participants from industry and other universities. This was a unique experience since most of the students had never participated in a workshop before. In addition, since the topic was introduced for the first time in Turkey, the audience would consist of participants who were new to this paradigm. Consequently, we thought that organizing such a workshop would be beneficial for both our students and external participants. In particular, with the organization of the workshop we had the following goals in mind:

- Trigger academic and industrial activities in the MDSD domain in Turkey.
- Show real-world example cases using different MDSD approaches to highlight the current state-of-the-art in the MDSD community.
- Show the lessons learned from MDSD.

- Share our ideas with respect to MDSD education on MDSD in Turkey.
- Trigger new research topics on MDSD.

For the First TModels workshop (TModels, 2009), we had around 45 registered participants, the Second TModels workshop (TModels, 2010) had around 30 registered participants. The majority of the participants consisted of our own students, participants from the Turkish software industry, and a few participants from other Turkish universities.

During the workshop, the project results have been presented as workshop papers. Because most students in the MDSD course wrote a workshop paper for the first time, they also received a short course on how to write workshop papers and in addition got extensive feedback on their papers by the instructor. The papers not only present experiences in an MDSD project but also highlight the obstacles in MDSD and provide some triggers for new research directions. As such, we consider the papers in the published workshop proceedings as both useful for novice MDSD developers and researchers.

Course evaluation

In this section, we report on the evaluation of the course by reflecting on the impact of the course on the students, and the evaluation of the course by the students.

Table 6 shows the overall grading results (over 100) of the students in each year for the particular course evaluation items. The first year homework was not graded and as such the results here are missing. In general, the students were highly motivated and also scored well for the course. Based on our experience we can state that defining and grading homework had an important impact on the overall motivation and understanding of both the concepts and the MDSD tools. The homework assignments were done individually and usually the grades for these assignments were quite high.

Table 6. Averages of the grades for the course evaluation items.

TASK	2008–2009	2009–2010	2010–2011	Avg. grade (over 100)
Homework	–	91	80	85
Mid-term project	72	79	80	77
Final project	83	74	86	83
Mid-term exam	78	63	69	70
Final exam	77	72	66	72
GPA	3.36	3.0	3.45	3.27

For the project report, the students had to also write a section in which they had to reflect on the lessons learned. Several items were recurring themes in the observations of students:

Identifying cases and problem statement

For the project, the students had to define their own cases and the related problem statement. Defining the cases was usually not a problem but very often the students needed additional help to define a proper problem statement within the context of the selected case. Defining a concrete problem statement required a sufficient understanding of MDSD topics and this appeared difficult in particular in the initial stages of the project. The division of the project into two parts including software language engineering for mid-term project and model transformations for the final project helped the students to reflect on the case and identify ''emerging'' and usually real problem statements. Students in the MDSD course that was given in the last two years had fewer problems in defining problem statements since they could access the project reports of the previous year(s) on the course website.

Immature tools

Most students needed quite a lot of time to analyze the different tools, install a selected tool, and use the tools. A commonly cited issue was the immaturity of the tools which was caused due to lack of proper documentation, lack of examples, bugs in the tools, or the lack for integration with other tools. In the first year of the course the immaturity of the tools was a bigger problem than the last year. We expect that, thanks to the developments in MDSD, the concerns about tool support will be somehow reduced in the coming years.

Difficulty of metamodeling

It appeared that the process of metamodeling was considered more difficult than expected after the lectures. During the lectures, many different examples of metamodels were given but once the students had to define their own metamodels they reported that defining the right metamodel abstractions was not that easy. There are different processes for modeling in the literature which most students had somehow applied before, but the process for metamodeling was quite new and required a different level of thinking. Despite the lack of a systematic metamodeling process and the related difficulties, most students noted that they still appreciated this new insight because it enabled a new perspective on

modeling in general. Regarding the different metamodeling approaches (metamodeling from scratch vs. UML profiling) it appeared that different students had different opinions about these two distinct appoaches. Some groups indicated that metamodeling from scratch was easier while others preferred profiling (UML) to define metamodels. A reflection on the projects showed that the difference was also caused due to the different selected cases. Important criteria here appeared to be the distance of the metamodel to the UML metamodel. Metamodels closer to the UML metamodel could be better and easily modeled using profiling, while other metamodels required the definition of metamodel from scratch.

Effective usage of model transformations

Once the metamodels were defined, the students usually had no problems in defining M2M and M2T. Although they had several options, in general, the students preferred using ATL (Jouault et al., 2008) for M2M and XPand (Stahl & Voelter, 2006) for M2T. The main (reported) reason for this was the more extensive documentation and the availability of examples in both tools. Many students indicated that the learning curve for the several tools was a bit high but after some experience with the tools they indicated that model transformations had a sufficient high pay-off since they could solve ''real'' problems related to such as reuse, portability, and interoperability.

Overall appreciation of technical spaces

During the lectures, we have adopted three different technical spaces for modeling (XML-based, grammar-based, and OMG metamodeling). The overall notion of technical spaces and the analysis of these seemed to be better understood and appreciated after the application of these in the project. Besides metamodeling, the students indicated that the notion of technical spaces broadened their vision of modeling in general. In many projects, the students themselves define the bridging between solutions in technical spaces. The motivation for this bridging became more understood during the realization of the project.

Table 7 shows the course evaluation by the students over three years. The evaluation criteria are defined by the department and used for all graduate and under-graduate courses. As it can be observed from the table, the students have indicated that they appreciated both the presented course content and the pedagogic forms. The evaluations can be considered as outstanding results considering the average in the computer engineering department of Bilkent University.

Table 7. Course evaluation by students over three years.

Evaluation criteria	2008–2009 Avg. evaluation	2009–2010 Avg. evaluation	2010–2011 Avg. evaluation
The instructor clearly stated course objectives and expectations from students.	4.89	4.91	4.67
The instructor stimulated interest in the subject.	4.89	4.55	4.44
The instructor was able to promote effective student participation in class	4.89	4.82	4.56
The instructor helped develop analytical, scientific, critical, creative, and independent thinking abilities in students.	4.89	4.91	4.67
Rate the instructor's overall teaching effectiveness in this course.	4.89	4.91	4.56
I learned a lot in this course.	4.56	4.91	4.56
The exams, assignments, and projects required analytical, scientific, critical, and creative thinking.	4.78	4.82	4.33

Related work

The Educators' Symposium at MODELS is being organized for several years now (Bezivin et al., 2010; EduSymp, 2011; Seidl & Clarke, 2010; Śmiałek, 2008). The workshop focuses on discussing the education of model-driven techniques to software engineers at universities and software industries. The motivation for organizing the workshop stems from the observation that while most computer science curricula include some education in modeling technologies, a more holistic approach of modeling in software engineering is rarely captured. We think that our experiences fit perfectly with the goals of the EduSymp workshops.

The panel discussion held during the Educators' Symposium at MODELS'2009 (Bezivin et al., 2010) indicated important concerns when teaching MDSD. A number of topics relevant to teaching modeling such as Notation, Semantics, Programming, Tooling, Suitability, and Metamodeling were discussed. An interesting observation that was made during the panel discussion is that MDSD has been initially an industry-driven approach, which was later on tackled and reshaped by researchers. There seems to be now an agreement that MDSD has a reasonable consistent theory with basic tooling support and empirical evidence for a significant return on investment for MDSD projects. What is missing in the MDSD context is education. We think that our effort in designing and

teaching an MDSD course is in alignment with this vision for dissemination of the MDSD topics.

In the EduSymp workshop in 2010 (Seidl & Clarke, 2010), the general consensus among the participants seemed to be that software modeling education should not be limited to teaching theory only but hands-on experience of the students is necessary. This is because students will be more motivated if they can use some tools, and the taught concepts will be better understood implementing practical applications. Our experiences underline these observations. We could indeed observe that the students got more motivated when starting with the projects and also mastered the MDSD concepts after realizing the project and implementing metamodels and the model transformations.

Several MDSD-related courses have been started elsewhere in the world. Gokhale and Gray (2005) describe their experiences in developing and teaching MDSD courses based on their research activities. The authors report on two special topic courses that were related to the theory and practice in MDSD. The courses were basically research-driven and open to both graduate and under-graduate students. As a consequence, in the adopted format of the course less emphasis was placed on the didactic aspects of teaching, while more focus was put on hands-on experiences, projects, and discussions. The course organization had also a practical goal; the enhancement and use of the CoSMIC and C-SAW tools. The course that we have organized was only open for graduate students, and due to the workshop organization within the course there was also a strong research focus. The main reason for organizing our course was educational and not targeted to support our research activities. As such our students had to analyze a broad set of MDSD tools and could select their own MDSD tool. Gokhale and Gray (2005) report on the similar lessons learned from our experience, including the relatively high learning curve involved in mastering the tools and the difficulties in comprehending the concepts of metamodeling.

Brosch et al. (2009) report on the advanced modeling course called Model Engineering that is started at the Business Informatics Group (BIG) of the Vienna University of Technology. The course is obligatory for business informatics master students and optional for master students of computer science. The course seems to include similar topics as in our MDSD course, including metamodeling, model transformation, code generation, and concrete syntax specifications. Further, the authors also state that besides a solid knowledge on MDSD, hands-on experience concerning the state-of-the-art techniques seemed to be important. The immature tool support in MDSD resulted in several problems but in general the students appreciated to work with the latest MDSD tools. Our experiences from our MDSD courses also confirm these observations.

Clarke et al. (2009) describe their experiences in how MDSD has been integrated into the software design course at Florida International University. During the course, the students had to work on a project for modeling and realizing communication applications. The authors of the paper further present the results of a survey that was provided to the students to obtain empirical evidence on how MDSD helps with understanding modeling concepts and the current state of the tools to support MDSD. The results of the survey showed that MDSD has a positive impact on helping students to better understand how models can be used during software design. Our experiences are in alignment with these findings. In particular, the notion of metamodeling and the discussion of different technical spaces increased the understanding and vision on modeling.

Conclusion

In this article, we have described our experiences of three-year teaching of the graduate course *Model-Driven Software Development*, at Bilkent University in Ankara, Turkey. To support the organization and planning of the course, the important topics were represented in a topic tree and these were later on distributed over 15 weeks in a semester. The selection of topics was done in parallel with the evaluation of the textbooks and the relevant scientific papers in the MDSD domain. The textbooks were evaluated based on the learning goals that we have defined for the course. For this we adopted both content analysis and instruction analysis. Since no textbook covered all the learning goals for the course, we selected the most feasible textbook and complemented this with a selection of important MDSD research papers in the course material. In our experience the adoption of research papers in the course helped to improve the scientific reading and writing skills, which is essential for graduate students.

Knowledge of tools and the ability to implement metamodels and model transformations was an important learning goal. During the projects, it appeared that different student groups used different Eclipse Modeling tools. As such, all students got an overview of practically all the Eclipse Modeling tools.

The organization of a workshop within a course was really a unique experience and helped increase not only the scientific reading, writing, and presentation skills of the students but also supported the popularization of the MDSD concepts in Turkey.

In the future, we plan to give the same course in a similar way. We expect that more and better textbooks will be published in the near future and the corresponding tools will be even more mature. This will facilitate the realization of the course further and the opportunity and need to introduce MDSD courses in the CS curriculum (CSC, 2008) will increase.

We hope that the course organization and the planning will be of help for our peer educators and colleagues.

Acknowledgments

I would like to thank all the Bilkent University, CS587 (MDSD course) students of the academic years 2008–2009, 2009–2010, and 2010–2011, for their motivation and their hard work during this course. Further, I would like to thank the anonymous reviewers for their valuable comments for the earlier versions of this article.

References

Bézivin, J. (2005). On the unification power of models. *Software and System Modeling*, *4*, 171–188.

Bézivin, J. (2006). Model driven engineering: An emerging technical space. In R. Lämmel, J. Saraiva, & J. Visser (Eds.), *Generative and transformational techniques in software engineering* (Vol. 4143) (pp. 36–64). Berlin, Heidelberg: Springer.

Bézivin, J., France, R., Gogolla, M., Haugen, O., Taentzer, G., & Varro, D. (2010). Teaching modeling: Why, when, what? In S. Ghosh (Ed.), *Models in software engineering* (Vol. 6002) (pp. 55–62). Berlin, Heidelberg: Springer.

Bilkent. (2011). *Bilkent University*. Retrieved from http://www.bilkent.edu.tr/bilkent-tr/index.html

Briand, L.C., Labiche, Y., & O'Sullivan, L. (2003). Impact analysis and change management of UML models. In *Proceedings of the International Conference on Software Maintenance* (*ICSM'03*), Amsterdam, Netherlands.

Brosch, P., Kappel, G., Seidl, M., & Wimmer, M. (2009). Teaching model engineering in the large. In *Educators' Symposium @ Models 2009*, Denver, USA.

Clarke, P.J., Wu, Y., Allen, A.A., & King, T.M. (2009). Experiences of teaching model-driven engineering in a software design course. In *Proceedings of the Models 2009 Educators' Symposium*, Denver, USA.

CSC. (2008). *ACM and IEEE computer science curriculum 2008: An interim revision of CS 2001* (Report from the Interim Review Task Force, The association for computing machinery and the IEEE computer society). New York: ACM/IEEE.

Deek, F., Turoff, M., & McHugh, J. (1999)A common model for problem solving and program development. *Journal of the IEEE Transactions on Education*, *42*, 331–336.

van Deursen, A., Visser, E., & Warmer, J. (2007). Model-driven software evolution: A research agenda. In Dalila Tamzalit (Eds.), *Proceedings 1st International Workshop on Model-Driven Software Evolution* (pp. 41–49). Nantes: University of Nantes.

ECMFA. (2011). *European conference on modelling foundations and applications*. Retrieved from http://www.ecmfa–2011.org/

ECTS. (2009). *ECTS users' guide*. Retrieved from http://ec.europa.eu/education/lifelong-learning-policy/doc/ects/guide_en.pdf

EduSymp. (2011). Software modeling in education. In *7th Educators' Symposium @ MODELS 2011*. Retrieved from http://edusymp.big.tuwien.ac.at/

EMP. (2011). *Eclipse modeling project*. Retrieved from http://www.eclipse.org/modeling/

Favre, J.M. & NGuyen, T. (2004). Towards a megamodel to model software evolution through transformations. In *SETRA Workshop* (pp. 59–74) [ENCTS]. Amsterdam: Elsevier.

Gokhale, A.S. & Gray, J. (2005). Advancing model driven development education via collaborative research. In *Proceedings of the Educators Symposium at the 8th International Conference, MoDELS 2005*. Montego Bay, Jamaica.

Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, *72*, 31–39.

Joyce, B., Weil, M., & Calhoun, E. (2003). *Models of teaching* (7th ed.). Englewood Cliffs, NJ: Prentice-Hall.

Kleppe, A. (2009). *Software language engineering: Creating domain-specific languages using metamodels*. Boston: Addison-Wesley Longman Publishing.
Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA explained: The model driven architecture – Practice and promise*. Boston: Addison-Wesley.
Kulm, G., Roseman, J., & Treistman, M. (1999). A benchmarks-based approach to textbook evaluation. *Science Books & Films*, *35*, 147–153.
Martin, R.C. (2002). *Agile software development, principles, patterns, and practices*. Upper Saddle River, NJ: Prentice Hall.
Mellor, S.J., Scott, K., Uhl, A., & Weise, D. (2004). *MDA distilled: Principle of model driven architecture*. Reading, Boston: Addison Wesley.
MODELS. (2011). *ACM/IEEE international conference series on model driven engineering languages and systems model conference*. Retrieved from http://www.modelsconference.org/
MOFM2T. (2011). Object management group. *MOF model to text transformation language (MOFM2T), 1.0 – Specification*. Retrieved from http://www.omg.org/spec/MOFM2T/1.0/
Seidl, M. & Clarke, P.J. (2010). Software modeling in education. In *Proceedings of the 6th Educators' Symposium at MODELS 2010*, Oslo, Norway.
SLE. (2011). *Software language engineering conference series*. Retrieved from http://planet-sl.org/sle2011/
Śmiałek, M. (2008). Promoting software modeling through active education. In *Educators Symposium at the 11th International Conference, MODELS 2008*, Toulouse, France.
Stahl, T. & Voelter, M. (2006). *Model-driven software development*. Boston: Addison-Wesley.
Tekinerdogan. (2011). *CS587-model-driven software development course home page*. Bilkent University, Ankara, Turkey. Retrieved from http://www.cs.bilkent.edu.tr/~bedir/CS587-MDSD/, 2010/2011
THEWUR. (2011). *Times higher education world university rankings*. Retrieved from http://www.timeshighereducation.co.uk/world-university-rankings/
TModels. (2009). *First Turkish workshop on model-driven software development*. Retrieved from http://www.cs.bilkent.edu.tr/Bilsen/TMODELS-2009/
TModels. (2010). *Second Turkish workshop on model-driven software development*. Retrieved from http://www.cs.bilkent.edu.tr/Bilsen/TMODELS-2010/
QVT. (2011). Object management group. *Meta object facility (MOF) 2.0 query/view/transformation (QVT)*. Retrieved from http://www.omg.org/spec/QVT/1.0/