DE GRUYTER OPEN

Applied Computer Systems

doi: 10.1515/acss-2014-0012

2014/16

Relationships between UML Sequence Diagrams and the Topological Functioning Model for Backward Transformation

Viktoria Ovchinnikova[1], Erika Asnina[2], Vicente García-Díaz[3],
[1–2]*Riga Technical University, Latvia*, [3]*University of Oviedo, Spain*

***Abstract* – The software system needs to be analyzed and designed before the program code is written. A Computation Independent Model (CIM) and a Platform Independent Model/ Platform Specific Model (PIM/PSM) from Model-Driven Architecture (MDA) will be partially considered in this paper. A Topological Functioning Model (TFM) will be considered as a formal CIM, and UML sequence diagrams – as a behavioral PIM/PSM of the software system. The paper presents a short overview of the TFM and sequence diagrams with their constructs, as well as the example of transformation from the sequence diagrams to the TFM.**

***Keywords* – Model transformation, model-driven architecture, topological functioning model, UML sequence diagrams.**

I. Introduction

Model-Driven Architecture (MDA) is one of the means for modeling and analyzing software systems. MDA consists of four models: a Computation Independent Model (CIM), a Platform Independent Model (PIM), a Platform Specific Model (PSM) and an Implementation Specific Model (ISM). The MDA models are described in detail in [1]. CIM and PIM/PSM will be partially considered in this paper. A Topological Functioning Model (TFM) specifies system functioning and will be considered as a formal CIM [2], [3], and a Unified Modeling Language (UML) sequence diagram – as a PIM/PSM of software system behavior.

Topological Functioning Modeling for Model-Driven Architecture (TFM4MDA) is a software development method for modeling and analyzing the software system within the MDA, where the CIM may contain 3 main parts [3]:

- a knowledge model that provides experts' visions of the enterprise operation with focus on its organizational structure and business specifics,
- a business model that is focused on the business goals and business scope such as resources, facts, rules and so on, and
- business requirements for the system that are set by business people.

In TFM4MDA, the knowledge model describes a problem domain (TFM of the system "as is") and the business requirements describe a solution domain (TFM of the system "to be"), specifying software system requirements. The business model represents 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, and there is a clear boundary between them.

Manual acquisition of the UML class diagram, sequence diagrams, use case diagram, object and activity diagram from the TFM is described and represented in [5], [6] and [7]. The UML class diagrams and object diagrams can be obtained from the graph of domain objects, which have been received from the TFM. The UML sequence diagrams and activity diagrams can be obtained from the posed goals (define the actions parts of the software system) in the TFM. The UML use case diagrams can represent the TFM either partially or fully depending on the needs of software systems analysts.

According to the early studies [5], [6], [7], the TFM can also be obtained from these diagrams by backward steps. The TFM obtained as a result of transformation of the UML sequence diagrams may contain less information than it is necessary for further backward transformation as well as for other diagrams that could be obtained from the TFM during forward transformations.

Reverse Engineering (RE) within TFM4MDA is necessary, when some changes are needed in the software system (for example, the legacy system needs to be migrated or integrated on the new technological platform or clients requirements were changed) that is described in [8]. In this research, RE from PSM/PIM (from the UML sequence diagram) to CIM (to TFM) within the topological functioning modeling is considered, and an example of this transformation is provided. Research results are necessary for further work, when full RE from ISM to PSM/PIM to CIM will be created.

The goal is to obtain theoretical and practical knowledge of transformation mappings between the TFM and sequence diagrams. For this purpose, it is necessary to find out the semantic similarities and differences of the sequence diagram and TFM constructs, to define the transformation mappings, and to demonstrate the transformation from the sequence diagrams to the TFM by the example.

The paper is structured as follows. Section II shortly describes the TFM and its constructs. Section III discusses the relationships between the TFM and the UML sequence diagrams. Section IV illustrates the example of this transformation. Section V contains information about the related work and Section VI – the conclusions and future work.

II. Topological Functioning Model and Its Constructs

The TFM specifies static and dynamic characteristics of the system; therefore, for creation of the TFM structure and behavior diagrams are needed.

The TFM has topological and functioning properties. The topological properties have the mathematical background and specify relationships among objects, i.e., specify cause-and-effect dependencies between software system functional characteristics. The topological properties include the following concepts, described in detail in [9]:

- neighborhood;
- connectedness;
- continuous mapping;
- closure.

The functioning properties are based on the principles of software system theory and include the following concepts, described in detail in [9]:

- inputs and outputs;
- cause-and-effect relations;
- cycle structure.

The TFM must have at least one main functioning cycle that indicates cause-and-effect relations among vitally important functional features of the system, as well as inputs from the external environment and outputs (reactions) to the external environment.

The 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 [4]. The functional feature characterizes the system that is needed for system goal achievement. It is a unique 11-tuple <Id, A, R, O, PrCond, PostCond, Pr, Ex, Req, Cl, Op> [4], [7]:

- Id – an identifier of the functional feature that is used in the topological graph G(X,Q);
- A – an object action (functional feature nature) in the functional feature;
- R – a result of the object action;
- O – an object that is used in this action or the result of this action;
- **PrCond** – preconditions that need to be completed before functioning feature execution;
- **PostCond** – post-conditions that need to be completed after functioning feature execution;
- **Pr** – providers that are responsible entities, which provide an action with some objects;
- **Ex** – executers that are responsible entities, which execute a concrete action;
- **Req** – requirements, which are realized by this functional feature;
- Cl – class that represents this functional feature;
- Op – operation that is responsible for action execution defined by a functional feature.

III. Relationships between the Topological Functioning Model and the UML Sequence Diagram

The UML sequence diagrams can specify interactions between objects, showing the order of these interactions and, hence, the behavior of the software system. The UML sequence diagram creation is described in detail in [11]. The UML sequence diagram constructs are considered in this section. Semantic similarities and differences between the sequence diagram and TFM constructs are researched to define the mapping of transformation.

A. The Semantic Similarities and Differences between Constructs of the Sequence Diagram and TFM

As written above, the TFM functional feature has the following parts: identifier (Id), actor (A), result (R), object (O), preconditions (PrCond), post-conditions (PostCond), provider (Pr), executer (Ex), requirements (Req), class (Cl) and operation (Op). The UML sequence diagrams consist of the active objects and their interaction with each other. The UML sequence diagram has [10]:

- Outside actor that is not part of the software system;
- Lifeline that represents an active object (role) in the interaction;
- Messages that are sent from one lifeline to another;
- Frames or fragments of the diagrams that are used for supporting the looping and conditional constructs.

Table I, Table II, Table III and Table IV show the semantic similarities and differences of the UML sequence diagram and TFM constructs in general. The UML sequence diagram lifeline types are considered in Table II, message types are considered in Table III and frames of the diagram – in Table IV. The tables do not include all the elements from the TFM constructs, because in the UML sequence diagrams all constructs have an identifier (Id) and, in this context, a class (Cl) and an operation (Op) are the same as an object (O) and object action (A), accordingly. Table II and III do not contain a provider (Pr) and an executer (Ex), because they represent only outside actors (systems or users). Table IV includes such TFM constructs (a kind of element configuration) as a cycle and a cause-and-effect relation, because the UML sequence diagrams have also the cycle as frame "loop", and some frames can establish a cause-and-effect relation between active objects.

TABLE I

The Matrix of the Semantic Similarities and Differences between Constructs of the Sequence Diagram and TFM

The UML sequence diagram constructs	TFM constructs						
	A	R	O	PrCond	PostCond	Pr	Ex
Outside actor			1			1	1
Lifelines			1				
Messages	1	1		1			
Frames				1			

The lifeline can be divided into the three types [12]:

- Lifeline <Boundary> is an active object for representing the system boundary (e.g., a user or system interface);
- Lifeline <Control> is an active object for modeling flows of control in behavior of the system;
- Lifeline <Entity> is an active object for representing some information that is processed by the system.

TABLE II

THE MATRIX OF THE SEMANTIC SIMILARITIES AND DIFFERENCES BETWEEN CONSTRUCTS OF THE SEQUENCE DIAGRAM LIFELINE TYPES AND TFM

The UML sequence diagram lifeline types	TFM constructs				
	A	R	O	PrCond	PostCond
Lifeline <Boundary>			1		
Lifeline <Control>			1		
Lifeline <Entity>			1		

Messages can be divided into the following types [13]:

- A synchronous message is a message that is sent from one lifeline to another and waits for response before being executed;
- An asynchronous message is a message that is sent from one lifeline to another and starts its action with no waiting for response;
- An object creation message is a message that is sent from one lifeline to another to create itself;
- An object deletion message is a message that is sent from one lifeline to another to terminate itself;
- A reply message is a message of response that is sent from one lifeline to another;
- A lost message is a message, where the receiving event is unknown;
- A found message is a message, where the sending event is unknown;
- An unknown message is a message, where the sending and receiving events are absent.

TABLE III

THE MATRIX OF THE SEMANTIC SIMILARITIES AND DIFFERENCES BETWEEN CONSTRUCTS OF THE SEQUENCE DIAGRAM MESSAGE TYPES AND TFM

The UML sequence diagram message types	TFM constructs				
	A	R	O	PrCond	PostCond
A synchronous message	1	1			
An asynchronous message	1	1			
An object creation message	1	1			
An object deletion message	1	1			
A reply message	1	1			
A lost message	1				
A found message	1				
An unknown message	1				

There are the following frames in the sequence diagram [13]:

- Alt – an alternative designates a fragment, in which one of some possible operands, expressed in the guards (true or false), will be chosen;
- Opt – an option designates a fragment, in which operand happens if its guard is true;
- Par – parallel designates a fragment, in which some operands execute in parallel;
- Loop – a loop designates a fragment that represents a loop while fragment guard is true (can also be loop(*N*) for looping fragment *N* times);
- Critical – a critical region designates a fragment, in which only one thread can run;
- Neg – negative designates a fragment, in which traces are defined to be incorrect;
- Assert – an assertion designates a fragment, in which if any sequence is not shown as an operand of the assertion, then it is invalid;
- Strict – a fragment with a strict sequencing between the behaviors of the operands;
- Seq – a fragment with a weak sequencing between the behaviors of the operands;
- Ignore(*a*, *b*) designates a fragment that shows that messages a and b will be ignored;
- Consider(*a*, *b*) designates a fragment that shows that messages a and b will be important.

TABLE IV

THE MATRIX OF THE SEMANTIC SIMILARITIES AND DIFFERENCES BETWEEN CONSTRUCTS OF THE SEQUENCE DIAGRAM FRAME TYPES AND TFM

The UML sequence diagram frame types	TFM constructs			
	Sequence of Cause and Effect	Cycle	PrCond	PostCond
Alt	1		1	
Opt	1		1	
Par	1			
Loop		1		
Critical	1		1	
Neg	1		1	
Assert	1		1	
Strict	1			
Seq	1			
Ignore	1		1	
Consider	1		1	

B. *The Mappings of Transformation*

Uldis Donins in his Doctoral Thesis [7] has provided and described the transformation from the TFM to the UML sequence diagrams. Fig. 1 shows the schematic representation of system structure creation.

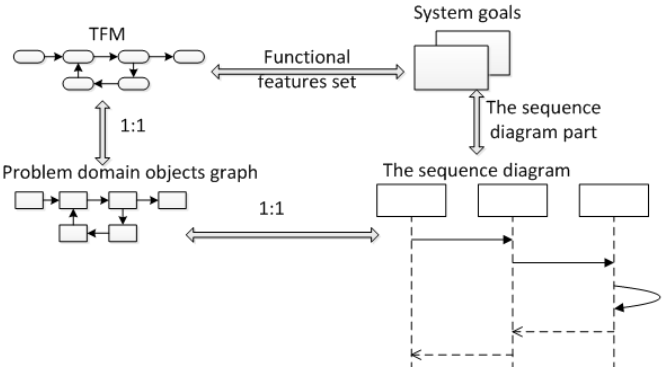

Fig. 1. The schema of software system structure creation.

The transformation from the TFM to the problem domain object graph is one-to-one, that means that all associations and the model structure need to be the same. The transformation

from the problem domain object graph to the UML sequence diagram is also one-to-one. The functional features can be chosen from the TFM and can describe system goals. One system goal can be described by one sequence diagram.

In the TFM, vertices are functional features identified as <action, result, object>, but in the UML sequence diagrams vertices are class objects. When transformation from the TFM to the UML sequence diagrams was performed, the conversion from 3 functional elements (action, result, object) to one structure element (class object) was created, as well as its behavior was obtained from the functional feature action and result.

The class objects, messages and frames were taken from the UML sequence diagrams, and the TFM functional feature <action (message), result (from message (e.g. return value)), object (class object)> was formed from these three elements in backward transformation.

These mappings can help to get TFM from the sequence diagrams, for this purpose backward transformation needs to be done. The systems goals and the TFM can be created and defined from the UML sequence diagrams. The problem domain object graph will not be used in this backward transformation, because the transformation is one-to-one from the UML sequence diagram to the problem domain object graph and from the problem domain objects to the TFM. This means that the transformation from the UML sequence diagram to the TFM can also be one-to-one in case of the systems with the predictable sequences of actions. Such a kind of the system is considered in this research.

IV. The Example of the Transformation

The game "Reversi", whose rules in detail are described in [14], is taken as an illustrative example of the defined transformation. The UML sequence diagrams created for this game as well as their transformation to the TFM are described in this section.

A. *The Game "Reversi" Rules*

The game "Reversi" rules are the following [14]:

- There are two players in the game. The first player plays with white discs, the second one with black ones;
- The board has 64 squares and is initialized with two white and two black discs in the middle of the board;
- The disc can change it color in the game, when the turn is done;
- At each turn, a player needs to place a disc on the allowed place from where the opponent's discs can be covered by diagonal, horizontal or vertical. The covered opponent's disc changes its color to a player's (who made the turn) disc color;
- The game is finished when both players cannot do the turn or when all squares are filled with discs or when only one disc color remains on the board. The player is a winner if his disc count is greater than opponent's disc count on the board.

B. *The UML Sequence Diagrams of the Game "Reversi"*

The UML sequence diagrams were created manually using Microsoft Visio tool. The system goals were achieved from the game rule description in [14]. The system goals are the following:

- Start the game;
- Finish the game;
- Do turn by a user player;
- Do turn by a computer player.

Fig. 2–5 show achievement of the system goals "Start the game", "Finish the game", "Do turn by a user player", "Do turn by a computer player", accordingly.

Fig. 2 contains outside actor "User", from which a boundary lifeline "UserPlayer" (the interface of the system for a user) is created. Lifelines "Game" and "Board" are the control lifelines, because there are game rules in class "Game" and the manipulation with the board in class "Board". Lifelines "Square" and "Disc" are entity lifelines, because they contain information only about their status. There are synchronous, object creation and reply messages and frames "loop" and "opt" in Fig. 2. Frame "loop" is necessary here for initializing boards. The board is not empty at the beginning of the game – four discs (2 white and 2 black) need to be placed in the middle of the board (the board size is 8x8). All squares need to be initialized (60 other squares are empty), but not all discs need to be created at the beginning of the game that is why frame "opt" is here.

Fig. 3 contains the same lifelines and messages (excluding object creation messages), it also contains frames "loop" and "opt". Frame "loop" is responsible for deletion of all squares in the board (and the board itself) and frame "opt" is responsible for deletion of existing discs in squares.

Fig. 4 contains the same lifelines and messages; additionally, it contains frames "loop", "alt" and "opt". The first frame "loop" is responsible for finding the possible turn for the player. If the square is empty (there is no disc), then a disc can be got. That is why here frame "opt" is needed. Frames "alt" are divided into the two parts:

- when the turn is possible, a player can do the turn;
- when the turn is impossible, a player skips the turn.

Frame "loop" in the first frame "alt" is responsible for all necessary disc color changes. Frame "opt" in the second frame "alt" will be executed if the game is finished. Frame "loop" in this frame "opt" is needed for player's disc counts.

Fig. 5 contains the same lifelines and messages, as well as it contains frames "loop", "alt" and "opt". Fig. 5 is similar to Fig. 4, but a computer player creates the board's clone and chooses the best turn of all possible turns, using this board's clone.

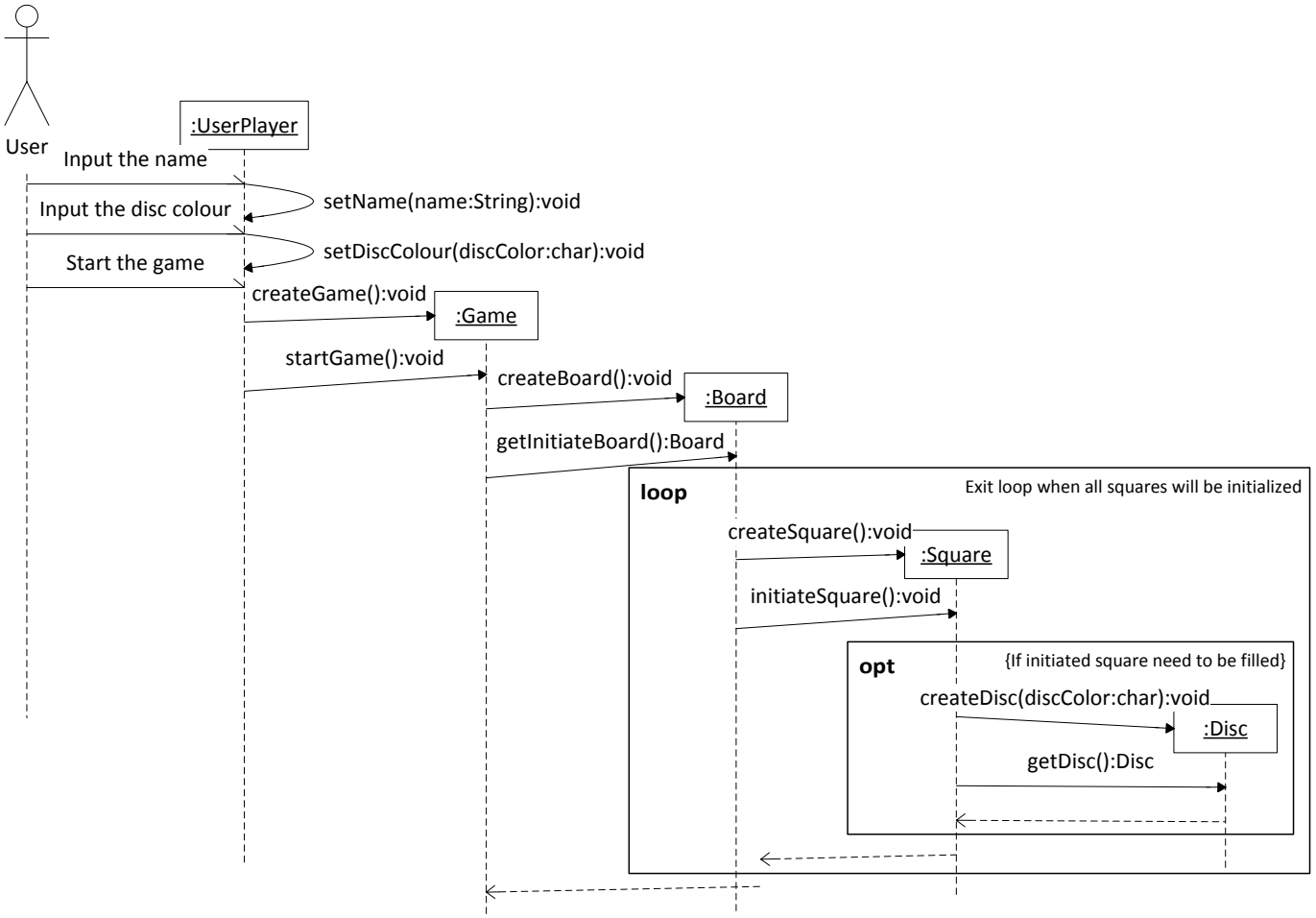

Fig. 2. The UML sequence diagram with the system goal "Start the game".

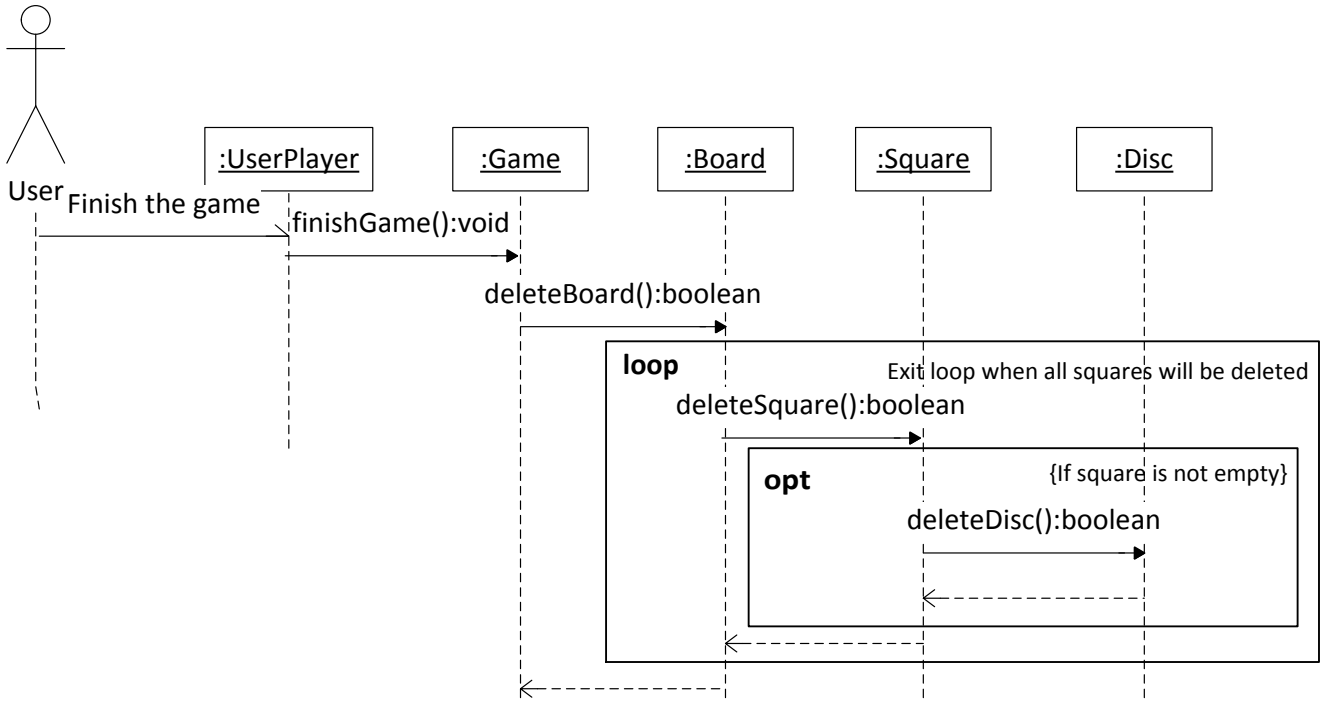

Fig. 3. The UML sequence diagram with the system goal "Finish the game".

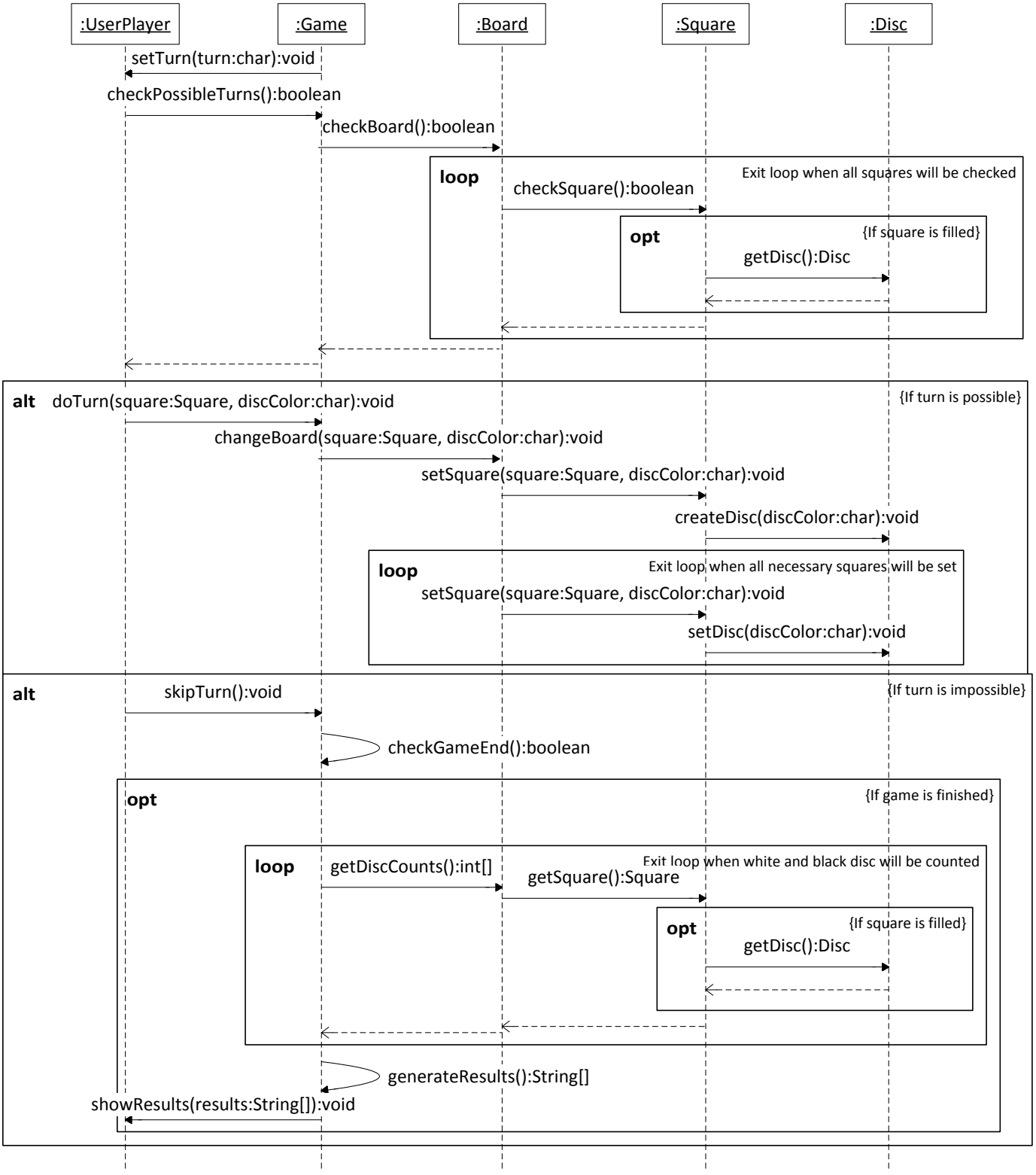

Fig. 4. The UML sequence diagram with the system goal "Do turn by a user player".

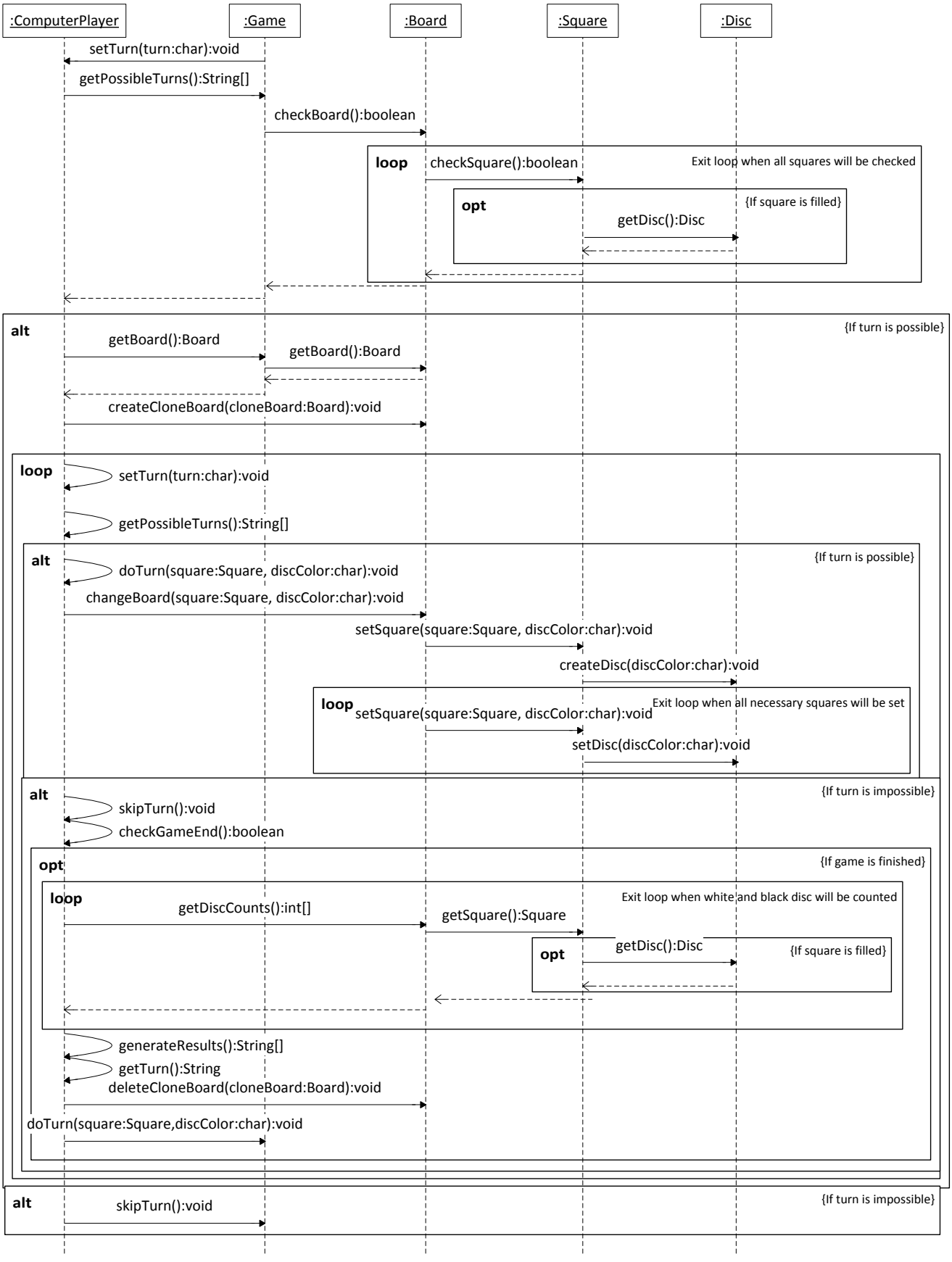

Fig. 5. The UML sequence diagram with the system goal "Do turn by a computer player".

TABLE V

CORRESPONDENCE OF THE FUNCTIONAL FEATURES, OBJECTS AND OPERATIONS

Functional features	Operation in the sequence diagram	Object, who does it	Outside/inside object
2.Entering the name	Input the name	User	Outside
3.Assigning the name	setName(name:String):void	UserPlayer	Inside
4.Entering the disc color to a user player	Input the disc colour	User	Outside
5.Assigning the disc color to a user player	setDiscColour(discColor:char):void	UserPlayer	Inside
6.Launching the game by a user	Start the game	User	Outside
7.Creating the game	createGame():void	Game	Inside
8.Starting the game	startGame():void	Game	Inside
9.Creating the board	createBoard():void	Board	Inside
10. Initializing of a board	getInitiateBoard():Board	Board	Inside
11.Creating the square	createSquare():void	Square	Inside
12.Initializing of a square	initiateSquare():void	Square	Inside
13.Creating the disc	createDisc(discColor:char):void	Disc	Inside
14.Getting the disc	getDisc():Disc	Disc	Inside
15.Exiting from the game by a user	Finish the game	User	Outside
16.Finishing the game	finishGame():void	Game	Inside
17.Deleting the board	deleteBoard():boolean	Board	Inside
18.Deleting the square	deleteSquare():boolean	Square	Inside
19.Deleting the disc	deleteDisc():boolean	Disc	Inside
20.Setting the turn by a user player	setTurn(turn:char):void	UserPlayer	Inside
20a.Setting the turn by a computer player	setTurn(turn:char):void	ComputerPlayer	Inside
21.Checking the possible turns	checkPossibleTurns():boolean	Game	Inside
22.Checking the board	checkBoards():boolean	Board	Inside
23.Checking the square	checkSquare():boolean	Square	Inside
24.Doing the turn by a game	doTurn(square:Square,discColor:char):void	Game	Inside
24a.Doing the turn by a computer player	doTurn(square:Square,discColor:char):void	ComputerPlayer	Inside
25.Changing the board	changeBoard(square:Square,discColor:char):void	Board	Inside
26.Setting the square	setSquare(square:Square,discColor:char):void	Square	Inside
28.Setting the disc	setDisc(discColor:char):void	Disc	Inside
29.Skipping the turn by a game	skipTurn():void	Game	Inside
29a.Skipping the turn by a computer player	skipTurn():void	ComputerPlayer	Inside
30.Checking the game end	checkGameEnd():boolean	Game	Inside
30a.Checking the game end by a computer player	checkGameEnd():boolean	ComputerPlayer	Inside
31.Getting the disc count in a board	getDiscCount():int	Board	Inside
32.Getting the square	getSquare():Square	Square	Inside
33.Generating the results in a game	generateResults():String[]	Game	Inside
33a.Generating the results by a computer player	generateResults():String[]	ComputerPlayer	Inside
34.Presenting the results	showResults(results:String[]):void	UserPlayer	Inside
35.Getting the possible turns from a game	getPossibleTurns():String[]	Game	Inside
35a.Getting the possible turns by a computer player	getPossibleTurns():String[]	ComputerPlayer	Inside
36.Getting the board from a game	getBoard():Board	Game	Inside
36a.Getting the board from a board	getBoard():Board	Board	Inside
37.Creating the board clone	createCloneBoard(board:Board):void	Board	Inside
38.Getting the turn by a computer player	getTurn():String	ComputerPlayer	Inside
39.Deleting the board clone	deleteCloneBoard(board:Board):void	Board	Inside

C. *The Transformation to the TFM*

Table V shows functional features, which compose the TFM created and operation names from the created UML sequence diagrams. Objects, which do this operation, are also presented, as well as these objects belonging to the system (outside or inside objects). Table VI shows the preconditions of the TFM functional features.

Fig. 6 represents the created TFM from the UML sequence diagrams.

The cycle structures in the TFM are created from frames "loop" in the UML sequence diagram, for example, the cycle that contains functional features 11-12-13-14-11 is responsible for the process of square creation.

Additionally, the cause-and-effect relations in the TFM are represented as frames "alt" and "opt" in the UML sequence diagrams. Frames "alt" and the "opt" are represented as the logical relations between the cause-and-effect relations in the TFM, for example, after functional feature 21 (Checking the possible turn in a game), next will be functional features 24 (Doing the turn in a game) or 29 (Skipping the turn in a game), – it depends on the possibility of a turn. The example with frame "opt" is the functional feature 30 (Checking the end of a game) after which functional feature 31 (Getting the disc count) will be executed if the game is finished; otherwise functional feature 20 (Setting the turn) will be performed.

TABLE VI

THE PRECONDITIONS OF THE TFM FUNCTIONAL FEATURES

The TFM functional feature	Preconditions
11	If all squares were not created
13	If initiated square needs to be filled
18	If all squares were not deleted
19	If square is not empty
23	If all squares were not checked
14	If square is filled
20, 20a	If turn is possible
26	If all necessary squares were not set
29, 29a	If turn is impossible
31	If a game is finished and If all white and black discs were not counted

V. RELATED WORK

There are other model transformations, too, such as Architecture Driven Modernization (ADM) implemented by Object Management Group (OMG) and Model Driven Reverse Engineering (MDRE). The ADM is a useful approach to the migration or integration of projects applied to the following processes [15]:

- The reverse modeling of the software system full source code;
- The model transformation from the source architectural model to the target model;
- The code generation (include the architectural model generation and the algorithmic source code) conversion to a programming language.

The MDRE is designed to overcome some problems, such as difficulty of prediction of how much time is needed for the reverse engineering, as well as difficulty of evaluation of the reverse engineering quality [16]. It uses the application domain model (expresses the domain concepts and their relationships and meanings) and the program model (provides the high-level representation of the software system functions). Examples of MDRE are presented in [16].

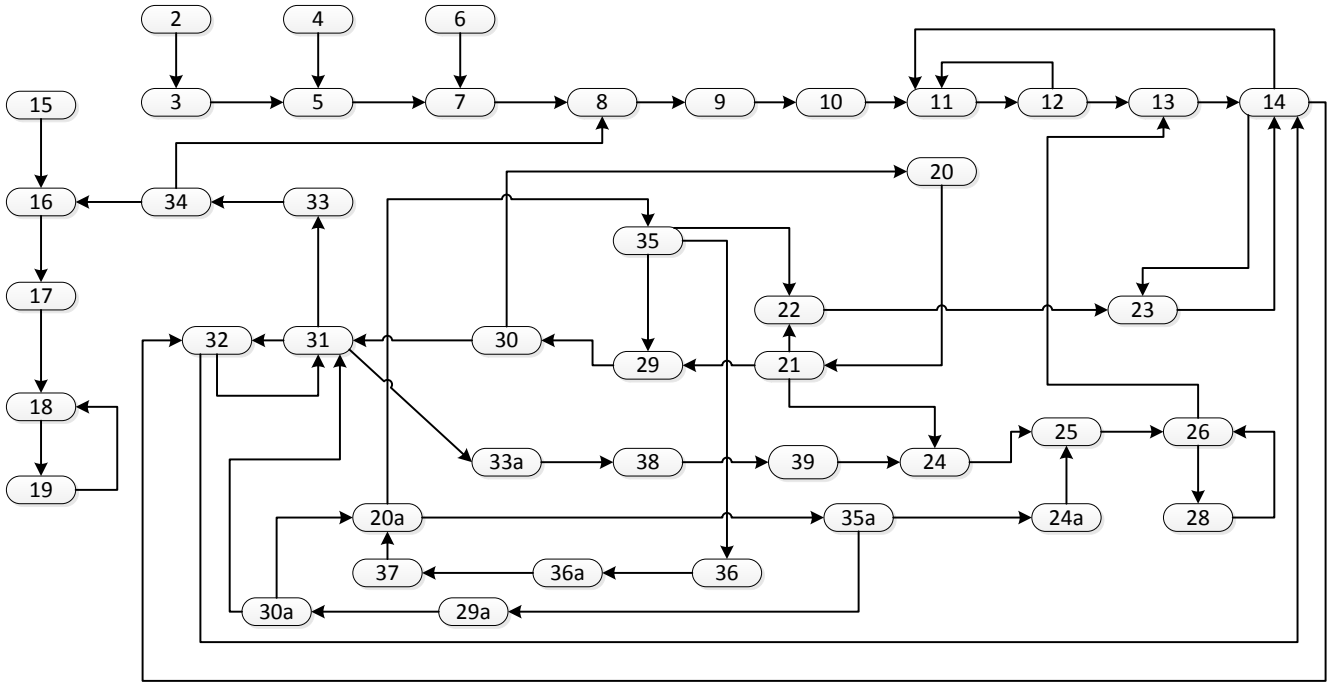

Fig. 6. The obtained TFM.

VI. Conclusion

It is important to analyze the software system before starting to write the source code for this system. In addition, it is necessary to design this software system by some business models for deep understanding of the software system and its creation by the posed rules in future. The reverse engineering helps to see how the business model is modified after the changes in the source code, for example, if the legacy system needs to be migrated or integrated to the new platform.

This research shows that it is possible to transform the UML sequence diagrams to the TFM (manually). The UML sequence diagrams and the TFM constructs have the similarities. Of course, some information is lost during the transformation, for example, parameters from the operations in the UML sequence diagrams are not presented in the TFM. Maybe some parameters for operations need to be added to the functional feature 11-tuple for keeping this information. The transformation from the UML sequence diagram to the TFM, excluding parameters of operations, is one-to-one according to the corresponding mappings described in the tables above.

The future research direction is related to complete development of rules for the reverse engineering from a source code to the PIM/PSM to the TFM using a set of UML diagrams as an assisting model.

References

[1] L. Favre,"MDA-Based Reverse Engineering," in *Reverse Engineering – Recent Advances and Applications*, 2012, ch. 3 [Online]. Available: http://www.intechopen.com/books/reverse-engineering-recent-advances-and-applications/mda-based-reverse-engineering http://dx.doi.org/10.5772/32473

[2] J. Osis and E. Asnina, "A Business Model to Make Software Development Less Intuitive," in *Proc. of the 2008 Int. Conf. on Innovation in Software Engineering, ISE08*, Vienna, 2008, pp. 1240–1246. http://dx.doi.org/10.1109/CIMCA.2008.52

[3] E. Asnina and J. Osis, "Topological Functioning Model as a CIM-Business Model," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, Ed. New York: IGI Global, 2011, pp. 40–64.

[4] J. Osis and E. Asnina, "Topological Modeling for Model-Driven Domain Analysis and Software Development: Functions and Architectures," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, Ed. New York: IGI Global, 2011, pp. 15–39.

[5] J. Osis, E. Asnina and A. Grave, "MDA Oriented Computation Independent Modeling of the Problem Domain," in *Proc. of the 2nd Int. Conf. on Evaluation of Novel Approaches to Software Engineering, ENASE*, Barcelona, 2007, pp. 66–71.

[6] J. Osis and E. Asnina, "Derivation of Use Cases from the Topological Computation Independent Business Model," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, Ed. New York: IGI Global, 2011, pp. 65 – 89.

[7] U. Donins "Topological Unified Modeling Language: Development and Application," Ph.D. dissertation, Riga Technical Univ., Riga, Latvia, 2012.

[8] V. Ovchinnikova and E. Asnina, "Reverse Engineering Tools for Getting a Domain Model within TFM4MDA," in *Proc. of the 11th Int. Baltic Conference on Databases and Information Systems Baltic DB&IS 2014 "Databases and Information systems"*, Tallinn, pp. 417–424.

[9] J. Osis and E. Asnina, "Is Modeling a Treatment for the Weakness of Software Engineering?" in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, Ed. New York: IGI Global, 2011, pp. 1–14.

[10] J. Rumbaugh, I. Jacobson and G. Booch, *The Unified Modeling Language Reference Manual* ADDISON-WESLEY, 1999 [Online]. Available: http://msdl.cs.mcgill.ca/people/tfeng/docs/The%20Unified%20Modeling%20Language%20Reference%20Manual.pdf

[11] D. Bell, *UML Basics: The sequence diagram*, 2004 [Online]. Available: http://www.ibm.com/developerworks/rational/library/3101.html

[12] K. Fakhroutdinov. *UML Sequence Diagrams* [Online] Available: http://www.uml-diagrams.org/sequence-diagrams.html#lifeline

[13] OMG.*OMG Unified Modeling Language. Version 2.4.1*. 2011 [Online]. Available: http://www.omg.org/spec/UML/2.4.1/

[14] S. MacGuire., *Strategy Guide for Reversi & Reversed Reversi*, 2011 [Online]. Available: http://www.samsoft.org.uk/reversi/strategy.htm

[15] OMG. *Architecture Driven Modernization (ADM)* [Online]. Available: http://www.omg.org/adm/

[16] S. Rugaber. *Model-Driven Reverse Engineering* [Online]. Available: http://www.cc.gatech.edu/reverse/repository/modelDriven.pdf

Viktoria Ovchinnikova received her 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 Research Assistant at the Department of Applied Computer Science, Riga Technical University. She is the author of one conference paper.

Her current research interests include reverse engineering and model-driven software development.

Address: Meža Str. 1/3, Riga, LV-1048, Latvia;
E-mail: viktorija.ovcinnikova@rtu.lv, viciyka@gmail.com

Erika Asnina received her Doctor's Degree (Dr. sc. ing.) in Information Technology with Specialization in System Analysis, Modeling and Design from Riga Technical University in 2006.

She has been an Associate Professor at the Department of Applied Computer Science, Riga Technical University since 2013. She is the author of 32 conference papers and book "Model-Driven Software Development: Architectures and Functions" awarded by the Latvian Academy of Sciences in 2011. Her research interests include software quality assurance, model-driven and object-oriented software development, and software engineering.

Address: Meža Str. 1/3, Riga, LV-1048, Latvia;
E-mail: erika.asnina@rtu.lv

Vicente García-Díaz is an Associate Professor at the Computer Science Department of the University of Oviedo. He has a PhD from the University of Oviedo in Computer Engineering. His research interests include model-driven engineering, domain specific languages, technology for learning and entertainment, project risk management, software development processes and practices. He has graduated in Prevention of Occupational Risks and is a Certified Associate in Project Management through the Project Management Institute.

Address: C/ Calvo Sotelo s/n. 33007 Oviedo (Asturias, España);
E-mail: garciavicente@uniovi.es