The Algorithm of Transformation from UML Sequence Diagrams to the Topological Functioning Model

Viktoria Ovchinnikova and Erika Asnina
Department of Applied Computer Science, Riga Technical University, Meza Street 1 k-3, Riga, Latvia

Keywords: Topological Functioning Model, Reverse Engineering, Model-driven Architecture.

Abstract: It is difficult and time-consuming to migrate a legacy system to some new platform or integrate it with other software system manually. High-level abstract models (domain models) of the existing software system must be got for further merging with new domain models. TFM4MDA (Topological Functioning Modeling for Model Driven Architecture) is an approach for software development from the high level of abstraction to the lower levels. The formal TFM (Topological Functioning Model) for software system analysis can be obtained stepwise from the low levels using RE (Reverse Engineering) techniques. The algorithm for transformation from UML sequence diagrams to the TFM is suggested in this research. It is based on the previous research results. Additional information about other approaches such as MDRE (Model-Driven Reverse Engineering) and ADM (Architecture Driven Modernization) is overviewed in order to use it for further analysis and full formalization of the transformation considered in our work.

1 INTRODUCTION

TFM4MDA (Topological Functioning Modeling for Model Driven Architecture) is an approach for software system development. It uses all models from MDA (Model Driven Architecture):

- CIM (Computation Independent Model) that is represented by the TFM (Topological Functioning Model);
- PIM/PSM (Platform Independent Model / Platform Specific Model) that should be represented by UML (Unified Modeling Language) diagrams, where UML class diagrams represent the structure of the software system and UML sequence diagrams – the behavior;
- ISM (Implementation Specific Model) – it is a source code.

In our case, RE (Reverse Engineering) is needed for software system analysis and examination, e.g., when we need to migrate a legacy system to other platform or integrate it with other software systems. RE gives an opportunity to get the structure and behavior of the software system by representing source code at the higher level of abstraction – in our case at the level of UML diagrams. The initial results about legacy system integration or migration within TFM4MDA are presented in (Ovchinnikova and Asnina, 2014a). They have been applied for choosing the tool and transforming code. Thus, eight tools were overviewed by five criteria in (Ovchinnikova and Asnina, 2014a) for defining their functions and availability. The main criteria are supported RE techniques, programming languages, and UML diagrams. In (Ovchinnikova and Asnina, 2014b) four tools have been selected from those eight and overviewed more deeply using the UML specification and available information about these tools. As well as some tools that lacked complete information have been installed and checked manually for getting necessary data. It was necessary to check which elements of the UML class and sequence diagrams exist in and are supported by these tools. The transformational mappings from the manually created UML sequence diagrams to the TFM are discussed in (Ovchinnikova, et al., 2014).

In this paper, the algorithm of transformation from UML sequence diagrams to the TFM is suggested. It states the main steps, which will be needed for creating the QVT (Query / View / Transformation) transformation as well as for necessary changes in the existing metamodel of the TFM in the future research. Additionally, related work with other approaches in the field is overviewed. The purpose of those approaches is

Ovchinnikova V. and Asnina E..
The Algorithm of Transformation from UML Sequence Diagrams to the Topological Functioning Model.
DOI: 10.5220/0005476603770384
In *Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering* (MDI4SE-2015), pages 377-384
ISBN: 978-989-758-100-7

similar, i.e. getting the domain model from code.

The paper is structured as follows. Section 2 represents information about the TFM4MDA in brief. Section 3 describes and shows the transformation algorithm from the UML sequence diagram to the TFM. Section 4 represents related work and Section 5 provides conclusions.

2 TFM4MDA IN BRIEF

As previously mentioned, the TFM4MDA uses all four MDA models: CIM, PIM/PSM and ISM for software development, analysis and modeling. The CIM can contain the following three main parts (Asnina and Osis, 2011c): Business requirements for the software system, Business Model, and Knowledge Model.

In the TFM4MDA the Business Model with Business requirements describe a solution domain and the Business Model with the Knowledge Model describe a problem domain. The TFM serves as this Business Model and maps the solution domain to the problem domain.

The conformity between models in MDA, TFM4MDA and reverse TFM4MDA is presented in (Ovchinnikova and Asnina, 2014b), where the solution domain is shown as the TFM of the system "TO BE" and the problem domain is shown as the TFM of the system "AS IS". The conformity between the problem domain and the solution domain is supported by continuous mappings between these two TFMs.

Figure 1 illustrates the RE within TFM4MDA (or reverse TFM4MDA). The TFM of the system "AS IS" fully include the knowledge about functionality of the legacy software system, but the TFM of the system "TO BE" may include it partially or fully. From the legacy software code we can got the PIM/PSM with the structure and the behavior of this system. The platform specific structural diagram of the software system can be represented by the UML class diagram and the platform specific object interaction diagram can be provided by the UML sequence diagram. From these UML diagrams the TFM of the legacy software system can be obtained by using transformation mapping rules among the constructs of the UML diagrams and the TFM.

The TFM contains and represents knowledge about the dynamic and static parts of the software system. Using RE, the UML class diagram is needed for providing the structure of the software system and the UML sequence is necessary for ensuring the behavior of the software system.

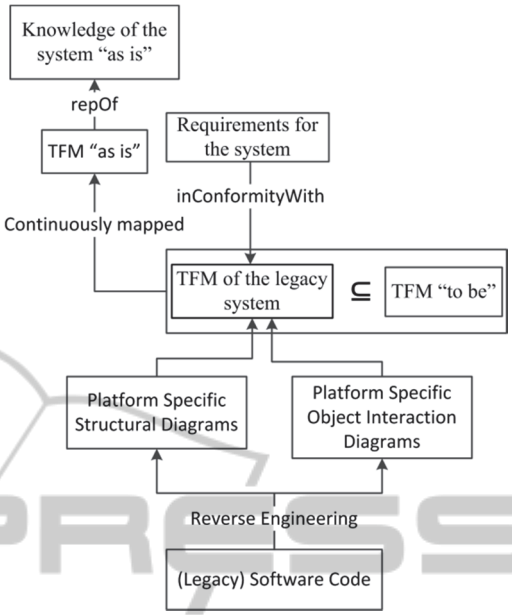

Figure 1: Reverse engineering within TFM4MDA.

The TFM can be characterized by two kinds of properties: functioning and topological. The functioning and topological properties allow modeling functional characteristics (features) of the business system. One functional feature represents one business process, activity or task execution. According to (Osis and Asnina, 2011a) the functioning properties are inputs and outputs, cycle structure and cause-and-effect relations. The topological properties are neighborhoods, connectedness, continuous mapping and closure (Osis and Asnina, 2011a).

The TFM is represented as a topological space (X, Q), where X is a finite closed set of functional features with topology Q on set X. The topology Q is represented in the form of the directed graph. Examples of the TFM are provided in (Osis and Asnina, 2011). A cause-and-effect relationship between two functional features of the system exists if the execution or calling of one functional feature is caused by the second functional feature and there are no another functional feature between them, as it is discussed in (Osis and Asnina, 2011b).

The functional features contain information of the system in the form of a unique 11-tuple <Id (identifier), A (object's action), R (result of the object's action), O (object), PrCond (preconditions), PostCond (post-conditions), Pr (providers), Ex (executers), Req (requirements), Cl (class), Op (operation)>. It is described in detail in (Donins, 2012) and (Osis and Asnina, 2011b).

The stepwise process of obtaining the TFM from system verbal descriptions is described in (Osis, et al., 2008). Another example of its construction is provided in (Asnina and Osis, 2010) with a focus on continuous mappings between solution and problem domains. According to (Donins, et al., 2011) the PIM can be represented as a topological UML class diagram with all elements taken from the TFM.

3 THE TRANSFORMATION ALGORITHM

The IDM toolset, which implements the IDM (Integrated Domain Modeling) approach and supports automatic creation of the TFM from the business use case descriptions, is implemented and described in (Shile and Osis, 2014). The transformation implemented in the tool works at the CIM level.

The manual transformations to the UML class, sequence, use case, object and activity diagrams from the TFM are discussed and provided in (Osis, et al., 2007), (Osis and Asnina, 2011d), (Donins, 2012). Uldis Donins (Donins, 2012) described mappings between constructs of the TFM and constructs of all TopUML (Topological UML) diagrams. The TopUML modeling is an extension of UML that helps in tracing the cause-and-effect relations between the solution and problem domains clearly.

In our case, we consider a reverse transformation, i.e. from PIM/PSM to the CIM. Donins' method can be only partially used for reverse transformation, i.e. for obtaining the TFM from the UML class (for structure of the software system) and sequence diagrams (for behavior of the system). The reason is that not all information can be kept and represented similarly at all levels of abstraction.

The mappings among TFM and UML sequence diagram elements and manual transformation from the UML sequence diagram to the TFM are described and provided in (Ovchinnikova, et al., 2014). As well as the example of this transformation is provided. The semantic differences and similarities between constructs of the UML sequence diagrams and the TFM are overviewed and analyzed in more detail in (Ovchinnikova, et al., 2014).

For automating this process, the transformation from the UML sequence diagram to the TFM can be done using QVT transformations. The schematic algorithm for this transformation is provided in this research.

The UML sequence diagram consists of the following elements (OMG, 2011):

- Outside actor – it is not a part of a software system;
- Lifeline – it is an active role (object), which interact with other roles;
- Message – an action send from one lifeline to other;
- Frame (fragment) – it can cover a part of the sequence diagram, which can be performed as a cycle or under some condition.

Figure 2 represents the transformation process, where a new XMI (XML (Extensible Markup Language) Metadata Interchange) output file is generated from all XMI input files:

- An XMI input file contains information about the UML sequence diagram, where it is known from which lifeline a message is sent and which lifeline receives it, as well as the information about frames;
- An XMI output file contains information about the TFM, i.e., a set of functional features and a set of topological relationships among functional features. As well all functional features have 11-tuple with information about themselves. However, it is not necessary to fill all 11 elements.

Figure 3 represents the next flowchart part (Part2) of the transformation algorithm. It takes each message from the first to the last one and checks whether this message is included in the frame or not. If it is included in the frame, then Part3 of the algorithm will be executed. Otherwise if is not included, then Part4 of the algorithm will be executed.

Figure 4 represents flowchart Part3, where the frame's name is taken for understanding whether there are conditions or loops. In this case we look at four more usable frames: *alt* (alternative), *opt* (option), *par* (parallel) and *loop*. In (Ovchinnikova, et al., 2014) the table with frame types of the UML sequence diagram is presented. The message will be executed only after the execution of the precondition for frames *alt* and *opt*.

The frame *loop* can be represented as a cycle in the TFM, there is no precondition before the message will be executed. The messages will be also executed without preconditions in the frame *par*, because all these messages will be done simultaneously. If frame's name is not *par* or *loop* then Part4 of this algorithm will be executed, otherwise the precondition will be set in the

functional feature as one parameter (i.e., precondition PrCond) in the 11-tuple. Similarly, if another frame is included in the existing frame, then we handle it the same.

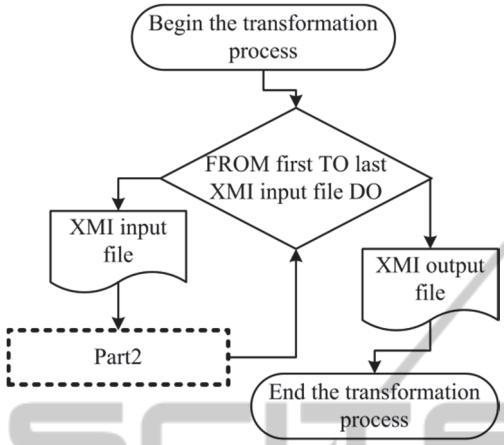

Figure 2: The transformation process from the UML sequence diagram to the TFM.

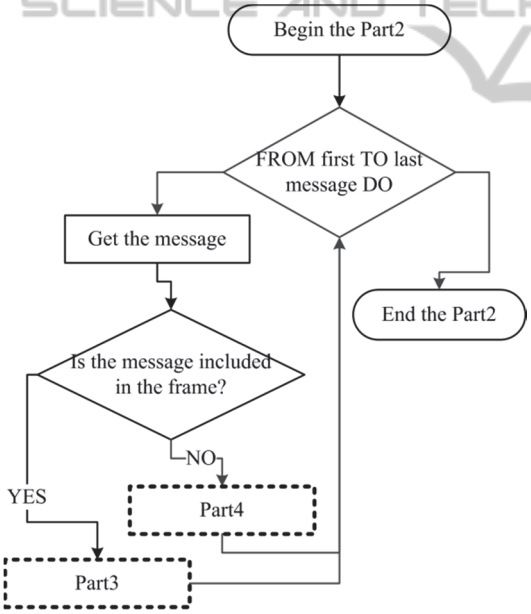

Figure 3: The second part of algorithm.

Figure 5 represents the flowchart Part4, where the message's name and the destination lifeline are taken for setting the functional features information. The chain of message invocations provides topological connection between cause and effect functional features. The destination of the message can be set as the object in the functional feature, as well as the message name can be set as object's action with some result of this action. When all information will be received, Part 4 of the algorithm finishes its work.

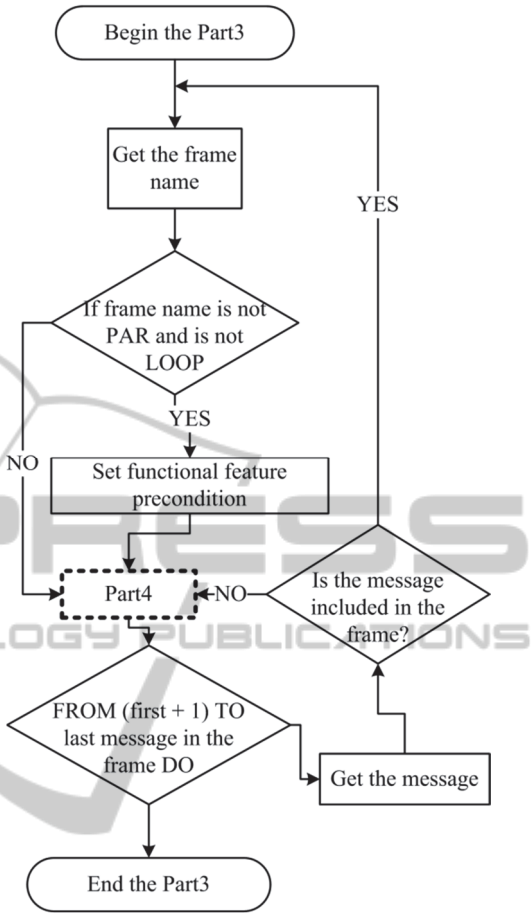

Figure 4: The third part of algorithm.

As the example the part of the UML sequence diagram are taken from (Ovchinnikova, et al., 2014) for providing the work of this algorithm. Figure 6 illustrates this example. After this algorithm realizing the following corteges are obtained:

- <1(Id), deleteBoard(A), null(R), Board(O), null(PrCond), null(PostCond), null(Pr), null(Ex), nul(Req), Board(Cl), deleteBoard():boolean(Op)> from the message deleteBoard();
- <2(Id), deleteSquare(A), null(R), Square(O), null(PrCond), null(PostCond), null(Pr), null(Ex), nul(Req), Square(Cl), deleteSquare():boolean(Op)> from the message deleteSquare() and this functional feature is invoked by the functional feature with id = 1;
- <3(Id), deleteDisc(A), null(R), Disc(O), If square is not empty(PrCond), null(PostCond), null(Pr), null(Ex), nul(Req), Disc(Cl), deleteDisc():boolean(Op)> from the message

deleteDisc() and this functional feature is invoked by the functional feature with id = 2.

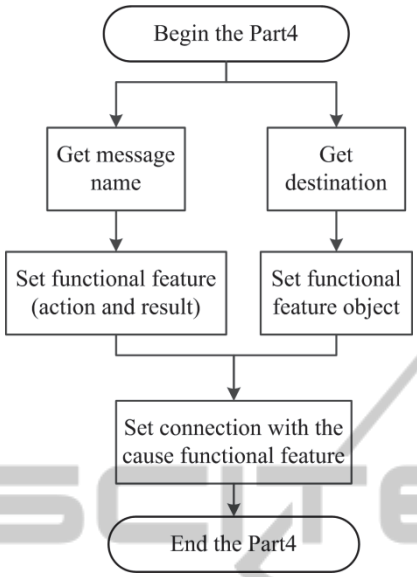

Figure 5: The fourth part of algorithm.

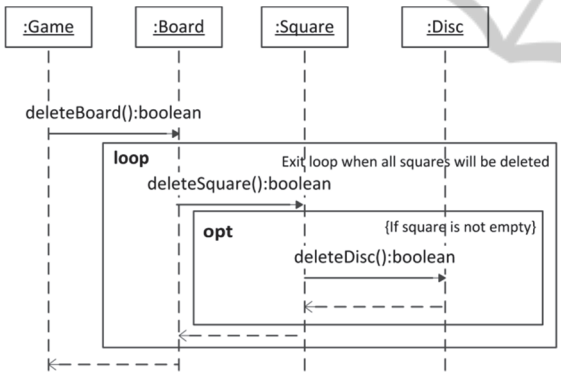

Figure 6: The example of the UML sequence diagram.

As well as the following two cause-and-effect relations are obtained: 1) the cause is the functional feature with id=1 and the effect is the functional feature with id=2; 2) the cause is the functional feature with id=2 and the effect is the functional feature with id=3.

This example shows that the semantic inconsistence exists between the functional feature of the *loop* and the UML sequence diagram frame *loop*. The functional feature loses information that all available squares must be deleted on the board. In case of composing the TFM from the textual description, the functional feature's name would indicate this information.

4 RELATED WORK

4.1 Model-driven Reverse Engineering

MDRE (Model-driven Reverse Engineering) can resolve such managerial problems as prediction of how many time need to be spend on the reverse engineering and the evaluation of the reverse engineering quality (Rugaber and Stirewalt, 2004).

MDRE uses two types of models, namely a program model and an application domain model Rugaber and Stirewalt, 2004). The program model provides the computed values of software system functions in a higher abstraction level than in the source code. The application domain model indicates domain concepts with their relationships and meanings, which are independent from the software system. These two types of models must have connections with each other.

As authors in Rugaber and Stirewalt, 2004) means, it is important to have the similar outputs from reversing reverse engineering. They defined two criteria, namely lucidity and thoroughness, for evaluation of the reversing reverse engineering. Results of the generated software system are similar to the original software system in case of lucidity. Domain concepts are connected to all the software system constructs in case of thoroughness.

MDRE can be applied using UML improvement tools, such as all UML diagrams and OCL (Object Constraint Language). It helps to create the more correct and full software system. The detailed example of MDRE is illustrated in (Rugaber and Stirewalt, 2004).

Authors in (Weijun, et al., 2009) also use MDRE and describe generation of OWL (Ontology Web Language) descriptions from UML models and WSDL (Web Services Descriptions Language) files.

The use of RE techniques, namely static and dynamic analysis, is suggested in (Favre, 2008) for generation of PSMs and PIMs from source code. Analysis of conformity of these transformations needs to be defined by formal specifications and metamodels. The hypothesis is that all information required by the software system can be taken from the source code. The PSMs and PIMs can be expressed by using UML and OCL. The OCL defines transformation between a source and a target metamodels in MOF (Meta Object Facility) terms. QVT (Query, View, Transformation) specification for metamodels transformation is presented as OCL contracts with names, a set of parameters, preconditions (assert relationships

between metaclasses, which belong to the source metamodel) and postconditions (after the transformation do something with the state of the models).

The transformation from the UML/OCL to the NEREUS language is described in (Favre, 2008). NEREUS language (Favre, 2005) is needed for certain expression of parts of the UML metamodels. All these transformations are described in detail in (Favre, 2008) and (Favre, et al., 2009). As well the detailed information of reverse engineering and MDA transformations are provided in (Favre, 2010) and (Favre, 2012).

4.2 Architecture Driven Modernization

ADM TF (Architecture Driven Modernization Task Force) has the following goals (Newcomb and Mansurov, 2005): successful modernization of existing software systems, revival of existing software systems and taking the existing software system more agile. According to (Newcomb and Mansurov, 2005) ADM creates modernized software systems from legacy systems using model transformations. ADM standards roadmap is:

- KDM (Knowledge Discovery Meta-Model) – it is an initial metamodel that shows the behavior, structure and data of the software system. As well it represents the software system above the procedural level that is why it can be used by multiple languages. It is a model of the semantic graph model;
- ASTM (Abstract Syntax Tree Meta-Model) – it is built upon the KDM and needed for representing the software system below the procedural level. It is a model of the syntax tree model;
- Pattern Recognition – it is needed for examination of the structure of metadata for getting anti-patterns and patterns if the existing software system. They can be used for definition of the transformation opportunities and requirements;
- SMM (Structured Metrics Package) – it is needed for getting metrics from the KDM, which transmit architectural, technical and functional problems for the data;
- Visualization – views of models in the form of graphs, charts or others realizations;
- Refactoring – improving, modularizing the existing software system, as well as getting the model-driven views of the software;
- Transformation – it defines mappings between the ASTM or the KDM and the target metamodel.

Author in (Khusidman, 2008) describes the ADM horseshoe, which consists of the following processes:

- Formal Transformation – getting the KDM from the source code;
- Abstraction Level Transformation – creating business rules, processes and vocabulary in SBVR (Semantic of Business Vocabulary and Rules) and BPMN (Business Process Model and Notation) from the KDM;
- Enhancement Transformation – upgrading BPMN and SBVR by adding business requirements;
- Abstraction Level Transformation – designing UML diagrams;
- Formal Transformation – generating the target code from the UML diagrams.

Authors in (Sadovykh, et al., 2009) provide methodology for migration of existing software system in C++ to the new software system in Java. They show transformation requirements (their case study is associated with all layers of MDA and ADM), analyze C++ and Java source code, as well as show differences and similarities of these programming languages. Additionally, they give information about testing the process of getting the legacy system for understanding where and what needs to be added or changed.

4.3 Knowledge Discovery Meta-model

The tutorial about KDM structure is available in (Mansurov, 2005). It can help in getting necessary knowledge from the KDM needed for creation of business models. The created business model needs to be understandable and improvable, as well there need to be a possibility to transform, modernize and maintain it. According to (Vasilecas and Normantas, 2011) the KDM can represent four layers of abstractions:

- Abstraction layer – it represents structure, build and conceptuality of the existing software system;
- Runtime resource layer - it represents the user interface, events, as well behavior of the existing software system;
- Program element layer – it represents the software system elements and their relationships;
- KDM infrastructure layer – it represents source files, configuration files and other files of the existing software system.

As well authors in (Vasilecas and Normantas, 2011) describe the main process of business rules derivation from the KDM. They use the GUIDE Business Rules Project (Group, 2001) formalizations for identifying the business rules. Four business rule categories are defined in these formalizations: business terms, facts, constraints and derivations in these formalizations. As well three steps need to be done for derivation the complete business rules. The first two are the derivation of terms and facts, and the third is the detection of structural derivations and assertions. After all these steps the system analysts have to check and analyze all obtained information.

In (Normantas and Vasilecas, 2012) authors represent their approach for business rule detection in the existing software system. They demonstrate three phases of this approach: preliminary study, knowledge extraction, and business logic abstraction. Each of these phases has additional steps for getting the necessary information. As well they represent their own example with obtained results that illustrates these phases.

Authors in (Perez-Castillo, et al., 2010) use QVT transformations for getting the BPMN models from the KDM model.

5 CONCLUSIONS

Modeling and analysis of the software system is the main part of the software system development. It is necessary for not losing the important functions, parameters and logic of the software system.

The transformation algorithm is suggested in this research. Certainly, it is necessary to check the algorithm in practice in order to validate its correctness and possibility to be implemented in code. As well some application information can be lost during transformation, such as parameters sending with messages. Maybe, the 11-tuple needs to be replenished for not losing such information.

If the message frames will be used in the UML sequence diagrams, additional check of logic needs to be provided, because TFM functional features are to be connected correspondingly the order of message invocations. As well the check of the connectedness of all created parts of TFM needs to be provided. For this task, the union operation can be used. Thus, all repeated elements will be deleted, except the original one, with which all connections from all other parts of the TFM will be set.

The information about others approaches also has been considered in the related work. This information will be used in further work for refining our approach. For example, the KDM can be considered as an alternative to UML diagrams.

The future research direction is related to analysis and possible modification of the TFM metamodel. It is needed for the transformation from the UML sequence diagrams to the TFM using QVT that will implement the algorithm discussed in this research. The current version of the TFM metamodel is implemented in the IDM toolset that is discussed in (Slihte and Osis, 2014). In order to obtain the necessary TFM, the QVTo (QVT Operational) transformation is planned to be implemented in the future, using modified or current version of the TFM metamodel. There is high possibility that this TFM will not be complete, because some information can be lost during transformation. The reason is that RE tools may work incorrectly.

Another point is that it is not clear what number and types of UML diagrams will be sufficient for generating a complete TFM. By now we assume that a set of sequence diagrams is sufficient for this task.

REFERENCES

Asnina, E. & Osis, J., 2010. Computation Independent Models: Bridging Problem and Solution Domains. In: *Proceedings of the 2nd InternationalWorkshop on Model-Driven Architecture and Modeling Theory-Driven Development (MDA & MTDD 2010), in conjunction with ENASE 2010.* Lisbon: SciTePress, pp. 23-32.

Asnina, E. & Osis, J., 2011c. Topological Functioning Model as a CIM-Business Model. In: *Model-Driven Domain Analysis and Software Development: Architectures and Functions.* Hershey - New York: IGI Global, pp. 40 - 64.

Donins, U., 2012. *Topological Unified Modeling Language: Development and Application. Ph.D. Thesis.,* Riga: RTU.

Donins, U. et al., 2011. Towards the Refinement of Topological Class Diagram as a Platform Independent Model. In: *Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development (MDA & MDSD 2011).* Lisbon: SciTePress, pp. 79-88.

Favre, L., 2005. Foundations for MDA-based Forward Engineering. *Journal of Object Technology,* 4(1), pp. 129-154.

Favre, L., 2008. Formalizing MDA-Based Reverse Engineering Processes. *Software Engineering Research, Management and Application,* pp. 153-160.

Favre, L., 2010. Reverse Engineering and MDA: An Introduction. In: *Model Driven Architecture for*

Reverse Engineering Technologies: Strategic Directions and System Evolution. Hershey - New York: IGI Global, pp. 1-14.

Favre, L., 2012. MDA-Based Reverse Engineering. In: D. A. Telea, ed. *Reverse Engineering - Recent Advances and Applications.* Rijieka: InTech.

Favre, L., Martinez, L. & Pereira, C., 2009. MDA-Based Reverse Engineering of Object Oriented Code. In: *Enterprise, Business-Process and Information Systems Modeling, LNBIP.* Berlin: Springer Berlin Heidelberg, pp. 251-263.

Group, B. R., 2001. *Defining Business Rules - What Are They Really?*. [Online] Available at: http://www.businessrulesgroup.org/first_paper/br01c0.htm [Accessed 27 January 2015].

Khusidman, V., 2008. *ADM Transformation.* [Online] Available at: http://www.omg.org/adm/ADMTransformartionv4.pdf [Accessed 25 January 2015].

Mansurov, N., 2005. *Knowledge Discovery Meta-model: Tutorial.* [Online] Available at: http://www.omg.org/news/meetings/workshops/ADM_2005_Proceedings_FINAL/T-2_Mansurov.pdf [Accessed 25 January 2015].

Newcomb, P. & Mansurov, N., 2005. *Architecture-Driven Modernization workshop.* [Online] Available at: http://www.omg.org/news/meetings/workshops/adm-2005.htm#tutorials [Accessed 25 January 2015].

Normantas, K. & Vasilecas, O., 2012. *Extracting Business Rules from Existing Enterprise Software System.* Berlin: Springer Berlin Heidelberg.

OMG, 2011. *OMG Unified Modeling Language. Version 2.4.1.* [Online] Available at: http://www.omg.org/spec/UML/2.4.1/ [Accessed 29 January 2015].

Osis, J. & Asnina, E., 2011a. Is Modeling a Treatment for the Weakness of Software Engineering?. In: *Model-Driven Domain Analysis and Software Development: Architectures and Functions.* Hershey - New York: IGI Global, pp. 1-14.

Osis, J. & Asnina, E., 2011b. Topological Modeling for Model-Driven Domain Analysis and Software Development: Functions and Architectures. In: *Model-Driven Domain Analysis and Software Development: Architectures and Functions.* Hershey - New York: IGI Global, pp. 15-39.

Osis, J. & Asnina, E., 2011d. Derivation of Use Cases from the Topological Computation Independent Business Model. In: *Model-Driven Domain Analysis and Software Development: Architectures and Functions.* Hershey, USA: IGI Global, pp. 65 -89.

Osis, J. & Asnina, E., 2011. *Model-Driven Domain Analysis and Software Development: Architectures and Functions.* Hershey - New York: IGI Global.

Osis, J., Asnina, E. & Grave, A., 2007. MDA Oriented Computation Independent Modeling of the Problem Domain. In: *Proceedings of the 2nd International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2007).* Barselona: INSTICC Press, pp. 66-71.

Osis, J., Asnina, E. & Grave, A., 2008. Formal Problem Domain Modeling within MDA. In: *Software and Data Technologies, Communications in Computer and Information Science.* Berlin: Springer-Verlag Berlin Heidelberg, pp. 387-398.

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

Ovchinnikova, V. & Asnina, E., 2014b. Overview of Software Tools for Obtaining UML Class Diagrams and Sequence Diagrams from Source Code within TFM4MDA. *Baltic Journal of Modern Computing,* 2(4), pp. 260 - 271.

Ovchinnikova, V., Asnina, E. & Garcia-Diaz, V., 2014. Relationships between UML Sequence Diagrams and the Topological Functioning Model for Backward Transformation. *Applied Computer Systems,* Volume 16, pp. 43-52.

Perez-Castillo, R., Garcia-Rodriguez de Guzman, I. & Piattini, M., 2010. Implementing Business Process Recovery Patterns through QVT Transformations. In: *Theory and Practice of Model Transformations, LNCS.* Berlin: Springer Berlin Heidelberg, pp. 168-183.

Rugaber, S. & Stirewalt, K., 2004. Model-Driven Reverse Engineering. *IEEE Software,* 21(4), pp. 45-53.

Sadovykh, A. et al., 2009. Architecture Driven Modernization in Practice – Study Results. In: *14th IEEE International Conference on Engineering of Complex Computer Systems.* Potsdam: IEEE.

Slihte, A. & Osis, J., 2014. The Integrated Domain Modeling: A Case Study. In: *Databases and Information Systems: Proceedings of the 11th International Baltic Conference (DB&IS 2014), Estonia, Tallinn, 8-11 June, 2014.* Tallinn: Tallinn University of Technology Press, pp. 465-470.

Vasilecas, O. & Normantas, K., 2011. *Deriving Business Rules from the Models of Existing Information Systems.* New York: ACM.

Weijun, S., Shixian, L., Defen, Z. & YuQing, Y., 2009. A Model-Driven Reverse Engineering Approach for Semantic Web Services Composition. *Software Engineering,* Volume 3, pp. 101-105.