An Approach to Generation of the UML Sequence Diagram from the Two-Hemisphere Model

Oksana Nikiforova, Konstantins Gusarovs
Riga Technical University
Riga, Latvia
konstantins.gusarovs@rtu.lv,
oksana.nikiforova@rtu.lv

Anatoly Ressin
SIA AssistUnion
Riga, Latvia
anatoly@assistunion.com

Abstract – Modelling for model-based development of software concerns the representation of both system structure and behavior. To this end, in order to satisfy this requirement Unified Modelling Language (UML) provides a range of static and dynamic diagram types; of these class and sequence diagrams are most frequently used. In contrast to the UML class diagrams, which are well researched and discussed in the literature, sequence diagrams are a 'dark horse' especially in regard layout and transformation. This paper proposes an approach to the generation of UML sequence from two-hemisphere model with the main attention to a dynamic aspect of the system.

Keywords – two-hemisphere model; UML sequence diagram, model transformation; finite-state machines; regular expressions.

I. INTRODUCTION

One of the trends for software developments, namely Model-Driven Software Development (MDSD) [1], is being widely introduced in order to support both the business analysis of the system being built as well as its implementation. According to MDSD, the development process is started with the modelling of the problem domain in order to produce the software domain model and to achieve once developed system model reuse [2].

The primary benefit of MDSD is an ability to provide a big-picture view of the architecture of the entire system. Usage of MDSD requires a common modelling notation and a system that can be used on all the stages of the development. Object Management Group (OMG) proposes its standard – Unified Modelling Language (UML) [3] that nowadays is being widely adopted. UML defines a notation for a set of the diagrams used for modelling the different aspects of the system – both static and dynamic. According to the research provided by Scott Ambler [4], the most popular UML diagrams in software development projects are the UML class and sequence diagrams. Static modelling is mainly done using UML class diagram that defines the general structure of the system and can be used as a basis for the implementation. The UML sequence diagram serves to define dynamic aspect presenting object interactions in the system. The UML sequence diagram includes both classes inside of the system as well as its environments represented as a set of the actors. These elements are exchanging the messages that are being placed on the lifelines that allow defining both the interaction patterns as well as the sequence of the interactions.

The UML class diagrams have been quite well studied in the MDSD-related researches and several methods exist for producing the UML class diagrams from the different types of the notations representing the problem domain. However, situation with the methodological modelling of the system dynamic is worse – only a few MDSD approaches focus on this question. As a result, issues associated with the transformations of the system dynamics are one of the main reasons why MDSD adoption is quite slow nowadays.

Since 2004, the research group lead by Oksana Nikiforova has been working on the applications of two-hemisphere model for the generation of different sets of the UML elements. This paper focuses on the UML sequence diagram, especially, its timing aspect. In this paper authors propose an approach that allows a way of transforming a two-hemisphere model into the UML sequence diagram based on the sequence diagram topology and using finite state machine (FSM) [5][6] as an auxiliary model within the series of model transformations. In addition, authors are going to analyze current limitations of the notational conventions proposed for the two-hemisphere model and argue the ideas of improving it in order to be able to receive results that are more precise.

The rest of the paper is structured as follows. Section 2 contains a short description of the two-hemisphere model. Section 3 describes related researches in the MDSD context providing an insight to similar existing methods and techniques. Section 4 provides a short description of the UML sequence diagram that is selected as a target model for the proposed transformation. The transformation method itself is described in the section 5 and a simple example is being analyzed in the section 6. In Section 7, method application results, as well as current limitations, are being analyzed and the possible solutions, in order to lift these limitations are being offered. Finally, the section 8 contains authors' conclusions and plans for the future work in this area.

II. TWO-HEMISPHERE MODEL AT A GLANCE

The two-hemisphere model-driven approach first published in 2004 [8] introduces an idea of joining elements both from the static and dynamic presentation of problem domain in the source model that consists of two diagram types (see Figure 1):

 ISBN: 978-1-61208-498-5

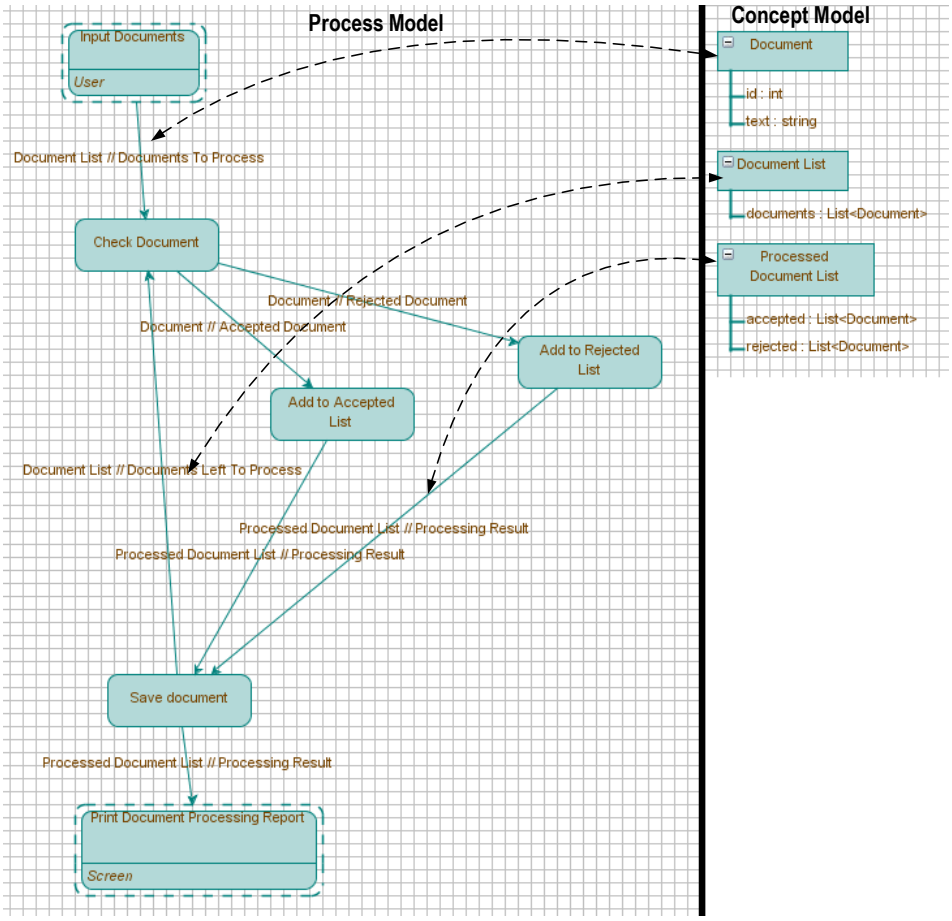

Figure 1. Two-hemisphere model for document processing created by BrainTool [7]

1) Concept model describes objects (or data types) used in the analyzed system presenting those in a form of concepts. Each concept has its attributes that are describing its inner structure. An attribute may be of the primitive type – such as an integer or string as a well as refer to another concept in the concept model.

2) Process model describes interactions performed inside the system. It consists of the processes connected with the data flows. Processes describe activities in the system and are divided into internal and external – external processes are defining system's interaction points with its environment and should only produce or consume the data flows whilst internal processes are defining activities inside the system and should both consume and produce the data. Data flows are used for the process interconnection and are describing the data migration inside the system. Each data flow is assigned to a concept of a given type thus linking both models.

A valid two-hemisphere model contains a single concept diagram as well as several process diagrams each of which might be describing a single activity inside the system, for example, it is possible to construct the process diagram for each of the use cases (or user stories) defined in the system requirements.

Currently, there exist several methods (and tools) for converting the two-hemisphere model into the UML class and sequence diagrams that are described in the papers [8]-[12]. Similarly to the situation in the MDSD area, transformations targeting the UML class diagram are a well-studied area.

III. Related Work

A two-hemisphere model-driven approach can be described as one of the branches in the UML-DFD method family tree. UML-DFD methods are based on the usage of the dataflow diagram (that is being called process diagram in the two-hemisphere model-driven approach). Original transformation method that allows dataflow diagram (DFD) conversion into the UML class diagram is described in 2004 in the paper [13]. It involved composite transformation from the system requirements into the UML class diagram that consists of the 9 steps:

1. System requirement identification.
2. Use case diagram creation.
3. Composition of the textual scenario for the each of the use cases.
4. A transformation of the use case diagram into the initial object diagram.
5. Reducing of the initial object diagram by analyzing object functional features and

 ISBN: 978-1-61208-498-5

grouping, dividing and removing some of the initial objects.
6. Constructing the data flow diagram using reduced initial object set.
7. Identification of the data flows and data vocabulary creation.
8. Modelling of the system behavior using the activity diagram.
9. Data flow transformation into the resulting UML class diagram.

After the initial publication of the UML-DFD approach, several ways of improving it have been offering by the different authors in 2004-2012 [14]-[17].

The main difference between the UML-DFD approach and the two-hemisphere mode-driven approach is an initial presentation of the system and a resulting data model type that has been produced. UML-DFD based methods create so-called anaemic data model [18][19]. The main principle covered in the anaemic data model design can be stated with a single phrase: "data are data", which means that domain classes should contain no business logic, which in turn should be contained in the appropriate services processing the data. Two-hemisphere model-driven approach in turn, produces so-called rich data model in which domain classes share the part of the system functional features. While it may seem that the anaemic data model breaks the idea of the Object-Oriented Programming (OOP), in practice it is widely used [20] due to the fact it well suits the commonly used Model-View-Controller (MVC) pattern [21].

UML-DFD family methods are producing an activity diagram that can be later translated into the UML sequence diagram. In turn, in 2013, a method for transforming the two-hemisphere model into the UML sequence diagram was offered by O. Nikiforova, L. Kozacenko and D. Ahilcenoka [12][22]. The transformation used in the proposed approach was similar to the transformation used for generating the UML class diagram from the two-hemisphere model [22].

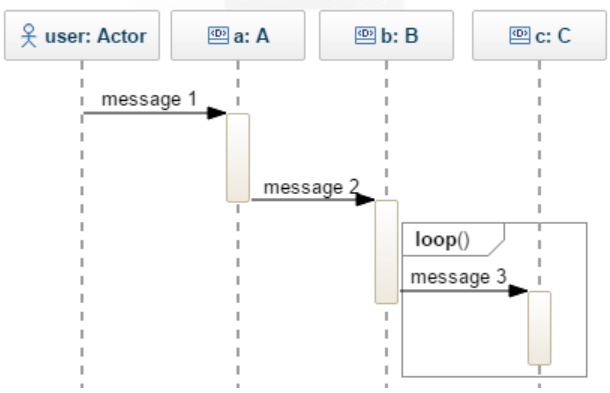

Figure 2. An example of the UML sequence diagram

The main idea behind the two-hemisphere model-driven approach was always a responsibility sharing, which in turn means, that the approach tends to find the most appropriate class for hosting one of the processes in process model as a method. As a result, both UML class and UML sequence diagrams generated from the two-hemisphere model will utilize the rich data model.

IV. UML Sequence Diagram at a Glance

The UML sequence diagram is one of the well-known UML artefacts used for the representation of a modeled system's dynamical features. It focuses on the definition of the object interaction in the correct sequence. The diagram's vertical axis is used to display the time. It is being directed downwards with the beginning of the interaction in the model's top. It is possible to define the following components of the UML sequence diagram:

- Object – is an instance of a class that reflects real system's object.
- Lifeline – represents the time of object existence and the participation in the interactions with the other diagram elements.
- Actor – is a specific type of object. It is not the part of the analyzed system; however, it interacts with it and represents the part of the system's environment.
- Message – represents a single communication fact between actors/objects. Message connects its sender and its receiver and can have additional arguments.
- Fragment – is used to combine several messages into the block. Fragments can represent different multi-message interaction patterns: parallel interaction, alternate interaction as well as loop (repeated) interaction etc.

An example of the simple UML sequence diagram is given in Figure 2. It consists of "user" actor, objects A, B and C and their appropriate lifelines, messages 1, 2 and 3 and a loop fragment that contains the third message.

V. Proposed Transformation Method

The two-hemisphere model uses the process diagram that should contain performers for the external processes (ones that determine system's integration with its environment). These performers are then transformed into the actors of the UML sequence diagram. Another diagram that is a part of the two-hemisphere model is a concept diagram holding the concepts that are transformed into objects of the UML sequence diagram.

As a result, all of the actors and the objects of a target UML sequence diagram are created. Next part of the transformation method involves the creation of the messages and the identification of the fragments in order to finalize the UML sequence diagram creation. This can be achieved by representing the process diagram in a different way and applying a transformation to a changed process diagram.

 ISBN: 978-1-61208-498-5

It is possible to transform the process model into a FSM [5][6] by transforming both the processes and the data flows into the transitions and inserting the state nodes between them. During this transformation, an additional change applies to the diagram's topology. The first change involves the creation of an initial node that is connected to all the external processes that only produce data flows and do not consume them. The second topology change is the creation of the final node that is connected to all the external processes that only consume and do not produce any of the data flows. In this case, the connection is made via special empty transition that is not in any way connected to the original model and is only used for the processing of an intermediate model. Former transformation can be described using the following pseudocode:

```
let de = Process Diagram Element
let parents = {de → [de]}
let children = {de → [de]}
let fsm = new Finite-State Machine

for each data flow d in data flows:
    let t = d.target
    let s = d.source

    children += {d → t}
    children += {s → d}

    parents += {t → d}
    parents += {d → s}

let initialProcesses = []
for each process p in processes:
    if not parents contains p:
       initialProcesses += p
let stateId = 1
let is = new State with stateId
let ts = new State with maximal
    allowed id
let nodeMap = {de → State}

fsm.addState(initialState)
fsm.addState(terminalState)

for each process p in initialProcesses:
    let n = new State with stateId
    nodeMap += {p → n}
    fsm.addState(n)
    let e = new Transition with p
    fsm.addTransition(e, is → n)
    stateId += 1

let open = [initialProcesses]
let closed = []

while open is not empty:
    let next = first entry from open
    let childs = children[next]
    let node = nodeMap[next]

    if childs is empty:
        let edge = fsm.findTransition(
            node → ts)

        if edge == null:
            let e = new Empty Transition
            fsm.addTransition(e, node → ts)

    for each child in children:
        let n = nodeMap[child]

        if n == null:
            n = new State with stateId
            stateId += 1

        nodeMap += {child → n}
        fsm.addState(n)

        let e = new Transition with child
        fsm.addTransition(e, node → n)

        if not closed contains child:
            open += child

        closed += next
```

In order to demonstrate the process model to the FSM transformation, a simple example is provided in Figure 3 and Figure 4. In this example, a process model containing two external and one internal processes interconnected with two data flows is transformed into the FSM .

After the initial FSM has been created from the process diagram, it is possible to minimize it and convert into the regular expression. In order to perform those operations, authors define the following formulas for the transition manipulation:

- `conj` – combines two transitions `a` and `b`, if transition b follows transition a (Formula 1):

$$conj(a,b) = \begin{cases} \emptyset, if\ a = \emptyset\ and\ b = \emptyset \\ b, if\ a = \varepsilon \\ a, if\ b = \varepsilon \\ a\ b, otherwise \end{cases} \quad (1)$$

- `disj` – combines two transitions `a` and `b`, if transition a is an alternative for transition `b` (Formula 2):

$$disj(a,b) = \begin{cases} \emptyset, if\ a = \emptyset\ and\ b = \emptyset \\ b, if\ a = \emptyset\ or\ a = \varepsilon \\ a, if\ b = \emptyset\ or\ b = \varepsilon \\ a, if\ a = b \\ (a)\Theta(b), otherwise \end{cases} \quad (2)$$

- `star` – marks a repeating transition `a` (Formula 3):

$$star(a) = \begin{cases} \emptyset, if\ a = \emptyset \\ \varepsilon, if\ a = \varepsilon \\ (a)*, otherwise \end{cases} \quad (3)$$

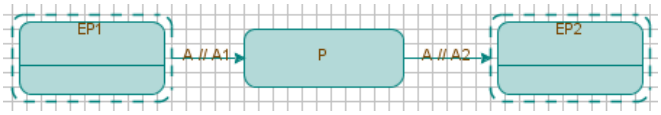

Figure 3. A source process diagram

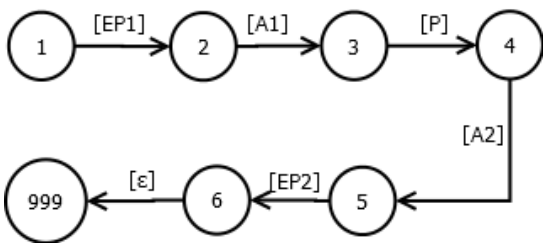

Figure 4. A finite state machine for the source process diagram

Using these formulas, it is possible to perform the FSM minimization using the following rules (where `ie` is incoming transition, `oe` is outgoing transition and $c_1 \ldots c_n$ are circles in a FSM graph):

- If a state has only one incoming and only one outgoing transition, it is possible to eliminate this state and merge those transitions into the single one (Formula 4):

$$newTransition = conj(ie, oe) \quad (4)$$

- If there are several transitions between two states, it is possible to merge these transitions into a single one (Formula 5):

$$newTransition = disj(ie, oe) \quad (5)$$

- If a state has only one incoming and only one outgoing transition and also has circles from itself to itself, it is possible to remove this state and create a new transition (Formula 6):

$$\begin{gathered} cDsij = \emptyset \\ for\ c\ in\ c_1, c_2, \ldots, c_n : cDisj = disj(cDisj, star(c)) \\ newTransition = conj(ie, conj(cDisj, oe)) \end{gathered} \quad (6)$$

After the minimization of the initial FSM is done, resulting the minimized FSM can be converted into the regular expression using the transitive closure method [23]:

```
let m = Finite-State Machine state count

for i = 0 to m - 1:
    for j = 0 to m - 1:
        if i == j:
            R[i, j, 0] = ε
        else:
            R[i, j, 0] = ∅

        for a in transitions:
            if a is transition from i to j:
                R[i, j, 0] = disj(R[i, j, 0],
                    a)

for k = 1 to m - 1:
    for i = 0 to m - 1:
        for j = 0 to m:
            let s = star(R[k][k][k - 1])
            let c1 = conj(s, R[k, j, k - 1])
            let c2 = conj(R[i, k, k - 1], c1)
            R[i, j, k] = disj(R[i, j, k - 1],
                c2)
let regex = ∅
for i = 0 to m - 1:
    if i is final state:
        regex = disj(regex, R[0, i, m - 1])
```

After the FSM has been transformed into the regular expression, all the empty transitions (ε) are being removed from it. Resulting regular expression in turn can be transformed into the UML sequence diagram by performing its tokenization and tokens one by one using the following algorithm:

```
let ci = null
let sequenceModel = new Sequence Model
let lastConcept = null

function checkCurrentInteraction(token):
    if token == null:
        return

    if token is not data flow:
        return

    if ci != null:
        let df = token.dataFlow
        ci += Receiver(df.concept)
        sequenceModel += ci
        ci = null

while there are tokens to process:
    let token = next token to process
    let nextToken = token after token

    match token:
        case '(':
            checkCurrentInteraction(
             nextToken)
            ci = null
            open new interaction fragment

        case ')':
            checkCurrentInteraction(
             lastConcept)
            ci = null
            close last opened
             interaction fragment

        case '*':
            mark last closed interaction
            fragment as 'repeat'

        case 'Θ':
            mark last closed interaction
            fragment as 'alternate'
            and attach next opened
            interaction fragment to it
            as a part of alternate execution

        case DataFlow df:
            checkCurrentInteraction(
             df.concept)
            ci = Sender(df.concept)
            lastConcept = df.concept

        case ExternalProcess with only
           outgoing data flows p:
            ci = Sender(p.performer) +
                 Message(p)

        case ExternalProcess with only
           incoming data flows p:
```

 ISBN: 978-1-61208-498-5

```
            ci += Message(p) +
                 Receiver(p.performer)
            sequenceModel += ci
            ci = null

        case InternalProcess p:
            ci += Message(p)
```

As a result, the UML sequence diagram, which corresponds to the appropriate process model, is generated. The resulting diagram contains actors, objects and messages as well as parallel and loop interaction fragments.

VI. An Example for a Transformation

In order to demonstrate the proposed transformation execution, authors have created a simple two-hemisphere model for an abstract use case describing document processing (shown in Figure 1). The user inputs a list of documents to be processed, the system in turn splits the document into the accepted and the rejected thus forming a processed document list that later is being outputted on the screen. The model itself consists of a three concepts: Document, Document List, and Processed Document List as well as two external processes that are responsible for the document input and the processing result output, four internal processes: Check Document, Add to Accepted List, Add to Rejected List and Save Document. Former processes are interconnected with data flows carrying the necessary information.

In order to be able to present both the FSM and the regular expression for the given two-hemisphere model, its elements are being marked with numbers that are presented in Table 1 and will be used in later descriptions.

TABLE I. Example Model Element Identifiers

Identifier	Element
1	Input Documents (External Process)
2	Documents To Process (Data Flow)
3	Check Document (Internal Process)
4	Rejected Document (Data Flow)
5	Add to Rejected List (Internal Process)
6	Processing Result (Data Flow)
7	Save Document (Internal Process)
8	Accepted Document (Data Flow)
9	Add to Accepted List (Internal Process)
10	Processing Result (Data Flow)
11	Documents Left To Process (Data Flow)
12	Processing Result (Data Flow)
13	Print Document Processing Report (External Process)

The first constructed FSM for the given model is not shown here due to its size; however, Figure 5 presents its minimized form before applying the transitive closure algorithm.

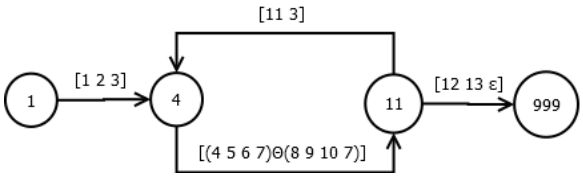

Figure 5. A minimized finite-state machine

The resulting regular expression that is obtained from this finite state machine is:

```
1 2 3 (4 5 6 7)Θ(8 9 10 7)(11 3 (4 5 6 7)Θ(8 9 10 7 ))* 12 13
```

This regular expression in turn can be transformed into the UML sequence diagram presented in Figure 6.

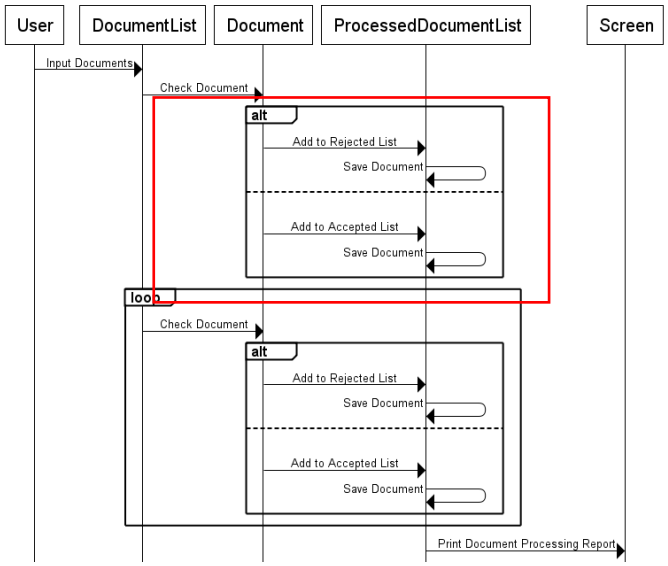

Figure 6. Resulting UML sequence diagram

VII. Analysis of Proposal

The proposed transformation method allows the generation of the UML sequence diagram from the two-hemisphere model by using finite-state machines that in turn are being converted into the regular expressions. However, authors would like to note some limitations of this approach that will become the main targets for the further improvements:

1. While resulting regular expression is correctly representing the source process model, it could be redundant. In the example shown in the previous section, it is not necessary to have the part marked with the red rectangle – since it will be repeated within the following loop frame. The post processing of the regular expression after applying the transitive closure is one of the further research directions that authors are planning to work on.

2. Currently, only the alternate interaction fragments are being generated. In the example above, it is fine

 ISBN: 978-1-61208-498-5

since the document processing in the source model has the alternative meaning – the document can only be added to the accepted or to the rejected list. However, there could be models where the several outgoing data flows are actually being sent in the parallel. It is necessary to improve the two-hemisphere model's process diagram notation in order to be able to distinguish between those cases.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, a method of generating the UML sequence diagram from the two-hemisphere model is described. Authors have studied the related work and have proposed the transformation algorithm that uses the finite-state machines and regular expressions in order to achieve the result.

The proposed transformation is able to generate the UML sequence diagram that is correctly representing the source process model in terms of interaction. However there are still some issues that require additional work and are described in the previous section. These improvements mark the first direction of the future work to be done by the paper authors.

Another direction of future research is the development of a produced regular expression interpretation algorithm that will allow to generate the UML sequence diagram for the anaemic data model based system. As it was mentioned before, the anaemic data model approach is currently being widely used in the industry and its adoption in terms of the two-hemisphere model could help in introducing the two-hemisphere model-driven approach to the target audience, i.e., enterprise system developers.

Finally, the last direction of future research is a study of different applications of the generated regular expression, e.g., using it directly for the code generation or producing UML diagrams.

ACKNOWLEDGEMENTS

The research presented in the paper is partly supported by Grant of Latvian Council of Science No. 12.0342 "Development of Models and Methods Based on Distributed Artificial Intelligence, Knowledge Management and Advanced Web Technologies for Applied Intelligent Software".

REFERENCES

[1] D. C. Schmidt, Model-Driven Engineering, http://www.cs.wustl.edu/~schmidt/PDF/GEI.pdf [retrieved 03/2016]

[2] A. Kleppe, J. Warmer and W. Ba, MDA Explained: The Model Driven Architecture™: Practice and Promise USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[3] Unified Modelling Language superstructure v.2.2, OMG Available: http://www.omg.org/spec/UML/2.2/Superstructure [retrieved 02/2016]

[4] Ambler, S. Modeling and Documentation 2013 Mini-Survey Results, Ambysoft, 2013,http://www.ambysoft.com/surveys/modelingDocumentation 2013.html [retrieved 03/2016]

[5] D. R. Wright, "Finite State Machines" (CSC215 Class Notes), http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html [retrieved 03/2016]

[6] S. S. Skiena, The Algorithm Design Manual, Springer, 1988, ISBN 978-0387948607

[7] O. Nikiforova et al., "BrainTool for Software Modeling in UML", Scientific Journal of RTU: Applied Computer Systems, Grundspenkis J. et al. (Eds), Vol.16, pp. 33-42, 2014.

[8] O. Nikiforova and M. Kirikova, "Two-hemisphere model Driven Approach: Engineering Based Software Development", Scientific Proceedings of CAiSE 2004 (the 16th International Conference on Advanced Information Systems Engineering), pp. 219-233, 2004.

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

[10] 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: Lecture Notes in Computer Science. Vol.4980. Berlin: Springer Berlin Heidelberg, pp.118-130, 2011. ISBN 9783642223853

[11] O. Nikiforova, K. Gusarovs, O. Gorbiks and N. Pavlova, "BrainTool. A Tool for Generation of the UML Class Diagrams", Proceedings of the Seventh International Conference on Software Engineering Advances, Mannaert H. et 26 al. (Eds), IARIA ©, Lisbon, Portugal, November 18-23, pp. 60-69 (Scopus), 2012.

[12] O. Nikiforova, L. Kozacenko and D. Ahilcenoka, "UML Sequence Diagram: Transformation from the Two-Hemisphere Model and Layout", Applied Computer Systems. Vol.14, pp.31-41, 2013.

[13] D. Truscan, J. M. Fernandes and J. Lilius, "Tool Support for DFD-UML based transformation." In: Proceedings of the IEEE International Conference and workshop on the engineering of Computer-Based Systems (ECBS'04) (Brno, Czech Republic, May 24-27, 2004), pp 378-397, 2004. IEEE Press, New York.

[14] T. N. Tran, K. M. Khan and Y. C. Lan, "A framework for transforming artifacts from Data Flow Diagrams to UML". In: Proceedings of the 2004 IASTED International Conference on Software Engineering (Innsbruck, Austria, Feb. 17-19, 2004). ACTA Press, Calgary, AB,

[15] F. Meng, D. Chu and D. Zhan, "Transformation from Data Flow Diagram to UML2.0 Activity Diagram",1/10, 2010 IEEE Journal.

[16] A. A. Jilani, M. Usman, A. Nadeem, I. M. Zafar and M. Halim, "Comparitive Study on DFD to UML diagrams Transformations", journal of WCSIT, vol. 1,no. 1,10-16,2011.

[17] K. Tiwari, A. Tripathi, S. Sharma and V. Dubey, "Merging of Data Flow Diagram with Unified Modeling Language." International Journal of Scientific and Research Publications, Volume 2, Issue 8, August 2012, ISSN 2250-3153

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

[19] M. Fowler, Anaemic Domain Model, http://www.martinfowler.com/bliki/AnemicDomainModel.html [retrieved 03/2016]

[20] The Anaemic Domain Model is no Anti-Pattern, it's a SOLID design | SAPM: Course Blog, https://blog.inf.ed.ac.uk/sapm/2014/02/04/the-anaemic-domain-model-is-no-anti-pattern-its-a-solid-design/ [retrieved 03/2016]

[21] Trygve MVC, http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html [retrieved 03/2016]

[22] O. Nikiforova, L. Kozacenko, and D. Ahilcenoka, "Two-Hemisphere Model Based Approach to Modelling of Object Interaction", ICSEA 2013 : The Eighth International Conference on Software Engineering Advances, pp. 605-611, 2013.

[23] T. Koshy, Discrete Mathematics With Applications, Academic Press, 2004, ISBN 0-12-421180-1

 ISBN: 978-1-61208-498-5