Baltic J. Modern Computing, Vol. 2 (2014), No. 4, 260-271

Overview of Software Tools for Obtaining UML Class Diagrams and Sequence Diagrams from Source Code within TFM4MDA

Viktoria OVCHINNIKOVA, Erika ASNINA

Riga Technical University, Meza iela 1 k-3, Riga, LV-1048, Latvia

{viktorija.ovcinnikova, erika.asnina}@rtu.lv

Abstract. Topological Functioning Modeling for Model Driven Architecture (TFM4MDA) is an approach for software development starting from formal domain models. We plan to enhance TFM4MDA with reverse engineering principles in order to decrease a number of errors during software migration to other platforms or integration with other systems. TFM4MDA foresees the preliminary analysis of the target software system's structure and behavior at the high-level of abstraction. Since we plan to work with legacy systems, the reverse engineering can be used for obtaining the structure and behavior of the software system from source code. For better integration with TFM4MDA the system structure and behavior should be represented by Unified Modeling Language (UML) class diagrams and sequence diagrams, correspondingly. This paper presents an overview of the selected tools that supports reverse engineering and the Eclipse platform. The goal is to check what elements of the UML sequence and class diagrams can be obtained by them. The tool owners' documentation and tool tests were used for getting and analyzing this information.

Keywords. Topological functioning model, model driven architecture, reverse engineering

1. Introduction

Reverse engineering can help when a software system needs to be migrated or integrated to other new and usable platforms. Reverse engineering provides the representation of the structure and the behavior of the system at the higher level of abstraction than code does that facilitates analysis, understanding and programming processes. The process of understanding the software system is time-consuming, because it is difficult to understand the source code quickly without any skeleton (model) of the software system.

In our work, the use of reverse engineering is necessary for supporting the migration or integration of the existing functionality of legacy software systems to new technological platforms and other software systems. It can be more suitable, because the migrated software system can work faster, can occupy less memory and can be more protected from viruses and other risks. Reverse engineering will be used within the software development approach called Topological Functioning Modeling for Model Driven Architecture (TFM4MDA). As mentioned in (Ovchinnikova and Asnina, 2014), a formal mathematical model, Topological Functioning Model (TFM), gives an opportunity to analyze and model both the solution and problem domains, as well as to

develop and transform problem domain artifacts (e.g., business and knowledge models, as well as diagrams at the high level of abstraction) into solution domain artifacts (e.g., software implementation specifications and business processes).

The goal of the research is to make a deeper overview of eight selected reverse engineering tools from (Ovchinnikova and Asnina, 2014) in order to determine those, which can be used in further research work. In order to satisfy this goal, we check provision of elements of the UML class and sequence diagrams by these tools. In other words, we make a comparison between elements of the UML sequence and class diagrams from Object Management Group (OMG) UML specification (WEB, g) and those elements that are supported by tools using information from tool owners' websites. Additionally, some of tools were installed and tested for getting more precise information.

The paper is structured as follows. Section II discusses the TFM within Model Driven Architecture (MDA) and legacy system integration and migration within TFM4MDA. As well, Section II describes reverse engineering and its techniques in brief. Section III overviews the selected tools and provides information about the UML class and sequence diagram elements supported by these tools. In Section IV related work are presented. Section V states conclusions.

2. Reverse Engineering from Source Code to TFM

2.1. Topological Functioning Model within MDA

Model Driven Architecture (MDA) is a framework for improving interoperability, portability and reusability through architectural separation of concerns in specifications (Favre, 2012). It uses models for defining the complete lifecycle of a real world system; requirement specifications, architecture and design descriptions and code are considered as models. MDA models describe an examined system at various levels of abstraction. It distinguishes (Favre, 2012):

- Computation Independent Model (CIM) that describes a system implementation independent from digital computation (business models, domain models, e.g., specified by UML). It should be traceable to PIM;
- Platform Independent Model (PIM) that describes a software implementation independent from platform specific details (analysis and logical models). It should be traceable to PSM and backward to CIM;
- Platform Specific Model (PSM) that describes a software implementation from platform specific viewpoints. It should be traceable to ISM and backward to PIM;
- Implementation Specific Model (ISM) that describes all the information that is needed for constructing an executable software system. It should be traceable to code and backward to PSM.

In its turn, the TFM is a model that specifies static and dynamic characteristics of the system. It is based on algebraic topology and system theory, and can be displayed as a mathematical digraph (Osis and Asnina, 2011b). The TFM has topological and functioning properties. Topological properties have the mathematical background. They

specify causal relationships among objects, i.e., specify cause-and-effect dependencies between system functional characteristics, and are based on properties of topological spaces. The topological properties include the following concepts: neighborhood, connectedness, continuous mapping and closure (Osis and Asnina, 2011b). Functioning properties are based on the principles of system theory and include the following concepts: inputs and outputs, cause-and-effect relations, and cycle structure (Osis and Asnina, 2011b).

A TFM is represented as a topological space (X, Q) in the form of a directed graph, where X is a finite set of functional features of the system under consideration, and Q is a topology among functional features of set X (Osis and Asnina, 2011c). The functional feature characterizes the system that is needed for system's goal achievements. It is a unique 7-tuple <A - action, R - result, O - object, PrCond - preconditions, PostCond - postoconditions, Pr - provider, Ex - executor>. All of these elements and topological structure are described in detail in (Osis and Asnina, 2011c).

The TFM within MDA is being used at the computation independent level. The CIM may contain three main parts, namely, a knowledge model, a business model, and business requirements for the system (Asnina and Osis, 2011). The knowledge model provides the experts' visions of the enterprise operation with focus on its organizational structure and business specificity. The business model is focused on the business goals and business scope such as resources, facts, rules and so on. The business requirements are requirements to the software system set by business people. The TFM specifies system functioning from the computation independent viewpoint and can serve as a formal CIM (Osis and Asnina, 2008), (Asnina and Osis, 2011). The knowledge model describes a problem domain (a "source" model), usually in the textual form. The business requirements describe a solution domain (a "target" model), specifying system requirements, which are dictated by business. The business model represents a part of the knowledge model and is a source of the business requirements to the system. Therefore, it represents both the problem and solution domains. The TFM represents the problem domain in a redesigned view. Requirements to the system are mapped to the TFM and verified. At the same time, new functionality that enhances the problem domain is determined. As a result, the TFM of the solution domain is constructed. When knowledge, business processes, system structure and system/software requirements are defined, use case diagrams can be created from the TFM of the solution domain (Osis et. al., 2008a), (Osis et. al., 2008b), (Osis et. al., 2007). The use case diagram can represent the TMF partially or fully (as necessary) and are specified by using corresponding UML diagrams and textual use case specifications.

2.2. Legacy System Migration and Integration within TFM4MDA

Let us explain the main motivation of the research started in (Ovchinnikova and Asnina, 2014). Fig. 1 illustrates our research in the context of MDA and TFM4MDA. In MDA, the conformity between code and a domain model (CIM) is grounded on the fact that code is produced on the base of transformations from CIM to PIM to PSM to code (Fig. 1, the left side). Business process logic and business structure implemented in code can be backward traced to PSM to PIM to CIM, by abstracting from platform specific and application specific details. MDA suggests using UML diagrams for specification of CIMs, PIMs and PSMs.

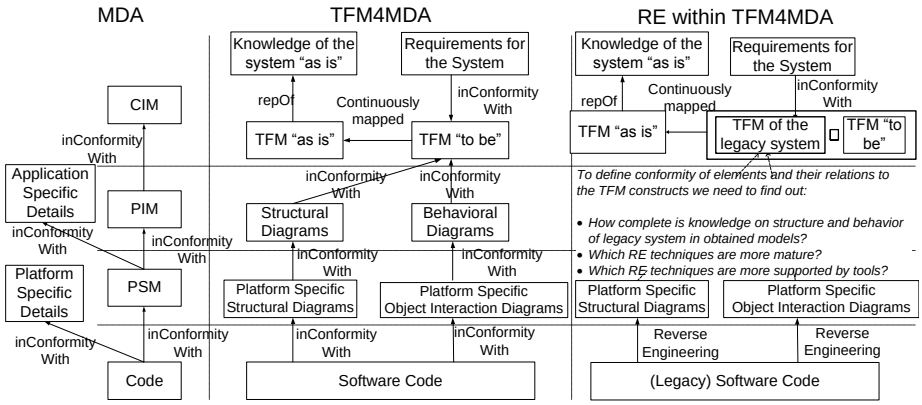

Fig. 1. Conformity between models in MDA, TFM4MDA and in case of legacy systems

In case of TFM4MDA, the CIM is represented by two pairs – the knowledge model and topological functioning model (TFM) of the system "as is" that represent the problem domain, and the requirements model and TFM of the system "to be" that represent the solution domain. The conformance of the solution domain to the problem domain is supported by continuous mapping between two TFMs (Fig. 1, the middle part). The continuous mapping is a mathematical mechanism that allows providing conformity between two topological spaces. Other models, PIMs and PSMs are specified by UML diagrams.

The question is how to guarantee this conformance within the TFM4MDA, when we need to migrate or integrate the legacy software system on the new technological platforms. The weak part here could be knowledge about functionality of the legacy system (Fig. 1, the right part). The TFM of the system "as is" will include this functionality completely. The TFM of the system "to be" will contain it completely of partially. Thus, it is worth to have a TFM of the structure and behavior of the legacy system in order to fasten the system analysis and to make preliminary verification of the integrated "target" system.

Reverse engineering techniques and tools can help to solve this weakness. They allow obtaining platform specific models (PSMs) from source code. The next step should be getting the TFM of the legacy system on the base of these PSMs. However, we have a lack of information about quality of the obtained PSMs, and their suitability for establishing conformity of PSM elements and their relations to the TFM constructs.

2.3. Reverse Engineering and its Techniques

Reverse engineering helps in understanding software behavior and structure by representing it at the high level of abstraction. It gives a chance to see incompleteness which can be done writing the source code or to compare a created domain model with its primary version (Ovchinnikova and Asnina, 2014).

Reverse engineering combines techniques from modeling of business processes, objects, data, and components. Converting source code to a domain model (UML diagrams) it represents elements which are necessary to describe a domain such as an

entity, its attributes, responsibilities, and relationships (it depends on a domain model type). Then the domain model described by UML diagrams can be manually (it is not automated) reverted to the necessary business models (Ovchinnikova and Asnina, 2014).

Traditional reverse techniques from source code to domain models are the following (Ovchinnikova and Asnina, 2014):

- static analysis – getting the static information that describes the structure of software;
- dynamic analysis – getting the dynamic information that describes how software behavior is organized.

The reverse engineering process can be enriched by static and dynamic analysis combinations. The static analysis creates PIMs or PSMs that can be enhanced by dynamic analysis. For example, Fig. 1 illustrates that the PIM layer displays structural and behavioral diagrams that can be represented by UML (the UML class diagram can represent structure of the system, and a set of UML sequence diagrams – behavior of the system).

Further these diagrams can be converted to the TFM (Ovchinnikova and Asnina, 2014). The UML use case diagram, class diagram, activity diagram, object diagram, collaboration and sequence diagrams can be obtained from the TFM manually as it is represents in (Osis et. al., 2008b), (Osis and Asnina, 2011d), (Donins, 2012). This means that the TFM (or its part) can be recreated from these diagrams by back (reverse) steps.

3. Selected Reverse Engineering Tools

In reverse direction within TFM4MDA, as it is illustrated in Fig. 1, we need to go from a bottom (source code) to the top (a domain model). Using reverse engineering tools, UML diagrams which display the structure and behavior of the system can be created from the source code, and then transforming these diagrams the TFM “to be” can be created (manually by now). It enables developers to enhance the TFM “to be” of the target system with functionality implemented in the legacy systems of the organization.

The next subsection more deeply discusses tools, which preliminary comparison by using selection criteria has been started in (Ovchinnikova and Asnina, 2014).

3.1. Brief Descriptions of the Selected Tools

Eight tools, namely Imagix4D, ArgoUML, AmaterasUML, jGRASP, Visual Paradigm for UML, Fujaba, EclipseUML, and MoDisco have been discussed in the (Ovchinnikova and Asnina, 2014). As mentioned previously, it is necessary to have the structure and behavior of the system for creation of the TFM. The UML class diagram can provide the structure of the system and the UML sequence diagrams can provide the behavior of the system. A possibility of creation of these diagrams needs to be supported in the selected tools. Tools that support obtaining the UML sequence and class diagrams from code, as well as what certain elements of those diagrams are really supported there, has been investigated deeply and the results are presented here.

Imagix4D (WEB, a) and jGRASP (WEB, c) tools are not tested more deeply in this research, because they can create only one necessary for our goal diagram, i.e., the UML class diagram, as well they are standalone tools and do not support Eclipse platform.

Both Visual Paradigm for UML and EclipseUML are commercial tools, but others four are free tools. This means that commercial tools may have limitations during a trial period, but free tools may have limitations in documentation completeness. Five of these tools (EclipseUML, ArgoUML, AmaterasUML, Fujaba, and MoDisco) were installed and tested for getting more information about elements of the UML class and sequence diagrams.

Fujaba (WEB, f) or Fujaba4eclipse is free academic tool. This tool has a possibility to create the UML class, activity and others diagrams. It provides reverse engineering for Java code. The UML class diagram can be obtained by reverse engineering. There is not information about export and import possibilities. This plug-in for Eclipse does not work correctly on the Eclipse Keepler and Eclipse Juno platforms, thus the class diagram is not created and only an empty fujaba model is provided after code transformation. Therefore, this tool will not be tested further in this research.

Visual Paradigm for UML (WEB, d) tool's 30-day trial version is provided for tool understanding and trying. It has four types of licenses:

- enterprise – there are provided all possible functions in the tool;
- professional – there are absent some of functions;
- standard - it is used for modeling, report creation and source code generation;
- modeler – it is used only for business process and software design.

All of these types of licenses can be tried by free with all necessary functions during 30 days. This tool gives users the possibility to create the UML class, sequence and activity diagrams and others. The reverse engineering of Java source files, C++ header files, .NET *.dll and *.exec files, CORBA IDL source file, Ada 9x sources files, XML, XML schema, PHP 5.0 source files, Python, Objective-C are provided in this tool. The reverse UML class diagram can be got from the mentioned source files, but only Java source files can be used for getting of the UML sequence diagram. The obtained UML class and sequence diagrams can be changed and completed by user in further. As well this tool supports XML (only XML documents exported from this tool can be imported back to it), XMI (version 1.0, 1.2, 2.1), Microsoft Excel and Visual Paradigm project export and import.

EclipseUML (WEB, e) tool includes a free 30-day evaluation license for tool testing and understanding. It has four types of licenses:

- academic – it used only for universities, institutes and so on, a student can't buy it for himself;
- viewer - it is used only for source code and model synchronization, as well diagrams can be viewed (but can't be created);
- mobile – a developer can install as many copies of this tool as he/she needs;
- floating – a developer can use the tool via connection to the server (1.6 developers per one license).

Only mobile and floating licenses can be used for free during 30 days. Others can be used only after paying for them. It is possible to create the UML sequence, class, activity and others diagrams in this tool. The reverse engineering of Java source files is also supported. The reverse UML package, class and sequence diagrams can be got from source code and completed by a developer in further. This tool provides XMI (version 2.1) export and import.

ArgoUML (Ramirez et. al., 2012) or ArgoEclipse as a plug-in for the Eclipse platform is a free tool. The UML class, activity and others diagrams can be created in ArgoEclipse. The user guide information illustrates that the UML sequence diagram can be created in this tool. However, the user guide information does not provide a lot of necessary information. The tool ArgoEclipse was installed in order to check whether the UML sequence diagram elements exist. As well the UML sequence diagram icon is provided in the tool, but it does not work. Thus, it is not possible to test which elements in the sequence diagrams exist. This tool supports XMI (version 1.0, 1.1 and 1.2) export and import, however the import does not work properly in the ArgoEclipse tool. The option for creation UML sequence diagrams exists in the ArgoUML (a tool that is independent from Eclipse) and only some elements (lifelines without type and synchronous, asynchronous, reply, object creation and deletion messages) can be used there. There is not much information about reverse engineering in ArgoUML and is not mentioned which programming languages can be used and which UML diagrams can be obtained from source code. But source files can be imported into ArgoUML. However, in such a case only a class diagram without any relations between elements will be created.

AmaterasUML (WEB, b) is a free tool. This tool gives a possibility to create the UML class, sequence, activity and others diagrams. There is no information about supported programming languages in reverse engineering at the tool owner's website (but it is stated that Java programming language is supported). The UML class diagram can be obtained during reverse engineering, for this it is necessary to move all UML classes from the created project to the working window. The UML sequence diagrams can also be automatically generated from Java stack traces or debug stacks (after minor manipulation). Information about import/export wasn't found at the tool owner's website.

MoDisco (Dupe, 2012) is a free tool. The UML diagrams cannot be created as such in this tool, because it is only the reverse engineering tool. It provides the reverse engineering for Java source code and other programming languages. This tool gives a possibility to create a KDM (Knowledge Discovery Metamodel) model as an XMI file of the existing project. The XML document or UML model can be created from this XMI file. After that, the necessary UML diagrams can be visualized using an assisting tool (e.g. Papyrus, UML Designer). The information about elements of the UML class and sequence diagrams is not provided at the tool owner's website, that is why this tool will not be included in the next subsection.

3.2. Elements of the UML Sequence and Class Diagrams in the Selected Tools

The structural and behavior aspects of a domain model can be represented by UML diagrams. For example, UML class diagrams capture the structural aspects of a domain model represented by classes, attributes, operations and relationships, but UML sequence diagrams capture behavior aspects of a domain model represented by objects and their interactions. A domain model created by using UML diagrams describes an examined problem space, but does not overview it completely. Implementation specifics usually cannot be specified by UML.

The UML defines 13 types of diagrams, divided into 2 categories (WEB, g):

- Structure diagrams – an object diagram, a class diagram, a composite structure diagram, a component diagram, a deployment diagram and a package diagram.
- Behavior diagrams – an activity diagram, a sequence diagram, a use case diagram, a communication diagram, an interaction overview diagram, a state machine diagram, and a timing diagram.

There are two incidence matrices of elements from UML sequence and class diagrams that exist in the selected tools illustrated in Table 1 and Table 2. Columns in tables represent tools' names. Rows in Table 1 represent elements of the UML class diagram, and rows in Table 2 represent elements of the UML sequence diagrams. More information about the elements of the UML sequence and class diagrams can be found in (WEB, g).

Table 1. The matrix of elements of the UML class diagram that exist in the selected tools

	Visual Paradigm for UML	EclipseUML	ArgoEclipse	AmaterasUML
Class	1	1	1	1
Class attribute	1	1	1	1
Attribute visibility	1	1	1	1
Attribute name	1	1	1	1
Attribute type	1	1	1	1
Class operation	1	1	1	1
Operation visibility	1	1	1	1
Return value type in the class operation	1	1	1	1
List of parameters in the class operation	1	1	1	1
Interface	1	1	1	1
Package	1	1	1	
Association	1	1	1	1
Aggregation	1	1	1	1
Composition	1	1	1	1
Dependency	1	1	1	1
Generalization	1	1	1	1
InterfaceRealization	1	1	1	1
Realization	1	1	1	1

Tools Visual Paradigm for UML, EclipseUML, and ArgoUML have all elements of the UML class diagram as it is illustrated in Table 1. The tool AmaterasUML also has all elements of the UML class diagram, excluding the "Package" element. All the tools mentioned in Table 1 can show the structure of the software system.

Table 2. The matrix of elements of the UML sequence diagram that exist in the selected tools

	Visual Paradigm for UML	EclipseUML	ArgoEclipse	AmaterasUML
Outside actor	1	1		1
Lifeline without type	1	1		1
Lifeline <Boundary>	1	1		
Lifeline <Control>	1	1		
Lifeline <Entity>	1	1		
Synchronous message	1	1		1
Asynchronous message	1	1		1
Object creation message	1	1		1
Object deletion message	1	1		
Reply message	1	1		1
Lost message	1	1		
Found message	1	1		
Unknown message				
Alt frame	1	1		
Opt frame	1	1		
Par frame	1	1		
Loop frame	1	1		
Critical frame	1	1		
Neg frame	1	1		
Assert frame	1	1		
Strict frame	1	1		
Seq frame	1	1		
Ignore frame	1	1		
Consider frame	1	1		

Both tools Visual Paradigm for UML and EclipseUML have all elements of the UML sequence diagram, excluding element "Unknown message", as it is illustrated in Table 2. The UML sequence diagram cannot be created in the tool ArgoEclipse, because this function does not currently work correctly in it. This is the reason why no one element is included in Table 2. The AmaterasUML has only some of the UML sequence diagram elements, shown in Table 2. Only two tools, namely Visual Paradigm for UML and EclipseUML, can show the behavior of the software system in detail. The ability to show the behavior of the software system in the AmaterasUML is limited, because it does not support any kind of the UML sequence diagram frames.

4. Related work

Architecture Driven Modernization (ADM) that is implemented by OMG is an approach that supports reverse engineering within MDA. It is used for existing software system modernization, understanding, improvement, modification, migration, translation into

another language and so on (WEB, h). The following tools can be used for this approach (Madiot, 2010):

- MoDisco tool which is discussed in this research can be used for ADM, because in this tool a KDM is created from source code;
- Mia-Mining is created for analyzing COBOL programs and providing information about their internal data and structure;
- Mia-Studio for generation model-to-text templates and developing model-to-model transformation rules.

Model Driven Reverse Engineering (MDRE) is another approach that supports reverse engineering within MDA. It is used for model of legacy system understanding, manipulation and discovery. The following tools can be used for this approach (WEB, i):

- MoDisco also can be used for MDRE for model discovery and understanding;
- ATL is a model transformation toolkit and language. It provides producing of target models from source models.

5. Conclusions

In the research on reverse engineering within TFM4MDA, the UML diagrams are planned to be used as an assisting model in a transformation chain from the source code to the TFM. In order to obtain them, we have investigated six tools, namely Visual Paradigm for UML, EclipseUML, ArgoUML, AmaterasUML, MoDisco and Fujaba. At the beginning of the research we have selected eight tools, but Imagix4D and jGRASP have been excluded from the further investigation, since they create only the class diagrams and do not support interaction with the Eclipse platform.

The VisualParadigm for UML and EclipseUML tools are commercial tools. As the research showed, they provide more functionality and more information in the user manual than the free tools. As well these tools have all necessary elements of the UML class and sequence diagrams as it is shown in tables above. The free tools have a little of information or it is absent at all in the user guides. The Fujaba tool is an academic tool and a user guide is not provided for it, as well this tool does not work on the Eclipse platform. MoDisco only creates KDMs, from which the UML diagrams can be created and refined by using assisting tools in further. The UML sequence diagram cannot be created in the ArgoEclipse tool due this function does not work properly. Summarizing the results, only three tools, namely VisualParadigm for UML, Eclipse UML and AmaterasUML supports a greater number of required elements.

The results of the research are mainly based on the information about the selected tools provided by the tools' vendors. Five of the six selected tools have been installed and support of the elements of the UML class and sequence diagrams has been determined. However, the reverse engineering is supported in the three tools that showed better results and must be investigated more deeply.

One of the future research directions is related to verification of quality of the obtained UML diagrams from the source code by experiments with VisualParadigm for UML, Eclipse UML and AmaterasUML. Another one foresees complete development of transformation rules from the received qualitative UML diagrams to the TFM.

References

Asnina E., Osis J. (2011). Topological Functioning Model as a CIM-Business Model. In: (Osis and Asnina, 2011a), 40 - 64.

Donins U. (2012). *Topological Unified Modeling Language: Development and Application*. PhD thesis, Riga Technical University, Riga, Latvia.

Dupe G. (2014). *MoDisco*, available at http://wiki.eclipse.org/MoDisco#Documentation.

Favre L. (2012). *MDA-Based Reverse Engineering*, available at http://www.intechopen.com/books/reverse-engineering-recent-advances-and-applications/mda-based-reverse-engineering.

Madiot F. (2010). *Architecture-Driven Modernization Case-Studies*, available at http://fmadiot.blogspot.com/2010/04/architecture-driven-modernization-case.html.

Osis J., Asnina E., Grave A (2008). Formal Computation Independent Model of the Problem Domain within the MDA. Information Systems and Formal Models, In: Jaroslav Zendulka (ed.) *Proceedings of the 10th International Conference*, (Apr., Hradec nad Moravici, CzechRepublic), Silesian University in Opava, Opava, 47 – 54.

Osis J., Asnina E.. (2008). A Business Model to Make Software Development Less Intuitive, In: M. Mohammadian (ed.) *Proceedings of the 2008 International Conference on Innovation in Software Engineering* (10 – 12 Dec. Vienna, Austria), IEEE Computer Society CPS, Los Alamitos, 1240 – 1246.

Osis J.,Asnina E.,Grave A. (2008). Formal Problem Domain Modeling within MDA. *Software and Data Technologies: Communications in Computer and Information Science 22*, 387 - 398.

Osis J., Asnina E., Grave A. (2007). MDA Oriented Computation Independent Modeling of the Problem Domain, In: Cesar Gonzalez-Perez, Leszek A. Maciaszek (eds.) *Proceedings of the 2nd International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2007* (23 – 25 Jul. Barcelona, Spain), INSTICC Press, Barselona, 66 - 71.

Osis J., Asnina E. (2011a). Model-Driven Domain Analysis and Software Development: Architectures and Functions, IGI Global, New York.

Osis J., Asnina E. (2011b). Is Modeling a Treatment for the Weakness of Software Engineering? In: (Osis and Asnina, 2011a), 1 -14.

Osis J., Asnina E. (2011c). Topological Modeling for Model-Driven Domain Analysis and Software Development: Functions and Architectures. In: (Osis and Asnina, 2011a), 15 -39.

Osis J., Asnina E. (2011d). Derivation of Use Cases from the Topological Computation Independent Business Model. In: (Osis and Asnina, 2011a), 65 -89.

Ovchinnikova V., Asnina E. (2014). Reverse Engineering Tools for Getting a Domain Model within TFM4MDA, In *Proceedings of the 11th International Baltic Conference on Databases and Information Systems Baltic DB&IS 2014*, (8-11 Jul. 2014, Tallinn, Estonia), Tallinn University of Technology Press, Tallinn, 417-424.

Ramirez A., Vanpeperstraete P., Rueckert A., Odutola K., Benett J., Tolke L., Wulp M.. (2011). *ArgoUML user manual*, available at http://argouml-downloads.tigris.org/nonav/argouml-0.34/manual-0.34.pdf.

WEB (a). *Imagix 4D*, available at http://www.imagix.com/products/source-code-analysis.html.

WEB (b). *AmaterasUML*, available at http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=AmaterasUML.

WEB (c). *jGRASP*, available at http://www.jgrasp.org/.

WEB (d). *Visual Paradigm for UML User's Guide*, available at http://www.visual-paradigm.com/support/documents/vpuserguide.jsp.

WEB (e). *EclipseUML*, available at http://www.omondo.com/.

WEB (f). *About Fujaba*, available at http://www.fujaba.de/about-fujaba.html.

WEB (g). *Unified Modeling Language, version 2.4.1*, available at http://www.omg.org/spec/UML/2.4.1/.

WEB (h). *Architecture-Driven Modernisation Task Force*, available at http://www.omgwiki.org/admtf/doku.php.
WEB (i). *Model Driven Reverse Engineering*, available at http://www.emn.fr/z-info/atlanmod/index.php/Model_Driven_Reverse_Engineering.

Author's information

Viktoria Ovchinnikova took bachelor's degree in Automation and Computer Engineering from Riga Technical University, Latvia, in 2013.

Currently she is the second year master's student and Scientific Assistant at Department of Applied Computer Science in Riga Technical University. She actively participates in the scientific research projects. She is an author of one conference paper.

Her research interests include programming, system modeling, reverse engineering and model-driven software development.

Erika Asnina received M.Sc. in computer systems in 2003 and doctor's degree (Dr.sc.ing.) in information technology with specialization in system analysis, modeling and design from Riga Technical University in 2006.

She is Associate Professor in the Department of Applied computer Science in Riga Technical University since 2013. She also worked 5 years as a Software Developer. She is author of 37 conference papers, four book chapters and one book. Her research interests include software quality assurance, model-driven and object-oriented software development, and software engineering.

Latvian Academy of Sciences has awarded her and her co-author, Janis Osis, for the book "Model-Driven Software Development: Architectures and Functions", which was recognized as one of the most significant theoretical achievements of Latvian Science in 2011. She was also awarded as a scholarship laureate of the target program "For Education, Science and Culture" of Latvian Education Fund in 2004 and 2005.

Received November 9, 2014, accepted November 13, 2014