DE GRUYTER OPEN

Applied Computer Systems

doi: 10.1515/acss-2014-0011

2014/16

BrainTool v2.0 for Software Modeling in UML

Oksana Nikiforova[1], Ludmila Kozacenko[2], Dainis Ungurs[3], Dace Ahilcenoka[4], Andrejs Bajovs[5], Nadezda Skindere[6], Konstantins Gusarovs[7], Maris Jukss[8], [1–7]*Riga Technical University, Latvia*, [8]*McGill University, Canada*

***Abstract* – It is a modern trend to develop a CASE tool for system modeling with an ability to transform models defined in different notations and also to generate a program code. Such a system modeling tool tries to bridge the gap between the system specification and the software components. A tool called BrainTool has been developed for generation of the UML diagrams from the initial presentation of problem domain by the two-hemisphere model. The paper presents the main components of BrainTool and compares it to other system modeling tools.**

***Keywords* – BrainTool, system modeling tool, two-hemisphere model, Unified Modeling Language (UML).**

I. Introduction

Models play an important role in the development of software systems. Currently, models and model transformations are the central component in software development and it is clear that the importance of models will increase [1]. Models can be used to specify the system in a graphical view, understandable to analysts, developers and even customers. Usually, the system model is organized as a set of diagrams, where specific notation is defined for each diagram and regulates diagram syntax and semantics. System models are abstractions that portray the essentials of a complex problem or structure by filtering out non-essential details, models make the problem easier to understand. Thus, the systematic approach to derivation of the system model from the information about the problem domain and the tool supporting the automation of this process is strictly required. Forester's research [2] confirms that tools to support models and modeling at the initial stage of software development are the modern trend in business process modeling and analysis. Therefore, the focus of the automation of software development is shifted from automatic code generation to the automatic modeling of the system itself.

The Unified Modeling Language (UML) [3] in this context is used to model system specification and serve as a "bridge" between the information about problem domain and the information required for definition of software components and their architecture. Currently, researchers are trying to achieve a high enough level of automation in the creation of the core UML diagrams and their derivation from information about the problem domain. Moreover, an increasing number of developers admit the necessity to model system at the initial stage of the software development project, and the models are increasingly used to specify the system and its processes at the business level [2], [4].

Authors of the paper offer so called two-hemisphere model driven approach [5] for the generation of the core UML diagrams, namely, class and sequence diagram. The goal of this paper is to present current research results achieved in the transformation of the two-hemisphere model into the UML diagrams and to introduce the tool supporting the approach.

The paper is structured as follows. The required functionality and main components of the tool suitable for system modeling and model transformation at the different levels of abstraction are specified in Section II. The components needed to implement such a tool are defined in Section III, where they are expressed by the components of BrainTool – the tool for the creation of the two-hemisphere model, UML diagrams generation and export them to some UML compatible tool. The authors analyse the variety of modern system modelling and code generation tools and define the position of BrainTool among them in Section IV. In the conclusion the authors discuss the technical solutions and lessons learned in solving the task of model transformation ant its support by a tool and stress the necessity to elaborate the area of tool development and model usage during software development.

II. The Main Functions and Components of a Model Transformation Tool

Computer-Aided Software Engineering (CASE) technologies are used in designing sophisticated tools to automate the software development process as much as possible. It is particularly useful where teams of engineers who may not share the same physical space design major software products. The goal of introducing CASE tools is the reduction of the time and cost of software development and the enhancement of the quality of the systems developed. CASE tools can be divided in the following groups: Requirement Analysis Tool, Structure Analysis Tool, Software Design Tool, Code Generation Tool, Test Case Generation Tool, Document Production Tool, and Reverse Engineering Tool [6].

According to [7], model transformation is the process of converting one model to another model of the same system. Tools should posses certain functionality to ensure their use and success. In [8] necessary functionality of the transformation tools is presented. One of the first is the ability to perform create/read/update/delete operations. It means that having the possibility to create new models and transformations or update existing ones is an important criterion to assess the "openness" or "extensibility" of a dedicated transformation tool. Other functions are an ability to suggest, when to apply transformations, and an ability to guarantee correctness of the transformations. Transformations should ensure syntactic correctness and semantic correctness to guarantee that a target model produced by the transformation is well formed.

Another function is an ability to deal with incomplete or inconsistent models. It is important to be able to transform models early in the software development life cycle, when requirements may not yet be fully understood or described in a natural language. An ability to test, validate and verify transformations are useful to ensure that the transformations are performed as desired. Tools that have an ability to specify bidirectional transformations require fewer transformation rules, since each transformation can be used in two different directions: to transform the source model into the target model, and to perform the inverse transformation, i.e. to transform the target model into the source model.

The final function from the list [8] is support for traceability and change propagation. To support traceability, the transformation tool needs to provide mechanisms to maintain an explicit link between the source and target models of a model transformation and to support change propagation, the transformation tool may have an incremental update mechanism and a consistency checking mechanism.

Thus, in general a model transformation tool should have the components to implement a model editor, a repository, its validation and transformation to another model (or code, which can also be considered as a model). Model Editor (CASE tool) is a part of the tool providing model creation and modification possibilities. Model Repository is the "database" for models, where they are stored.

Transformation Definition Editor is used for transformation definition construction and modification. Transformation Definition Repository is storage for transformation definitions. There, a set of basic scripts written in a general purpose programming language and powerful graphical transformations can reside. And, finally, Model Validator is a component used to check if the model is well enough defined and has no potential problems that can affect the transformation result.

III. BrainTool as a Model Transformation Tool

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 requirements stated in the previous section.

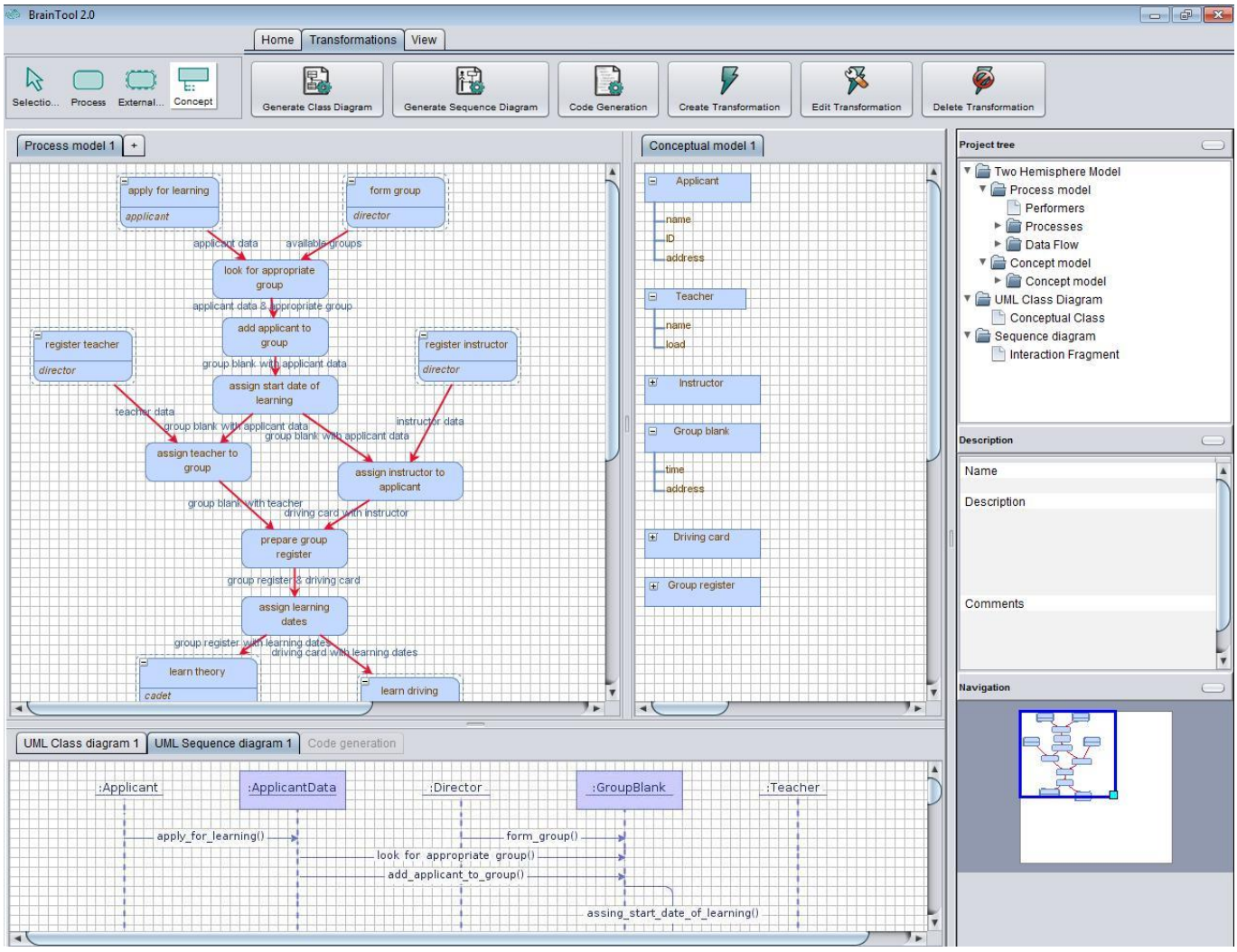

Fig. 1. General view of BrainTool.

The problem of automatic generation of the UML diagrams from the formal and still customer-friendly presentation of problem domain has not been solved yet [6]. The authors of BrainTool propose to generate UML diagrams from the so-called two-hemisphere model [5] of the problem domain, which presents information about processes, information flows between these processes and pre-defined types of these information flows.

A. Modeling of the Two-hemisphere Model

The main idea of representing the initial information about the system with two interrelated models – the business process model and the concept model was introduced in 2002 [10]. The name of the approach, the two-hemisphere model, was defined in [5], where the hypothesis about how to use two interrelated models to share the responsibilities between object classes was demonstrated on the example of a driving school.

By analogy with the human brain, which consists of two hemispheres harmonically interrelated for an adequate human behavior, the two-hemisphere model requires the harmony of the presentation of the problem domain for future software development with two interrelated models. Here, the business process model displays behaviour of the system; the concept model displays a skeleton of the system's static structure.

The main benefit of the two-hemisphere model is that it can be created and often already is created by the business analyst at the customer's side. A Standish group survey [11] shows that about 83% of companies are engaged in business process improvement and redesign. This implies that many companies are familiar with business process modeling techniques or at least employ particular business process description frameworks [3], [11]. On the other hand, the practice of software development shows that functional requirements can be derived from the problem domain description as much as 7 times faster than if trying to elicit them directly from users [1]. Both facts mentioned above and the existence of many commercial and open source business modeling tools are a strong motivation to base software development on the business process model, rather than on any other soft or hard models. So far, the first task the authors had to solve was to develop the modelling environment or a two-hemisphere model editor, which would provide an opportunity to create a model, to manipulate with it and to save the model in a format suitable for further transformations. Fig. 1 shows the general view of BrainTool 2.0, which is developed to support the two-hemisphere modeling of the problem domain.

The screen of BrainTool is divided into three parts. The information panel on the topside of the screenshot shows the list of elements defined by the two-hemisphere model. The working area of the tool is divided into three drawing frames – the process model, the concept model and the resulting class or sequence diagram (at the bottom). Bottom part is also used to create the sub-process diagrams. The tree view of all objects defined in all models (including the resulting model) is shown on the right side of the screenshot in Fig. 1.

Fig. 2 and Table I present the notation of the two-hemisphere model. The two-hemisphere model of the system has to define internal processes of the system, which have to be enclosed by external processes performed by a set of performers (users or other systems). The data flow coming from one process to another is defined as the collection of data processing, where the structure of the exact data flow has to be defined by a conceptual class, called a concept in the two-hemisphere modeling notation. Process diagram (Graph G1 in Fig. 2) presents steps of some fragment of business logic of the system (or a scenario) and usually is defined on the left side of the two-hemisphere model. Concept diagram (Graph G2 in Fig. 2) presents conceptual classes of the system.

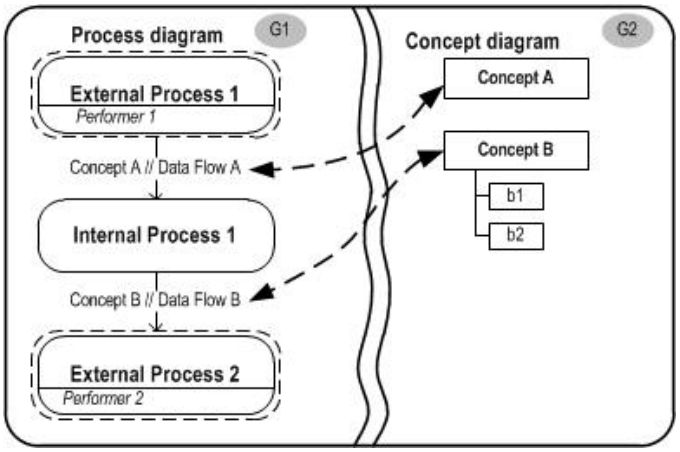

Fig. 2. Notation of the two-hemisphere model.

Concept diagram is similar to some kind of Entity Relationship (ER) diagram [12], but without presentation of relationships between classes, which are avoided at this level of abstraction in the two-hemisphere model. The model can have one or many process diagrams (scenarios) and only one (general for the whole system) concept diagram.

TABLE I

ELEMENTS OF TWO-HEMISPHERE MODEL AND THEIR NOTATION

ELEMENT	ELEMENT APPEARANCE
External process and performer	External Process 1 *Performer 1*
Internal process	Internal Process 1
Data flow	Concept A // Data Flow A
Concept	Concept B b1 b2

B. Transformation to the UML Class Diagram and Its Layout

The essence of the two-hemisphere model driven transformations is illustrated in Fig. 3. The business process model (graph G1 in Fig. 3) is interrelated with the concept model (graph G2 in Fig. 3) as follows – concepts in the concept model define data types for dataflows between business processes. The main idea of the transformation is based on graph theory. The business process model is transformed into intermediate model (graph G3 in Fig. 3),

when edges of the business process model become nodes of the intermediate model, and nodes of the business process model become edges of the intermediate model. The intermediate model serves as a base for construction of the communication model. The meaning of objects in an object-oriented philosophy gives a possibility to share responsibilities among class objects, where the data flow outgoing from the internal process becomes the object-owner of this process for performing it as an operation. The concept model allows determining classes with attributes [13], [14], [15].

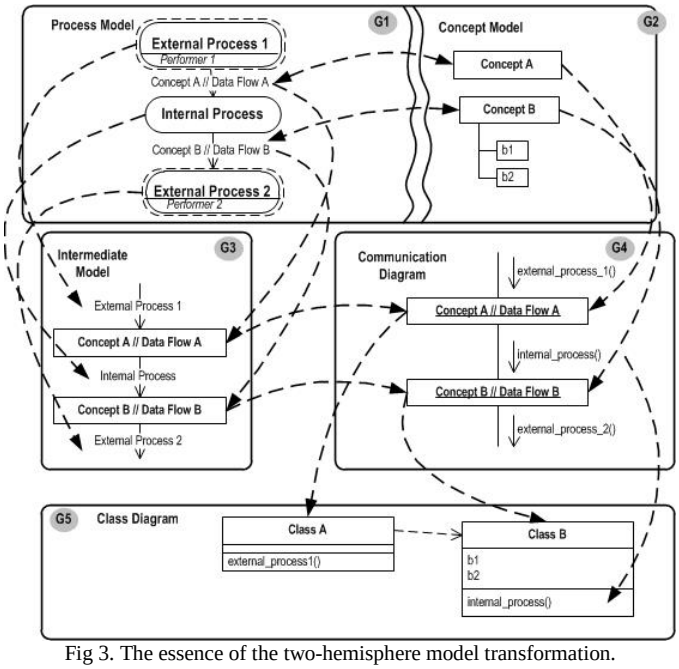

Fig 3. The essence of the two-hemisphere model transformation.

In correspondence with [16] the transformation is the automatic generation of a target model from a source model according to a transformation definition. In the case of BrainTool, the source model is the two-hemisphere model consisting of the process diagram, the set of concepts and linkage of the concept to the data flows. The target is the UML class diagram, which is a set of classes, class methods, class attributes, interfaces and relationships between classes and interfaces.

The first transformation task is to generate classes of resulting UML class model. Classes are created from concepts and retain their attributes. The following high-level pseudocode expresses the idea of this transformation:

```
func generate_classes(process_model pm, concept_model cm, class_model clm)
    for each concept in cm do
        clm.create_class_from(concept)
    for each process in pm do
        u_inputs = node.input_set().cardinality
        outputs = len(node.outputs())
        u_outputs = node.output_set().cardinality
        if u_inputs = 1 and u_outputs = 1 and outputs != 1 then
            for each output in node.outputs() do
                clm.create_class_from(output)
                clm.define_generalization(output, node.input_set())
```

An example in Fig. 3 shows the simplest transformation case, when a process has one input event and one output event. However, based on combinatorics we have determined 19 transformation cases, which are described in [14]. The transformation cases differ from one another by the number and combinations of input and output events and their types expressed as concepts. All 19 cases allow conclusions about classes – owners of methods, and about relations among classes. Cardinalities (the number of different concepts linked to data flows) of process inputs and outputs are used to determine generalization between the classes created. Also this information is used to define aggregation, association and dependency between elements of the UML class diagram [14]. Processes from the process model become class methods as a result of the transformation, which can be expressed in the following pseudocode fragment:

```
func assign_methods(process_model pm, concept_model cm, class_model clm)
    for each process in pm do
        classes = node.get_classes(clm)
        inter = null
        if len(classes) > 1 then
            inter = clm.create_or_get_interface(classes, node)
                for each c in classes do
            c.add_method(node)
            if inter != null then
                clm.define_realisation(c, inter)
```

Method assignment to classes gives the possibility to define interfaces and realization relationship in the UML class diagram. As a result, the target model consists of classes with methods and attributes, interfaces with methods and five types of relationships: generalization, dependency, aggregation, association and realization.

Next step after creating the class model via the transformation is its layout. A proper layout is essential for easy viewing and quick understanding of a diagram. Crossing lines of class relationships, overlapping classes and obtrusive spacing, along with some other issues can cause misinterpretation of an otherwise properly transformed diagram, leading to difficulties in its usage and comprehensibility. And since a manual layout can take time, especially for larger diagrams consisting of many classes and a variety of relationships, an automatic layout algorithm have been proposed by authors of the paper in [ICSEA 2014].

C. Transformation to the UML Sequence Diagram and Its Layout

The same principle of graph transformation can also be used also to generate elements of the UML sequence diagram and additionally to keep also the time aspect as a sequence of message sending. Process diagram is constructed in a manner that it is possible to tell a sequence of processes and pass it via transformation.

General idea of two-hemisphere model transformation to the UML sequence diagram is shown in Fig. 4.

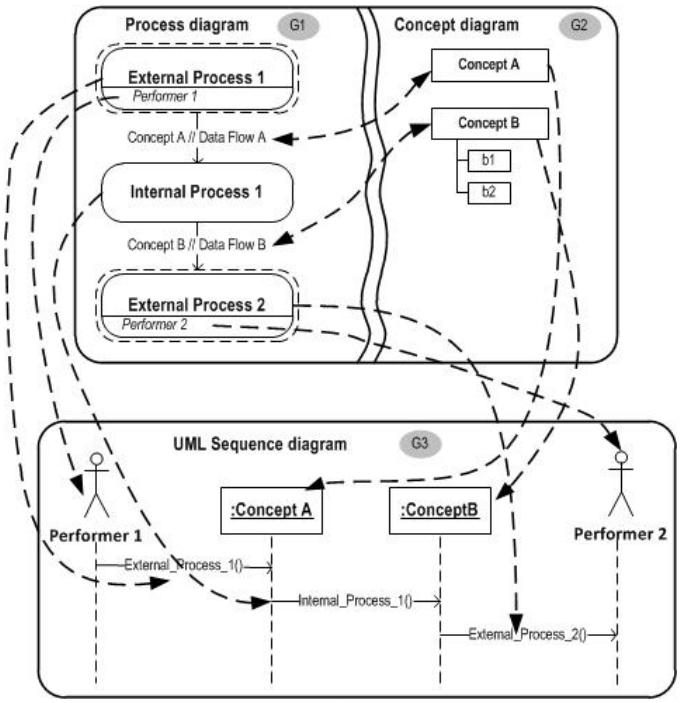

Fig. 4. Two-hemisphere model transformation to UML sequence diagram.

The pseudocode shows concept, process and data flow transformation to the relevant UML sequence diagram elements [17].

```
func generate_sq_object (concept_model cm, sequence _model sqm)
   for each concept in cm do
      sqm.create_object_from(concept)
func generate_sq_actor (process_model pm, sequence _model sqm)
   for each ext_process in pm do
      sqm.create_actor_from(ext_process_performer)
func generate_sq_message (process_model pm, sequence _model sqm)
   for each ext_process in pm do
      inflows = node.get_in_flows(ext_process)
      outflows = node.get_out_flows(ext_process)
      if len(inflows) = 0 then
         mess_sender = ext_process_performer
          if len(outflows) > 1 then
              for each flow in outflows
                  mess_receiver = flow.get_Concept()
                  sqm.create_mess_from(mess_sender,  mess_receiver)
      if len(outflows) = 0 then
         mess_receiver = ext_process_performer
          if len(inflows) > 1 then
              for each flow in inflows
                  mess_sender = flow.get_Concept()
                  sqm.create_mess_from(mess_sender,  mess_receiver)
    for each int_process in pm do
      inflows = node.get_in_flows(int_process)
      outflows = node.get_out_flows(int_process)
      for each iflow in inflows do
          for each oflow in outflows do
              mess_sender = iflow.get_Concept()
              mess_receiver = oflow.get_Concept()
              sqm.create_mess_from(mess_sender,  mess_receiver)
```

Processes from the process diagram (Graph G1) are transformed into messages of the UML sequence diagram (Graph G3). Concepts from the concept diagram (Graph G2) with their relevant data flows from the process diagram help to determine senders and receivers in the sequence diagram. It is important to assign performers to external processes, as they become actors in the sequence diagram. Actors are external entities, which begin interaction. There are many transformation cases that can occur transforming two-hemisphere model into the UML sequence diagram. Altogether, there are nine different cases described in [17], they show a two-hemisphere model and corresponding UML sequence diagram.

Another factor that needs to be taken in consideration is an automatic diagram layout after it is created by transformation. Diagram must be semantically and syntactically correct and well-layouted to be comprehensible. In order to layout a diagram special criteria have to be satisfied. There are general diagram criteria, which can be applied to UML sequence diagram (not all of them) and also specific criteria for a sequence diagram. The authors of [17] summarize twelve important criteria for a sequence diagram. The authors of this paper have developed and implemented UML sequence diagram layout algorithm in BrainTool that satisfies the most important layout criteria [ICSEA 2014], such as

- precise sequence of messages;
- avoid object and lifeline overlapping;
- elements need to be arranged orthogonally;
- diagram flow;
- minimize crossings;
- message arrow length minimization;
- reduction of long message arrow number;
- minimize longest message arrow length.

D. Export/Import of the UML Diagrams from BrainTool to Other UML Compatible Tools

The modern trend, which is model driven software development, states the ability to define tool sets, which are integrated to make a complete development environment. In our case, the set can be defined as a tool for the UML class diagram creation (it is BrainTool) plus the powerful modeling environment for manipulating UML diagrams and further code generation from them. Therefore, to become compatible with such a modelling environment created tool has to have export abilities of the generated UML class diagrams for their use in other modeling tools.

Initially the Sparx Enterprise Architect was selected as the environment to import the UML class diagrams created in BrainTool. However, the authors faced problems importing the XMI files containing the information about the UML class diagrams (defined according to the existing standard [18]) Import into several other UML modelling tools failed on occasion as well as loss of geometry was observed. The authors of the paper stated the task to check abilities of import/export between several different tools, and the results of the experiment are shown in Table II.

TABLE II

MODEL INTERCHANGE BETWEEN UML COMPATIBLE TOOLS

	ArgoUML	SPARX Enterprise Architect	Visual Paradigm	StarUML
ArgoUML		• Attribute types • Method argument types • Geometry • Relations	• Import failed	• Geometry • Method return values became method arguments • Attribute types • Method argument types • Relations • Attribute and method names • Private/public/protected modifiers
SPARX Enterprise Architect	• Import failed		• Import ok. All data imported	• Attribute and method names • Attribute types • Method argument types • Private/public/protected modifiers
Visual Paradigm	• Import failed	• Geometry • Relations		• Access violation. Program will be closed. Import failed.
StarUML	• Geometry • Elements from class model were moved to design model	• Method argument names	• Import failed	
Self-generated XMI	• Import failed	• Geometry	• Geometry	• Attribute and method names • Attribute types • Private/public/protected modifiers • Geometry

Cells of Table II show lacking information during import of XMI file generated from the tool listed on the left side of the table and imported into the tool listed at the top of the table.

For example, during the export of the UML class diagram from the ArgoUML tool and import of it into the SPARX Enterprse Architect tool, the model lost attribute types, method argument types, diagram geometry and relations. Otherwise, import of the UML class diagram from SPARX Enterprse Architect tool into the ArgoUML tool failed at all.

The main lesson learned about the model interchange between various modelling tools can be stated as a lack of XMI support in current solutions. The problem can be solved, for example, by introducing a certification standard for tools that generate and interchange the UML diagrams. However, such a certification needs to be introduced by a well-recognized development community, e.g., the vendor of the object-oriented philosophy – Object Management Group.

Currently, the problem of the model interchange has been solved by the authors adapting the export of the generated UML class diagram in correspondence with the format required by the Sparx Enterprise Architect tool.

IV. COMPARISON WITH OTHER UML MODELING AND MODEL TRANSFORMATION TOOLS

Since the beginning of the 1980s. numerous cases of model generated software systems have been offered to attack problems regarding software productivity and quality [19]. CASE tools developed up to that time were oversold on their "complete code-generation capabilities" [20].

Nowadays, similar arguments are introduced by the Object Management Group (OMG) Model Driven Architecture (MDA) [6], using and integrating Unified Modeling Language (UML) models [2] at different levels of abstraction. Manipulation with models enables the automation of software development with CASE tools supporting model driven software development [16], [21], [22], [23]. Most of today's tools combine a number of functions in a more or less open fashion. The traditional CASE tools provide a model editor and a model repository. A code generator based on a scripting language and plugged into a CASE tool provides the transformation tool and transformation definition editor. In that case, the transformation repository is simply text files [16]. Authors have listed several tools offering creation of the UML class and interaction diagrams in Table III, but they are mainly UML editors, where a developer creates all the diagrams manually with limited ability to generate new elements.

The variety of "model-driven" tools can be divided into tools to support code generation from the UML model, and into tools created for the definition of the system model itself. The second group of the tools is the so-called "UML editors", where tool developers propose different levels of automation of the model creation itself.

BrainTool demonstrated in this paper can be classified as a tool for creation of the UML diagrams, where the result of the generation is expressed in XMI format – is importable either into UML editors, like:

- UMLet 11.3 [24];
- Umbrello [25];
- Together [26];

or code generation tools, like

- Sparx Enterprise Architect [27];
- UML Studio [28];
- Visual Paradigm for the Unified Modeling Language [29];
- ArgoUML [30];
- MagicDraw [31];
- IBM Rational Software Architect [32];
- Eclipse [33].

As for now, the generation of the UML class diagram from the existing source code (e.g., Java) is widely used by software developers to visualize and understand the software structure.

TABLE III

COMPARISON OF BRAINTOOL WITH OTHER TOOLS GENERATING THE UML DIAGRAMS

Tool / **Criteria**	**Visual Paradigm**	**Sparx EA**	**IBM RSA**	**Visual Studio**	**ReDSeeDS**	**BrainTool**
Initial information for generation of the UML diagrams	System req-ts & use-case diagram	System req-ts & use-case diagram	System req-ts & use-case diagram & program code	Program code	System req-ts	Two-hemisphere model
Model editor for initial information	Text editor	Text editor	Text editor	Text editor	Text editor	Graphical editor
Transformation base to UML Class diagram	Reverse transformation code-to-model; Formal transformation model- to-code	Reverse transformation code-to-model; Formal transformation model- to-code	Text-to-model via transformation configuration mechanism	Formal transformation text-to-model	Formal transformation text-to-model using language MOLA	Formal transformation model-to-model
Transformation base to UML Sequence diagram	Linguistic analysis	Linguistic analysis	Linguistic analysis	Formal transformation text-to-model	Linguistic analysis	Formal transformation model-to-model
Class (UML Class Diagram)	Automatically	Automatically	Automatically	Automatically	Automatically	Automatically
Attribute (UML Class Diagram)	Automatically	Automatically	Automatically	Automatically	Automatically	Automatically
Method / Operation (UML Class Diagram)	Automatically	Automatically	Automatically	Automatically	Automatically	Automatically
Association (UML Class Diagram)	Automatically	Manually	Automatically	Automatically	Automatically	Automatically
Class Interface (UML Class Diagram)	No information	Automatically	Automatically	Automatically	Automatically	Automatically
Dependency (UML Class Diagram)	Automatically	Manually	Automatically	Automatically	Automatically	Automatically
Aggregation (UML Class Diagram)	Automatically	Automatically	Automatically	Automatically	Automatically	Automatically
Generalization (UML Class Diagram)	Automatically	Manually	Automatically	Automatically	Automatically	Automatically
Implementation (UML Class Diagram)	Automatically	Automatically	Automatically	Automatically	Automatically	Automatically
Actors (UML Sequence Diagram)	Borrowed from use-cases	Borrowed from use-cases	Borrowed from use-cases	No	Automatically	Automatically
Objects (UML Sequence Diagram)	Manually	Manually	Manually	Automatically	Automatically	Automatically
Lifelines (UML Sequence Diagram)	Manually	Manually	Manually	Automatically	Automatically	Automatically
Operations (UML Sequence Diagram)	Manually	Manually	Manually	Automatically	Automatically	Automatically
Operation ordering (UML Sequence Diagram)	Manually	Manually	Manually	Automatically	Automatically	Automatically
Interaction frames (UML Sequence Diagram)	Manually	Manually	Manually	Automatically	Automatically	Automatically
Operation parameters (UML Sequence Diagram)	Manually	Manually	Manually	Automatically	Automatically	No
Links between objects (UML communication diagram)	Manually	Manually	Manually	No	Automatically	Automatically
Graphical representation of the UML class diagram	Yes	Yes	Yes	Yes	via Sparx EA	Yes
Graphical representation of the UML sequence diagram	Yes	Yes	Yes	Yes	via Sparx EA	Yes
Graphical representation of the UML communication diagram	Yes	Yes	Yes	No	via Sparx EA	Not yet
Automatic layout of UML class diagram	No	Yes	Yes	Yes	via Sparx EA	Yes
Automatic layout of UML sequence diagram	No	Lawless ordering of objects at the top of diagram	No	Yes	via Sparx EA	Yes
Export abilities to UML compatible tools	Has special export format	Has special export format	Has special export format	No	at least to Sparx EA	Defined by XMI and importable in the tools supporting the standard specification

Several examples of these tools are the following:

- ESS-Model, which allows obtaining a class diagram with associations and inheritance by simple drag-and-drop of source files – Java .java and .class and Delphi .pas and .dpr [34].
- AgileJ Structure View, which allows displaying Java-specific information, is IDE specific and automatically handles layout elements of a class diagram [35].
- BOUML is a free modelling tool, which allows UML modelling, Java, C++, PHP, Python, Idl code generation, as well as class diagram generation from C++, Java and PHP source files [36].
- ObjectAid as Eclipse plug-in provides a visual representation for Java source files. ObjectAid does not execute any reverse engineering; it displays source files in a different view. When a developer adjusts the diagram or the source file, the other view is adjusted accordingly [37].

Another variety of tools generate the UML class diagram from a predefined data structure. For example Sparx Enterprise Architect has this feature. One more tool, which allows generating a class diagram from a data model, is Visual Paradigm. Generation of the UML class diagram from pre-defined data structures requires a solid contribution of the software specialist to define all these structures. It is already modelling of the UML class diagram itself. In contrast to these tools, BrainTool generates the class diagram from initial information about the system, which is understandable for the business analyst and does not require software knowledge for modelling of business processes. Usually, the UML class diagram is constructed in the analysis phase, before code writing. Therefore, the tool, which generates the class diagram at the initial stage of the project, is very useful. It allows automatic creation the static structure of the developed system and serves as a base for a further code generation, avoiding mistakes and mismatches between requirements and implementation.

Attempts to receive UML interaction diagrams from the requirements in a natural language are one of the popular lines of research. For example, ReDSeeDS [38] supporting tool proposes the linguistic analysis of system requirements and generates several elements of the UML sequence diagram, based on predefined format of requirement specification. However, the tool has no graphical presentation of the resulting diagram and exports the result to Sparx Enterprise Architect.

On the other hand, Visual Studio supports the ability to generate the UML sequence diagram from the source program code. This is different direction from the approach offered by the authors of the paper and this tool can be interesting for comparison only in a diagram presentation aspect, like the diagram layout implementation, or export to other UML compatible tools.

Tools, like Sparx Enterprise Architect [27], Visual Paradigm [39] or Rational Software Architect [32], give the ability to reflect to the existing UML diagram elements, if they are already created in other UML diagrams, but still, initially, these elements are identified manually.

There are several tools that provide automatic diagram layout, e.g., Borland Together [26] ((not listed in Table III) supports automatic UML sequence diagram layout, but uses a lawless set of layout criteria). Sparx Enterprise Architect [27] is the tool that also provides automatic UML sequence diagram layout however, it does not satisfy all the mentioned criteria of layout.

Thereby the authors believe that currently abilities for the generation of the UML diagrams offered by the two-hemisphere model driven approach and supported by BrainTool are the most expansive, but the authors still have to refine the tool with additional functionality expected by users in popular UML editors.

V. Conclusion

The main idea of the research presented in this paper is to show the main functionality of BrainTool to automatically generate UML diagrams from scratch. Moreover, the tool has to be able:

1) to work with the initial presentation of the problem domain expressed in terms of the two-hemisphere model;
2) to validate the initial model and to identify problematic elements, which break the transformation process;
3) to generate the UML diagrams from it based on the pre-defined transformations;
4) to visualize the target model in the form of the UML class or sequence diagram; and
5) to export the generated diagrams into the UML compatible tool. (Huge sentence, it's best to break it into several for clarity...) The authors share several lessons they have learned in engineering a model transformation tool within the scope of the paper.

One of the key lessons learned during the experiment with the BrainTool implementation is the selection of the development environment and technologies. Deeper post-experimental analysis of available technologies and the essence of the tool as a software product allow the authors to claim that any general purpose programming language or specific environment for such tool creation may be used. In this case, the determining factor is the available resources and previous experience of the developers. It is possible to use a general purpose programming language for transformation definition without integrating transformation language support into a model editor. General purpose language allows defining more universal transformation rules. Also it enables possible contributors not familiar with specific transformation languages to define their own model transformations. As another advantage it makes tool components more independent from each other, thus easing parallel development of them.

One of the problems solved during the development of BrainTool is in the area of the target model layout and its import into other modeling tools. The issue of diagram layout is not the problem of the tool; it is a problem of the algorithm itself. The authors have offered such an algorithm satisfying

the requirements for element placement in the modeling area for the UML class and sequence diagrams. However, the authors claim that any solution can be easily integrated within BrainTool. The authors consider the layout to be another kind of transformation where the source model is the generated class diagram "as is" and the target is the model laid out. Using such a point of view makes it easy to add automatic diagram layout at any development stage.

Another problem still not solved is a lack of standardization and certification support in model interchange. The authors would introduce plug-ins for different tools, where each plug-in can also be considered transformation from UML class diagram to tool-specific XMI document, if it is still impossible to integrate the unified standard of the diagrams into the tools themselves.

The working version of BrainTool enables one to operate with the two-hemisphere model and to generate the set of the elements of the UML diagrams, which in turn can be used to generate code fragments. Comparison of BrainTool with other UML modeling or model transformation tools shows that BrainTool has several abilities not yet realized in advanced modeling tools, but the developers of BrainTool still have to implement a piece of functionality.

The further efforts of the authors will be turned to the refinement of the transformation rules to be able to extend the set of the UML elements with the aim to improve code generation abilities in general.

Acknowledgment

The research presented in the paper is supported by Accenture Latvian Branch, project No. L7950 "Development of Model Transformation Tool Prototype", and by the Latvian Council of Science, 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, "Trends in business process analysis – from verification to process mining," in *Proc. of the 9th Int. Conf. on Enterprise Inform. Syst.*, ICEIS 2007, Funchal, Portugal. 2007, pp. 5–9.

[2] K. Vollmer, C. Richardson and C. Clair, "The Importance Of Matching BPM Tools To The Process," Forrester Research Inc., 2010.

[3] OMG, "UML Unified Modeling Language Specification", August 2011. [Online]. Available: http://www.omg.org/spec/UML/ [Accessed: Sept. 21, 2013].

[4] P. Rittgen, "Quality and perceived usefulness of process models," in *Proc. of the 2010 ACM Symp. on Appl. Computing*, Sierre, Switzerland, 2010, pp. 65–72. http://dx.doi.org/10.1145/1774088.1774105

[5] O. Nikiforova and M. Kirikova, "Two-hemisphere model driven approach: engineering based software development," in *The 16th Int. Conf. Advanced Information Systems Engineering*, CAiSE 2004, vol. 3084, June 7–11, 2004, Riga, Latvia. pp. 219–233. http://dx.doi.org/10.1007/978-3-540-25975-6_17

[6] G. Krishnamurthy "CASE Tools," [Online]. Available: http://www.umsl.edu/~sauterv/analysis/F08papers/View.html [Accessed: Sept. 22, 2013].

[7] J. Mukerji, J. Miller, "MDA Guide Version 1.0.1." OMG, no. omg/2003-06-01 June 2003. [Online]. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01 [Accessed: Sept. 22, 2013].

[8] T. Mens, P. Gorp, "A Taxonomy of Model Transformations," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 152, 2006, pp. 125–142, March 2006. http://dx.doi.org/10.1016/j.entcs.2005.10.021

[9] RTU, Official page of "BrainTool" tool. [Online]. Available: http://braintool.rtu.lv [Accessed: Sept. 20, 2013]

[10] O. Nikiforova, "General framework for object-oriented software development process," *Scientific Proc. of Riga Technical University*, vol. 13, pp.132–144, 2002.

[11] H. Peyret and D. Miers, "The shifting market for business process analysis tools," Forrester Research Inc., 2010.

[12] P. Chen, "The entity relationship model – towards a unified view of data," *ACM Trans. Database Systems*, TODS, vol. 1, no. 1. pp. 9–36, 1976. http://dx.doi.org/10.1145/320434.320440

[13] O. Nikiforova, N. Pavlova and J. Grigorjevs, "Several Facilities of Class Diagram Generation from Two-Hemisphere Model," in *23rd Int. Symp. on Computer and Information Sciences*, ISCIS, Oct. 27–29, 2008, Istanbul, Turkey. pp. 1–6.

[14] O. Nikiforova and N. Pavlova, "Foundations on generation of relationships between classes based on initial business knowledge," in *17th Int. Conf. on Information Systems Development*, ISD, Paphos, Cyprus, Aug. 25–27, 2008., Springer-Verlag: New York. pp. 289–297. http://dx.doi.org/10.1007/b137171_30

[15] O. Nikiforova and N. Pavlova, "Open Work of Two-Hemisphere Model Transformation Definition into UML Class Diagram in the Context of MDA," in *Software Engineering Techniques*, *LNCS*, Springer Berlin Heidelberg, vol. 4980, 2011, pp. 118–130. http://dx.doi.org/10.1007/978-3-642-22386-0_9

[16] A. Kleppe, J. Warmer and W. Bast, *MDA Explained: The Model Driven Architecture – Practise and Promise.* 1st. ed., Addison-Wesley, 2003.

[17] O. Nikiforova, D. Ahilcenoka, D. Ungurs, K. Gusarovs, L. Kozacenko, "Several Issues on the Layout of the UML Sequence and Class Diagram," *Proc. of the 9th Int. Conf. on Software Engineering Advances*, ICSEA 2014, Oct. 12–16, 2014, Nice, France, pp. 40–47, Available: http://www.thinkmind.org/

[18] *Information Technology – XML Metadata Interchange (XMI)*, ISO/IEC 19503:2005(E), 2005.

[19] R. Balzer, "A 15 year perspective on automatic programming," *IEEE Transactions on Software Engineering*, vol. 11, issue 11, pp. 1257–1268, Nov. 1985. http://dx.doi.org/10.1109/TSE.1985.231877

[20] J. Krogstie, "Integrating enterprise and IS development using a model driven approach," in *13th Int. Conf. on Information Systems Development. In: Advances in Theory, Practice and Education*. Springer Science+Business media, Inc. 2005, pp.43–53. http://dx.doi.org/10.1007/0-387-28809-0_5

[21] D. Kundu, D. Samanta and R. Mall, "Automatic code generation from unified modelling language sequence diagrams," *IET Software*, vol. 7, issue 1, 2013, pp. 12–28. http://dx.doi.org/10.1049/iet-sen.2011.0080

[22] D. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley Publishing, Inc., Indianapolis, Indiana, 2003.

[23] S. J. Mellor and M. J. Balcer, *Executable UML. A Foundation for Model-Driven Architecture*. Boston: Addison-Wesley, 2002.

[24] UmLet, Official page of "UMLet" tool. [Online]. Available: http://www.umlet.com/ [Accessed: Sept 21, 2013].

[25] KDE e.V., Official page of "Umbrello" tool. [Online]. Available: http://uml.sourceforge.net [Accessed: Sept 21, 2013].

[26] Borland, Official page of "Together" tool. [Online]. Available: http://www.borland.com/products/together/ [Accessed: Sept 21, 2013].

[27] Sparx, Official page of "Sparx Enterprise Architect" tool. [Online]. Available: http://www.sparxsystems.com [Accessed: Sept 21, 2013].

[28] PragSoft Corporation, Official page of "UML Studio" tool. [Online]. Available: http://www.pragsoft.com/prod_umls.html [Accessed: Sept 21, 2013].

[29] Visual Paradigm, Official page of "Visual Paradigm for the Unified Modeling Language" tool. [Online]. Available: http://www.visual-paradigm.com/product/vpuml/ [Accessed: Sept, 21, 2013].

[30] Tigris, Official page of "AgroUML" tool. [Online]. Available: http://argouml.tigris.org/ [Accessed: Sept 21, 2013].

[31] No Magic Inc, Official page of "MagicDraw" tool. [Online]. Available: http://www.nomagic.com/ [Accessed: Sept 21, 2013].

[32] IBM, Official page of "Rational Software Architect" tool. [Online]. Available: http://www.ibm.com/developerworks/rational/products/rsa/ [Accessed: Sept. 18, 2013].

[33] Eclipse Foundation, Official page of "Eclipse" tool. [Online]. Available: Internet: http://www.eclipse.org// [Accessed: Sept. 18, 2013].

[34] Eldeam, Official page of "ESS-Model" tool. [Online]. Available: http://essmodel.sourceforge.net/index.html [Accessed: Sept. 18, 2013].

[35] AgileJ Ltd, Official page of "AgileJ Structure View" tool. [Online]. Available: http://www.agilej.com/index.html [Accessed: Sept. 18, 2013].

[36] Bruno Pages, Official page of "BOUml" tool. [Online]. Available: http://www.bouml.fr/ [Accessed: Sept. 18, 2013].

[37] ObjectAid LLC, Official page of "ObjectAid" tool. [Online]. Available: http://www.objectaid.com/ [Accessed: Sept. 15, 2013].
[38] ReDSeeDS, Official page of "ReDSeeDS" development system. [Online]. Available: http://www.redseeds.eu/ [Accessed: Sept. 18, 2013].
[39] Visual Paradigm, Visual Paradigm for UML, "Generate Sequence Diagram from Use Case Flow of Events," May 2011. [Online]. Available: http://www.visual-paradigm.com/product/vpuml/tutorials/gensdfromfoe.jsp [Accessed: Sept. 15, 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 at the Department of Applied Computer Science, Riga Technical University, where she has been working for the faculty since 2000. Her current research interests include object-oriented system analysis and modeling, especially the issues in 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 Research 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

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 modeling tools.

E-mail: dainis.ungurs@rtu.lv

Dace Ahilcenoka received the Master Degree in Computer Systems from Riga Technical University, Latvia, in 2014.

She is presently a Research Assistant at 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

Andrejs Bajovs received the Master Degree in Computer Systems from Riga Technical University, Latvia, in 2014. He is presently Java developer in IT solutions development company ProfIT.

His current research interests include original ways of code generation in model driven architecture.

E-mail: andrej_bv@inbox.lv

Nadezda Skindere received the Master Degree in Computer Systems from Riga Technical University, Latvia, in 2014.

Her current research interests include certification on system modeling tools within the framework of Model Driven Software Development.

E-mail: nadezda.skindere@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

Maris Jukss is a second year PhD student at the Modelling, Simulation and Design Lab in School of Computer Science at McGill University, Canada. His current research interests are in efficient and usable model transformations.

E-mail: maris.jukss@mail.mcgill.ca