Applied Computer Systems
ISSN 2255-8691 (online)
ISSN 2255-8683 (print)
May 2020, vol. 25, no. 1, pp. 51–56
https://doi.org/10.2478/acss-2020-0006
https://content.sciendo.com

Anemic Domain Model vs Rich Domain Model to Improve the Two-Hemisphere Model-Driven Approach

Oksana Nikiforova[1], Konstantins Gusarovs[2*]
[1, 2]*Riga Technical University, Riga, Latvia*

***Abstract* – Evolution of software development process and increasing complexity of software systems calls for developers to pay great attention to the evolution of CASE tools for software development. This, in turn, causes explosion for appearance of a new wave (or new generation) of such CASE tools. The authors of the paper have been working on the development of the so-called two-hemisphere model-driven approach and its supporting BrainTool for the past 10 years. This paper is a step forward in the research on the ability to use the two-hemisphere model driven approach for system modelling at the problem domain level and to generate UML diagrams and software code from the two-hemisphere model. The paper discusses the usage of anemic domain model instead of rich domain model and offers the main principle of transformation of the two-hemisphere model into the first one.**

***Keywords* – Anemic domain model, code generation, model transformation, rich domain model, system modelling, two-hemisphere model.**

I. Introduction

In general, there are two ways of looking at any software system. One way is to consider just data, including variables, arguments, data structures and files where the operations are examined only within the framework of the data. The other way of viewing the software system is to consider just the operations performed on the data where data are of the secondary importance. According to current trends in software development, data and operations are viewed at equal importance, in spite of the fact that sometimes data have to be stressed and other time operations are more critical. Object-oriented software development assumes the definition of system objects according to problem domain as the primary artefacts, including the information about data and operations together [1]. Therefore, one of the fundamental tasks is to define an object structure and to share the responsibilities of object, i.e., to define the operations for objects to perform. Two-hemisphere model driven approach [2] shares responsibilities between objects and serves as a basis for generation UML [3] diagrams and software code from them.

As far as the two-hemisphere model contains two interrelated models of problem domain, the approach can produce UML class and sequence diagrams in correspondence to the so-called Rich Domain Model (RDM) [4], where domain objects representing business entities are also carrying the business logic and are able to solve the business requirements [5]. While this approach might seem aligned with the basic principles of object-oriented software development, it also imposes limitations on the system code. Several authors are claiming that storing the logic in the domain model violates the separation of concern principle by making the domain objects not only responsible for storing the information, but also operating it [6]–[8]. These authors are advertising the usage of the so-called Anemic (or Anaemic) Domain Model (ADM), which can be best described with a single sentence “data are data”. One of the main ideas of ADM is the usage of business entities to solidly store the information. Business logic that is required for system to meet the actual requirements should in turn be contained at the business logic level.

The paper is focused on the discussion about advantages of anemic domain model and usage of the two-hemisphere model as a basis for such type of domain model generation. The next section discusses the differences between anemic and rich domain models. Section 3 describes two-hemisphere model-driven approach transition to the anemic model and shows the UML sequence diagrams created according to rich and anemic domain models. Section 4 offers the list of changes that should be made in the approach to have all the advantages of the anemic model. Finally, the last section contains the very brief conclusions and some of the areas for the future research.

II. Anemic vs Rich Domain Model

Rich data model is based around the basic principles of an object-oriented paradigm stating that objects can hold its properties along with behaviour [4]. It means that any object can be described with a dual nature – one part of which is data and the second is methods for working with these data. This idea is used in the current two-hemisphere model driven approach, for example, when data from the conceptual model are combined with the behaviour from the business process model in order to create classes holding both parts of the system description [9]. RDM basically says that objects should have

[*] Corresponding author’s e-mail: konstantins.gusarovs@gmail.com

this dual nature in order to perform their tasks in the software system. This gives some advantages in further code editing during the software system support – in order to change the data and algorithms that are working with these data; it is usually necessary to introduce changes only to a single class in a programming language code – the class that is responsible for both data storage and processing.

It is possible to see that ADM approach is, in turn, aligned with the Model-View-Controller (MVC) paradigm [10] that nowadays is widely adopted by multiple frameworks, e.g., AngularJS [11]. MVC offers to present the software system as a set of three components – Model responsible for storing the information and operating it, View responsible for Model rendering in a form suitable for the user interaction and a Controller that interconnects these two components. However, since being firstly presented MVC paradigm has undergone several changes. It is possible to see in [11] that the model is referred to only as data storage, and the business logic that is responsible for the business requirement implementation is being stored in services that are representing, cited "reusable business logic independent of views". This kind of concern separation is being sometimes referred to as a Model-View-Service-Controller (MSVC) model [7] and adds additional flexibility to the original MVC. MVC offered to split the system into the three main components to make those pluggable and replaceable [10], which means that the single Model-Controller composition can successfully serve several different Views – for example, one Web Service can be used by both desktop and a mobile application. Separation of the Model component into Model and Service with former responsible solely for the information storage and later – for the business logic provides another level of flexibility where single View-Controller-Service composition can use different models (e.g., different database engines) without the need for the code changes. By analysing the MSVC principles, it is possible to see that RDM would not fit inside since it ties both business information and business logic in a single component. ADM seems to be more elegant and flexible solution, while it might seem to be violating some of the OOP principles.

One might argue that MVC/MSVC approach is not "object-oriented enough", since it seems that all the parts of this model are separate blocks that are composed together and do not use the advantages of the object-oriented paradigm. However, the authors of this paper claim that OOP principles are still applied – only on a different level. It is possible to inherit the Model class and override their data storage behaviour. It is possible to do the same things on different MVC/MSVC levels by keeping the same class interfaces but changing what and how they do. Thus, this means that all the OOP principles are still being used and provide the same advantages that they do in RDM. However, in this case, the abstraction level is shifted a little bit towards higher separation of responsibilities, i.e., data storage and its processing are separate concepts in the system that can be changed independently.

III. Workflow Generation From the Two-Hemisphere Model

Fig. 1 shows an example of the two-hemisphere model for the room booking in a hotel. The model is developed by BrainTool [12] – a tool by this paper authors' research group, which gives the ability to create the two-hemisphere model and to generate the UML class and sequence diagrams from it [13]. The two-hemisphere model contains both the information that is necessary to work with in this system as well as information on how the data should be processed. Data are described with the help of the conceptual model that consists of the concepts and their attributes. These concepts are then used in the business process diagram that describes which processes consume what data, and what data do they produce.

This model describes a booking process in a hotel that starts with booking inquiry consisting of room details and book dates (from-to). Next, the system should check the room availability and decide if it is possible to satisfy the request. In case of a positive outcome, personal data are requested and booking is confirmed and stored in the database. Otherwise, booking request is rejected, and a user is asked to revise booking details. This process might repeat several times and could end up with booking cancellation or successful room search followed by already described personal data request and confirmation. The model describes processes that are required for the implementation of this business process as well as data that are utilised (in form of the conceptual model).

By looking at this example, it is possible to see that the model itself does not enforce the usage of RDM or ADM – it does not, for example, define responsibilities for the objects in the system, nor it defines the objects/classes at all. Thus, the authors state that it should be possible to transform this model using both data modelling approaches and yielding results that would be different from the data processing approach yet should describe the same business processes that happen in the system. To demonstrate this approach, the authors have chosen the UML sequence diagram [3] as a target model of transformation. The choice of this diagram has several advantages from the perspective of result demonstration – the UML sequence diagram shows which objects communicate one with another in the system, and which data do they pass between during their communication. The UML sequence diagram should also preserve the information that is described in the initial business process model, and it should be possible to trace it back in order to check the accuracy of the transformation itself.

The result of transformation of this model into the UML sequence diagram using the algorithm described in [10] is shown in Fig. 2 and it is based on ADM principles. This sequence diagram corresponds to the MSVC architecture having a single controller that communicates with the services and a user; services in turn communicate with the database; business entities, however, do not contain any business logic and are not present there. Fig. 3 provides an example of how the same process model could be transformed into the UML sequence diagram using an RDM approach. It is possible to see that business entities obtain the methods responsible for the business logic implementation. The authors of this paper claim that the ability to perform transformation from the two-hemisphere model to a similar sequence diagram (or even software code) would benefit the two-hemisphere model driven approach.

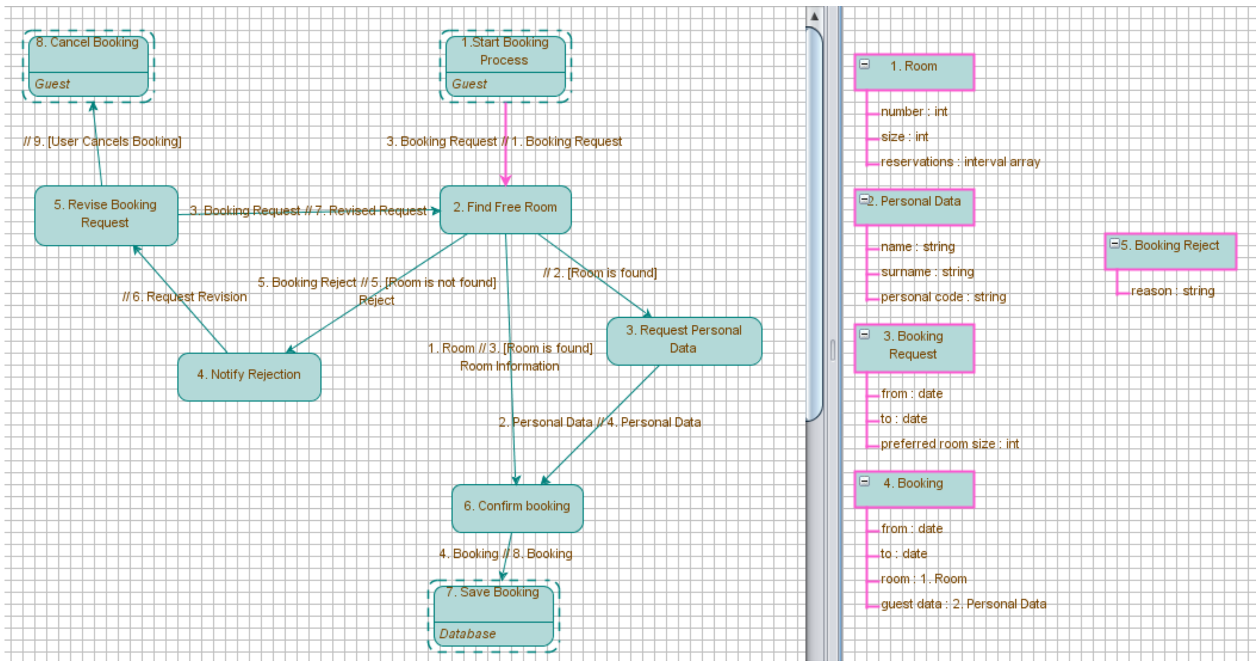

Fig. 1. Two-hemisphere model of room booking developed in BrainTool [12].

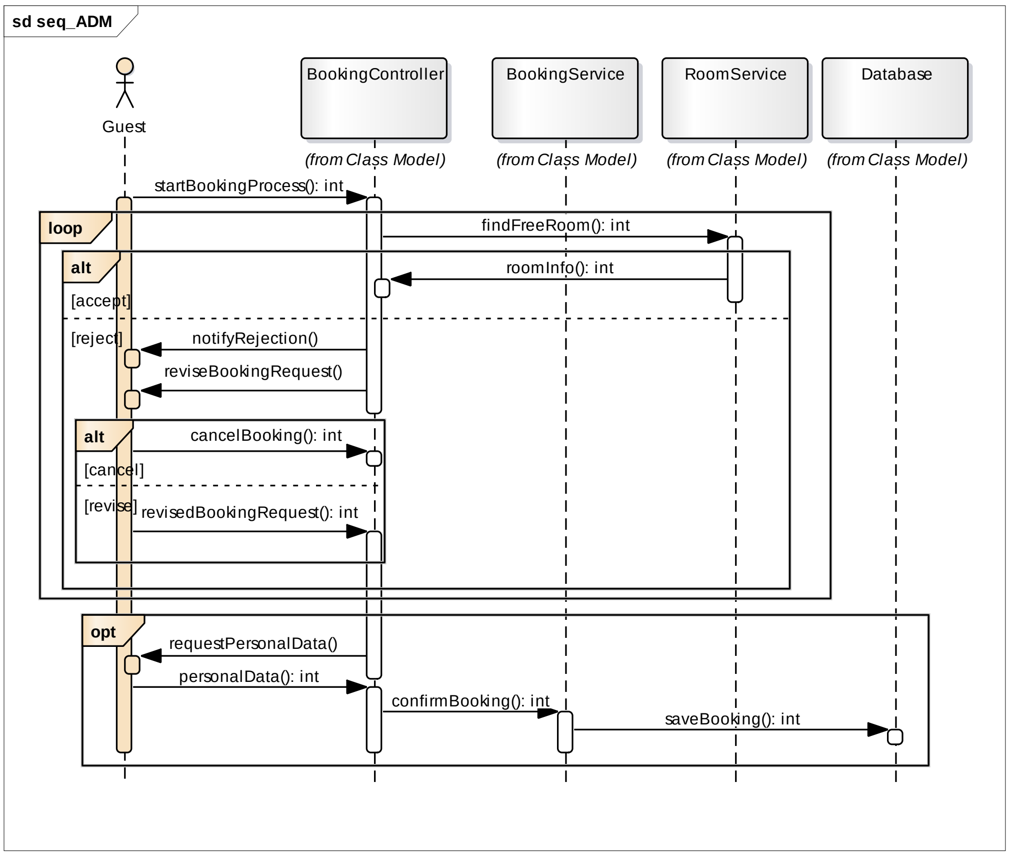

Fig. 2. Sequence diagrams for ADM.

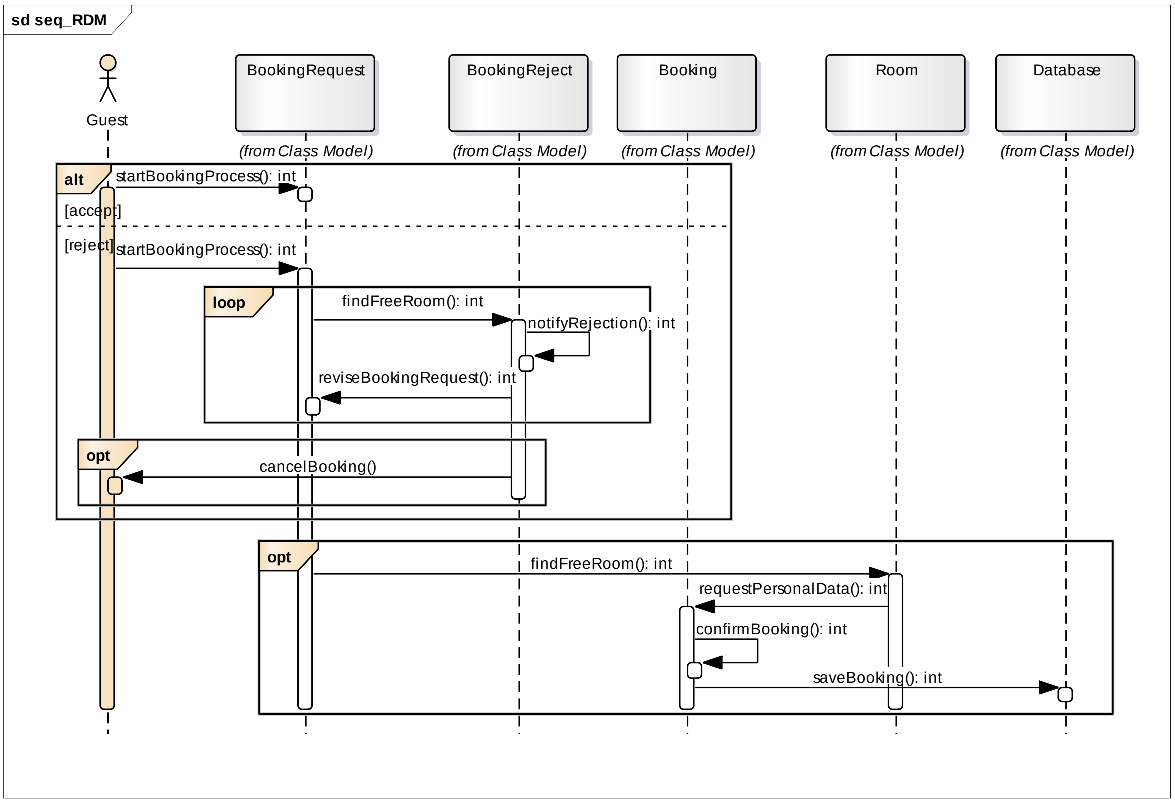

Fig. 3. Sequence diagrams for RDM.

By checking both the UML sequence diagrams against the initial business process model, it is possible to see that all the processes presented in it were moved to the appropriate transformation results. Their invocation sequence is also preserved and still can be traced to the initial data flows. Only difference between two resulting models is objects that are performing business logic as well as the data they pass between them. In case of RDM, data model itself (which is derived from the initial conceptual model) is responsible for the actions that take place in the system.

In case of ADM, data become "only data" and are passed between the classes that are responsible for their processing. In other words, an approach utilising ADM distinguishes between what is processed and what is responsible for processing. While in case of RDM, a booking object itself obtains personal data from a user; in case of ADM this is the responsibility of BookingService object. Other classes behave in a similar way. Thus, the main difference here is which object is responsible for business logic invocation. Rich Data Model states that objects can perform these operations, while Anemic Data Model insists on separation of domain classes and business logic classes, i.e., domain objects are used only to describe necessary data structures, thus becoming "only data".

To be able to perform such a transformation, the authors would like to propose several improvements to the transformation algorithm described in [14] as well as to the two-hemisphere model itself. Former transformation algorithm is based on considering the process diagram of the two-hemisphere model as a Finite State Machine (FSM). This allows applying to it FSM minimization algorithms, such as state reduction etc. The resulting minimized FSM, in turn, could be used for further processing. In [14], the authors proposed to create the regular expression out of it and later parse it to produce the resulting artefacts. However, several problems with such an approach were noted, and in this paper the authors would like to present a result of further study of such an approach.

During their research, the authors have defined several additional algorithms that can be used for FSM that represents process model processing. Due to the size limitations, those algorithms are not presented here, however, the next section and Fig. 4 provide a short insight into the results of its application to a process model.

IV. List of Improvements to Introduce in Two-Hemisphere Model

Fig. 4 shows the so-called workflow model of the appropriate process diagram. This model contains the information about process invocation sequence, data these processes receive as well as data these processes can produce. In the model itself, processes are not part of classes (they are not appropriate class methods), which means that it is possible to produce both RDM and ADM code from this model. This can be achieved via implementation of the next transformation step that could analyse processes in this model, their inputs and outputs and decide which of the classes should own the appropriate business process as its method. In case of RDM, such transformation is already present and discussed, for example, in [14] and [15].

```
(booking_request) = 1. start_booking_process();
repeat: {
  (room_info, reject) = 2. find_free_room(booking_request, revised_request);
  (reject) = 4. notify_rejection(reject);
  (revised_request, reject) = 5. revise_booking_request(reject);
}
disj{
  case1: {
    8. cancel_booking(reject);
  }
  case2: {
    (booking) = 3. request_personal_data(room_info);
    (booking) = 6. confirm_booking(booking);
    7. save_booking(booking);
  }
}
```

Fig. 4. Workflow model of the hotel booking process.

In case of ADM, however, such methods should be developed and are one of the directions for the future research in the area of two-hemisphere model-based approach.

However, such methods are not only direction for further studies. By inspecting this model, it is possible to see that it contains all the processes in a correct order grouped under loops and possible disjoints. However, further inspection of the workflow model will allow identifying some additional problems that should be solved in order to improve the transformation techniques and make the two-hemisphere model driven approach ADM-capable.

Additional limitations that can be noted on the workflow model presented in Fig. 4 are the following.

- Processes 4 and 5 should only be invoked if the room was not found, i.e., process 2 invocation resulted in rejection. There is no such information in the initial two-hemisphere model and, thus, this information is not represented in the workflow model.
- Process 5 should not consume any data flows since it does not require any additional data. In this case, it is possible to see that process 5 is actually similar to the external process – it only provides the information but does not consume anything. Current model notation does not allow for such cases to appear.
- Process 8 should be invoked only as a result of the invocation of process 5, if a user chooses to cancel booking. Again, it should not consume any data flows.
- Processes 3 and 6 can be invoked in a parallel.

To successfully resolve these problems, the authors propose the following changes to the two-hemisphere model.

- It should be possible for a single data flow to carry more than one concept. In the presented model, this is not required, however, in other cases this might be necessary.
- It should be possible for a data flow to have an execution condition, i.e., guard. This will allow for further analysis of the generated workflow to produce more accurate artefacts, such as UML sequence diagrams or the actual code of the system.
- It is also possible that process or even a sequence of processes might have an execution condition – again, in real life branches in the program source code usually consists of more than one operation. Even if data flow is executed, it should not mean that the process it is directed into should be executed as well.
- The given example contains a single use case and all the necessary actions can be put in a single service. However, in real life one system might be described with multiple process diagrams and some of these might contain the same processes. Using this information, it should be possible to define common service signatures.
- It is necessary to introduce a new element to the two-hemisphere model – control flow. Control flow is like the existing data flow; however, it should not carry any data. It should only be responsible for the definition of actual process invocation sequence.

V. Conclusion And Future Work

In this paper, the authors describe several limitations that are currently present in the two-hemisphere model driven approach, analyse those and offer several improvements to both the initial model being used by the method and the algorithms involved in the approach. These improvements are offered based on the previous work in the same area as well as the analysis of artefacts being produced by the approach at the current moment. The authors plan to introduce these improvements to the method soon and analyse how these will affect the resulting artefacts.

The paper also discusses different ways of modelling the data in the software systems – the so-called Rich and Anemic data models that are widely used and differ in the ways how the static part of an object-oriented system (i.e., data structures) and its dynamic part (i.e., algorithms for the appropriate data processing) are represented in the resulting source code. In this paper, the authors show that the same two-hemisphere model might be used to produce UML sequence models that support one or another approach.

From the authors' point of view, it is important to have all the transformation techniques for the two-hemisphere model flexible enough, since it might be required to produce both ADM and RDM based code. Therefore, when working on the improvements to the method, the authors also plan to analyse the best way to achieve this.

References

[1] L. Leimane, O. Nikiforova, "Mapping of Activities for Object-Oriented System Analysis", *Applied Computer Systems*, 2018, vol. 23, no. 1, pp. 5–11. https://doi.org/10.2478/acss-2018-0001

[2] O. Nikiforova, "System Modeling in UML with Two-Hemisphere Model Driven Approach", *Applied computer systems*, 2010, vol. 41, no. 1, pp. 37–44. https://doi.org/10.2478/v10143-010-0022-x

[3] OMG, UML Unified Modeling Language Specification. Available at http://www.omg.org

[4] K. Cemus, T. Cerny, L. Matl, and M. J. Donahoo, "Aspect, Rich and Anemic Domain Models in Enterprise Information Systems," *42nd International Conference on Current Trends in Theory and Practice of Computer Science*, 2016. https://doi.org/10.1007/978-3-662-49192-8_36

[5] K. Gusarovs, O. Ņikiforova, "Workflow Generation from the Two-Hemisphere Model", *Applied Computer Systems*, 2017, vol. 22, pp. 36–46. https://doi.org/10.1515/acss-2017-0016

[6] L. Marques, "A defense of so-called anemic domain models". Slides of D-Lang-Silicon-Valley Meetup @ January 28, 2016. Available at http://files.meetup.com/18234529/luis_marques_anemic_domain_models.pdf

[7] E. Evans, *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[8] M. Fowler, "Anaemic Domain Model." Available at http://www.martinfowler.com/bliki/AnemicDomainModel.html

[9] N. El Marzouki, Y. Lakhrissi, O. Nikiforova, M. El Mohajir, "The application of an automatic model composition prototype on the-Two hemisphere model driven approach," *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems*, WITS, 2017. https://doi.org/10.1109/WITS.2017.7934673

[10] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80", *Journal of Object-Oriented Programming*, 1988, vol. 1, no. 3, pp. 26–49.

[11] AngularJS, Developer Guide: Conceptual Overview. Available at https://docs.angularjs.org/guide/concepts

[12] O. Nikiforova, L. Kozacenko, D. Ungurs, D. Ahilcenoka, A. Bajovs, N. Skindere, K. Gusarovs, M. Jukss, "BrainTool v2.0 for Software Modeling in UML", *Applied Computer Systems*, 2014, vol. 16. no. 1, pp. 33–42. https://doi.org/10.1515/acss-2014-0011

[13] O. Nikiforova, K. Gusarovs, "Comparison of BrainTool to Other UML Modeling and Model Transformation Tools," *AIP Conference Proceedings*, vol. 1863, 2017. https://doi.org/10.1063/1.4992503

[14] O. Nikiforova, K. Gusarovs, A. Ressin, "An Approach to Generation of the UML Sequence Diagram from the Two-Hemisphere Model", *Proceedings of the Eleventh International Conference on Software Engineering Advances*, 2016.
K. Gusarovs, O. Nikiforova, A. Giurca, "Simplified Lisp Code Generation from the Two-hemisphere Model," *Procedia Computer Science*, 2016, vol. 104, pp. 329–337. https://doi.org/10.1016/j.procs.2017.01.142

Oksana Nikiforova received the Doctoral degree in information technologies (system analysis, modelling and design) from Riga Technical University, Latvia, in 2001.
She is presently a Professor at the Department of Applied Computer Science, Riga Technical University, where she has been on the faculty since 1997. Her current research interests include object-oriented system analysis, design and modelling, especially the issues related to Model Driven Software Development.
E-mail: oksana.nikiforova@rtu.lv
ORCID iD: https://orcid.org/0000-0001-7983-3088

Konstantins Gusarovs received the Master degree in computer systems from Riga Technical University, Latvia, in 2012. He is presently the fourth year Ph. D. Student and Researcher at the Department of Applied Computer Science, Riga Technical University, as well as Java Developer at C.T.Co Ltd. His current research interests include object-oriented software development and automatic acquisition of program code.
E-mail: konstantins.gusarovs@gmail.com