doi: 10.2478/acss-2013-0003
2013/ 14

Improvement of the Two-Hemisphere Model-Driven Approach for Generation of the UML Class Diagram

Oksana Nikiforova[1], Konstantins Gusarovs[2], Olegs Gorbiks[3], Natalja Pavlova[4], [1-4]*Riga Technical University*

***Abstract* – In this paper an ability to apply the two-hemisphere model-driven approach for creation of the UML class diagram is discussed and the way to avoid the limitations of the approach is offered. The result of the proposed improvement of the two-hemisphere model-driven approach is the increased number of elements of the UML class diagram available for automatic generation and several statements for semi-automatic transformation of business process diagram and the concept diagram into software components. As a result, the authors can ascertain that it is possible to apply the improved two-hemisphere model-driven approach in practice in the real software development, and not only for academic purpose.**

***Keywords* – two-hemisphere model, UML class diagram, model transformation, binary route matrix, BrainTool.**

I. Introduction

Many ways of software development and a lot of methodologies have been discovered and described to this date. However, there is no one, which could be called an ideal or the best method.

For many years scientists tried to make the software development process more effective and faster. This was done not only for the sheer love of science. Making the development process more effective (meaning to make it less time consuming, producing less bugs and making the entire process cheaper) is necessary to ensure the possibility of developing more complex software systems.

The main focus of this paper is on the so called *Model-Driven Software Development* (MDSD) [1]. It claims to automatically transform an independent system presentation (from the implementation platform) into the specific software components. There are many methods within MDSD. One of these methods, which proposes to create the UML class diagram from initial presentation of problem domain was developed in Riga Technical University in 2004 and is called the Two-hemisphere model-driven approach [2]. More detailed description of this approach can be found in Section 2.

Unfortunately, the current version of the transformations defined by the two-hemisphere model-driven approach has a number of limitations. There is impossible to obtain the resulting complete UML class diagram, which includes all necessary elements such as interfaces and super classes. Definition of the association types in the resulting UML class diagram is not completely suitable either.

On the other hand, a number of the MDSD methods (and the two-hemisphere model-driven approach as one of these) suggest the effective way of how to transform the user defined scenarios of the software system into the UML class diagram. It is not reasonable to dismiss this practice completely. On the contrary, it is necessary to improve the existing methods and discover the new ones if required.

This paper describes the way of improving the two-hemisphere model-driven approach, which is supposed to avoid certain limitations of the method.

The paper is structured as follows. The next section describes the initial version of application of the two-hemisphere model meant for generation of the UML class diagram. The third section describes the core ideas of the proposed method improvement. The result analysis is given in Section 4. The comparison of the two-hemisphere approach to other analogue methods is discussed in the fifth section and the conclusion is given in Section 6.

II. Description of Two-Hemisphere Model-Driven Approach (Initial Version)

Two-hemisphere model-driven approach proposes to generate UML class diagram from the so-called two-hemisphere model of the problem domain, which presents information about the processes, information flows between these processes and pre-defined types of these information flows. The main idea of displaying the initial information about the system with the help of two interrelated models was introduced by Nikiforova in 2002 [3]. These two initial interrelated models are: the business process model (the shorter name is – the process model), which displays behaviour of the system and the model of conceptual classes (the shorter name is – the concept model), which displays a skeleton of the system static structure.

The meaning of objects in an object-oriented philosophy gives the possibility to share responsibilities between the objects based on the direct graph transformation, where the data outflow from the internal process in the process model becomes the owner of this process for performing it as an operation in object communication. The title of the approach as the two-hemisphere model driven was defined by Nikiforova and Kirikova in 2004 [2], where the hypothesis of how to use two interrelated models to share the responsibilities between object classes was demonstrated on the abstract example and later in a real project [4].

In general, the two-hemisphere model driven approach uses the transformation of graph converting nodes of the source graph into the edges of the target graph and the edges of a source graph into nodes of the target graph. The essence of the transformation is illustrated in Figure 1.

The business process model (graph G1 in Figure 1) is interrelated with the concept model (graph G2 in Figure 1) as shown below. Certain concept in the concept model defines the data type for one or several data flows between business processes. The business process model is transformed into an intermediate model (graph G3 in Figure 1), where the edges (i.e. data flows) of the business process model become nodes of an intermediate model, and nodes of the business process model (i.e. processes) become edges of an intermediate model.

Semantics of the nodes and edges of the intermediate model is the same as of the UML communication diagram (graph G4 in Figure 1), where nodes of the intermediate model are the edges (i.e. objects) of communication diagram, and nodes of the intermediate model are the edges (i.e. messages to perform the operation) of the communication diagram.

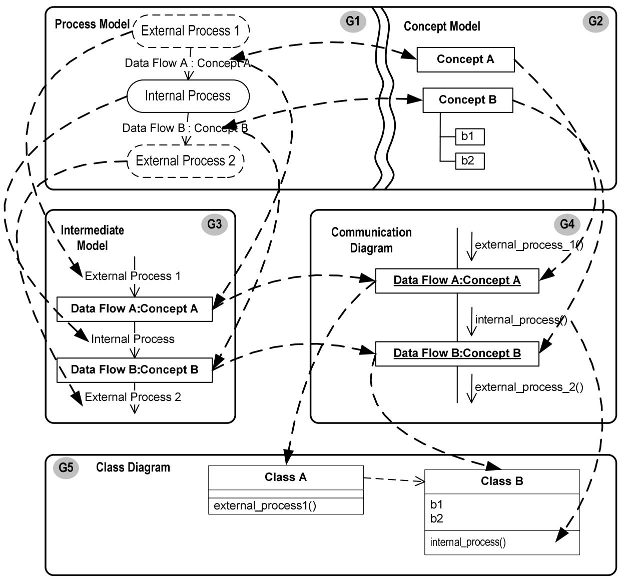


Fig. 1. Two-hemisphere model transformation into class model (initial version)

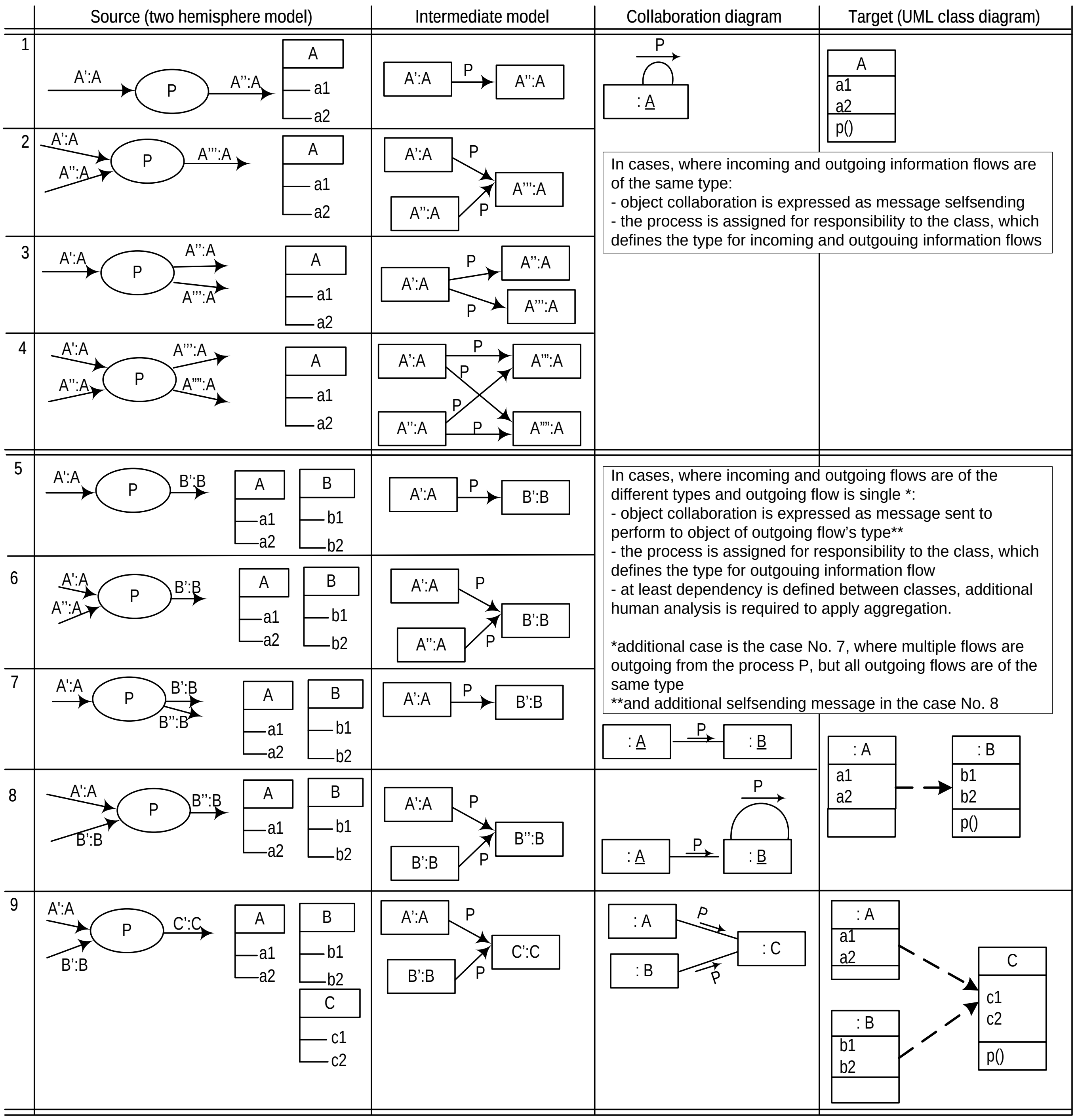
The communication diagram itself serves as the basis for the definition of classes-owners of methods in the UML class diagram (graph G5 in Figure 1). Details of application of the two-hemisphere model are provided in [5], [6], [7].

Analysis of different situations, which may appear when drawing the process model, i.e. number of incoming and outgoing data flows, variety of their types, etc., has provided the possibility to define various transformation cases depending on the number of process inputs and outputs and their cardinality (a set of differently typed data flows incoming or outgoing from the process). These transformation cases are implemented according to definition of relationships between generated classes, which are expressed in [8]. Possible variations of the two-hemisphere model and the corresponding fragments of the UML class diagram are summarized in Figure 2.

At the moment the transformations are defined only for the cases where either data flows (both incoming and outgoing) are defined of the same type or the process has a single outgoing data flow or multiple outputs are of the same type. In cases, when direct transformation is not possible, an interface class for implementation of problematic process is created or a developer of the two-hemisphere model has to create the sub-process diagram for the problematic process, let for any process would be defined single information flow outgoing from the process.

Successful application of the two-hemisphere model transformation into the UML class diagram served as a motivation to support these transformations with the software system. The first software prototype of the tool supporting two-hemisphere model based transformation was introduced in 2008 [5], [9]. The prototype used textual information in special format as a source and produced the text file containing description of the resulting UML class diagram as a specification, where classes, attributes, methods and relationships were listed in a pre-defined format. Analysis of these generated text files gave authors an opportunity to refine transformations for definition of relationships between the classes; the results are published in [8].

Currently, the ability to apply the two-hemisphere model for generation of the UML sequence diagram with the attention on the timing aspect is investigated, and preliminary results are published in [10]. So far, the continuing research in the area of model-driven software development and an increasing demand in the industry for automation of the ability to bridge the gap between the problem domain and software components, served as the motivation to develop the two-hemisphere model driven approach supporting tool – the BrainTool [11], [12], which allows to draw the two-hemisphere model in the manner suitable for the problem domain expert and to generate the UML class diagram from it.



In cases, where incoming and outgoing information flows are of the same type:
- object collaboration is expressed as message selfsending
- the process is assigned for responsibility to the class, which defines the type for incoming and outgouing information flows

In cases, where incoming and outgoing flows are of the different types and outgoing flow is single *:
- object collaboration is expressed as message sent to perform to object of outgoing flow's type**
- the process is assigned for responsibility to the class, which defines the type for outgouing information flow
- at least dependency is defined between classes, additional human analysis is required to apply aggregation.

*additional case is the case No. 7, where multiple flows are outgoing from the process P, but all outgoing flows are of the same type
**and additional selfsending message in the case No. 8

Fig. 2**.** Transformation cases defined by initial version of two-hemisphere model driven approach

Figure 3 demonstrates a two-hemisphere model, where the problem domain is the two-hemisphere model driven approach itself for development of the BrainTool. The modeller of the problem domain should in the first place model any process of the operating system; then some of other processes have to be placed to unable the definition of the information flow between any two processes. The information flow, at first, should be created and linked to processes and then it becomes possible to define the data type for this information flow, etc.

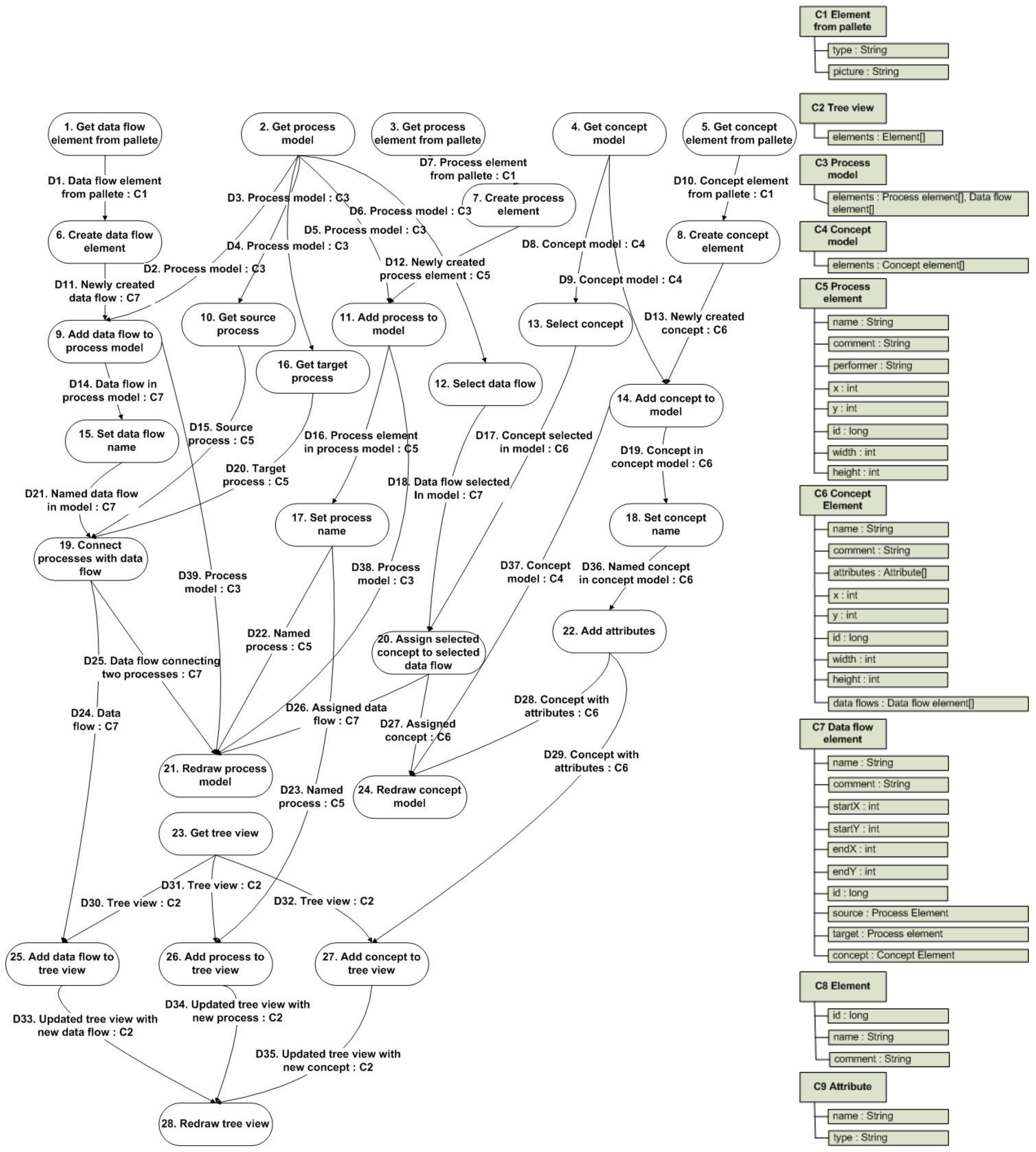


Fig. 3**.** Two-hemisphere model of the two-hemisphere model-driven approach itself

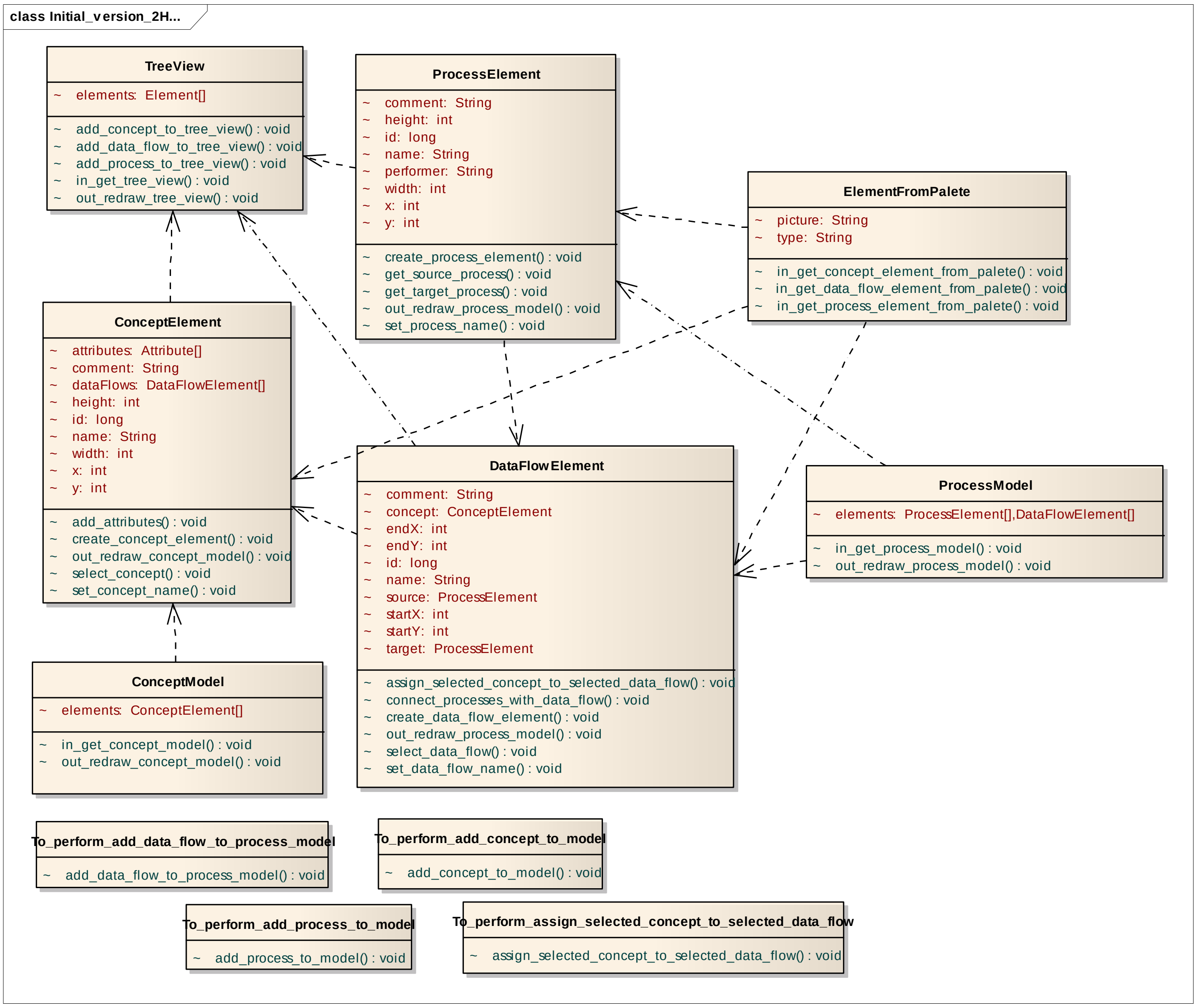

class Initial_version_2H...
TreeView
~ elements: Element[]
~ add_concept_to_tree_view() : void
~ add_data_flow_to_tree_view() : void
~ add_process_to_tree_view() : void
~ in_get_tree_view() : void
~ out_redraw_tree_view() : void
ProcessElement
~ comment: String
~ height: int
~ id: long
~ name: String
~ performer: String
~ width: int
~ x: int
~ y: int
~ create_process_element() : void
~ get_source_process() : void
~ get_target_process() : void
~ out_redraw_process_model() : void
~ set_process_name() : void
ElementFromPalete
~ picture: String
~ type: String
~ in_get_concept_element_from_palete() : void
~ in_get_data_flow_element_from_palete() : void
~ in_get_process_element_from_palete() : void
ConceptElement
~ attributes: Attribute[]
~ comment: String
~ dataFlows: DataFlowElement[]
~ height: int
~ id: long
~ name: String
~ width: int
~ x: int
~ y: int
~ add_attributes() : void
~ create_concept_element() : void
~ out_redraw_concept_model() : void
~ select_concept() : void
~ set_concept_name() : void
DataFlowElement
~ comment: String
~ concept: ConceptElement
~ endX: int
~ endY: int
~ id: long
~ name: String
~ source: ProcessElement
~ startX: int
~ startY: int
~ target: ProcessElement
~ assign_selected_concept_to_selected_data_flow() : void
~ connect_processes_with_data_flow() : void
~ create_data_flow_element() : void
~ out_redraw_process_model() : void
~ select_data_flow() : void
~ set_data_flow_name() : void
ProcessModel
~ elements: ProcessElement[],DataFlowElement[]
~ in_get_process_model() : void
~ out_redraw_process_model() : void
ConceptModel
~ elements: ConceptElement[]
~ in_get_concept_model() : void
~ out_redraw_concept_model() : void
To_perform_add_data_flow_to_process_model
~ add_data_flow_to_process_model() : void
To_perform_add_concept_to_model
~ add_concept_to_model() : void
To_perform_add_process_to_model
~ add_process_to_model() : void
To_perform_assign_selected_concept_to_selected_data_flow
~ assign_selected_concept_to_selected_data_flow() : void

Fig. 4. Class model generated by initial version of the approach

The process model on the left side of Figure 3 presents the process for drawing the two-hemisphere model. The concept model on the right side of Figure 3 demonstrates the conceptual structure of the two-hemisphere model. Figure 4 shows the resulting UML class diagram, which is generated from the two-hemisphere model according to the transformations defined by the approach and which should be used to implement the software system of BrainTool.

The four classes (located separately at the bottom of Figure 4) are defined, which demonstrate the limitation of the initial version of the transformations defined by the approach. These limitations relate to the transformation cases, where the process has several outputs typed by different concepts and therefore the transformation does not define, which of the classes should perform this process as an operation. These four "problematic" processes are the following (numeration see in Figure 3):

- No.9 add data flow to process model,
- No.11 add process to model,
- No. 14 add concept to model,
- No. 20 assign selected concept to selected data flow.

The current version of the approach requires creation of the sub-process diagrams for these processes to ensure the ability for transformation. However, if these processes take place, further decomposition does not provide a way to define the owner of the operation. Therefore, the approach needs several addition transformations to complete the problem. To improve the current version of the two-hemisphere model-driven approach, the authors have selected these problematic situations for a more profound analysis and try to define at least a semi-automatic support for redrawing the problematic process. The result of this activity is described in the next section.

III. Improvement of two-hemisphere model-driven approach

To get rid of problematic processes, which have multiple outputs of different data types the authors suggest to add creation and analysis of the transition matrix for the two-hemisphere model. For creation of the transition matrix after initial creation of the business process diagram, concept diagram and concept assignment to data flows a modeller should be asked to perform primary validation of the developed model. This can be done by asking series of questions if some concept is required to perform the exact process and to get the defined concept in the output. This allows generating methods with the same names for different classes even with different sets of arguments. For example, in process No.9 in Figure 3 should initiate the following question: who is responsible for "add data flow to process model" operation – data flow itself or a process model?. Depending on an answer class containing this method is being chosen. This is first option. Another option is to introduce method with same name but different signatures in both classes.

The matrix of the required transitions is square matrix MxM, where M is an amount of concepts in a two-hemisphere model. Each cell of matrix contains the set of processes, which takes the data flow of the concept defined in column as an input and outputs the data flow of another concept defined in row. Such matrix for two-hemisphere model shown in Figure 3 and 4 is presented in Table 1.

TABLE I
The Matrix of the Required Transitions

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	∅	∅	∅	∅	{7}	{8}	{6}	∅	∅
C2	∅	{25,26,27}	∅	∅	∅	∅	∅	∅	∅
C3	∅	∅	{9,11}	∅	{10,11,16}	∅	{9,12}	∅	∅
C4	∅	∅	∅	{14}	∅	{13,14}	∅	∅	∅
C5	∅	{26}	{11}	∅	{11,17}	∅	{19}	∅	∅
C6	∅	{27}	∅	{14}	∅	{14,18,20,22}	{20}	∅	∅
C7	∅	{25}	{9}	∅	∅	{20}	{9,15,19,20}	∅	∅
C8	∅	∅	∅	∅	∅	∅	∅	∅	∅
C9	∅	∅	∅	∅	∅	∅	∅	∅	∅

Based on the matrix of the required transitions the binary route matrix can be created for validation of the two-hemisphere model. The binary route matrix is square NxN matrix where N is the amount of processes in the two-hemisphere model. The cell of the binary route matrix is equal to 1 if the path exists in the business process model between the two processes and it is listed in the corresponding cell of the matrix of the required transitions. The cell of the binary route matrix is equal to 0 if no such path exists. The binary route matrix is not included in this paper because of its large size (28 rows *28 columns). It is applied to the two-hemisphere model presented in Figure 3 in order to perform model validation according the following constraints, which must be satisfied:

1. Each node (process) in the business process model graph should be visited meaning the sum of values on each route matrix column except for the columns representing external input processes (processes having only outputs) should be greater than 0.

2. A route must exist from any external input process (processes having only outputs) to at least one external output process (processes having only inputs).

If the validation, which uses the binary route matrix is successful, the method should proceed to the next step – creation of class method signatures. Signatures are created by using the matrix of the required transitions. During signature creation the owner of the created method is also being detected. The owner class of the process is the output concept type, but the arguments of this method are the concepts that are required to get the process output data flow by this process according to the required transition matrix.

The outgoing external processes are an exception – they are owned by the input data flow concept. The external input processes have no inputs and return the owner. The external output processes have only one input (the owner) and have no return. For example, process P has three inputs with types C1, C2 and C3. This process has two outputs with types C4 and C5. The setting in the required transition matrix is such as in order to get the data flow with type C4, the data flows with types C1 and C3 are required, while in order to get the data flow with type C5, the data flow with type C2 is required. As a result, the class C4 will have method P(C1, C3):C4, while the class C5 will have method P(C2):C5.

If one of the input data flow assigned concepts is the same as the process owner class it may be removed from the argument list assuming the same object. This can be done automatically, however the authors would like also to enable a step by step validation of each created method for the purpose of checking, which argument can actually be removed. For example method signature generation for the concept *Process element* is done in following way:

— *ProcessElement* → *AddProcessToModel*(*ProcessElement* , *ProcessModel*): *ProcessElement* . Argument *Process element* is being automatically removed.

— *ProcessElement* → *AddProcessToModel*(*ProcessModel*): *ProcessElement*.

Where the following notation is used:

— Owner Concept → Process (Argument list – concepts assigned to input data flows): Resulting Concept (determined in the way described above).

When all the method signatures are determined a class model is being created. In the initial version of the approach only dependency between the classes is formally identified as the relationship based on the following definition:

— Dependency relationship from class A to class B is defined if there is the process P, which has an incoming dataflow (one or several) defined by concept A and outgoing flow (one or several) defined by concept B.

In addition to this, the authors suggest to define several more types of the relationships between the classes in the improved version of the approach. They are the following:

— Aggregation relationship from class A to class B if A is contained in B as an attribute.

— Association relationship between classes A and B is defined if relations between A and B exist in two ways (for example, aggregation from A to B and dependency from B to A).

— Generalization relationship and realization relationship definition process are described below.

After the aggregation, dependency and association relationships are defined, the created model is subject to iterative analysis in order to determine generalization relationship based on class attributes and methods: if A and B contain the same set of attributes a superclass C is being introduced for them. At this stage the modeller is asked to provide a class name for C. For example if A is ProcessElement and B is ConceptElement C may be named DiagramElement. This process is being repeated iteratively while at least one new class is being introduced as a result of iteration. After completing the superclass creation process an interface detection process is performed. In order to create an interface A and B should have the method with the same signature and they should be inherited from different superclasses (A from C, B from D, for example) thus making it impossible to escalate method in superclass. This process is being repeated until iteration produces no new interfaces.

As a result of added and improved transformations the UML class diagram containing classes with attributes and methods (with argument types and return types) and 5 different relationships – aggregation, dependency, generalization, association and realization is produced. The resulting class model generated for the two-hemisphere model, presented in Figure 3, is shown in Figure 5.

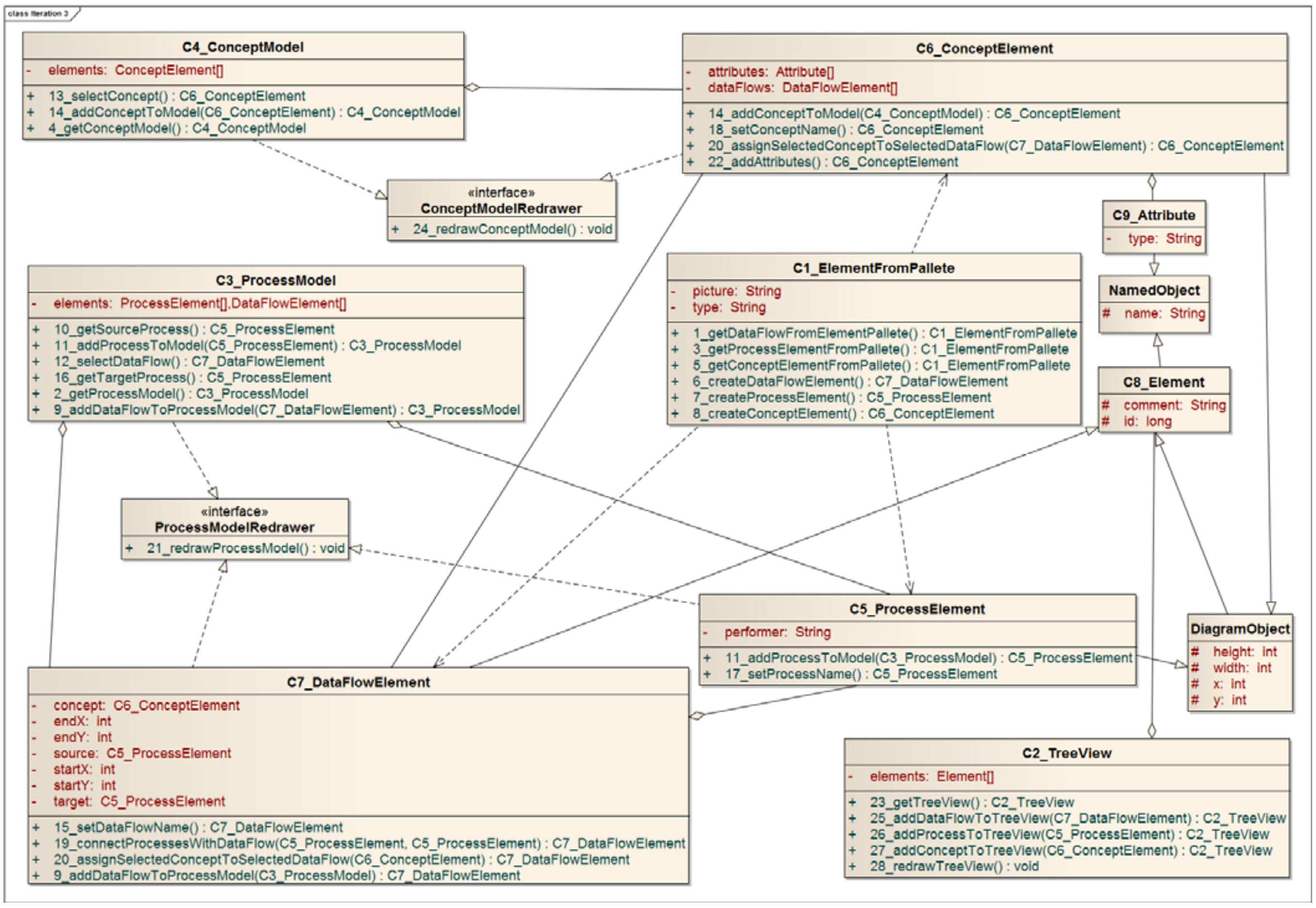

class iteration 3
C4_ConceptModel
- elements: ConceptElement[]
+ 13_selectConcept() : C6_ConceptElement
+ 14_addConceptToModel(C6_ConceptElement) : C4_ConceptModel
+ 4_getConceptModel() : C4_ConceptModel
C6_ConceptElement
- attributes: Attribute[]
- dataFlows: DataFlowElement[]
+ 14_addConceptToModel(C4_ConceptModel) : C6_ConceptElement
+ 18_setConceptName() : C6_ConceptElement
+ 20_assignSelectedConceptToSelectedDataFlow(C7_DataFlowElement) : C6_ConceptElement
+ 22_addAttributes() : C6_ConceptElement
«interface» ConceptModelRedrawer
+ 24_redrawConceptModel() : void
C9_Attribute
- type: String
NamedObject
# name: String
C1_ElementFromPallete
- picture: String
- type: String
+ 1_getDataFlowFromElementPallete() : C1_ElementFromPallete
+ 3_getProcessElementFromPallete() : C1_ElementFromPallete
+ 5_getConceptElementFromPallete() : C1_ElementFromPallete
+ 6_createDataFlowElement() : C7_DataFlowElement
+ 7_createProcessElement() : C5_ProcessElement
+ 8_createConceptElement() : C6_ConceptElement
C3_ProcessModel
- elements: ProcessElement[],DataFlowElement[]
+ 10_getSourceProcess() : C5_ProcessElement
+ 11_addProcessToModel(C5_ProcessElement) : C3_ProcessModel
+ 12_selectDataFlow() : C7_DataFlowElement
+ 16_getTargetProcess() : C5_ProcessElement
+ 2_getProcessModel() : C3_ProcessModel
+ 9_addDataFlowToProcessModel(C7_DataFlowElement) : C3_ProcessModel
C8_Element
# comment: String
# id: long
«interface» ProcessModelRedrawer
+ 21_redrawProcessModel() : void
C5_ProcessElement
- performer: String
+ 11_addProcessToModel(C3_ProcessModel) : C5_ProcessElement
+ 17_setProcessName() : C5_ProcessElement
DiagramObject
# height: int
# width: int
# x: int
# y: int
C7_DataFlowElement
- concept: C6_ConceptElement
- endX: int
- endY: int
- source: C5_ProcessElement
- startX: int
- startY: int
- target: C5_ProcessElement
+ 15_setDataFlowName() : C7_DataFlowElement
+ 19_connectProcessesWithDataFlow(C5_ProcessElement, C5_ProcessElement) : C7_DataFlowElement
+ 20_assignSelectedConceptToSelectedDataFlow(C6_ConceptElement) : C7_DataFlowElement
+ 9_addDataFlowToProcessModel(C3_ProcessModel) : C7_DataFlowElement
C2_TreeView
- elements: Element[]
+ 23_getTreeView() : C2_TreeView
+ 25_addDataFlowToTreeView(C7_DataFlowElement) : C2_TreeView
+ 26_addProcessToTreeView(C5_ProcessElement) : C2_TreeView
+ 27_addConceptToTreeView(C6_ConceptElement) : C2_TreeView
+ 28_redrawTreeView() : void

Fig. 5. Resulting class model using improved transformation

IV. Analysis of the Results

To perform analysis of the improvement offered for class diagram models are compared each to other. They are the following:

1) The UML class diagram generated by the initial version of the approach. It is shown in Figure 4.
2) The UML class diagram created on the basis of improvement of transformations. It is shown in Figure 5.
3) The class structure of BrainTool itself created by BrainTool's developers manually and can be treated as a "as-is" model. This model is not included in this paper due to the limited paper size, but it is also compared to other models, because this is the model of the real operating software.
4) The UML class diagram showing how the class model should look like, which can be named "TO-BE" model. This model was obtained on the basis of BrainTool development analysis and contains the improved AS-IS model with purpose to remove logical problems in it. TO-BE class model is shown in figure 6.

To estimate the obtained result, the first three models are compared with TO-BE model. Authors have selected four comparison criteria describing the core set of the UML class diagram:

1. Classes (how many correct classes were generated).

2. Attributes (how many attributes in each class are the same as in the TO-BE model).

3. Methods (how many methods in each class are the same as in the TO-BE model).

4. Associations (how many among the obtained associations are correct comparison to the TO-BE model).

The names in the TO-BE model and in the obtained models may differ, therefore the comparison were done manually on the basis of the obtained elements semantics. If the meaning of a differently named elements is the same, in the comparison process these elements are used as the same elements.

To estimate the result the authors calculated the difference between the TO-BE model and each of the obtained models. To find this difference the Pearson Squared distance were calculated:

$$X^2_{\text{Pearson}} = \sum((O_i\text{-}E_i)/E_i^{1/2})^2 = \sum(O_i\text{-}E_i)^2/E_i$$

The Euclid distance is not suitable for such purpose, because the result depends on the number of elements in the TO-BE model. Indeed, the Pearson Squared distance allows elimination of this defect. Using the Pearson Squared distance, the TO-BE model elements are the hypothesis H_0, which is correct by definition.

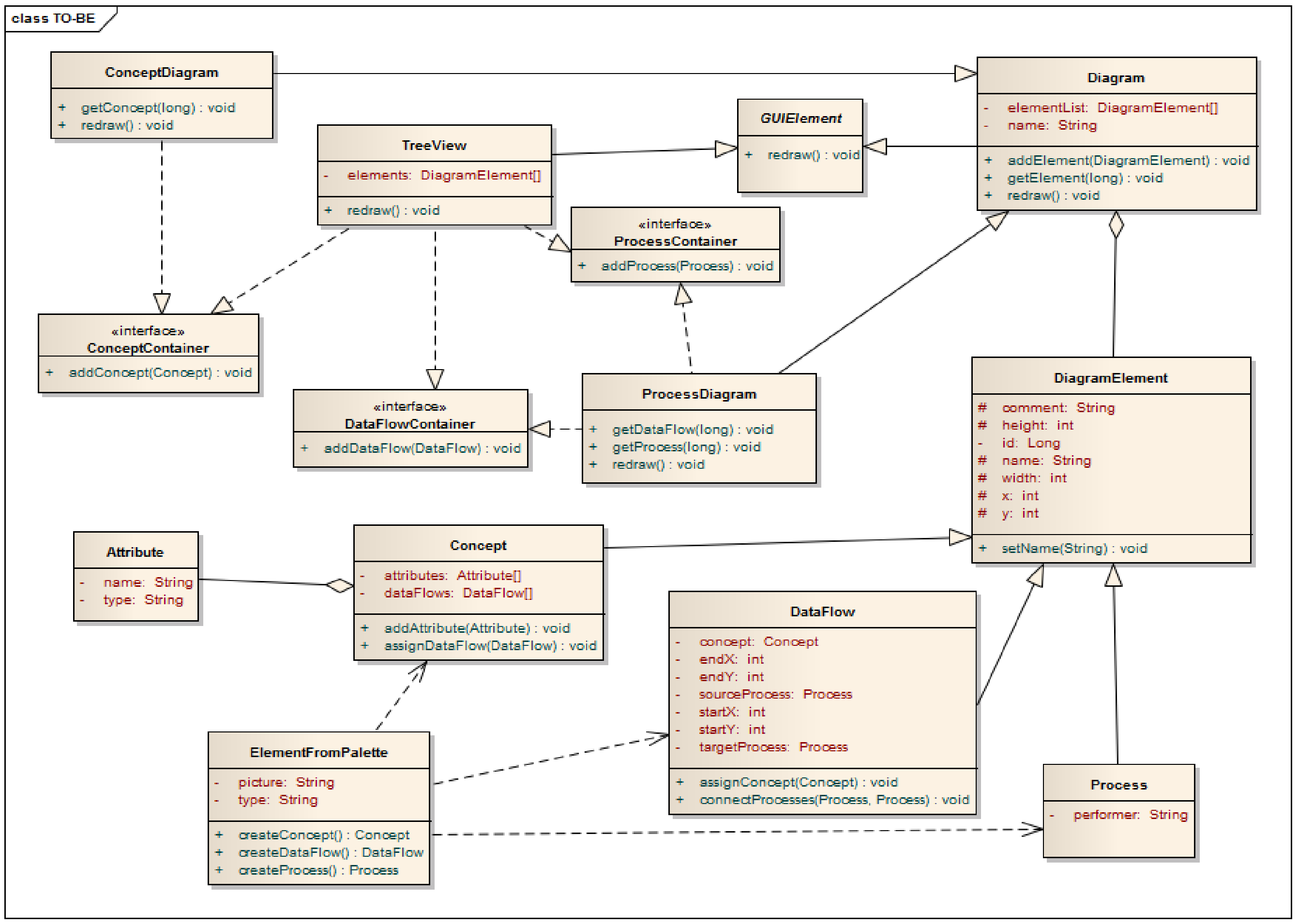

class TO-BE
ConceptDiagram
+ getConcept(long) : void
+ redraw() : void
TreeView
- elements: DiagramElement[]
+ redraw() : void
GUIElement
+ redraw() : void
Diagram
- elementList: DiagramElement[]
- name: String
+ addElement(DiagramElement) : void
+ getElement(long) : void
+ redraw() : void
«interface» ProcessContainer
+ addProcess(Process) : void
«interface» ConceptContainer
+ addConcept(Concept) : void
«interface» DataFlowContainer
+ addDataFlow(DataFlow) : void
ProcessDiagram
+ getDataFlow(long) : void
+ getProcess(long) : void
+ redraw() : void
DiagramElement
# comment: String
# height: int
- id: Long
# name: String
# width: int
# x: int
# y: int
+ setName(String) : void
Attribute
- name: String
- type: String
Concept
- attributes: Attribute[]
- dataFlows: DataFlow[]
+ addAttribute(Attribute) : void
+ assignDataFlow(DataFlow) : void
DataFlow
- concept: Concept
- endX: int
- endY: int
- sourceProcess: Process
- startX: int
- startY: int
- targetProcess: Process
+ assignConcept(Concept) : void
+ connectProcesses(Process, Process) : void
ElementFromPalette
- picture: String
- type: String
+ createConcept() : Concept
+ createDataFlow() : DataFlow
+ createProcess() : Process
Process
- performer: String

Fig. 6. TO-BE class model

For each of the criteria vector of N elements was constructed. The vector for the TO-BE model contains a number of elements of current criteria. In the obtained model vector a number of right elements are present minus the number of extra elements. Thus, the vector for the classes criteria for the TO-BE model is (zero shows that such class is not present in the TO-BE model, yet it is present in the compared model):

(1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0)

For the model, generated with the initial version of the approach, such vector is:

(0,0,1,1,1,1,1,1,0,0,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1)

For other criteria only those elements from classes were brought to the comparison vectors, that are present in both models, because the method error associated with the class structure difference is included in the first criteria. For example, the attributes criteria vector for the TO-BE model:

(1,0,0,7,2,2,7,2,1)

It means that the first class from model class list contains one correct attribute, the second class contains zero correct attributes, etc. For the obtained by an improved version of the approach model in comparison to the TO-BE model such vector is:

(1,-1,-1,4,1,2,7,2,1)

The task is to find the distance from the H_0 model and each of the obtained models and to compare the results to determine, how effective each method is and how much improvement was obtained in comparison to the initial version of the approach.

One exception was found during the obtained models analysis. While the obtained model contains the elements, the TO-BE model contains no such elements. As a result, the normalisation procedure cannot be performed. To solve this problem, we need to understand the possible values, which could be correct.

For example, if the TO-BE model contains three elements, whereas an obtained model contains no right elements, then the Pearson Squared distance would be $(0-3)^2/3=3$. In the opposite case (the TO-BE model contains no-elements, but obtained model contains three extra elements) the distance, quite logically, should be the same. Indeed, in case when the TO-BE model contains no elements and the obtained model contains three extra elements, the 3 must be simply added to the calculated distance. For example, for the attribute criteria described above, the Pearson Squared distance is:

$X^2_{Pearson}=(1-1)^2/1+1+1+(4-7)^2/7+(1-2)^2/2+(2-2)^2/2+(7-7)^2/7+(2-2)^2/2+(1-1)^2/1=3.79$

The results of entire calculations are described in the Table 2. Because the result of the each obtained model is a three-dimensional vector, the module of this vector is calculated. To be able to evaluate the obtained results, these vectors were normalised. As seen in the table above, the class diagram that was generated on the basis of improved version of the approach is located rather close to the class diagram created manually under the development of BrainTool, in comparison to the class diagram generated by the initial version of the method.

TABLE II
REQUIRED TRANSITION MATRIX

	Class criteria	Attributes criteria	Methods criteria	Associations criteria	Normalised module of the vector
Model AS-IS	5	3.5	4.25	10.08	0.09
Initial version	11	76.79	37.38	98	0.95
Improved method	9	3.79	36.13	13.66	0.29

Actually, the resulting distance is so close, that the authors can assert that an improved version of the approach can be used to generate class diagrams for actual use with the benefit of its automatic generation.

The proposed experiments are also applied to several problem domains studied during definition of the initial method. They are the fragments of driving school administration, hotel room booking, and insurance policy agreement. All of these give the same results for comparison of initial version to the improved one; however, they are not shown in this paper due to paper size limitation.

V. Related work and comparison to other modelling approaches

The object-oriented approach is based on representation of the objects, which interact in the developed system. The most commonly used model of the system in this approach is class diagram. The first skeleton of the class diagram was proposed in 1986 by G.Booch [13] and refined in 1991 by J.Rumbaugh [14] with the suggestion to denote class with a rectangle, which includes class name, names and types of the attributes and names, types and parameters of operations. The idea of automatic generation of objects interaction has been present actually since the first outlines in the object representation area.

During the first studies of the object-oriented approach entity-relationship (ER) diagrams served as the basis for obtaining class diagram [15]. In other words, it was the change of notation. Classes have only names and attributes in this case. On the other hand, different tools propose to create ER diagram with the notation of UML class diagram. Therefore, we can conclude that ER and class diagram are very close, and represent similar kind of information.

Automatic generation of the system static structure is researched not only in the object-oriented approach. Aspect-oriented approach proposes methods for obtaining system static structure. The automatic generation of some view of system structure based on system behaviour representation is the popular topic for researches. Amparo Navasa etc. in [16] propose to generate system architectural view by using case and sequence diagrams. They developed an appropriate tool, yet this tool does not offer the possibility to construct initial models and to receive graphical representation of results.

Studies of automatic class generation can be found in the object-oriented development area as well. Attempts to receive class diagrams from the requirements in natural language are one of the popular lines of research. For example, the approach LIDA [17], which proposes linguistic analysis of system requirements with the help of developed tool. The LIDA tool helps to perform analysis only, it generates nothing but only has graphical representation of class diagram, where all the elements are defined by the user. CM-Builder [18] provides the method and the tool that generates classes, attributes, associations and defines the multiplicity of associations with the help of linguistic analysis of the requirements. The methods and special kinds of relationships – aggregation and generalisation are not generated with this tool. NL-OOML [19] is another tool. It generates only classes with the attributes and methods without any relationships. GOOAL [20] is yet another tool based on linguistic analysis of system requirements. It generates classes, attributes and methods. GOOAL obtains associations with semi-formal transformation and does not define special kinds of relations such as aggregation and generalisation and multiplicity. TRADE [21] is the approach, which provides model-to-model transformation for class diagram generation. At the basis is the requirements model, which consists of mission statement, function refinement tree and use case model. TRADE proposes a guided specification sequence diagram, and guided translation to conceptual model. It receives classes, attributes, methods, associations with multiplicity and aggregations, but cannot define generalisations. TRADE announced tool development, yet did not publish any outlines about it, thus the tool is not available for download, either as a free or as a commercial version.

Conference MODELS 2012 declares the workshop for comparison of different modelling methods. It insists that the problem of formal obtaining of the models and model elements remains a topical issue and has not been solved yet. But methods announced for this workshop relate to aspect-oriented and service-oriented development areas. The methods discuss the problem of system structure generation from this point of view, for example [16] mentioned above or [22] which provides an improved class diagram from some initial class diagram. SBVR2UML [23] proposes transformation from semantics of business vocabulary and rules to class diagram with linguistic analysis. It generates all elements of class diagram, and possesses a tool. All the above mentioned methods except TRADE have appropriate tools. For initial information they provide editor for textual requirements. CM-Builder represent class diagram in textual way; LIDA, GOAL, NL-OOML and SBVR2UML provide graphical representation of generated class diagram. Two-hemisphere approach proposes to generate elements of the class diagram from initial two-hemisphere model. It has a tool, which provides the possibility to create initial model and transform it into class diagram in the formal way.

Table 4 shows comparative analysis of the two-hemisphere approach to other methods and tools, described above. Part of the criteria for comparison of methods is mentioned in [23] - the core elements of class diagram, the rest is added to more descriptive comparison.

TABLE 4.

COMPARISON OF 2HMD APPROACH TO OTHER MODELLING APPROACHES PROVIDING THE POSSIBILITY TO CREATE THE UML CLASS DIAGRAM

Method / **Criteria**	**CM-Builder**	**LIDA**	**GOOAL**	**NL- OOML**	**SBVR2UML**	**TRADE**	**2HMD approach**
Initial information	System req-ts	System req-ts	System req-ts	System req-ts	Semantics of Business Vocabulary and Rules specification	Requirements model	Two-hemisphere model
Class diagram elements							
Classes	Yes	Manually	Yes	Yes	Yes	Yes	Yes
Attributes	Yes	Manually	Yes	Yes	Yes	Yes	Yes
Methods	No	Manually	Yes	Yes	Yes	Yes	Yes
Associations	Yes	Manually	Semi-NL	No	Yes	Yes	Yes
Multiplicity	Yes	Manually	No	No	Yes	Yes	Manually
Aggregation	No	No	No	No	Yes	Yes	Yes
Generalization	No	No	No	No	Yes	No	Yes
Transformation base	Linguistic analysis	Linguistic analysis	Linguistic analysis	Linguistic analysis	Linguistic analysis	Semi-formal transformation model-to-model	Formal transformation model-to-model
Tool support							
Tool available	Yes	Yes	Yes	Yes	Yes	No	Yes
Model editor for initial information	Text editor	Text editor	Text editor	Text editor	Text editor	--	Graphical editor
Graphical representation of class diagram	No	Yes	Yes	Yes	Yes	--	Yes

The table shows, that methods are mainly based on the linguistic analysis and pressure on the user to write documentation in the defined form. It can cause the problems associated with the quality of problem area description. E.g., it may produce similar classes due to used synonyms, or redundant relationships due to incorrect wordings. TRADE [21] method proposes transformations from the initial models, however, transformations cannot not be executed in full automatic mode, and tool for TRADE has not been announced. Two-hemisphere approach proposes formal transformation to class diagram, which is based on the model and is automated by the tool. The tool in this case solves the problem of obtaining the system structure from initial information of a system.

VI. Conclusion

The initial version of the two-hemisphere model-driven approach gives the possibility to generate UML class diagram from the two models – the business process model and the concept model. This is a great benefit, because instead of manual creation of the UML class diagram directly from information about the problem domain based on the principles of object-oriented analysis, the two-hemisphere model-driven approach allows using already existing business artefact – a business process diagram, which is widely used in many enterprises, and the structure of information flows between the processes is definable under description of user stories.

Therefore, the two-hemisphere model that is created with minimal efforts and intuitively understood by customer can be used for automatic generation of class diagram skeleton, which can be later reviewed and used in software development.

The main benefit of the approach improvement is in the expansion of the set of generable elements of the UML class diagram. The improved approach allows defining generalisation and realisation relations. Another improvement is that there are no more problematic processes requiring additional decomposition of the customer. The improved approach in the method signatures defining allows avoiding this limitation. The next step of method improvement can move the focus on to the study of class aggregation heuristics, due to its ambiguity also under manual creation of the UML class diagram. The authors suggest that the proposed matrix can be useful for formalisation of the aggregation identification.

Based on the analysis of obtained results the authors can state, that the UML class diagram, generated with the improved approach is closer to the "ideal" result than the UML class diagram generated with the initial approach. Actually it closely approached the possibility to use it in software development.

Furthermore, comparison with other advanced approaches for generation of the UML class diagram from some kind of the presentation of problem domain shows the applicability and appropriateness of the two-hemisphere model and its supporting BrainTool in software development.

The future direction of the research could be to introduce all the transformations offered in the paper into the next version of BrainTool.

Acknowledgements

The research presented in the paper is partly supported by the Grant of Latvian Council of Science No. 09.1269 "Methods and Models Based on Distributed Artificial Intelligence and Web Technologies for Development of Intelligent Applied Software and Computer System Architecture".

References

[1] J. Krogstie, "Integrating enterprise and IS development using a model driven approach." In: 13th International Conference on Information Systems Development – Advances in Theory, Practice and Education. Vasilecas O. et al. (Eds). Springer Science+Business media, Inc. 2005. pp.43-53.

[2] O. Nikiforova and M. Kirikova, "Two-Hemisphere Model Driven Approach: Engineering Based Software Development." In: CAiSE 2004 16th International Conference on Advanced Information Systems Engineering June 7-11, Proceedings, 2004, pp. 219-233.

[3] O. Nikiforova., "General framework for object-oriented software development process" In: Scientific Proceedings of Riga Technical University (13), 2002, pp. 132–144.

[4] O.Nikiforova, M.Kirikova, N.Pavlova, „Two-Hemisphere Driven Approach: Application for Knowledge Modelling", In proceedings of the Seventh IEEE International Baltic Conference on DB and IS (BalticDB&IS'2006), O. Vasilecas, J. Eder, A. Caplinskas (Eds.), Vilnius, Lithuania, 2006, pp. 244-250

[5] O. Nikiforova and N. Pavlova "Development of the Tool for Generation of UML Class Diagram from Two-Hemisphere Model." In: Proceedings of The Third International Conference on Software Engineering Advances (ICSEA), International Workshop on Enterprise Information Systems (ENTISY). Mannaert H., Dini P., Ohta T., Pellerin R. (Eds.), Published by IEEE Computer Society, Conference Proceedings Services (CPS), 2008, pp. 105-112.

[6] O. Nikiforova "Two Hemisphere Model Driven Approach for Generation of UML Class Diagram in the Context of MDA", e-Informatics Software Engineering Journal - Volume 3, Issue 1, Huzar Z., Madeyski L. (eds.), Wrocław University of Technology, Institute of Applied Informatics, Wrocław University of Technology, Wrocław, Poland, Copyright by Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, Poland, [Online]. Available: http://www.e-informatyka.pl/wiki/e-Informatica_-_Volume_3, 2009, pp. 59-72.

[7] 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, 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques, Revised Selected Papers, Huzar Z. et al (Eds.), LNCS Sublibrary: SL 2 – Programming and Software Engineering, Springer, 2011, pp. 118-130.

[8] O. Nikiforova and N. Pavlova, "Foundations on generation of relationships between classes based on initial business knowledge." In: Information Systems Development: Towards a Service Provision Society. Springer US, 2009, pp. 289–297.

[9] O.Nikiforova, N.Pavlova and J.Grigorjev, "Several Facilities of Class Diagram Generation from Two-Hemisphere Model" In: 23rd International Symposium on Computer and Information Sciences (ISCIS 2008). Istanbul, Turkey, 27-29 October 2008, [Online]. Available: http://ieeexplore.ieee.org/, 2008, pp.1-6.

[10] O.Nikiforova, "Object Interaction as a Central Component of Object-Oriented System Analysis." In: International Conference „Evaluation of Novel Approaches to Software Engineering" (ENASE 2010), Proceedings of the 2nd International Workshop „Model Driven Architecture and Modelling Theory Driven Development" (MDA&MTDD 2010), Osis J., Nikiforova O. (Eds.), Greece, SciTePress. 2010, pp. 3-12.

[11] Website of BrainTool. [Online]. Available: braintool.rtu.lv/
[12] O. Nikiforova, N. Pavlova, K. Gusarovs, O. Gorbiks, J. Vorotilovs, A. Zaharovs, D.Umanovskis, J. Sejans, "Development of the Tool for Transformation of the Two-Hemisphere Model to the UML Class Diagram: Technical Solutions and Lessons Learned", Proceedings of the 5th International Scientific Conference „Applied Information and Communication Technology 2012", held on 26-27 April 2012, in Jelgava, Latvia, 2012, pp. 11-19.
[13] G. Booch, "Object-oriented analysis and design with applications", Addison Wesley, 1986.
[14] J. Rumbaugh, M.Blaha, W.Premerlani, F.Eddy, W Lorensen, "Object Oriented modelling and design", Englewood Cliffs: Prentice-Hall, Inc/, New Jersey, 1991.
[15] P.P. Chen, "Historical Events, Future Trends, and Lessons Learned." Software pioneers, Springer-Verlag New York, Inc. New York, 2002, pp. 296 – 310.
[16] A.Navasa, M. A.Pérez-Toledano, J. M. Murillo, "An ADL dealing with aspects at software architecture stage" Information and Software Technology 51, 2009, pp. 306–324.
[17] S.V.Overmyer and O.Rambow, "Conceptual Modelling through Linguistics Analysis Using LIDA." 23rd International Conference on Software engineering, July 2001.
[18] H. M. Harmain and R.Gaizauskas "CM-Builder: A Natural Language-Based CASE Tool for Object- Oriented Analysis." Automated Software Engineering. 10(2), 2003, pp.157-181.
[19] G.S.Anandha, G.V. Uma "Automatic Construction of Object Oriented Design Models [UML Diagrams] from Natural Language Requirements Specification" PRICAI 2006: Trends in Artificial Intelligence, LNCS 4099/2006, 2006, pp. 1155-1159.
[20] H. G. Perez-Gonzalez and J.K. Kalita, "GOOAL: A Graphic Object Oriented Analysis Laboratory." 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '02), NY, USA, 2002, pp. 38-39.
[21] E.Insfrán, O.Pastor, and R.Wieringa, "Requirements Engineering-Based Conceptual Modelling." Requir. Eng., 7, 2002 pp. 61-72.
[22] A.Hovsepyan, S.Baelen, Y.Berbers and Joosen W., "Generic Reusable Concern Compositions." In Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications (ECMDA-FA '08), Ina Schieferdecker and Alan Hartman (Eds.). Springer-Verlag, Berlin, Heidelberg, 2008, pp. 231-245.
[23] H.Afreen, I.S.Bajwa, B.Bordbar, "SBVR2UML: A Challenging Transformation," fit, 2011 Frontiers of Information Technology, 2011, pp.33-38.

Oksana Nikiforova earned her Ph.D. in information technologies (system analysis, modelling and design) in Riga Technical University, Latvia, in 2001.
At present, she is Full Professor at the Department of Applied Computer Science, Riga Technical University, where she has been studying and working since 1999. Her current research interests include the object-oriented system analysis and modelling, with special focus on the issues in the framework of Model-driven Software Development (MDSD). She has published widely in these areas and has been awarded several grants. She participated and managed several research projects related to system modelling, analysis and design, as well as participated in several industrial software development projects.
She is member of RTU Academic Assembly, Council of the Faculty of Computer Science and Information Technology, RTU Publishing Board, RTU Scientific Journal Editorial Board, etc. She co-chairs the workshops focused on MDSD – MDA 2009 in conjunction with ADBIS, MDA&MTDD 2010 and MDA&MDSD 2011 in conjunction with ENASE. She was awarded a RTU Young Scientist of the Year 2009.
E-mail: oksana.nikiforova@rtu.lv

Konstantins Gusarovs took master's degree in computer systems in Riga Technical University, Latvia, in 2012.
At present he is scientific assistant at the Department of Applied Computer Science in Riga Technical University.
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

Olegs Gorbiks took his master's degree in computer systems from Riga Technical University, Latvia in 2012.
Currently, he is the first year Ph.D. student and scientific assistant at the Department of Applied Computer Science, Riga Technical University.
He is PHP developer in Ambergames Corp . His current research interests include object-oriented software development and automatic obtaining of program code.
E-mail: olegs.gorbik@rtu.lv

Natalja Pavlova earned the Ph.D. degree in information technologies (system analysis, modelling and design) from Riga Technical University, Latvia in 2008.
Currently she is leading researcher at the Department of Applied Computer Science, Riga Technical University, where she studied and worked since 2003. For two years she worked as programmer and for five years as software quality assurance specialist. Her current research interests include system analysis and design, object-oriented software development and automatic obtaining of system models and program code. She has published widely in these areas and has been awarded several grants. She has participated in several industrial software development projects.
E-mail: natalja.pavlova@rtu.lv