DE GRUYTER OPEN

Applied Computer Systems doi: 10.1515/acss-2015-0014

2015/18

A Prototype of Description Language for the Two-Hemisphere Model

Konstantins Gusarovs[1], Oksana Nikiforova[2], Maris Jukss[3]
[1,2] *Riga Technical University, Latvia*, [3] *McGill University, Canada*

***Abstract* – Nowadays, it is a modern trend to develop a CASE tool for system modelling with an ability to transform models defined in different notations and also to generate a program code. However development of such a tool often involves experimentation with transformation algorithms that may require changes to the source model structure. Since CASE tools are basically used to represent a model in diagram's form, implementing experimental changes in a modelling tool can require additional effort. In order to solve this problem, authors propose a way of describing the two-hemisphere model using Domain Specific Language. This paper covers the language's syntax as well as provides an example of the two-hemisphere model defined with its help.**

***Keywords* – Two-hemisphere model, Domain Specific Language, system modelling, Extended Backus–Naur Form.**

I. Introduction

In the last 10 years business process modelling has become one of the most popular trends in software development [1]. Models are being used in various stages of software development process [2], however, most often they are being applied at the initial business analysis process. Usually, the system model is organized as a set of diagrams where specific notation is defined for each diagram thus specifying diagram's syntax and semantics. Various types of notations were developed over the last decade and researchers seem to be concentrating their efforts in developing new notations as well as appropriate transformations in order to use those models in practice [1]. Continuous improvement of the transformation method can require several changes to the source models, however, before introducing such changes it is often necessary to perform various experiments in order to determine if the change proposed will result in actual method's improvement. Since many CASE tools are using graphical notation representing models in a diagram form, such experimentation can require additional work in order to introduce a method of creating diagrams with enriched notation.

In order to solve this problem, i.e., necessity to enrich the notation without introducing too many changes into the CASE tool, it is possible to use various methods that will enable experimentation on existing models in order to enrich them. Usually, the CASE tool consists of several isolated components: model editor, model repository, model validation module, transformation editor and transformation repository [3]. In order to change a model's notation and perform transformation tests it is not necessary to apply changes to all of those modules. It is possible to achieve results only by changing the model repository as well as transformation repository. In cases when models are stored using one of the plain text based formats, e.g., XML, it is possible to achieve the result by manually changing the model's structure. However, this approach cannot be used in case when the model is stored using binary formats. Even the model stored in XML file can be hard to modify by human – it could have complex structure and a single element of the model could be represented by several locations of the file. In order to solve this problem authors propose to use Domain Specific Language (DSL) for model description that can be modified with minimal effort, demonstrate the ability to develop such language for the two-hemisphere model and apply it to the two-hemisphere model based approach.

The paper is structured as follows. Domain specific languages are described in the second section. The two-hemisphere model driven approach is described in the third section. The proposed domain specific language is described in the fourth section. The fifth section covers the technologies used by authors for language development, and the sixth section provides an example of the two-hemisphere model defined with the help of the proposed language. Finally, the seventh section contains authors' conclusions as well as covers several areas of the future work.

II. Domain Specific Languages

In comparison to General Purpose Languages (GPL) Domain Specific Languages are specialized for describing concrete problem domain/domain of knowledge [4], [5]. DSLs usually trade the generality of the programming language in favor of expressiveness in a limited area. Use of a domain specific language enables additional verification, analysis and optimization possibilities that would be harder to achieve using GPL [5]. In case of business modelling it is possible to define Extensible Markup Language (XML) as a general purpose language and its dialects as a DSL – while general XML offers more flexibility and can be used to describe any information (in this case a model), its specific dialect is more expressive and describes problem domain more effectively.

While the development of DSL can offer additional benefits it could be costly and the resulting language should be able to cover all the development costs [5]. It is worth developing one when its potential users have more domain knowledge and less programming expertise which applies to business modelling.

While working on the improvements of the two-hemisphere model driven approach authors of this paper have defined the necessity for such a language since continuous development also includes changes to a metamodel that have to be

evaluated. As a result it is vital to have a way of representing the two-hemisphere model and its new features in a human readable form that can be rapidly integrated into the model and transformation repositories in order to perform tests and an analysis. Since the two-hemisphere model itself is a set of diagrams, it is quite a time-consuming task to implement experimental features in a functional model editor. Considering this, authors decided that it is worth implementing a DSL language for describing the two-hemisphere model.

In order to develop a domain-specific language it is necessary to define the basic element of the problem domain and create their appropriate representations using a chosen syntax. After all the basic constructs are defined, validation and semantical rules have to be developed thus finalizing the DSL's design. Depending on DSL's syntax it is possible to express it using different notations. For example, when a domain-specific language is XML based, it is possible to specify it using XML schemas [6]. Text based DSL can be described using Extended Backus-Naur Form (EBNF) [7]. Authors of this paper are using the latter approach for the syntax definition.

In case of Model-Driven Software Development (MDSD) the graphical model itself is usually considered the domain specific language [1]–[5] so there are very few related works on the topic. Authors consider that this is happening due to the large amount of purely-theoretical researches in this area. There are very few model-driven approaches having a tool supporting the transformations proposed by their authors since the development of such a tool is a task requiring a lot of effort. There is also existing a risk connected to such a development of such a tool – in case the initial theories and proposed approaches fail to prove being correct it would require rewriting of the tool's code. This is proven also by the experience of authors in the development of the CASE tool supporting model transformations for the two-hemisphere model driven approach which is described in the next section.

III. The Two-hemisphere model Driven Approach

The two-hemisphere model was first presented by Oksana Nikiforova and Marite Kirikova in 2004 [8]. The first attempt to transform the two-hemisphere model into a Unified Modelling Language (UML) class diagram required the use of the intermediate model, i.e., the UML sequence diagram, in order to gain the results. Later the approach was improved by Oksana Nikiforova and Natalia Pavlova in 2008 [9] by using the UML communication diagram as well as enriching the resulting UML class diagram with more elements.

Several versions of a tool, named BrainTool, supporting the two-hemisphere model driven approach were developed in Riga Technical University in 2012 [10] and 2013–2014 [11] further improving transformation algorithms, eliminating the necessity for the intermediate results as well as introducing new result of the transformation – the UML sequence diagram, and enriching resulting models with additional information. During the development of both initial and following versions of the BrainTool the model itself was improved and enriched with additional information (e.g., single performer of the external process – see the next paragraph which was changed to many) which resulted in a change of metamodel and caused large amount of the model editor. Another problem that authors encountered was the inability to prove that the changes to metamodel would actually be useful before implementing those in the tool itself. This means that the tool has to support the new untested metamodel in order to test it. All this leads to the large amount of work to be done in order to simply test some theories. As a result of the lessons learned during the tool development, authors state that another way of model representation is required. The requirements for this way of representation are the following:

- It should be possible to change the metamodel with the least effort that graphical representation incorporates. Graphical representation is easy to read but even a small change of it requires several changes in both the model repository (in order to store the new information) and the model editor (in order to make it editable).
- The model representation should be easy to read and conceive by a human thus keeping the advantages of the graphical model representation.
- It should be possible to change the metamodel without changing the tool itself so the actual tool would receive only the tested changes thus eliminating the necessity of code rollbacks if some improvement fails to prove its worth.

As a result authors have made a decision of using the text-based DSL for model representation.

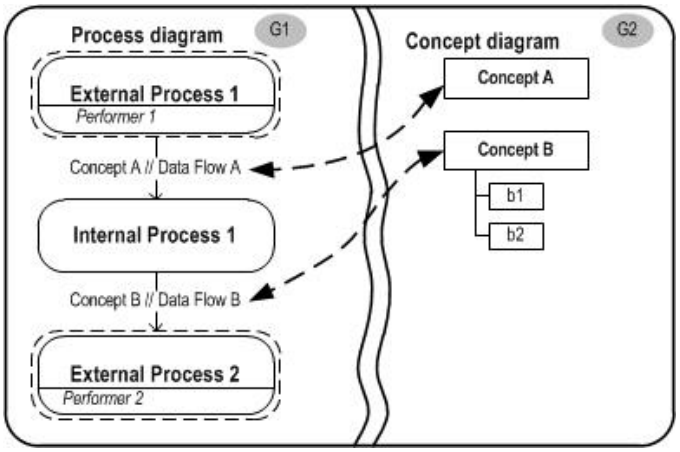

Fig. 1. Notation of the two-hemisphere model.

The two-hemisphere model consists of two models used in the transformation process serving as a source for the produced UML class and sequence diagrams. The overall model structure is presented in Fig. 1. Graph G1 in Fig. 1 is referred as the process model and consists of the process nodes connected with data flows. The data flows in turn are representing information exchange between those processes and are linked with the concepts of the concept model (Fig. 1, Graph 2). Each of the concepts can be described by its name and attributes, where an attribute consists of the name and

type. It is worth mentioning that in addition to using primitive types (e.g., integer, string etc.) it is possible to set the attribute's type to one of the concepts as well as define cardinality of the attribute – defining if it is referring to a single appropriate object's entry or array. In turn each of the processes in the process model can be defined as an internal or external process. The main difference between internal and external processes is a restriction on incoming/outgoing data flows – external processes can only produce or accept them whilst internal process both consume and produce information. It is possible to define one or more performers for a process.

Using this notation it is possible to identify the main elements of the model in order to create appropriate DSL constructions for the element representation. Such elements are process, concept and data flow. The two-hemisphere model driven approach states that the system can be described using several different process models but only one common concept model is present. Using this information it is possible to define the final DSL element – process model. As described in the following section this minimal set of four basic elements is enough to define two-hemisphere models with the help of DSL.

IV. The Two-Hemisphere Model Domain Specific Language

As defined in the previous section the two-hemisphere model DSL language contains four elements: concept, process, data flow and process diagram. Before describing these elements it is necessary to introduce three special syntax constructions used in later definitions. These constructions are presented in Fig. 2.

```
character          = any Unicode character
                     except '\' and '"';
escape_sequence    = '\b' | '\t' | '\n' |
'\f'
                   | '\r' | '\"' | '\'';
string_character   = character
                   | escape_sequence;
string             =
                '"' string_character* '"';

identifier_letter = 'A' - 'Z' | 'a' - 'z'
digit             = '0' - '9';
identifier_letter_or_digit =
                    identifier_letter
                  | digit;
identifier        = identifier_letter
             identifier_letter_or_digit*;

glue              = 'WITH' | 'AND';
```

Fig. 2. Common DSL constructions.

`String` is used to represent character sequences and its syntax corresponds to a string literal syntax in the Java programming language. It is a sequence of Unicode characters enclosed in double quotation marks with escape sequences for specific characters – such as tabulations, line breaks etc.

`Identifier` is used to name the elements of the two-hemisphere model and its syntax corresponds to the identifier syntax in various programming languages (e.g., Java, C, Pascal etc.) with the exception of underscore symbol not being allowed. `Identifier` consists of Latin letters and numbers and should start with a letter.

`Glue` is a special token that has no specific semantical meaning but is used in the two-hemisphere model element definition when the element's internal structure is being described. `Glue` is one of the following words: "`WITH`", "`AND`". Both words have the same meaning in a parsing context but can be used in order to increase the expressiveness of the model definition.

The next element of the DSL language describes the concept and its attributes. EBNF definition of this element is presented in Fig. 3.

```
cardinality      = 'SINGLE' | 'ARRAY'
                   | 'COLLECTION';
cardinal_def     = 'HAVING' cardinality
                   'CARDINALITY';

type             = identifier;
concept_id       = identifier;
attribute_name   = string;

name_def         = glue 'NAME' string;
description_def  = glue 'DESCRIPTION'
                   string;
comment_def      = glue 'COMMENT'
                   string;
attribute_def    = glue 'ATTRIBUTE'
                   attribute_name '('
                   type ')' [cardinal_def];
concept_param    = name_def
                   | attribute_def
                   | description_def
                   | comment_def;

concept          = 'DEFINE CONCEPT'
                   concept_id
                   concept_param*
                   'END CONCEPT';
```

Fig. 3. Concept definition in the proposed DSL language.

Concept definition starts with keywords "`DEFINE CONCEPT`" and consists of one or more concept field definitions. Concept field is either its name, description, comment for it or one of the concept attributes. For each of the attributes its name and type have to be defined. It is also possible to define appropriate attribute's cardinality which can take one of the following values: "`SINGLE`", "`ARRAY`", "`COLLECTION`". Concept definition is terminated by keyword "`END CONCEPT`". In case there are multiple concept name, description or comment definitions parsing error is being triggered. In case when the concept's name is not defined it is assigned automatically in a form "`Concept X`" where `X` is an auto-incremented number starting from 1.

Description of the process in the proposed DSL language is similar to a concept definition and starts with keyword "DEFINE PROCESS". Possible fields in a process definition include process type definition (i.e., internal/external), one or more performers definition and name definition. If there are several name, type, comment or description definitions, parsing error is being triggered. When the process name is not defined it is being assigned automatically in the way similar to the automatic concept name assignment. Process definition ends with keyword "END PROCESS". EBNF definition of a process element is shown in Fig. 4.

```
process_type      = 'INTERNAL' | 'EXTERNAL';
process_id        = identifier;

process_type_def  = glue 'TYPE'
                    process_type;
performer_def     = glue 'PERFORMER' string;
process_param     = name_def
                  | description_def
                  | comment_def
                  | process_type_def;

process           = 'DEFINE PROCESS'
                    process_id
                    process_param*
                    'END PROCESS';
```

Fig. 4. Process definition in the proposed DSL language.

EBNF definition of a data flow element is shown in Fig. 5. Data flow fields are enclosed between keywords "DEFINE DATA FLOW" and "END DATA FLOW" and may contain name, description, comment definitions as well as a definition of a concept that is assigned to a data flow. Similarly to concept and process definitions, duplication of name, comment and description fields is not allowed however the data flow definition introduces additional restrictions – processes and concepts used in it should be defined. In case when the name is not present, it is automatically generated.

```
data_flow_id      = identifier;

concept_ref_def   = glue 'CONCEPT'
                    concept_id;
dataflow_param    = name_def
                  | description_def
                  | comment_def
                  | concept_ref_def;

data_flow         = 'DEFINE DATA FLOW'
                    data_flow_id
                    'FROM' process_id
                    'TO' process_id
                    dataflow_param*
                    'END DATA FLOW';
```

Fig. 5. Data flow definition in the proposed DSL language.

The last DSL's element is the definition of a single process model. Its EBNF definition is shown in Fig. 6. It features the same basic fields and rules applied to them as the previously described elements – name, comment and description. Field specific to this elements is a reference to a defined process. If parser is unable to find the referenced process, an error occurs. Data flows are not being included in the process model definition since it is possible to identify those using processes included in the process model. If it is impossible to define data flow's adherence to a single process diagram (i.e., the processes it connects belong to different process models), an error is triggered.

```
process_model_id  = identifier;

process_ref_def   = glue 'PROCESS'
                    process_id;
process_m_param   = name_def
                  | description_def
                  | comment_def
                  | process_ref_def;

process_model     = 'DEFINE PROCESS MODEL'
                    process_model_id
                    process_m_param*
                    'END PROCESS MODE';
```

Fig. 6. Process model definition in the proposed DSL language.

V. Implementation Details

After defining the proposed DSL's structure authors analyzed the possibilities to implement its parser in order to use it in the improvement of the two-hemisphere model driven approach. Since language syntax is defined using EBNF notation it would be possible to create a recursive descent parser [7], however, current state of art in a parser development allows to use tools that are able to generate the parser's code using the rules defined in a syntax similar to EBNF [12], [13]. Authors have chosen to use the ANTLR tool [13] that allows generating Java code for parsing defined grammar rules. The main reason for choosing this tool was the fact that it produces Java code – the last version of the BrainTool [11] was developed using Java. ANTLR uses grammar rules that are defined in a notation similar to EBNF. The example of ANTLR's rules, that correspond to the `identifier` construction, is shown in Fig. 7.

```
fragment
IdentifierLetter
    :  [a-zA-Z]
    ;

fragment
IdentifierLetterOrDigit
    :  [a-zA-Z0-9]
    ;

Identifier
    :  IdentifierLetter
       IdentifierLetterOrDigit*
    ;
```

Fig. 7. Identifier construction defined in ANTLR.

One of the main advantages over the raw EBNF notation is a possibility to use character classes similar to the ones used in regular expressions. Java code generated by the ANTLR tool contains the class that implements observer pattern and should be extended in order to process grammar constructions that appear in a source code being parsed. For each grammar construction ANTLR creates two methods called `enterGrammarConstruction` and `exitGrammarConstrution` that will be invoked when the parser begins and finishes specific grammar construction processing. Examples of such methods generated for `process_type` construction are shown in Fig. 8.

```
void enterProcessType(@NotNull
        THMLParser.ProcessTypeContext ctx);

void exitProcessType(@NotNull
        THMLParser.ProcessTypeContext ctx);
```

Fig. 8. Observer methods generated by ANTLR.

In order to transform DSL specification of a two-hemisphere model it is necessary to extend ANTLR's generated observer class and override its methods responsible for appropriate model element processing. As a result new element can be added to a model by performing steps: 1. define new ANTLR grammar rule; 2. define parsing logic in observer class.

This way it is possible to rapidly change the model notation, test it and analyze the results gained. It is not necessary to change the model editor and possible to perform experiments on a two-hemisphere model by changing its metamodel using fewer actions than it would be when using the CASE tool. As it was already stated, CASE tool consists of model editor, model repository, model validation module, transformation editor and transformation repository [3]. DSL language combines the first three elements into a set of grammar rules and parsing process observer that can be changed with less effort comparing to the tool with a graphical interface.

VI. Application of the Proposed DSL for a Simple Example of the Two-Hemisphere Model

In order to demonstrate proposed capabilities of DSL authors present a simple two-hemisphere model that corresponds to an abstract use case of adding entry into the database. It contains a single concept that describes a structure of the entry being added and a single process model with three processes: receive entry from user, validate it and save in a database. This model created by the latest version of the BrainTool is shown in Fig. 9 and an appropriate DSL definition of the same model is presented in Figure 10.

While sharing the same information as the graphical two-hemisphere model, DSL definition is significantly larger and might be harder to read. However, DSL syntax modification requires less effort than graphical notation modification. Another information that is currently missing from the proposed DSL but is present in CASE tool supporting the two-hemisphere model driven approach, is information on element geometry – this information is not vital for model definition and transformation and is only required for displaying the two-hemisphere model.

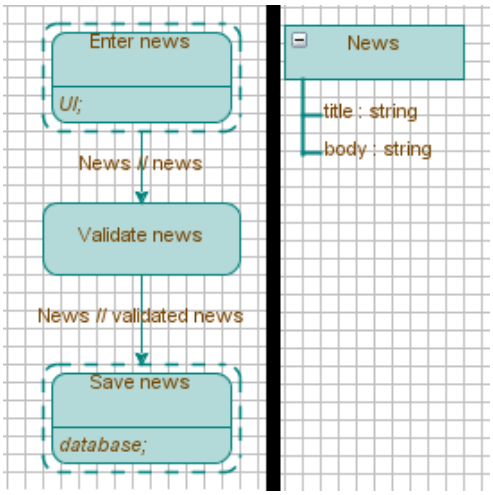

Fig. 9. Example of the two-hemisphere model.

```
DEFINE CONCEPT News
   WITH NAME "News"
   AND ATTRIBUTE "title"(string)
   AND ATTRIBUTE "body"(string)
END CONCEPT

DEFINE PROCESS EnterNews
   WITH NAME "Enter news"
   AND PERFORMER "UI"
   AND TYPE EXTERNAL
END PROCESS

DEFINE PROCESS ValidateNews
   WITH NAME "Validate news"
END PROCESS

DEFINE PROCESS SaveNews
   WITH NAME "Save news"
   AND PERFORMER "database"
   AND TYPE EXTERNAL
END PROCESS

DEFINE DATA FLOW EnteredNews FROM EnterNews
  TO ValidateNews
   WITH NAME "news"
   AND CONCEPT News
END DATA FLOW

DEFINE DATA FLOW ValidatedNews FROM
  ValidateNews TO SaveNews
   WITH NAME "validated news"
   AND CONCEPT News
END DATA FLOW

DEFINE PROCESS MODEL ValidateAndSaveNews
   WITH PROCESS EnterNews
   AND PROCESS ValidatedNews
   AND PROCESS SaveNews
END PROCESS MODEL
```

Fig. 10. DSL definition of the sample model.

VII. Conclusion And Future Work

This paper presents the DSL language developed for representing a two-hemisphere model. The proposed language allows representing the model in a text form using specific syntax that is described in the paper using EBNF notation. The proposed DSL allows a two-hemisphere model definition and its validation based on semantic rules. While the proposed domain-specific language allows the definition of the two-hemisphere model it stores no information on model element geometry that is currently being stored by the BrainTool. However, the proposed DSL can be modified in a rapid way which is usable in both model and its transformation improvement enabling more experimentation possibilities than the graphical notation of the model. The proposed DSL language provides wide possibilities for future use and experimentation. The future work that the authors are planning could include DSL enrichment with additional elements that are currently missing from it. It is possible to define syntax constructions for representing the geometry data for the elements being defined, thus allowing the usage of DSL as the main way of storing the two-hemisphere model. In this case it would be possible to integrate the DSL into the before mentioned BrainTool CASE tool.

Another direction of the future work includes the two-hemisphere model and its transformation improvement using capabilities of the proposed language. In order to enable this, it is necessary to integrate the proposed DSL into the BrainTool. Since the proposed solution combines model repository, model editor and model validator and produces Java language objects compatible with the BrainTool's implementation, all the changes made to the transformation modules would mainly consist of actual improvements to the approach.

It is also possible to enrich the proposed DSL with constructions that will support UML artefacts such as a class diagram or a sequence diagram. Such an enrichment of the language proposed will allow its usage in other tools and approaches and will increase the contribution of this research.

To sum up, the proposed way of the two-hemisphere model definition has the following advantages over the XML model description used in the BrainTool: it is easy to modify the metamodel enriching the two-hemisphere driven model approach with new capabilities, models can be edited without necessity to use graphical CASE tool; large amount of experimentation with different model elements is enabled.

As a final conclusion authors would like to state that the main goal – to create a new way of two-hemisphere model representation that will allow to easily modify its metamodel and test new features – was reached. The authors plan to widely use the proposed DSL in their future research in the model-driven software development area.

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] W.P.M. van der Aalst, "Business process management: A comprehensive survey," *ISRN Software Engineering*, vol. 2013, 2013. http://dx.doi.org/10.1155/2013/507984

[2] M. Brambilla, J. Cabot, M. Wimmer, *Model-Driven Software Engineering in Practice*. 1ed. USA: Morgan & Claypool Publ., 2012.

[3] A. Kleppe, J. Warmer, W. Bast, "*MDA Explained: The Model Driven Architecture – Practise and Promise*, Addison-Wesley, 2003, 170 p.

[4] L. Benoît, C.-E. Jitia, E. Jouenne, "DSL classification," *OOPSLA 7th workshop on domain specific modeling*. 2007.

[5] M. Mernik, J. Heering, A. M. Sloane, "When and how to develop domain-specific languages," *ACM Computing Surveys*, vol. 37(4), pp. 316–344, 2005. http://dx.doi.org/10.1145/1118890.1118892

[6] W3C XML Schema [Online]. Available: http://www.w3.org/XML/Schema [Accessed: Oct. 6, 2015].

[7] D. Grune, C.J.H. Jacobs, *Parsing Techniques – a Practical Guide*. Ellis Horwood, Chichester, England, 1990.

[8] O. Nikiforova., M. Kirikova, "Two-hemisphere model Driven Approach: Engineering Based Software Development," in *Scientific Proc. of CAiSE 2004 (the 16th Int. Conf. on Advanced Information Systems Eng.)*, pp. 219–233, 2004.

[9] O. Nikiforova, N. Pavlova, "Development of the Tool for Generation of UML Class Diagram from Two-hemisphere model," in *Proc. of The 3rd Int. Conf. on Software Eng. Advances*. Mannaert H., Dini C., Ohta T., Pellerin R. (Eds.), *IEEE Computer Society*, CPS, pp. 105–112, 2008. http://dx.doi.org/10.1109/icsea.2008.37

[10] O. Nikiforova, K. Gusarovs, O. Gorbiks, N. Pavlova, "BrainTool. A Tool for Generation of the UML Class Diagrams," in *Proc. of the Seventh Int. Conf. on Software Eng. Advances*, Mannaert H. et al. (Eds), IARIA ©, Lisbon, Portugal, November 18-23, 2012, pp. 60-69 (Scopus)

[11] O. Nikiforova, L. Kozacenko, D. Ungurs, D. Ahilcenoka, A. Bajovs, N. Skindere, K. Gusarovs, M. Jukss, "BrainTool v2.0 for Software Modeling in UML," *Scientific Journal of RTU: Applied Computer Systems*, Grundspenkis J. et al. (Eds), vol. 16, 2014, pp. 33–42. http://dx.doi.org/10.1515/acss-2014-0011

[12] The LEX & YACC Page [Online]. Available: http://dinosaur.compilertools.net/ [Accessed: Oct. 8, 2015].

[13] ANTLR [Online]. Available: http://www.antlr.org/ [Acc: Oct. 8, 2015].

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

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 Riga Technical University. 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

Maris Jukss is a third year PhD student at the Modelling, Simulation and Design Lab in the 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