Behavior Model Mapping

Judith Michael and Heinrich C. Mayr

Application Engineering Research Group, Alpen-Adria-Universität, Klagenfurt, Austria

Abstract. The work presented here is part of a comprehensive project that aims at supporting user centered software development from requirements elicitation to program generation. This paper focuses on transforming validated "preconceptual" requirements models into conceptual ones (a UML dialect) which then are input for a program generation engine (OlivaNova). In particular, we discuss a set of rules and their prototypical implementation, that map networks of so-called CooperationTypes (as models of business processes) to state charts. This differs from other studies that mostly deal with transformations or mappings of structure models.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

1 Introduction & Vision

Despite of all methodological work that enhanced the construction of correct software from formal specifications, software development projects still often fail. They may implement a given formal specification (e.g. a UML model) but not meet the needs of the real stakeholders: the users. I.e., what is specified often differs from what is really needed. However, due to their technical nature design specifications cannot be understood and validated by the average user. Consequently, requirements have to be modeled in a way such that the stakeholders can understand and validate them. There are several approaches to such kind of user centered requirements modeling, e.g. [1, 2].

A user validated requirements specification defines the final system, i.e. is just another representation of it. So, system development in reality is a kind of model transformation which can be automated to a large extent. Figure 1 gives an overview of our approach to such a software development / model transformation process: (1) Semantics extraction from natural language requirements specification [3]. (2) Generation of a user centered first cut "conceptual predesign" requirements model using an easy to understand Interlingua like KCMP [4], (3) Transformation of the validated requirements model into a conceptual one (a UML dialect) which again (4) is input for a program generation engine; figure 2 shows tools and methodologies that have been developed to support these steps (1)-(4).

We focus here at step (3). The transformation of structure models has been investigated since long beginning with the transformation of ER-Schemata into Relational Database Schemata. Therefore, we concentrate on the transformation of predesign behavior models into conceptual ones. More concrete: we transform networks of so-

Michael J. and C. Mayr H..
Behavior Model Mapping.
DOI: 10.5220/0003561300030013
In *Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development* (MDA & MDSD-2011), pages 3-13
ISBN: 978-989-8425-59-1
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

called CooperationTypes (as models of business processes) into state charts which are represented by an XML file that serves as input for the OlivaNova Modeler (step (4)).

In order to make our intentions and solutions more transparent, we shortly introduce into Model Driven Architecture (MDA) and Model Driven Software Development (MDSD) in section 2. Section 3 outlines the main modeling concepts of KCPM (Klagenfurt Conceptual Predesign Model), i.e. the "language" we use for requirements modeling. In Section 4 we discuss the most important aspects of behavior model transformation in detail and propose solutions that fit within the given framework. A transformation engine prototype that realizes these solutions is presented in section 5. The paper ends with a short summary and an outlook to what will be investigated next.

Fig. 1. The software development / model transformation process in four steps.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

Fig. 2. Overview from Requirements Engineering to Software Development.

2 MDA and MDSD

The main idea of Model-Driven Architecture (MDA) is to separate the specification of the operation of a system from the details of the way that system uses the capabilities of its platform [5]. MDA consists of three types of models, the Computation Independent Model (CIM), the Platform Independent Model (PIM) and the Platform Specific Model (PSM), which are structured into three basic layers (see figure 3).

The OO-Method Group headed by Oscar Pastor at the Technical University of Valencia has aligned OO Design with the MDA-perspective [6] and built the basis for commercial software that has been developed by CARE Technologies [7]: *OlivaNova Model Execution* (ONME) includes the *OlivaNova Modeler*, which generates applications from conceptual models, and a set of compilers branded *OlivaNova Transformation Engine*. [8]

Creating Application Software using ONME meets the main aspects of Model Driven Software Development (MDSD): ONME employs techniques to generate automatically executable software from formal models [10]. For model creation (i.e. the PIM) the OlivaNova Modeler provides three views: the object, the dynamic and the functional view.

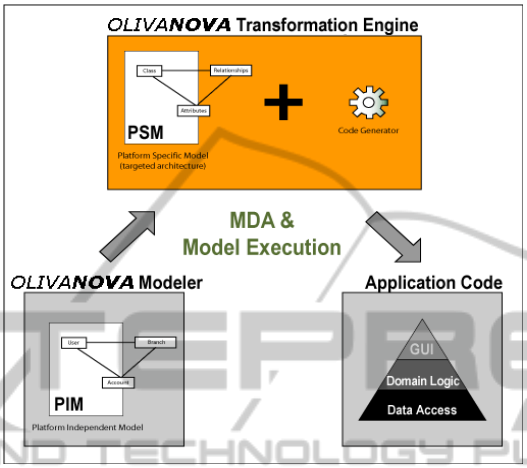

Fig. 3. MDA and Model Execution [9].

ONME extends the PIM by specific platform information (e.g., the database system to use, the target programming language) and thus produces the PSM. Finally ONME generates the application code, composed of scripts for creating a database, the business logic and several user logics for desktop and web applications.

3 KCPM

Though MDSD is based on models it comes with a substantial problem, namely the kind of meta-models/modeling languages used (today mostly UML) for compiling the PIM: The modeling concepts offered by these meta-models and the resulting complexity of real world application models are hard or even not to understand for the main stakeholders, i.e. the end-users. Thus, end-users are not able to process them and to find out if they reflect the factual requirements. This leads to the well-known fact that many software projects fail although delivering software which is correct w.r.t. its specification (the PIM), but which does not fulfill the stakeholder's requirements. Thus a more user-centered approach to modeling is needed.

KCPM (*Klagenfurt Conceptual Predesign Model*) [2] is intended to be such an approach: It comes with few concepts and two kinds of representations, a graphical and a glossary like, i.e. a tabular one, the latter being easy to understand mainly for business people. The main KCPM modeling concepts are: ThingType and ConnectionType for static aspects, OperationType and CooperationType for dynamic aspects; CooperationTypes are combined to business process models.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

Figure 4 shows a cutout of the KCPM meta-schema. The central concept is ThingType for modeling similar concrete or abstract things which exists in the real world (e.g., *book, title, author, car, color*). ThingTypes may be related to each other, ConnectionTypes are abstractions of these relationships. Each ThingType may be involved into one or more ConnectionTypes. (e.g., *A book has a title. An author writes a book*). A Perspective (as part of a ConnectionType) refines the role by which a ThingType is involved into that ConnectionType. An OperationType models an activity which instances of one or more given ThingType can execute. The ThingType then is called the "*executing actor*". Typically there are also one or more ThingTypes which are allowed to request the execution of the operation; they are called "*calling actors*". Resources ("*involved ThingTypes*") are parameters of the activity, i.e. the "things" on which the executing actor does something (read, manipulate, send…; e.g., *The reader loans a book in the library*).

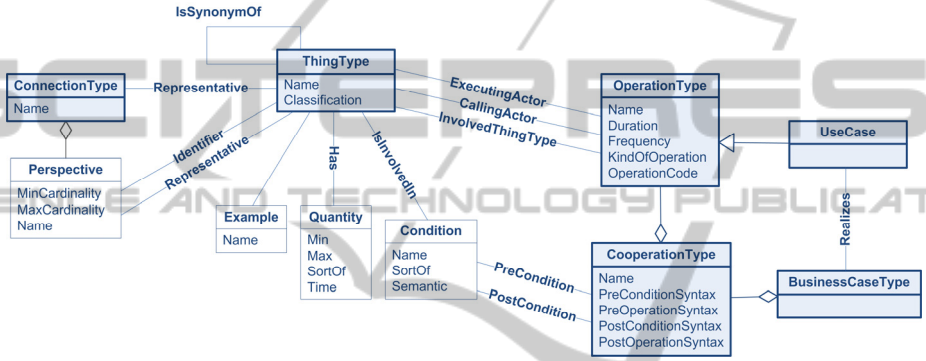

Fig. 4. Meta-Schema of KCPM.

Operations can be executed sequentially or concurrently which is reflected by the concept of CooperationType (see figure 5) that combines OperationTypes and pre- and postconditions [3] by logical expressions.

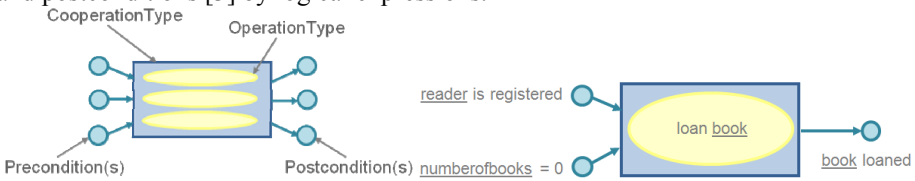

Fig. 5. Graphical notation of a CooperationType and Example.

> *E.g.: When a registered reader hasn't yet loaned a book, (i.e., his number of books loaned is zero) and he loans one, the OperationType loan book is executed, and the reader's number of loaned books is increased (see example figure 5).*

For reasons of simplicity, we disclaim here to discuss the logical expressions (modeled by PreConditionSyntax etc. as Attributes of ConnectionTypes) as well as other KCPM concepts and refer the reader to [11].

4 Behavior Meta-model Mapping

We now proceed to the mapping of KCPM schemata (corresponding to CIM from a MDA perspective) to conceptual ones (see figures 1 and 2), the latter using the meta-model of ONME, i.e. a subset of UML (for detailed information about the ONME Meta-Model see [12]). The result of the mapping is a PIM.

The mapping of structural (static) model elements, i.e. ThingTypes and ConnectionTypes, leads in the case of UML as a target to classes and associations. This is not very challenging and has been discussed and implemented in [13], where a set of mapping rules has been introduced. However, note that the concept of ConnectionType is more general than that of association. It can be mapped to associations if at least two involved ThingTypes are mapped to classes, or it can e.g. mapped to a class attribute, if one involved ThingType is mapped to a class and the other one to a value domain.

OperationTypes are mapped to services (class methods); thus, the ingoing and outgoing parameters of the KCPM OperationTypes are mapped to inbound and outbound parameters of their associated services in the OlivaNova Modeler.

The question now is how to map CooperationTypes. OlivaNova Modeler offers two concepts for behavior modeling: State Transition Diagrams (STD) and Object Interaction Diagrams (OID).

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

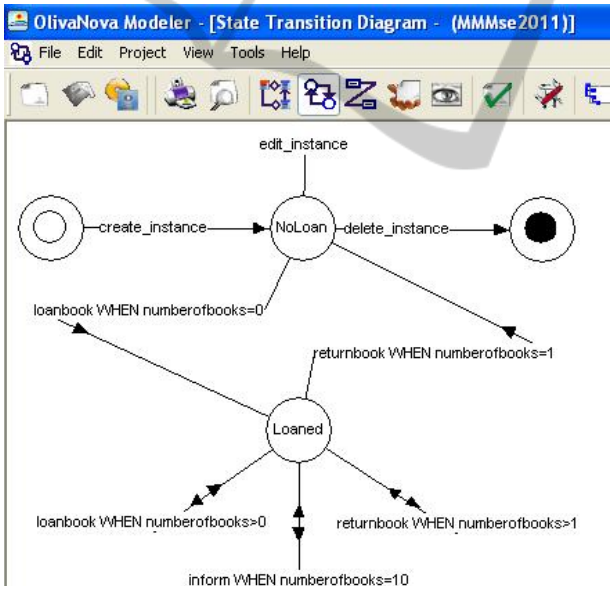

Fig. 6. STD Example.

4.1 State Transition Diagram

The OlivaNova Modeler provides a *default State Transition Diagram* consisting of an initial, a final and an intermediate state for each class. In this default STD all transitions start and end at the intermediate state, except the creation and destruction

transitions. A State Transition Diagram (STD) describes valid object life cycles for the instances of a given class and thus reflects the objects' behavior (see figure 6).

The OlivaNova Modeler differentiates three kinds of states: *initial state*, *simple* or *intermediate*, and *final state*. State transitions have an agent which can execute a defined service. The execution of this service causes the state transition to the next one (e.g.: from *NoLoan* to *Loaned* in figure 6) when a condition (e.g.: number of loaned books equal 0) is fulfilled.

4.2 Object Interaction Diagram

The OID models the inter-object communication (by Triggers and Global Services). There exists only one Object Interaction Diagram per system – as opposed to State Transition Diagrams that are to be defined for each class.

Since global services can be treated similar to local ones (class services) we concentrate here on the OID concept of triggers: A trigger forces a service to be executed automatically depending on specified conditions. *E.g., figure 7 shows a trigger where an "inform" service is called because a reader tries to loan too many books.*

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

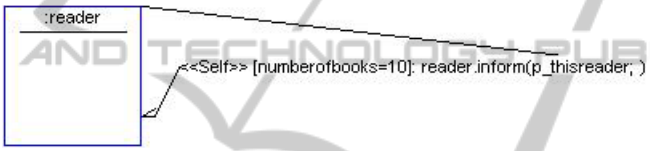

Fig. 7. Example of an Object Interaction Diagram.

4.3 Steps of the Behavior Model Mapping Process

In order to define the mapping of CooperationTypes to State Transition Diagrams, we assume that the static schema elements have already been mapped (to classes, attributes and services). Then, the mapping of CooperationTypes can be done by executing the following steps for each ThingType:

(1) Select one ThingType (class). Check its occurrence in a pre- or postcondition of any CooperationType.

(1a) There is no occurrence: The default STD is used; the process ends.

(1b) The ThingType occurs in at least one CooperationType: Continue with (2).

(2) The next goal for the STD creation is now to find out, if another STD needs to be defined, which means that the default STD is not enough to fulfill the requirements. In this step a list of state-candidates is provided to the designer. This means to show a list of those CooperationTypes, where the selected ThingType occurs in pre- or post-conditions.

(3) The designer has to make a decision based on the list of state-candidates:

(3a) If there are no appropriate state-candidates the default STD is used and this process ends for this ThingType.

(3b) If there are sufficient states, the next step is to map those identified pre- and postconditions to states of the respective ThingType (class).

(4) Map the identified pre- and postconditions to states of the class and define transitions (services in the OlivaNova Modeler) from and to these states and their WHEN-conditions:

(4a) State: Pre- and Postconditions of each CooperationType, where the attributes of the ThingType have certain values or value-areas.

(4b) Transition (service): OperationType; Postconditions define in which state the transition ends – the same or another state. Preconditions define from which state the transition has started.

AND

(4c) WHEN condition (control condition in the OlivaNova Modeler): Preconditions of a CooperationType if an attribute of the selected ThingType is involved in this precondition of the OperationType in (4a).

Example: We refer to the example discussed in figure 5: A registered reader loans a book and hasn't loaned one before. This means the state changes from *no loan* to *loan*.

The pre-condition of a CooperationType has the name "no books loaned" and an involved ThingType reader may have an instance with the attribute numberofbooks and value "=0". This precondition is the first state *no loan*.

A second state is mapped from the postcondition "books loaned" (and the value ">0" for the attribute numberofbooks of the involved ThingType reader).

The transition starts in the state *no loan* and results in the state *loan*. The OperationType loan of that CooperationType is the service of the transition. The WHEN condition is a concatenation of the attribute/value pairs of the precondition, in this case "numberofbooks =0". (result see figure 6)

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

4.4 Mapping of Triggers

The OlivaNova Modeler provides triggers only in conjunction with the execution of services. I.e., triggers can't be fired, scheduled or time-controlled autonomously. To describe a trigger, the corresponding class (the trigger becomes a part of), the destination (Self, Object, Class or For All), the condition when it should be fired, and the action (service) which should be executed must be defined.

The corresponding class is the image of the involved ThingType (having InvolvementType="in" within the OperationType in question).

The destination is the image of the executing actor (a class) of the OperationType that corresponds to the service (the action) to be executed.

The condition of the trigger (formula) is the appropriate Precondition of the CooperationType where the OperationType is a part of.

E.g.: A reader gets *informed* by Email if he tries to loan one book more than the maximum number of loans allowed: The OperationType *inform* has the calling actor *system* and there exists a ThingType with the InvolvementType="in" of the class *reader*. The executing actor is again the *reader*, i.e. the destination is "self". The corresponding CooperationType has a precondition *maxnumber_reached* which becomes the condition for the trigger ("numberofbooks=10"). The action *inform* will be executed when the trigger is fired. (result see figure 7).

5 Prototype

We have implemented a prototype which imports a KCPM schema (exported by the KCPM tool in form of a XML file), maps the concepts to the ONME target concepts and creates again a XML file formatted according to the import interface specification of ONME (see figure 8). The result of such an import can be seen in figure 6.

Firstly after loading the KCPM XML file, the mapping of the static structure to static concepts of the target concepts is finished (classes, attributes, services). This is executed automatically to a large extent, although there are still some decisions to be made by the developer.

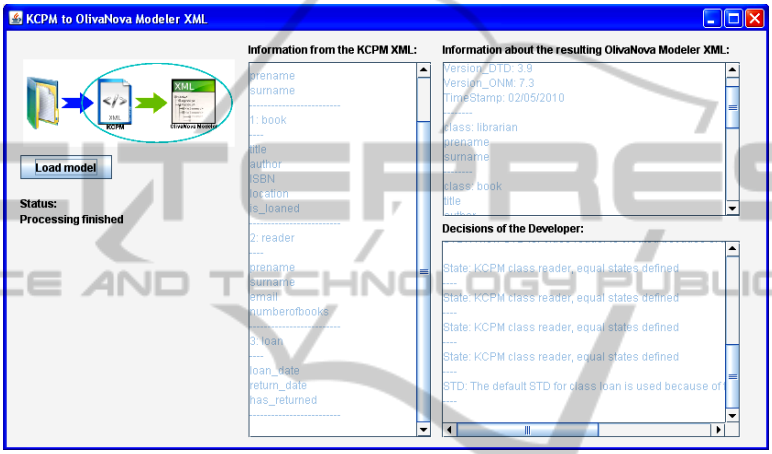

Fig. 8. Main screen of the prototype.

As a next step, the algorithm sketched in section 4 is executed: each ThingType within the KCPM schema is checked if there are possible state-candidates to create a STD. The developer gets a list with those candidates and can choose if he wants to create a STD or use the default one. If a customized STD is chosen, the developer has to check the states and afterwards the STD is automatically created.

Simplified STD Example: For loaning a book in a library, there are two possible state candidates: The *reader* has loaned a *book* or he hasn't (see figure 6). The *create*, *delete* and *edit* transitions are generated automatically in our prototype because of restrictions of the OlivaNova Modeler. The five other transitions with their services, start and target state and WHEN-conditions are gained out of further Cooperation-Types (loan a book when the reader hasn't loaned one before, loan a book when he has already loaned one, inform the reader when he has loaned 10 books, return a book when he still has at least one more loaned, return the last book).

If there is an OperationType within a CooperationType where the calling actor is "system", a trigger is created via the mapping rules defined in section 4 (see figure 7).

The ONME import interface demands a couple of further tags which had to be defined besides of the mapping of the behavior model parts; e.g., basic project information, coordinates for the user interface, checksums.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

The output XML file can be imported using the OlivaNova Modeler import function. This routine checks the generated XML - e.g. if it is well-formed, if the needed tags are contained – and in case of success creates a new model file which contains the mapped information (see figure 6 for the dynamic view of such an imported model). To generate sourcecode further information has to be added (define the functional and presentation model). As a next step the ONME validation process checks if the model is complete, well defined and syntactically correct. In case of success, source code is generated based on added PSM information.

6 Conclusions and Outlook

We claimed that user centered requirements modeling and a subsequent process of automated model transformations help to produce application systems which meet the users' needs. Since behavioral aspects (functional requirements) are an important part of requirements they have to be studied, modeled and mapped like structural aspects.

We have shown that it is possible to map the KCPM concepts for behavior modeling to the concepts of the MDA tool OlivaNova Modeler as well as the concepts for structural aspects. We have closed the gap between conceptual predesign modeling and MDA step by step, but there is still further work to do.

Currently, to our knowledge there exists no comparable MDA tool-suite to ONME on the market. That's why we used the OlivaNova Modeler for our research. In future the development of other tools has to be observed in order to strengthen the position of KCPM as an Interlingua to arbitrary MDA approaches and tools (e.g., Microsoft SQL Server Modeling [15], Eclipse Modeling Framework[16]).

The ON Modeler offers three different kinds of services that are mapped to methods in the source code: events, transactions and operations. Transactions are units of processing and can be composed of different services (events, operations, user functions): in case of an error, transactions are rolled back. In contrast to that, operations (composed of events and user functions) are executed where possible and not rolled back when failing. Within this paper we restricted ourselves to the mapping of OperationTypes to events or, in special cases, to user functions (events for state changes of the object, user functions for self-implemented-not-generated methods). For the mapping of concurrent operations (two or more OperationTypes in a CooperationType) the OlivaNova concepts of operations and transactions have to be regarded in more detail. The same applies for the use of global services.

Another challenge for subsequent projects is to extend KCPM by concepts for modeling the presentation layer in order to capture the user requirements w.r.t. menu shape and structure, actions, navigations and other presentation aspects. In this context the work of Jean Vanderdonckt about model based user-interface design [14] and Bernhard Thalheim about modeling information systems by means of storyboarding [1] may be helpful and should be considered within the on-going research.

Finally it is worthwile to evaluate the applicability of our framework to object-oriented web modeling approaches like Object-Oriented Hypermedia Design Model (OOHDM) [18], UML-based Web Engineering (UWE) [19], Object-Oriented

Hypermedia Method (OO-H) [20] or Object-Oriented Web Solution (OOWS) [21]. As mentioned in [17] there's a lack of tool support (only VisualWade for OO-H offers a commercial tool). However, still no tool provides full code generation support. So its an open research question if our approach could contribute to fill that gap.

We thank the reviewers for their valuable comments; some of them will stimulate our current research.

References

1. Hui Ma, Klaus-Dieter Schewe, Bernhard Thalheim: Storyboarding - High-Level Engineering of Web Information Systems. In: Proceedings of WISE'2009.
2. Christian Kop, Heinrich C. Mayr: Conceptual Predesign – Bringing the Gap between Requirements and Conceptual Design, Proceedings of the Third International Conference on Requirements Engineering, Colorado Springs, USA, 1998.
3. Günther Fliedl, Christian Kop, Heinrich C. Mayr: From Scenarios to KCPM Dynamic Schemas: Aspects of Automatic Mapping, Proc. Natural language processing and information systems – NLDB, Bonn, 2003.
4. Christian Kop, Heinrich C. Mayr: An Interlingua based approach to derive state charts from natural language requirements specifications, Proc. Seventh IASTED International Conference on Software Engineering and Applicatios, Maria del Rey, USA, pp. 538-543, 2003.
5. Joaquin Miller, Jishnu Mukerji (Ed.): MDA Guide Version 1.0.1, Object Management Group, Framingham, Massachusetts, June 2003.
6. Oscar Pastor, Emilio Insfrán, Vincente Pelechano, José Romero, José Merseguer: OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. In Proc. of the 9th International Conference on Advanced Information Systems Engineering (CAISE 1997), Barcelona, Spain. LNCS 1250. 145–158, 1997.
7. Care Technologies, MDA, URL: http://www.care-t.com/technology/mda.asp, 2009.
8. Roger Chiang, Keng Siau, Bill C. Hardgrave: Systems analysis and design: techniques, methodologies, approaches, and architectures, M.E. Sharpe, 2009.
9. Oscar Pastor, Juan Carlos Molina, Emilio Iborra: Automated Production of Fully Functional Applications with OlivaNova Model Execution, ERCIM News No. 57, April 2004.
10. Tom Stahl, Markus Völter, Sven Efftinge, Arno Haase: Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management, 2. Auflage, dPunkt, 2007.
11. Christian Kop: Rechnergestützte Katalogisierung von Anforderungsspezifikationen und deren Transformation in ein konzeptuelles Modell, Diss., Univ. Klagenfurt, 2002.
12. Oscar Pastor, Juan Carlos Molina: Model-Driven Architecture in Practice. A Software Production Environment Based on Conceptual Modeling, Springer-Verlag, 2007.
13. Christian Kop, Heinrich C. Mayr, Nataliya Yevdoshenko: Requirements Modeling and MDA – Proposal for a Combined Approach, Proc ISD 2006, Springer Verlag, 2007.
14. Jean Vanderdonckt: Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In S. Buraga and I. Juvina, Eds., Proc. 5th Annual Romanian Conf. On Human-Computer Interaction ROCHI'2008, pp. 1-10. Matrix ROM, Bucarest, 2008.
15. Microsoft SQL Server Modeling CTP, URL: http://msdn.microsoft.com/data.
16. Eclipse Modeling Framework (EMF), URL: http://www.eclipse.org/modeling/emf/.
17. Wieland Schwinger et al.: A survey on web modeling approaches for ubiquitous web applications. IJWIS 4(3): 234-305, 2008.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

18. Gustavo Rossi, Daniel Schwabe: Model-Based Web Application Development. In Web Engineering: Theory and Practice of Metrics and Measurement for Web Development, E. Mendes and N. Mosley, Eds. Springer, 203–333, 2006.
19. Nora Koch, Andreas Kraus: The expressive Power of UML-based Web Engineering. In Proc. 2nd Int. Workshop on Web-oriented Software Techn. (IWWOST), 21–32, 2002.
20. Jaime Gómez, Cristina Cachero, Oscar Pastor: Extending a Conceptual Modelling Approach to Web Application Design. In Proc. 12th Int. Conf. on Advanced Information Systems Engineering (CAISE 2000), Stockholm, Sweden. LNCS 1789. 79–93, 2000.
21. Oscar Pastor, Joan Fons, Vicente Pelechano, Silvia Abrahão: Conceptual Modelling of Web Applications: The OOWS Approach. In Web Engineering: Theory and Practice of Metrics and Measurement for Web Development, E. Mendes and N. Mosley, Eds. Springer, 277–302, 2006.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS