DE GRUYTER OPEN

Applied Computer Systems doi: 10.1515/acss-2015-0013

2015/18

Comparison of the Two-Hemisphere Model-Driven Approach to Other Methods for Model-Driven Software Development

Oksana Nikiforova[1], Ludmila Kozacenko[2], Dace Ahilcenoka[3], Konstantins Gusarovs[4], Dainis Ungurs[5], Maris Jukss[6]
[1–5] *Riga Technical University, Latvia*, [6] *McGill University, Canada*

***Abstract* – Models are widely used not only in computer science field, but also in other fields. They are an effective way to show relevant information in a convenient way. Model-driven software development uses models and transformations as first-class citizens. That makes software development phases more related to each other, those links later help to make changes or modify software product more freely. At the moment there are a lot of methods and techniques to create those models and transform them into each other. Since 2004, authors have been developing the so called 2HMD approach to bridge the gap between problem domain and software components by using models and model transformation. The goal of this research is to compare different methods positioned for performing the same tasks as the 2HMD approach and to understand the state of the art in the area of model-driven software development.**

***Keywords* – Two-Hemisphere Model, Model-Driven Software Development, problem domain modelling, UML modelling.**

I. Introduction

Developers have always been looking at ways to improve software development. Since then the development process has undergone several paradigm shifts, for example, from functional to object-oriented. One of the newest paradigms is model-driven approach, which is based on models and their interrelated transformations.

The so-called "software crisis" [1], identified in 1968, led to new solutions to formalize and structure development process. A lot of different development models and their improvements appeared both heavy-weight and light-weight and their combinations. Crisis still exists [2] and its causes still remain the same as in 1968, e.g., projects fail or they are completed with greater time and resource costs than originally estimated, the software is low quality or is missing functionality. Therefore, developers are still searching for ways to solve the "software crisis".

One of the new and promising techniques is Model-Driven Software Development (MDSD), which is based on models and their transformations [3]. MDSD proposes to use models at different stages of software project. Recent research [4], [5] and [6] show, that mostly methods and transformations associated with later development stages are studied. So on the one hand models are used, but on the other – models in the later development stages are not connected to the initial stages of the software development. As a result, significant amount of quality is lost in final software solution. The authors of this paper offer the two-hemisphere model driven (2HMD) approach. It is one of the methods to link information about problem domain and system analysis to system design phase and further software development. This paper aims to show 2HMD approach and compares it with other methods for the initial information modelling and model transformation to the design stage model.

The paper is structured as follows. The next section describes the position of the 2HMD approach within the framework of MDSD and offers a brief history of the approach development. Section 3 explains the basic elements of the approach, where the main principles of the transformations used to generate the UML diagrams from the two-hemisphere model are mentioned in the fourth section. Section 5 is a summary of the results of the 2HMD approach in comparison to other MDSD methods able to generate UML diagrams from the problem domain model. Several conclusions on the application of BrainTool and directions for future research are stated in the sixth section.

II. The 2HMD Approach within the Area of MDSD

MDSD is a relatively new paradigm, however, it already has a vast amount of model transformation techniques, which are classified by [7], [8] and [9]. The research done by [4] studies 85 papers about requirement engineering using model-driven approach and shows an increasing number of different model-driven methods.

Models are the main elements in MDSD, however their nature or use is not innovative in software development. Back in 1971, Ned Chapin created a flowchart diagram [10], a graphical means of documenting a sequence of operations. After that Larry Constantine and his colleagues from IBM developed data flow diagram [11] notation, which helped to split complex system into modules, depicting processes and information flows. One of the most significant models is the entity-relationship diagram (ER), offered by Peter Chen in 1976 [12]. It shows system structure, using entities and relationships between them. ER can be considered a predecessor of the Unified Modelling Language's (UML) class diagram. Subsequently, in 1997, Object Management Group introduced UML [13]. With an increase of software complexity, models serve as a sketch in summarizing knowledge about the system. In contrast, MDSD utilizes models not only as an auxiliary, but also as one of the key artefacts. Models are like building blocks that can be combined to construct system skeleton, which can then be

directly transformed into system code. Therefore, effort should be made to create correct and accurate models, as this will ensure the correctness of the final system. As accented by several authors in [14], [15], the way software is developed, without the use of model driven approach, can be compared to the 17th and 18th century hand crafting. At the time the components were handmade and unique. It was impossible to establish a standardization process, because it was impossible to create tools for mass production of said components. Nowadays software is developed similarly to that. Each software project is a unique piece of handy work that requires the developer to actively participate in its creation. Despite the existence of reusable component approach, software component addition is still a manual task [14]. One of the main reasons for that is a lack of unified approach, tool and standardization.

MDSD is an attempt to move over from manual to partial or completely automated development, because it offers standards and formal methods for the creation of new components. MDSD approach creates software by the use of modelling [15]. Drawing differs from modelling in the fact that it creates an image that may or may not confer with rules and semantics, while modelling on the other hand is more complex, because it clearly defines semantics. Because of that, modelling offers several advantages, such as model validation, ability to run and transform models, as well as debugging. The two-hemisphere model driven (2HMD) approach [16] is positioned as one of methodologies that can be used in the process of software development based on model driven development principles.

2HMD approach ideas were first published in 2002. In [17] the author proposes object oriented software framework that is based on the use of two linked models, where one of the models displays the structure of the system while the other displays the systems processes. The innovation at the time was the combination of these two models in a single abstraction level, which was not achieved in any other notations or modelling tools [18]. The [16] continues the development of the framework described in [17], as well as names the model used in it as the two-hemisphere model. The title is chosen because in cognitive psychology [19] the human brain is divided into two-hemispheres, one of which governs logic and the other- concepts. Coordinated work of both hemispheres ensures proper human functionality. Similarly the two-hemisphere concept is taken over to software development, because it is based on the display of problem domain in two mutually connected diagrams. Since the left hemisphere of the brain governs logic, the left part of the 2HM displays the business processes. And the right side displays conceptual class diagram.

Initially 2HMD approach was developed as a manual approach for the development of design of model object oriented system. However, after the emerging and evolution of model driven architecture, the author of [16] expresses an idea for the use of automated transformations in 2HMD approach. The author of [20] for the first time introduces transformation principles from the two-hemisphere model, focusing on which class objects will execute which methods. In 2008 a prototype of a tool is developed [21]. In 2010, the author in [22] conceptually describes transformation from the two-hemisphere model to the UML sequence diagram, focusing on the time aspect of methods' call. During this time transformation for the generation of the UML class diagram is refined regarding different types of class relationships [22].

In 2012 a tool named BrainTool (version 1.0) is developed that supports the 2HMD [23]. The tool supports the creation, validation and transformation of the 2HM to the UML class diagram as well as export of the XML file with the UML class diagram to Sparx EA 7.5. In the same year the authors develop the software that converts 2HM description in XML file from BrainTool into the UML sequence diagram with a defined element set and layout, usable by Sparx EA. In [24] the author discusses several issues within the model interchange between tools due to the problems with standard for file format exchange between various tools, so far a decision to adapt XML file structure for UML from Sparx EA is made. This is done to prove that at least theoretically tool compatibility is possible.

Inspired by the evolution in the area of model-driven software development with respect to modelling tools and IDEs for their development, the authors in 2013 developed BrainTool version 2.0 from scratch in Java programming language, using JGraphX [24] library for graph visualization. The tool offers a new, more mainstream GUI and features an improved transformation implementation. The research described in [25] showcases that BrainTool v2.0 supports 2HM creation, validation as well as the UML class and sequence diagram generation. It also supports the export of generated diagrams to Sparx EA 7.5. Year 2014 brings the research into the UML diagram layout [26], offering a new UML class diagram layout algorithm. As well as following research into the UML sequence diagram automatic layout, the author of [26] offers the algorithm that features a wider array of criteria and elements, compared to a variety of other available sequence diagram layout algorithms.

III. The Basic Elements of the Two-Hemisphere Model

Two-hemisphere model consists of two diagrams – business process diagram and conceptual class diagram. The inclusion of these diagrams is not random, it is not only based on the previously mentioned analogy with the human brain, but also based on the information shown in these diagrams helping to describe the system from different points of view, which is important in system development.

Business process modelling, as mentioned in [27], developed as a result of solutions made by Management Science and Computer Science in the 70ies of the 20th century. Besides, nowadays the importance of business process modelling has not decreased. The importance of business process modelling is confirmed in [28] presenting regular studies on the importance and usability of these processes. The studies confirm, that management of business processes is important and companies pay attention to it. The research also shows, that companies over time learn the

existing business process modelling notations and methodologies. Thereby, one advantage of using two-hemisphere model is that there is no need to make additional models to use it, but the user can be sure, that the business model consisting of elements required by two-hemisphere in the organization already exists. As the two-hemisphere model serves as a bridge between the problem domain and software design phase, the business model is understandable to both − business people and developers.

The inclusion of concept model in the approach is motivated by the principles of object-oriented paradigm and general context of data analysis. Usually at the beginning of software development data dictionary is created or there is any other agreement about the terminology used in software development and documentation. In [29] the author describes conceptual modelling as basis of software development, without which good design cannot be performed. Conceptual models are high-level software description, which contains concepts. Any kind of things, events and living beings that are important to a given problem domain can be considered as concepts. Concepts are described by attributes, but methods show actions specific to these concepts. Peter Chen's [12] created Entity-Relationship (ER) diagram as mentioned in [30] was used in database design, but later it was also used in software system design as conceptual model. In [31] it is indicated, that nowadays it is topical to use ontology not only in artificial intelligence (robot, agent) systems, but also to create unified terminology, so that all stakeholders could communicate. In [32] it is shown that ontology represents classes of objects, class relations, attributes and axioms. It provides the basis for choosing the model that represents problem domain concepts as the other model. Therefore the other diagram of two-hemisphere model is the conceptual model, consisting of concepts and its attributes, where model notation is similar to the same in ER diagram.

The two-hemisphere model consists of one conceptual model and one or more process models. A fragment of two-hemisphere model of conference support system is shown in Fig. 1. BrainTool [9] (see its general view in Fig. 1) is positioned as one of the CASE tools, which enables system modeling and model transformation according to the requirements stated in the previous section.

2HMD approach uses its own notation considering process model elements, which is described in Fig. 1. It combines good practices from the data flow diagram [11] and BPMN [33] notations to represent business processes, leaving only the minimum range of elements, that further will be used in transformations. The conceptual model in two-hemisphere model is similar to ER diagram, consisting of concepts and their attributes (see Fig. 1).

The difference between the classical ER diagram and two-hemisphere conceptual model is that the conceptual model does not include relations between concepts, the relations are generated taking into account the links between concepts and process model's data flows. The linkage between these two models is an innovative usage of two models in software development. The assigning of data structure (concept) from the conceptual model to process model's data flow provides the linkage between models [17]. Every data flow has only one assigned concept, but one concept might define the data type of many data flows.

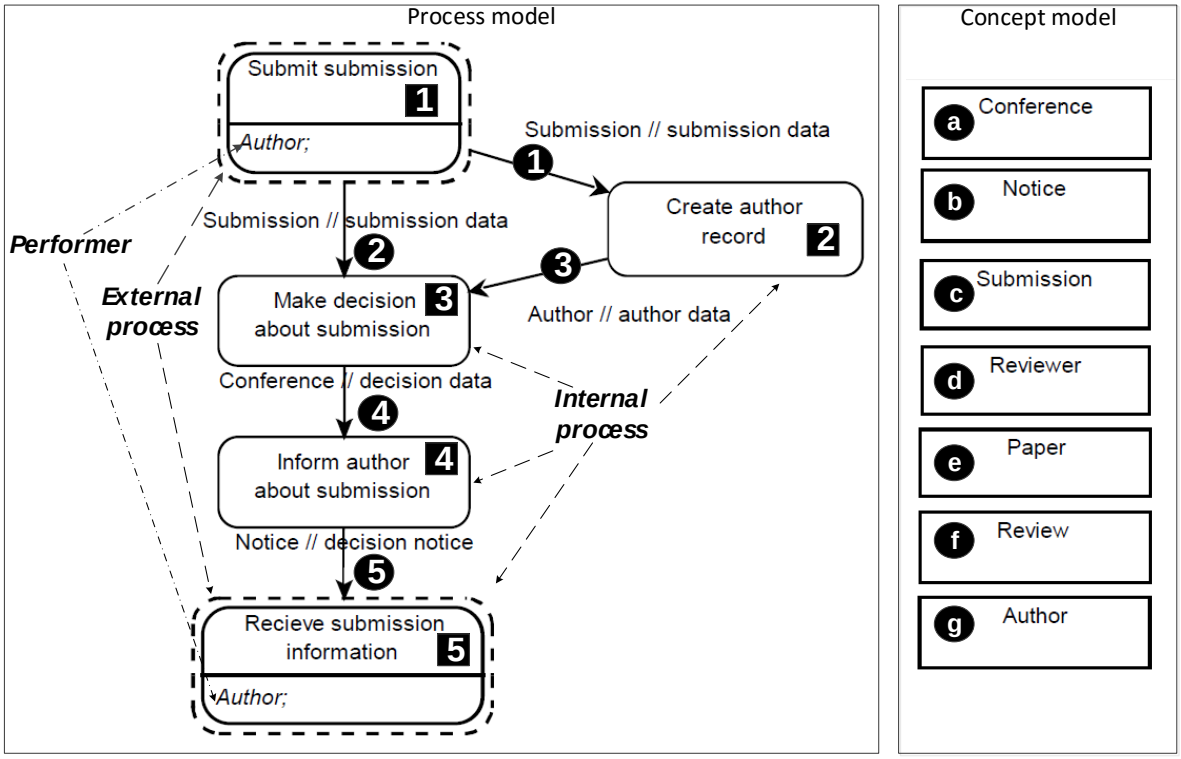

Fig. 1. The fragment of the two-hemisphere model for conference support system.

Further transformations are basis for determining responsibilities to object classes. They allow the assignment of methods to class objects based on object-oriented paradigm. The linkage also provides basis to define the relationship type between classes in the UML class diagram.

IV. The Essence of the Two-Hemisphere Model Transformations to the UML Diagrams

The 2HMD approach enables generation of UML diagrams. Since 2002, the approach has undergone several changes, which were caused by addition of new resulting UML diagrams and generated diagram element amount. Currently the 2HMD approach has defined transformations, which are described in subsections below. The transformations to the UML class and sequence diagrams are supported by a tool, called BrainTool [34].

This diagram choice is not accidental; it is based on the following arguments. Firstly, when designing a system, it is important to reflect the system from a different point of view, thus showing both structural side as well as dynamic side of the system. The structure helps to show the parts and components of the system, while the dynamic parts allow to see how these parts interact with each other. The UML use case diagram is typically used in system design as a basis for requirements specification [13].

In addition, class and object interaction diagrams are main reflectors of the system's structural and dynamic aspect. Secondly, as show in [35] study only more than 50 % of the UML users regularly utilize class diagram. Sequence and use case diagrams are used approximately by 50 % of users. In turn, state machine diagrams show class method executed order, which is another type of dynamic representation and is used in software testing. Therefore it can be stated that the 2HMD approach provides minimum necessary set of diagrams, which are used in system design.

A. The UML Class Diagram Transformations

The transformations from the two-hemisphere model into the UML class diagram are shown in Table I. The transformation description briefly explains the essence of which elements of the two-hemisphere model are taken as the basis for which UML class diagram elements generation. In details, these transformations are explained in authors' papers [16], [22], [25]. Fig. 2 shows the UML class diagram, which is generated from the two-hemisphere model shown in Fig. 1.

The layout algorithm for the UML class diagram offered by authors operates in four major steps [26]. Prior to these steps, the algorithm gathers data on all the classes and their relationships in the diagram and places them in specific data types and constructs, for easier usage. First of all, each class is assigned a score. Then all the classes are divided into small groups. The groups are created around the classes with the highest scores. The third step covers the layout of individual groups.

TABLE I

List of the UML Class Diagram Generated from the Two-hemisphere Model

Element	**Transformation description**
Class	For each concept in conceptual diagram a class is created
Class attribute	Each concept's attribute is transferred to the relevant class as a class attribute.
Class method	Class methods can be determined by the link between conceptual and process diagram. Every data flow has an assigned concept, in other words every data flow belongs to a class. Class method is an internal or external process. Class method is added to a class, which is defined on the outgoing data flow.
Dependency	Dependency is obtained if incoming and outgoing data flows have different concepts. As well as in cases, when process has several incoming data flows, and one of the incoming data flows has the same concept as the outgoing data flow.
Aggregation	Aggregation is obtained, if a process has several incoming data flows with different concepts and one or more outgoing data flows with the same concept.
Association	Association is obtained, when two classes have different bidirectional relationships, for example, there is aggregation from A to B and dependency from B to A.
Generalization	Generalization is obtained after aggregation, dependency and association is defined. When a process has one or more incoming data flows with the same concept and there are several outgoing data flows with the same concept, generalization rule is applied.
Interface	If several concepts, which are transformed to classes, can be given the same method for executing, and it is not possible to clearly define which one will execute it, interface is created. This method will be given to interface, and classes will inherit the method from interface.

B. The UML Sequence Diagram Transformations

The transformations from the two-hemisphere model into the UML sequence diagram are summarized in Table II. For the moment not all the elements of the UML sequence diagram are defined to be generated from the two-hemisphere model. Additional to the elements the same in the UML class diagram, the transformations are focused to the dynamic aspect of the two-hemisphere model, i.e. sequence of the processes to be performed within the time aspect. Fig. 3 shows the UML sequence diagram, which is generated from the two-hemisphere model shown in Fig. 1.

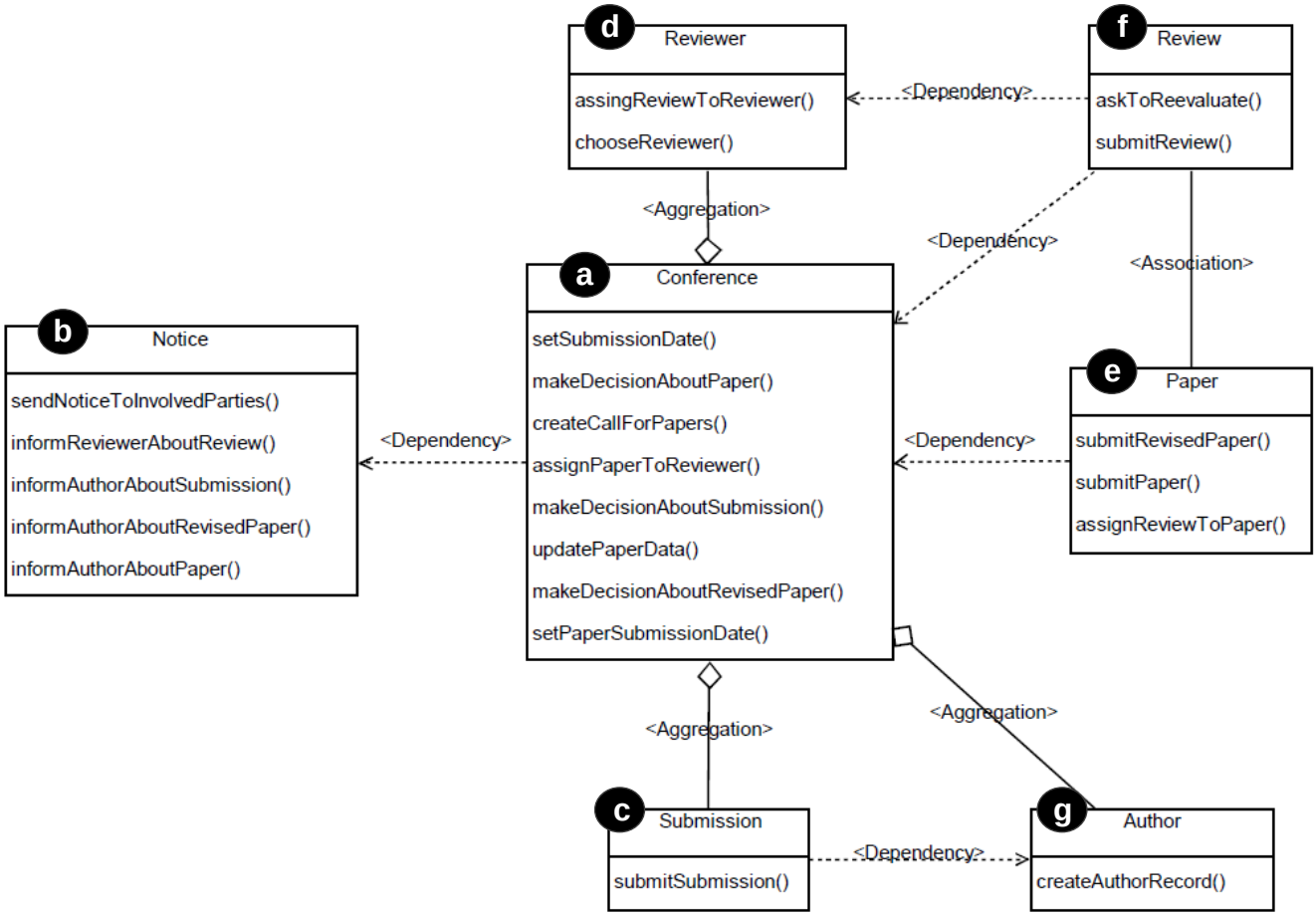

Fig. 2. The UML class diagram generated from the two-hemisphere model.

Considering the specificity of sequence diagram, where the objects are allocated horizontally at the top of the diagram and the life lines are drawn vertically top-down, the authors propose to use an algorithm, which is based on topology-shape-metrics planarization step [36] and uses one principle of force-directed approach [37] – object tends to attract the objects with which it communicates. The algorithm places the elements as close as possible and tries to arrange the communicating participants beside, based on priorities. The priorities are calculated considering object attraction forces – the more messages between the elements the higher the priority for them to be beside. The layout algorithm calculates the distance between the elements considering the length of messages and class object names.

Algorithm places elements as close as possible by taking into account the diagram flow (e.g., interacting objects are being placed beside if possible) [26].

TABLE II

LIST OF THE UML SEQUENCE DIAGRAM GENERATED FROM THE TWO-HEMISPHERE MODEL

Element	**Transformation description**
Object Lifeline	Lifeline is created for each concept, which interacts.
Object	Every concept, which is assigned to the data flow in a process diagram, is transformed as the object's class.
Actor	Every performer from process diagram is transferred as an actor
Message	Internal processes are transformed into messages (only between two class objects).
	External processes are transformed into messages (between actor and class object or vice versa).
Set of messages	Usually processes which have several incoming or outgoing data flows are grouped in interaction fragments.

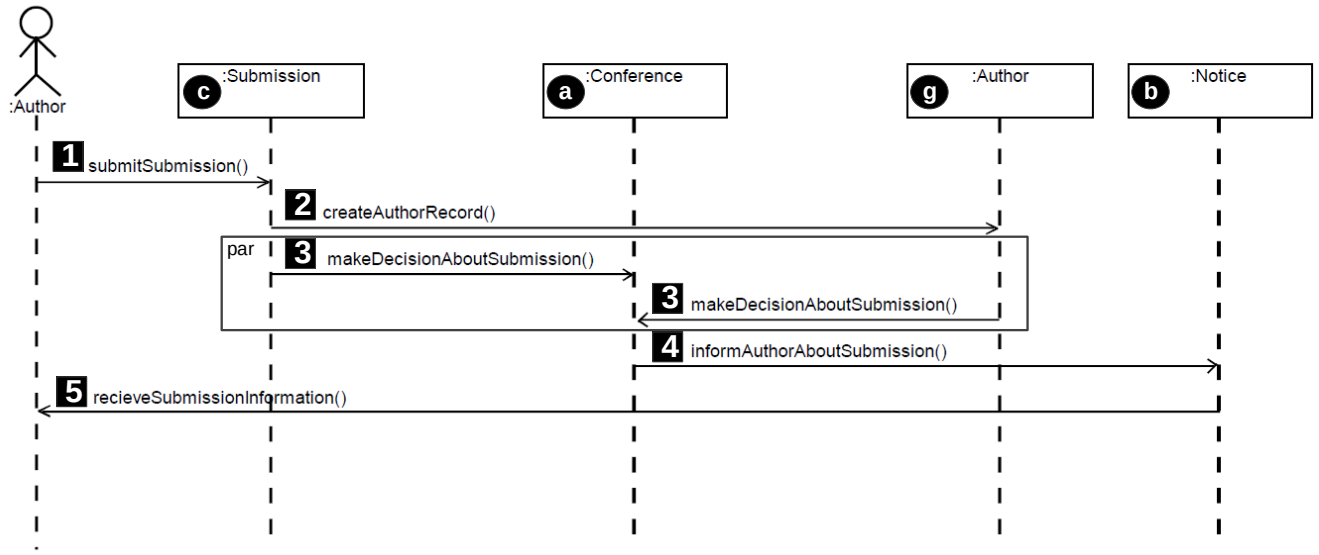

Fig. 3. The UML sequence diagram generated from the two-hemisphere model.

C. UML State Machine Diagram Transformations

Since it is possible to generate the UML sequence diagram from the two-hemisphere model, the authors are investigating the ability to generate the UML state diagram presenting the behavior of the certain class. The initial result of this research is published in [22], but still the defined transformations are not implemented in BrainTool. The basic principles of this set of transformations are described in Table III.

TABLE III

LIST OF THE UML STATE DIAGRAM POTENTIALLY GENERATED FROM THE TWO-HEMISPHERE MODEL

Element	**Transformation description**
Initial state	One initial state is created for the state diagram.
Final state	At least one final state is created for the state diagram.
State	Each state is reflected as incoming or outgoing message.
Transition	Defined according to sequence of method calls.
Fork/Join pseudo state	The separate "branch" of states is defined for each class behavior in certain sequence diagram using fork/join pseudo state construction.

D. UML Use Case Diagram Transformations

For the moment authors are investigating the ability to generate the UML use case diagram from the two-hemisphere model. Transformations still are at the theoretical point of the research and are not implemented in BrainTool. Authors offer to obtain actors using an external performer of the process. This way all performers from all process diagrams are transferred to the use case actors. Each process diagram in 2HM can be considered as a use case. Associations between use cases and actors are defined by the performer which is mentioned in the respective process diagram.

V. COMPARISON

To compare the 2HMD approach with other methods, which support model-driven principles, international electronic libraries were used, such as ACM Digital Library, EBSCO Host un ScienceDirect and also related literature [5], [38], [39] about method comparison were used. The methods were selected using MDSD keywords such as "problem domain model", "computer independent model", "platform independent model". In addition, only papers in English were selected and those which have been published after 2000, because model-driven development appeared around this time.

After automatic search was completed authors of the paper examined each paper in detail and selected only those which were longer than five pages. Table IV shows summarized comparison results of 21 methods, which have been chosen based on the selection strategy described above. The methods are arranged in alphabetical order by their developer names, which are listed in the second column. We do not give references to method descriptions in order not to expand the list of the bibliography due to limitations of paper volume, but it is possible to search the method by its author's name. In the third column of the table problem domain notations offered by the emethod developers are listed. In the fourth column solution domain notations are listed, which are obtained from problem domain by help of transformations. This way it is possible to evaluate the knowledge that software developer should have to work with the given methods. Table IV shows that approximately 64 % of problem domain methods use other notation rather than UML.

TABLE IV

THE TYPE OF MODEL PRESENTATION OFFERED AT DIFFERENT ABSTRACTION LEVEL IN COMPARED METHODS

No.	Method	Problem domain	Solution (software) domain
1.	Bousetta, et al	**UML** Use case, BPMN	UML class, sequence + Business rules
2.	Cetinkaya, et al	BPMN	DEVS
3.	De Castro, et al	e^3 value & BPMN	UML Use case, activity
4.	Fatolahi, et al	**UML** Use case scenarios	UML State machine
5.	Harbouche, et al	**UML** activity with collaboration	UML State machine
6.	Trujillo, u.c.	i* with refinement	UML profile for data warehouse
7.	Kherraf, et al	**UML** activity	UML Class
8.	Kardos, Drozdova	Data flow diagrams	UML use case, activity, class, sequence
9.	Koch, et al	**UML** use case, activity	UML activity, class, state machine
10.	Meertens, et al	Archimate models and SVBR	Mendix, Microflow
11.	Nikiforova, et al	Two-hemisphare model	UML use case, class, sequence
12.	Osis, et al	Topological functioning model	TopUML profile
13.	Penserini, et al	i*	Not given
14.	Prat, et al	CommonKADS model	UML class, activity
15.	Rodriguez, et al	**UML** activity, BPSec process model	UML class, use case (UMLsec)
16.	Rodriguez-Dominguez, et al	**UML** class	UML class
17.	Raj, et al	SBVR dictionary, rules	UML activity, class, sequence
18.	Wu, et al	**UML** use case, activity	UML sequence, state machine, class
19.	Zdravkovic, et al	OeBTO process model	IBM UML profile service
20.	Zhang, Mei, et al	Feature model	Not given
21.	Zhang, Feng, et al	Processes in OWL	UML class

These other notations mainly are BPMN, data flow diagrams or other process notation diagrams. Most used UML diagrams in business level are the use case diagrams (62 %) and activity diagrams (62 %). Otherwise, at the solution domain 86 % of the methods use UML. The leader of UML diagrams is the class diagram (63 %), followed by activity diagram (31 %), then sequence, state machine and use case diagrams (26%). In 15% of the cases specific UML diagram is not offered to use.

Table V combines comparison results for the same 21 methods in correspondence with the following 11 criteria:

1. Business objects.
2. Business processes.
3. Design level system structure.
4. Design level system behavior.
5. Level of automation. This criterion provides insight in system's readiness to be used. "M" or "Manual" shows that no transformation rules are present. "P" or "Partial" shows that some transformation rules are present. "A" or "Automated" shows that the method has tool support that has implemented transformations.
6. Layout.
7. Transformation direction. "O" − one-way transformation shows that the target model is generated from source model. "T" − two-way transformation shows that the target model can be generated from the source model and the source model can be generated from the target model.
8. Result of transformations. Criterion describes the type of automated transformation result. It can be diagram, XML file, or some other type.
9. Model traceability. This criterion describes how changes are passed down to subsequent model abstraction levels. "+" means that changes automatically transfer to other models. "P" or "Partially" means that it offers to make changes in other abstraction levels. "-" means, that it warns if changes would affect other abstraction levels or outright forbid making such changes.
10. Area of application. If the method is offered for the specific application the area is defined in the table according to the evaluation of the 10th criterion, otherwise it is noted "G" (general).
11. Approbation type. This criterion shows how the method is showcased by its authors: Case Study (CS), System Development (SD) or no testing (-).

A star (*) in Table V next to several cases of the 5th criterion shows methods that claim to have a tool support, however authors have failed to locate such tools. A question mark (?) next to several cases of the 8th criterion shows that, since authors could not gain access to the tool, it was impossible to determine how the method represents the model after transformation.

All the methods represent processes at the business level, however not all of them display system objects. Only 71% of the methods do that. In Design level 90% of methods display system structure and 85% show system dynamics. However in both CIM and PIM levels, if a method does not display one half of the system, it always displays the other half. In general only half of the methods display all four of the mentioned criteria. Authors point out that a situation appears where in higher abstraction levels objects and processes are displayed, however in lower levels one of these elements is not included. This is notable in the 6th and 16th method where dynamic aspect is lost in design level.

TABLE V

METHOD COMPARISON BASED ON SELECTED CRITERIA

No.	**Criteria**										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1.	+	+	+	+	P	−	O	−	−	G	CS
2.	+	+	+	+	A	−	O	?	−	Modelling and simulation	−
3.	+	+	+	+	A*	−	O	?	−	G	CS
4.	−	+	−	+	A*	−	O	XML	−	Web application	CS
5.	−	+	−	+	A*	−	O	?	−	Artificial intelligence	CS
6.	+	+	+	−	A	−	O	SQL	+	Data warehouse	SD
7.	+	+	+	+	M	−	O	−	−	G	CS
8.	−	+	+	+	M	−	O	−	−	G	CS
9.	+	+	+	+	P	−	O	−	−	Web application	−
10.	+	+	+	+	M	−	O	−	−	G	−
11.	+	+	+	+	A	+	O	XML	−	G	CS
12.	+	+	+	+	P	−	O	−	−	G	CS
13.	+	+	+	+	A	−	O	XML	−	G	CS
14.	+	+	+	+	P	−	O	−	−	Knowledge engineering	CS
15.	−	+	+	+	A*	−	O	?	−	G	CS
16.	+	+	+	−	A	−	O	XML	−	middleware	CS
17.	−	+	+	+	A*	−	O	?	−	G	CS
18.	+	+	+	+	A*	−	O	?	−	GUI modelling	−
19.	+	+	+	+	M	−	O	−	−	Web application	−
20.	−	+	+	−	M	−	O	−	P	G	−
21.	+	+	+	+	P	−	O	−	−	G	−

Breakdown between method automation is rather homogeneous, five methods or 23 % have not defined transformations, the same amount – 23 % have defined transformations. For 23 % of the methods authors have gained access to tools that implement them, 28 % of the methods claim they have a tool, but authors have not managed to locate or access it.

Three methods with tool support offer to store models in XML files with XMI structure. However, it is not entirely clear if the 13th and 16th method XML files are compatible with other UML tools, like it can be done in the 2HMD approach. Since the 6th method is used for the creation of data warehouse, the result is displayed as SQL code. All methods support only one-way transformations.

Only the 6th method provides traceability and the 20th method describes it but only theoretically, because the method does not have an implementation tool. None of the methods except the 2HMD focus on layout problems. That means that even with tool support additional time will be consumed to deal with the layout. Twelve methods or 57 % are not specific enough for the development of a particular information system. Some methods are more specific and are used for the creation of web based or other application software. 62 % of the methods show their practical application using system analysis; one method is showcased by the use of a full-fledged system. 33% of methods do not even describe any possible applications. This means that the potential users of the method method cannot determine how the method will work in practical environment and if it will work at all.

In conclusion of the comparison authors point out that only three methods satisfy the criterion most. These are the 6th, 11th (2HMD approach) and 13th method. The 6th method should be used in case of creating data warehouses, is showcased in a real system environment and is the only method that supports traceability. The 11th and 13th method are more general and can be used in the development of variety of software systems. The 11th method, also known as 2HMD, surpasses the 13th method in the fact that it also supports automatic layout. That is valuable as it lets the developers to save time and resources otherwise spent to deal with this problem. It is also unknown if the 13th method supports XML file export to other tools.

VI. CONCLUSION

A relatively new development paradigm − MDSD − is based on the models and their transformations up to the code, which gives an opportunity to improve the quality of software development due to linking models built at various levels of abstraction. Linking would provide the consistency throughout the whole development process. The 2HMD approach, its twelve-year evolution, the basic elements and possible transformation into UML diagrams supported by BrainTool is presented in the paper as one of MDSD methods. Authors have made a comparison of the 2HMD approach with twenty MDSD methods based on eleven selected criteria. The comparison result shows that only few methods are supported by a tool, where some of them are just the tool's prototypes. Therefore, the main limitation of MDSD methods application for software development is a lack of automatization to support model transformations offered by a wide range of MDSD methods. So far, the main contribution of the research group is the work on the development and formalization of the two-hemisphere model-driven approach

with the ability to implement the defined model transformation in a tool.

The need to create another new software development approach is justified by the fact that the 2HMD approach links the problem domain with software system at the design phase, thus preparing a wide set of design artefacts for future code generation directly from problem domain model. The benefit in this case, first of all, is reduced time and human resource consumption. Secondly, the design of software system based on formally defined transformations of problem domain model allows reducing the risks of "manual" software class design. These risks are the loss of information, inconsistency and duplication.

Based on the research performed and results achieved authors should stress the following conclusions:

- The two-hemisphere model contains enough information about the problem domain to generate elements for design model of system structure and behavior;
- The approach offered by authors is formalized to develop the tool to support model transformations defined by the method;
- The transformations offered by the approach pay attention also to the diagram layout and use newly invented layout algorithm for UML diagrams.
- Only one method from the compared ones supports the traceability of the model at different abstraction levels. This indicates that traceability is possible, but not sufficiently developed, so it is a potential direction for the improvement.

Additionally to supporting the traceability, the future research directions can deal with the layout of the two-hemisphere model, refinement of the element set to be generated from the two-hemisphere model and expand the notational conventions of the model itself.

ACKNOWLEDGMENT

The research presented in the paper is supported by Latvian Council of Science, Project No. 342/2012 "Development of Models and Methods Based on Distributed Artificial Intelligence, Knowledge Management and Advanced Web Technologies".

REFERENCES

[1] Haigh, T., "Crisis, What Crisis? Reconsidering the Software Crisis of the 1960s and the Origins of Software Engineering," Thomas Haigh, 2011. Available: http://www.tomandmaria.com/tom/Writing/SoftwareCrisis_SofiaDRAFT.pdf

[2] Jones, C., "Software Quality in 2012: A Survey of the State of the Art," *Software Quality Group of New England*, 2012. Available: http://sqgne.org/2012-13.html

[3] Stahl, T., Volter, M., Bettin, J., Haase, A., et al. *Model-Driven Software Development: Technology, Engineering, Management*, 1 ed. USA: John Wiley & Sons, 2006. 446 p.

[4] Loniewski, G., Insfran, E., Abrah, S. "A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development," in *Proc. of 13th Int. Conf. MODELS 2010*, Norway, Oslo, October 3-8, II part. Berlin: Springer Berlin Heidelberg, pp. 213–227, 2010 http://dx.doi.org/10.1007/978-3-642-16129-2_16

[5] Sharifi, H.R., Mohsenzadeh, M., Hashemi, S.M., "CIM to PIM Transformation: An Analytical Survey," *IJ of Computer Technology & Applications*, vol. 3, pp.791–796, 2012.

[6] Kriouile, A., Gadi, T., Balouki, Y., "IM to PIM Transformation: A criteria Based Evaluation," *IJ of Computer Technology & Applications*, vol. 4, no. 4, pp. 616–625, 2013.

[7] Czarnecki, K., Helsen, S., "Feature-Based Survey of Model Transformation Approaches," *IBM Systems Journal-Model-driven software development*, vol. 45, pp. 621–645, 2006.

[8] Mens, T., Gorp, P.V., "A Taxonomy of Model Transformations," *Electronic Notes in Theoretical Computer Science (ENTCS)*. vol. 152, pp. 125–42, 2006. http://dx.doi.org/10.1016/j.entcs.2005.10.021

[9] Lano, K., Kolahdouz-Rahimi, S., Poernomo, I., "Comparative Evaluation of Model Transformation Specification Approaches," *IJ of Software and Informatics*, vol. 6, no. 2, pp. 233–269, 2012.

[10] Chapin, N., *Flowcharts*. New York, USA: Petrocelli Books, 1971

[11] Stevens, W., Myers, G., Constantine, L., "Structured Design," *IBM Systems Journal*, vol. 13, no. 2, pp. 115–139, 1974. http://dx.doi.org/10.1147/sj.132.0115

[12] Chen, P.P., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, vol. 1, pp. 9–36, 1976.

[13] OMG, *OMG Unified Modelling Language™ (OMG UML), Superstructure*. OMG, 2014. Available: http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF

[14] Guttman, M., Parodi, J., *Real-Life MDA*. 1st ed. USA: Morgan Kaufmann, 224 p., 2007.

[15] Brambilla, M., Cabot, J., Wimmer, M., *Model-Driven Software Engineering in Practice*. 1st ed. USA: Morgan & Claypool Publishers, 182 p., 2012.

[16] Nikiforova, O., Kirikova, M., "Two-Hemisphere Model-Driven Approach: Engineering Based Software Development." in *Proc. of 16th International Conference*, CAiSE 2004, Riga, Latvia, June 7–11, vol. 3084. Springer Berlin Heidelberg, pp. 219–233, 2004. http://dx.doi.org/10.1007/978-3-540-25975-6_17

[17] Nikiforova, O., "General Framework for Object-Oriented Software Development Process," in *Scientific Proc. of RTU. Computer Sciences*, vol. 13, pp. 132–144, 2002.

[18] Gartner Predicts 2002: Top 10 Predictions, Gartner, 2002.

[19] Anderson, J.R., *Cognitive psychology and its implications*, Worth Publishers, 469 p., 2010.

[20] Nikiforova, O., Kirikova, M., Pavlova, N., "Two-Hemisphere Driven Approach: Application for Knowledge Modelling," in *Proc. of the Seventh International Baltic Conf. on Databases and Information Systems*, (Baltic DB&IS 2006), O. Vasilecas, J. Eder, A. Caplinskas (eds.), Lithuania, Vilnius, 3–6 July, IEEE, pp. 244–250, 2006. http://dx.doi.org/10.1109/DBIS.2006.1678503

[21] Nikiforova, O., Pavlova, N., Grigorjevs, J., "Several Facilities of Class Diagram Generation from Two-Hemisphere Model in the Framework of MDA," in *Proc. of the 23rd International Symposium on Computer and Information Science*, Istanbul, Turkey, October 27–29, p. 6., 2008. http://dx.doi.org/10.1109/iscis.2008.4717956

[22] Nikiforova, O., "System Modelling in UML with Two-Hemisphere Model Driven Approach," *Scientific Journal of RTU. Computer Sciences*. 2010, vol. 21, pp. 37–44.

[23] Nikiforova, O., Gusarovs, K., Gorbiks, O., Pavlova N., "BrainTool A Tool for Generation of the UML Class Diagrams," in *Proc. of the 7th International Conference on Software Engineering Advances*, Lisbon, Portugal, 18-23 Novemer, IARIA, 60-69.lpp (2012)

[24] JGraph. jGraph-connectiong the dots. Available: http://www.jgraph.com/

[25] Nikiforova, O., Kozacenko, L., Ungurs, D., Ahilcenoka, D., Bajovs, A., Skindre, N., Gusarovs, K., "BrainTool for Software Modelling in UML," *Scientific Journal of RTU: Applied Computer Systems*, Grundspenkis J. et al. (Eds), vol. 16, pp. 33–42, 2014. http://dx.doi.org/10.1515/acss-2014-0011

[26] Nikiforova, O., Ahilcenoka, D., Ungurs, D., Gusarovs, K., Kozacenko, L., "Several Issues on the Layout of the UML Sequence and Class Diagram," in *Proc. of the 9th Int. Conf. on Software Eng. Advances*, Mannaert H. et. al (Eds), October 12–16, Nice, France, IARIA, pp. 40–47, 2014.

[27] Polak, P., "BPMN Impact on Process Modelling," in *Proc. of the 2nd International Business and Systems Conference BSC*, Riga, Latvia, November 5, pp. 26–35, 2013.

[28] Harmon, P, Wolf, C.: The State of Business Process Management, BPTrends, 2014. Available: http://www.bptrends.com/

[29] Johnason, J., Henderson, A., *Conceptual Models. Core to Good Design*, 1st ed. Morgan & Claypool Publishers, 2011. 110 p.

[30] Hesse, W., "Ontologies in the Software Engineering process," in *Proc. of the 12th Int. Workshop on Exploring Modelling Methods for Systems*

Analysis and Design, (EMMSAD-2007), Trondheim, Norway, 11–15 June, pp. 1–13, 2007.

[31] Graudina, V., Grundspenkis, J., "Algorithm of Concept Map Transformation to Ontology for Usage in Intelligent Knowledge Assessment System," in *Proc. of the 12th International Conference on Computer Systems and Technologies*, Vienna, Austria, June 16–17, 2011. http://dx.doi.org/10.1145/2023607.2023627

[32] Mädche, A., Schnurr, H.P., Staab, S., Studer, R., "Representation-Language-Neutral Modelling of Ontologies," J. Ebert, U. Frank (Hrsg.): Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik. Proc. "Modellierung 2000". Koblenz: Fölbach-Verlag, pp. 143–150, 2000.

[33] OMG, *Business Process Model and Notation*, Available: http://www.bpmn.org/

[34] RTU, *BrainTool webpage*, Available: http://braintool.rtu.lv

[35] Dobing, B., Parsons, J., "How UML is used," *Communications of the ACM*. May 2006, vol. 49, no. 5, pp. 109–113, 2006.

[36] Eiglsperger, M., "Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach," Thesis. Tübingen, Germany: Eberhard Karls Universitat, p. 173, 2003.

[37] Dwyer, T., "Three dimensional UML using force directed layout. Australian Symposium on Information Visualisation," *Australian Computer Society, Inc*, pp. 77–85, 2001.

[38] Siqueira, F. L., Silva, P. S. M., "Analyzing CIM to PIM Transformations Using the WRSPM model," in *Proc. of the 2nd Int. Conf. on Advanced Communications and Computation*, Venice, Italy, October 21–26, IARIA, pp. 41–50, 2012.

[39] Al-Jamini, H., Ahmed, M., "Transition from Analysis to Software Design: A Review and New Perspective," in *Proc. of Int. Conf. on Soft Computing and Software Engineering*, vol. 3, no. 3, pp. 169–176, 2013.

Oksana Nikiforova received the doctoral degree in information technologies (system analysis, modeling and design) from Riga Technical University, Latvia, in 2001.
She is presently a Professor with the Department of Applied Computer Science, Riga Technical University, where she has been on the faculty since 2000. Her current research interests include object-oriented system analysis and modelling, especially the issues on Model Driven Software Development.
E-mail: oksana.nikiforova@rtu.lv

Ludmila Kozacenko received the Master degree in computer systems from Riga Technical University, Latvia, in 2014. She is presently a scientific assistant at the Department of Applied Computer Science, Riga Technical University. Her current research interests include transformation approach classification and realization of transformation using general purpose programming language Java.
E-mail: ludmila.kozacenko@rtu.lv

Dace Ahilcenoka received the Master degree in computer systems from Riga Technical University, Latvia, in 2014.
She is presently a Scientific Assistant with the Department of Applied Computer Science, Riga Technical University. Her current research interests include UML diagram layout, algorithms of diagram layout.
E-mail: dace.ahilcenoka@rtu.lv

Konstantins Gusarovs received the Master degree in computer systems from Riga Technical University, Latvia, in 2012. He is Java developer in Forticom Ltd.
His current research interests include object-oriented software development and automatic obtaining of program code.
E-mail: konstantins.gusarovs@gmail.com

Dainis Ungurs received the Master degree in computer systems from Riga Technical University, Latvia, in 2014.
His current research interests include original ways of UML class diagram layout in system modelling tools.
E-mail: dainis.ungurs@rtu.lv

Maris Jukss is a third year PhD student at the Modelling, Simulation and Design Lab in School of Computer Science at McGill University, Canada. His current research interest is efficient and usable model transformations.
E-mail: maris.jukss@mail.mcgill.ca