The Validation Possibility of Topological Functioning Model using the Cameo Simulation Toolkit

Viktoria Ovchinnikova and Erika Nazaruka
Department of Applied Computer Science, Riga Technical University, Setas Street 1, Riga, Latvia

Keywords: Topological Functioning Model, Execution Model, Foundational UML, UML Activity Diagram.

Abstract: According to requirements provided by customers, the description of to-be functionality of software systems needs to be provided at the beginning of the software development process. Documentation and functionality of this system can be displayed as the Topological Functioning Model (TFM) in the form of a graph. The TFM must be correctly and traceably validated, according to customer's requirements and verified, according to TFM construction rules. It is necessary for avoidance of mistakes in the early stage of development. Mistakes are a risk that can bring losses of resources or financial problems. The hypothesis of this research is that the TFM can be validated during this simulation of execution of the UML activity diagram. Cameo Simulation Toolkit from NoMagic is used to supplement UML activity diagram with execution and allows to simulate this execution, providing validation and verification of the diagram. In this research an example of TFM is created from the software system description. The obtained TFM is manually transformed to the UML activity diagram. The execution of actions of UML activity diagrams was manually implemented which allows the automatic simulation of the model. It helps to follow the traceability of objects and check the correctness of relationships between actions.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Development of the software system is a complex and stepwise process. At the beginning an analyst needs to ensure the description of functionality of the software system. Generally, this description is represented as a large amount of documents, which consist of text with figures, tables and multiple links to other documents. The description and requirements of functionality can be represented as a formal Topological Functioning Model (TFM) in the form of oriented graph with vertices (functional characteristics of the system) and causal relationships between them. The TFM can be constructed, using the TFM Editor in the Integrated Domain Modeling (IDM) toolset, implemented by Armands Slihte and provided in (Slihte, 2015).

During manual validation of the TFM mistakes can be made - important functional features, relationships between them or logic of TFM can be unnoticed or used incorrectly. It can happen, because the TFM is represented as a figure of an oriented graph, without any traceable execution and simulation. Generally, this graph consists of a large amount of vertices and relationships between them. It represents the full scenario of system functionality and its relationships.

The simulation of models can help to see some incorrect places in the model and to fix these places in the early stage of development of a software system. The simulation of execution models is more traceable and understandable than manual validation. The foundational subset for the execution UML models (fUML) standard is provided for modeling each behavior as an activity in the execution model. Currently, this standard ensures only UML activity diagrams for modeling of an activity, and each activity can be represented as the UML activity diagram in the execution model (OMG, 2015b). Cameo Simulation Toolkit from NoMagic uses the fUML standard for representing the execution model.

The TFM can be manually transformed to the Unified Modeling Language (UML) activity diagram following to mappings between elements of the TFM and the UML activity diagram, provided in (Donins, 2012). After that the execution of actions need to be ensured in the UML activity diagrams. The simulation of executions of UML activity diagrams can be provided using the Cameo Simulation Toolkit (NoMagic, 2015). The UML

Ovchinnikova, V. and Nazaruka, E.
The Validation Possibility of Topological Functioning Model using the Cameo Simulation Toolkit.
In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2016)*, pages 327-336
ISBN: 978-989-758-189-2
Copyright © 2016 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

activity diagram can be validated during simulation of execution of actions. The hypothesis is that the TFM also can be validated, according to simulation of execution of UML activity diagrams.

The UML activity diagram is chosen, because its visual structure and vertices names are similar to the TFM. But TFM has more information of software system, simple structure of the oriented graph and it is the formal model as compared with the UML activity diagram. The UML activity diagram includes decision-making nodes (such as decision, merge, fork and join nodes), data flows and concurrences (OMG, 2015a) and its execution rules are based on principles of Petri nets. Dynamic of the activity diagram is similar to a state chart diagram:

- Edges with its guards in the activity diagram are similar to signals in the state chart diagram;
- Initial and final nodes are equal;
- State chart diagram is represented without Petri nets similar decision-making nodes.

The main goal of this research is to assess the possibility of TFM validation in the Cameo Simulation Toolkit for not losing important information of functionality. To accomplish this goal it is necessary to perform the following tasks:

- Obtain the TFM from the system descriptions;
- Manually obtain the activity diagram from the TFM, using mappings rules;
- Add necessary functionality to the activity diagram for its execution;
- Provide execution and simulation of the activity diagram;
- Validate the activity diagram and bind results with the TFM.

The main sources of information are scientific papers, UML and Cameo Simulation Toolkit specification and notifications.

The paper is structured as follows. Section 2 describes Topological Functioning Model, fUML, and Cameo Simulation Toolkit in brief, as well as related work. Section 3 illustrates the example and results of execution of the UML activity diagram. Section 4 provides discussions, conclusion and future work.

2 BACKGROUND

2.1 TFM in Brief

The TFM has been invented at Riga Technical University (RTU) by Janis Osis in 1969. At that time a topological model was used for mathematical definition of functionality of complex mechanical systems in a holistic way (Osis and Asnina, 2011).

The formal TFM can be represented as a Computation Independent Model (CIM) in Model-Driven Architecture (MDA) (Asnina and Osis, 2011c). It can describe the functionality and structure of the software system in the form of the oriented graph. The figure of the graph with vertices that depict functional characteristics of the system named in human understandable language, and causal relationships between them provided as oriented arrows is more perceived, precise and clear then the large text of description of the software system.

A TFM is provided as a topological space (X, Q), where X is a set of functional features and Q is a set of relationships between elements in X (Osis and Asnina, 2011b). The obtainment of TFM is performed by the followings steps (Osis and Asnina, 2011):

- Provide descriptions of functional features;
- Provide the cause-and-effect relations between them;
- Separate the TFM from the created topological space.

The functional feature represents business process, task execution or activity in the software system (Osis and Asnina, 2011a). It is a unique cortege <A, **R**, **O**, **PrCond**, **PostCond**, Pr, **Ex**>, where A denotes the object's action, R is the set of results of the object's action, O denotes the object set, PrCond and PostCond represent the pre- and post-conditions respectively, Pr is the provider, Ex is the set of executors (Osis and Asnina, 2011b).

The relationships between functional features define the cause from which the execution of the effect is depended. Therefore, the relationships between functional features are named the cause-and-effect relations. Cause-and-effect relations may be in logical relationships using logical operators AND, OR and eXclusive OR.

The TFM is characterized by the topological and functioning properties (Osis and Asnina, 2011a). The topological properties are connectedness, neighborhood, closure and continuous mapping. The functioning properties are cause-and-effect relations, cycle structure, inputs and outputs.

Rules of derivation and the obtainment process of the TFM from the software system description is provided by examples and described in detailed in (Asnina, 2006), (Osis et al., 2007), (Osis et al., 2008) and (Osis et al., 2008). Construction of the TFM with attention put on continuous mappings between problem and solution domain is provided in

(Asnina and Osis, 2010). The TFM can be obtained automatically from the business use case descriptions, which can be created in the IDM toolset (Osis and Slihte, 2010), (Slihte et al., 2011), (Slihte and Osis, 2014). It also can be manually created in the TFM Editor from the IDM toolset.

The UML use case diagram can be obtained from the TFM, according to (Osis and Asnina, 2011d), (Donins, 2012). According to (Osis and Donins, 2010), (Donins et al., 2011) topological class diagram, manually derived from the TFM, can be represented as a Platform Independent Model (PIM) in MDA.

2.2 Executable UML in Brief

Executable UML (xUML) is an earlier name of Executable and translatable UML (xtUML) (Mellor and Balcer, 2002), (J.Mellor, 2003), (xtUML, 2015). It is a methodology that is fully automated with rules for execution and it uses the UML notation. xtUML is a programming language, but UML is a set of object-oriented notations (xtUML, 2015). xtUML helps to create detailed specifications of software system requirements and execute these specifications. The model figure can only be created with UML notation, but xtUML provides creation of models that are the templates of executed systems (xtUML, 2012). The testing, independent of system implementation and design, and validation, according to system requirements, processes of software system can be done and be traceable before implementation of the software system. In such a way defects or some unused objects can be noticed and resolved. The 100% target source code can be obtained and translated from this executable model with its provided behavior (xtUML, 2012), but this target source code will not be complete implementation of the software system. xtUML modeling can be used as agile modeling.

xtUML is designed for modeling control, data and processing. Control and data can be modeled using graphical diagrams – class, component and state machine. The Object Action Language (OAL) is used for modeling the processing (xtUML, 2016). This language is similar to Java, Python, C++ and Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) languages. The difference is that OAL tries to be an abstract, simple, translatable and model-aware language as it is suggested in (xtUML, 2016). The OAL is independent of the target language and can be translated to it using a model compiler (xtUML, 2016). The main tool for modeling, execution and translation xtUML models is BridgePoint.

2.3 The Foundational Subset for Execution UML Models in Brief

The fUML standard encompasses most activity and object-oriented modeling. The fUML specification (OMG, 2015b) does not change OMG specification. The semantics for a subset of UML is refined by this standard (OMG, 2015a). It simplifies the model structure of the execution UML, some elements from the UML superstructure specification are excluded in the fUML (OMG, 2015b). The fUML standard is aimed at xtUML modeling standard and strongly influenced by it.

The execution model must fully designate its own behavior, it means that all classifier behavior and operation method need to be fully specified in it. The fUML supports only activity as a user-defined behavior. Each behavior of the model needs to be modeled as an activity, using fUML specification (OMG, 2015b). Currently, the fUML specification provides graphical UML activity diagrams for modeling each activity in the execution model (OMG, 2015b). It means that for each behavior (e.g. method of an operation of a class or an effect behavior of a transition on a state machine) needs to be provided an additional graphical activity diagram (Cabot, 2011). Drawing these additional diagrams is time-consuming and errors can be made in the process (Cabot, 2011).

It provides the library fuml-1.1.0.jar file that is freely available (java2s, 2015) and can be used during modeling of the execution model. The specification of this library and all included elements are discussed and provided in (OMG, 2015b).

The action language for fUML (Alf) is used for fUML with the similar goal as the OAL language is used for xtUML (OMG, 2013). The UML modeling elements are represented in a textual type using Alf language. The mappings between Alf and fUML syntax exist that provides the execution semantics for Alf (OMG, 2013). It is similar to C or Java programming languages. The main tool for this language is called Alf.

2.4 Cameo Simulaton Toolkit in Brief

The tool MagicDraw is a commercial tool produced by NoMagic. The Cameo Simulation Toolkit is a MagicDraw plug-in for modeling the complete execution model based on fUML standard (NoMagic, 2015). The tool MagicDraw version

18.1, Professional Java edition and Academic Seat license is used in the research.

It provides the simulation of the execution UML model. The Cameo Simulation Toolkit for executions of models uses different kind of engines, such as (NoMagic, 2014):

- Behaviors – interaction (sequence diagram), state machine (state machine diagram) and activity (activity diagram). The sequence diagram can be simulated based on UML semantics, the state machine can be simulated based on the (World Wide Web Consortium) W3C (State Chart XML) SCXML standard and the activity based on fUML standard. W3C SCXML is an event-based state machine language (W3C, 2015);
- Classes – the tool creates a simulation where the class can be executed and the runtime value of the type of this class is created. If a classifier behavior is defined, then it also executes. If the selected class is the SysML Block and contains Constraint Properties, then parametric will be executed;
- Instance Specifications – values and the runtime object need to be created and used for the execution of the model. These objects and values can be automatically changed during execution and simulation.

It is necessary to write a script in the provided scripting languages (Ruby, Groovy, Python, JavaScript and BeanShell) for representing the behavior of one action. The execution of these models can be traceable and the entire process of simulation is documented in the console and log file during simulation. There are four kinds of colors (green - visited, red - active, yellow - breakpoint and orange – last visited), which are used during simulation (if necessary, the colors can be changed). The simple user interface of the software system can be developed using provided components (buttons, text fields and so on) and supplying each component with the activity that is provided by a feature in tool MagicDraw. It will be executed with simulation of the execution model in parallel. The verification and validation of the model, as well as user guides with stepwise examples and its descriptions are provided in this tool.

The MagicDraw supports imports of UML v. 1.4, XML Metadata Interchange (XMI) (v. 1.0, 1.2 and 1.4), Eclipse Modeling Framework (EMF) UML 2.2.x, custom diagrams, provides dynamical import of Rich Text Format (RTF) (or parts of them) documents into reports, data from excel and Comma-separated values (.csv) file.

It exports EMF UML 2.2.x, custom diagrams, data into excel and .csv files. It also provides export of the current diagram, selected elements of the diagram or all diagrams as bitmap (Portable Network Graphics (.png), Joint Photographic Experts Group (.jpeg)) or vector (Scalable Vector Graphics (.svg) and others) and the export of the UML state machine diagram to the standard SCXML file.

2.5 Related Work

Authors in (Abdelhalim et al., 2012) provide simulation of their system “CubeSats”, using the execution of the behavior diagram in Cameo Simulation Toolkit. This diagram is related to parametric diagrams in ModelCenter, which is related to the analytical diagram in Matlab. It is necessary to find a way to increase the storing of energy and collect the necessary data. Authors in (Panthithosanyu et al., 2014) provide external device execution, by using opaque behavior with JavaScript for supporting communication between the simulated model and the device. The model is obtained with System Modeling Language (SysML) and executed in Cameo Simulation Toolkit. Authors in (Berardinelli et al., 2015) provide the execution of communication between nodes in Wireless Sensor Networks (WSN) for testing the consumption of energy and improve the power of nodes.

The mentioned authors use the SysML, UML class diagram, state machine diagram and activity diagram for providing the execution. They describe the functionality of the system or device (as a description) and then create diagrams in Cameo Simulation Toolkit. It is the same as transformation from text to the UML diagram. In our case the description is provided as TFM and transformation is from TFM to UML diagram for its further execution in tool.

3 RESULTS OF EXECUTION OF UML ACTIVITY DIAGRAM

3.1 TFM of the Problem Domain

A part of a sport event organization process is taken as an example. A short version of the system “Registration at the sport event” description is as follows: “The visitor can visit and leave the sport event website after doing some tasks in the sport event website.

Table 1: Functional features of the problem domain.

Id	Name	Result	Executer	Precondition
1	Visiting sport event website		Visitor	
2	Requesting sport event data		Visitor	
3	Providing a sport event data	Sport event data	Sport event website	
4	Leaving sport event website		Visitor	
5	Requesting participants list		Visitor	
6	Providing participants list	Participants list	Sport event website	
7	Requesting registration form		Visitor	
8	Providing registration form	Registration form	Sport event website	
9	Filling participants data	Participant data	Visitor	(If registration is available)
10	Checking participants data		Sport event website	
11	Determining of a price	Price	Sport event website	(All mandatory fields are filled) (Entered data are correct)
12	Providing a price		Sport event website	
13	Sending a payment for participation	Payment	Visitor	
14	Receiving payment		Sport event website	
15	Adding participants to participants list		Sport event website	(If payment is received)
16	Assigning identifiers to participants	Participant id	Sport event website	
17	Assigning groups to participants	Participant group	Sport event website	
18	Sending registration confirmation	Registration confirmation	Sport event website	
19	Receiving registration confirmation		Participant	

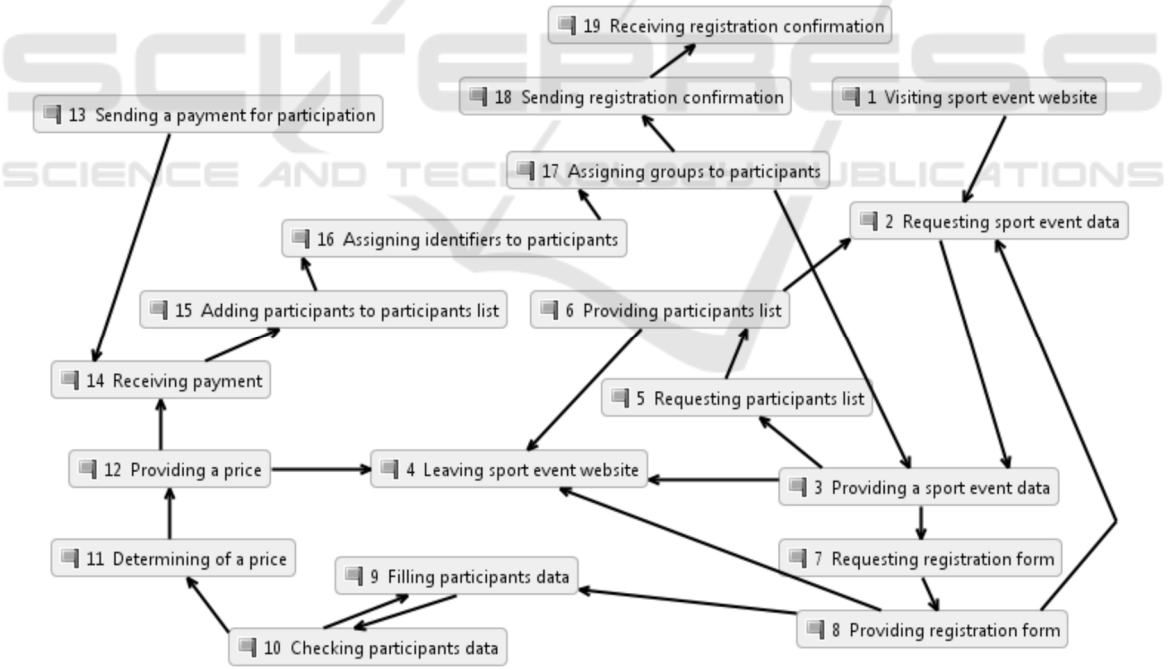

Figure 1: TFM of the problem domain.

He can request sport event data and after that the website returns the requested data (price, date, description and place) to the visitor. The visitor can request the list of participants and see all participants in the list or can request a registration form, register to the sport event and fill participant data (name, surname, gender, birthday, e-mail, mobile phone number, country, name of the team, distance). When participant's data is added it needs to be checked. If participant's data is correct and all mandatory fields

are filled, then the price of participation needs to be automatically determined and provided, according to the distance, count of participants and the date of registration. After that the visitor needs to pay for participation. When the sport event website receives the payment, the visitor becomes a participant. The participants are added to the participants list, unique identifiers and existing groups are assigned for each participant. Registration confirmation is send by the e-mail. After that the visitor receives the registration confirmation". Table 1 represents functional features information. Others unique cortege elements are empty or similar (Postcond is empty, Action is similar to functional feature name, Object is similar to result, Provider for 1 and 4 functional feature is Visitor and for others Sport event website).

Figure 1 provides the TFM with functional features and cause-and-effect relationships between them. The TFM is separated from the created topological space, where external functional features (some inputs and outputs), without direct relations (cause-and-effect) with internal functional features is considered, but not considered in the TFM.

The cycles in the TFM are the following:

- checking data (9 – 10 - 9);
- requesting sport event website information (2 – 3 – 5 – 6 – 2 and 2 – 3 – 7 – 8 - 2);
- and the main one is registration process (3 – 7 – 8 – 9 – 10 – 11 – 12 – 14 – 15 – 16 – 17 - 3).

3.2 TFM to UML Activity Diagram

Uldis Donins in his Doctoral Thesis (Donins, 2012) provided mappings between elements of TFM and elements of the UML activity diagram. The following mappings exist:

- Action in functional feature (TFM) is provided as an action (UML activity diagram);
- Cause-and-effect relationship (TFM) is provided as an edge (UML);
- Preconditions of the functional feature (TFM) is provided as guards on edges outgoing from the decision node (UML);
- Logical relationship (TFM) is provided as decision (Figure 2 a), merge (Figure 2 b), fork (Figure 2 c) or join nodes (Figure 2 d) and their combination (UML);
- Input and output functional features (TFM) are provided as initial and final nodes (UML) correspondingly.

Uldis Donins suggests that TFM can be split up in several parts in more advanced scenarios. Each part provides UML activity diagrams. In our case the TFM is represented as one UML activity diagram. It

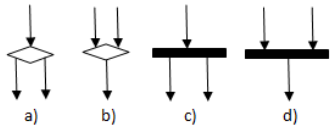

Figure 2: UML activity diagrams nodes.

is not divided into different activity diagrams. But it is necessary to define the main entry (initial node), main exit (final node) and end of flows (if exist) in the UML activity diagram.

Additionally to mappings provided by Uldis Donins, new mapping – end of flow - is added between elements of the TFM and elements of the UML activity diagram. It is necessary when a flow is divided to two or more different flows and one of them goes outside and must not to interrupt the work of other(s) flow(s). The result of functional features (instead of preconditions) is provided as guards on edges outgoing from the decision node in our case.

Figure 3 represents the activity diagram manually obtained UML from the TFM with added opaque behavior and behavior, represented by another inside activity diagram (it will be more discussed in the next subsection). The UML activity diagram is obtained following the mappings and after that the execution is added. Almost all logical relationships are provided as decision and merge nodes in the UML activity diagram in our case. Two logical relationships are provided as join and fork nodes when a new flow is appeared and one flow is divided into two flows accordingly.

3.3 Execution of UML Activity Diagram

In the first section is described why is chosen UML activity diagram. The more information of state chart diagram execution is provided in the Cameo Simulation user guide (NoMagic, 2014). States of the state chart diagram are executed within the inside activity diagram, which describes behavior of this state.

In our case the same method is taken for the UML activity diagram – some activities behavior is provided with another inside activity diagram and others with opaque behavior as it is represented in Figure 3.

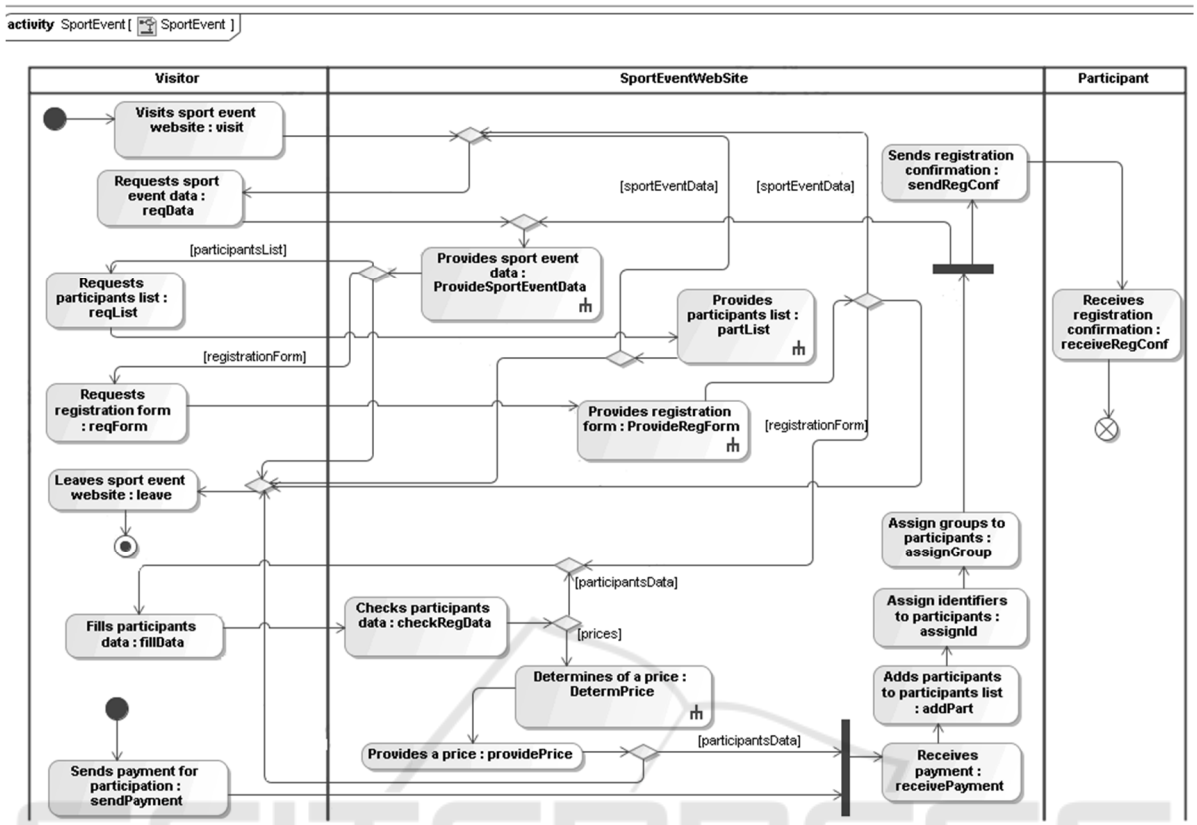

Figure 3: The obtained UML activity diagram with provided behavior.

In Figure 3 activities with inside activity diagrams are represented with the activity diagram name after ":" (partList) and special symbol (Figure 4 on the left). Activities with opaque behavior are represented with opaque behavior name after ":" (visit) (Figure 4 on the right).

Figure 4: Activity with the inside activity diagram (on the left side) and opaque behavior (on the right side).

Figure 5: Class RegistrationForm attributes.

It is necessary to write scripts for execution of all activities. For this task BeanShell language was chosen. For all opaque behaviors is written "print ("Information")" in this language. All output information is provided in the console, while UML activity diagram executes. It is also necessary to create an inside activity for getting information from objects, its instance and for doing manipulations (e.g. create, edit, delete) with them.

Figure 5 represents class RegistrationForm and its attributes. It is necessary for managing objects during execution of the UML activity diagram. Next Figure 6 illustrates instances "maya" and "jack" of class RegistrationForm. Figure 7 on the left represents class ParticipantsData that has an attribute that is the list of RegistrationForm instances. Figure 7 on the right illustrates instance "participantsData" of class ParticipantsData.

Figure 8 represents the inside activity diagram in "Provides participants list" (Figure 4). Activity "readParticipants" provides getting of necessary and existing instances. It is needed to define the type (it is ParticipantsData in our case, see Figure 7) of this instance.

Figure 9 represents activity "participantsData" (in Figure 8) input and output objects. In this activity it is possible to read the instance's attribute data.

Input (object) is object ParticipantsData and output (result) is the list of RegistrationForm.

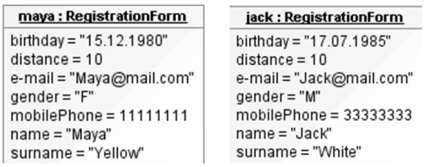

Figure 6: Class RegistrationForm instances.

Figure 7: Class ParticipantsData's attribute (on the left) and instance (on the right).

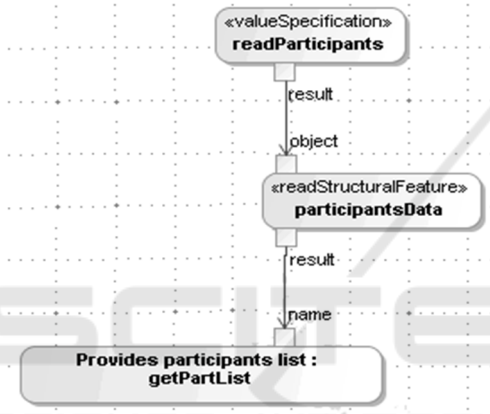

Figure 8: Inside activity diagram.

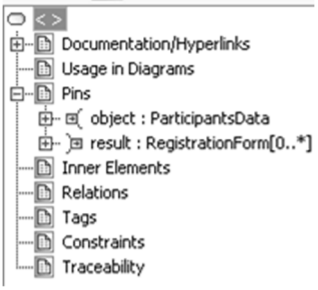

Figure 9: Activity "participantsData" input and output.

Activity "Provides participants list" in Figure 8 has opaque behavior. The object name with type RegistrationForm[0..*] is the input object to this activity (as a parameter sent to the method). The following script is written for printing results in the console, using BeanShell language:

```
for (i : name)//parameter in method
print(i);//one record from list
```

Figure 10 represents the choice option during execution of UML activity diagram, which provides traceability. It is possible to choose the necessary guard. The window with a question appears and if the guard name shown is chosen by clicking the "yes" button then the execution will continue (going to edge with chosen guard). If button "no" is clicked then the next available guard is offered.

Figure 10: Decision example during activity diagram execution.

Figure 11: Console output during activity diagram execution.

Figure 11 illustrates part of console outputs. It is the result of execution of activity "Provides participants list" that is provided in Figure 4 and instances data from Figure 6.

In summary, the execution model is obtained with objects described as classes and its instances, and with activities, where management occurs with these objects. The model is executed manually by choosing the next possible step (the guard in the

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

activity diagram) if it exists. Derivation of results is traceable during simulation of the model. All results are represented in the console as a text in our case.

4 DISCUSSIONS AND CONCLUSION

The obtained results in this research represent the manual validation and traceability during execution of the UML activity diagram by choosing the necessary guards. At first, the full TFM is transformed to the UML activity diagram. The main input, main output and flow end are additionally detected and are marked in the UML activity diagram. It is necessary to create the class diagram and its instances for providing the object management feature. Simulation of this diagram helps to test and validate the software system in its initial state of development. It allows going through all possible paths (or scenarios) from inputs to outputs, checking needed functional characteristics and input/output sets. It helps to prevent mistakes and ensures that no vital information is lost. Authors have come to the conclusion that it is possible to validate TFM using UML activity diagram execution, but only partially. This is due to the fact, that information can be lost during manual transformation from TFM to the UML activity diagram.

The Cameo Simulation Toolkit is a tool that provides execution of UML activity or state chart diagrams by defining each activity as the inside activity diagram with extended opportunity (special activities with their already defined tasks). This tool provides the management with objects during execution. The execution of diagrams can be simulated and outputs results can be shown in console. It also extends possibility from MagicDraw to visualize the execution process by creating forms and binding it with the activity.

One of disadvantages is that the transformation from TFM to UML activity diagram is manual. Some necessary information and relationships can be lost during transformation. It is planned to automate this transformation in the future. Synchronization between TFM and the generated UML activity diagram is also planned.

Another disadvantage is that it is necessary to create inside activity diagrams for execution of activities and write scripts for each activity. It is a time-consuming process. Authors were not able to write complex scripts for managed objects, because the functionality of implementation of the chosen script language was limited. Future validation of this tool requires assessment of other language implementations. The TFM currently does not store information of objects such as object characteristics (attributes). It is necessary to analyze the necessity of providing such information in the TFM in the future. Compared to TFM, the UML activity diagram increases the count of diagram elements due to using decision, join, merge and fork nodes. It also complicates the reading of the diagram.

Future work is related to analyzing the possibility of automating the simulation of all possible paths and of documenting the results (e.g. listing all errors) of these paths and object management (and the input and output object counters) in it. The direct simulation of TFM execution (omitting the transformation to UML) is planned in the future for its validation according to software system functionality requirements.

REFERENCES

Abdelhalim, I., Schneider, S. and Treharne, H., 2012. An Optimization Approach for Effective Formalized fUML Model Checking. In 10th International Conference on Software Engineering and Formal Methods (SEFM 2012). Thessaloniki, 2012. pp. 248-262.

Asnina, E., 2006. The Computation Independent Viewpoint: a Formal Method of Topological Functioning Model Constructing. Applied computer systems, 26, pp.21-32.

Asnina, E. and Osis, J., 2010. Computation Independent Models: Bridging Problem and Solution Domains. In Proceedings of the 2nd InternationalWorkshop on Model-Driven Architecture and Modeling Theory-Driven Development (MDA & MTDD 2010), in conjunction with ENASE 2010. Lisbon: SciTePress. pp.23-32.

Asnina, E. and Osis, J., 2011c. Topological Functioning Model as a CIM-Business Model. In Model-Driven Domain Analysis and Software Development: Architectures and Functions. Hershey - New York: IGI Global. pp.40 - 64.

Berardinelli, L. et al., 2015. Energy Consumption Analysis and Design of Energy-Aware WSN Agents in fUML. In 11th European Conference on Modelling Foundations and Applications 2015. L'Aquila, 2015. pp. 1-17.

Cabot, J., 2011. The New Executable UML Standards: fUML and Alf. [Online] Available at: http://modeling-languages.com/new-executable-uml-standards-fuml-and-alf/ [Accessed 17 January 2016].

Donins, U., 2012. Topological Unified Modeling Language: Development and Application. PhD Thesis. Riga: RTU.

Donins, U. et al., 2011. Towards the Refinement of Topological Class Diagram as a Platform Independent Model. In Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development (MDA & MDSD 2011). Lisbon: SciTePress. pp.79-88.

J.Mellor, S., 2003. Executable and Translatable UML. [Online] Available at: http://www.embedded.com/design/prototyping-and-development/4024510/Executable-and-Translatable-UML [Accessed 17 January 2016].

java2s, 2015. Download fuml-1.1.0.jar. [Online] Available at: http://www.java2s.com/Code/Jar/f/Downloadfuml110jar.htm [Accessed 30 November 2015].

Mellor, S. and Balcer, M., 2002. Executable UML: A Foundation for Model-Driven Architectures. Boston: Addison-Wesley Longman Publishing Co.

NoMagic, 2014. Cameo Simulation Toolkit 18.1 user guide. [Online] Available at: http://www.nomagic.com/files/manuals/Cameo%20Simulation%20Toolkit%20UserGuide.pdf [Accessed 30 November 2015].

NoMagic, 2015. Cameo Simulation Toolkit. [Online] Available at: http://www.nomagic.com/products/magicdraw-addons/cameo-simulation-toolkit.html [Accessed 30 November 2015].

OMG, 2013. Action Language for Foundational UML (Alf). [Online] Available at: http://www.omg.org/spec/ALF/1.0.1/ [Accessed 17 January 2016].

OMG, 2015a. OMG Unified Modeling Language. Version 2.4.1. [Online] Available at: http://www.omg.org/spec/UML/2.4.1/ [Accessed 30 November 2015].

OMG, 2015b. Documents Associated With Semantics Of A Foundational Subset For Executable UML Models. [Online] Available at: http://www.omg.org/spec/FUML/1.1/ [Accessed 30 November 2015].

Osis, J. and Asnina, E., 2011a. Is Modeling a Treatment for the Weakness of Software Engineering? In Model-Driven Domain Analysis and Software Development: Architectures and Functions. Hershey - New York: IGI Global. pp.1-14.

Osis, J. and Asnina, E., 2011b. Topological Modeling for Model-Driven Domain Analysis and Software Development: Functions and Architectures. In Model-Driven Domain Analysis and Software Development: Architectures and Functions. Hershey - New York: IGI Global. pp.15-39.

Osis, J. and Asnina, E., 2011d. Derivation of Use Cases from the Topological Computation Independent Business Model. In Model-Driven Domain Analysis and Software Development: Architectures and Functions. Hershey - New York: IGI Global. pp.65 -89.

Osis, J. and Asnina, E., 2011. Model-Driven Domain Analysis and Software Development: Architectures and Functions. Hershey - New York: IGI Global.

Osis, J., Asnina, E. and Grave, A., 2007. MDA Oriented Computation Independent Modeling of the Problem Domain. In Proceedings of the 2nd International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2007). Barselona: INSTICC Press. pp.66-71.

Osis, J., Asnina, E. and Grave, A., 2008. Computation Independent Representation of the Problem Domain in MDA. e-Informatica Software Engineering Journal, 2(1), pp.29-46.

Osis, J., Asnina, E. and Grave, A., 2008. Formal Problem Domain Modeling within MDA. In Software and Data Technologies, Communications in Computer and Information Science. Berlin: Springer-Verlag Berlin Heidelberg. pp.387-98.

Osis, J. and Donins, U., 2010. Formalization of the UML Class Diagrams. In Evaluation of Novel Approaches to Software Engineering. Berlin, 2010. Springer-Verlag. pp. 180-192.

Osis, J. and Slihte, A., 2010. Transforming Textual Use Cases to a Computation Independent Model. In Osis, J. and Nikiforova, O., eds. Model-Driven Architecture and Modeling Theory-Driven Development : Proceedings of the 2nd International Workshop (MDA & MTDD 2010). Lisbon, 2010. SciTePress. pp. 33-42.

Panthithosanyu, K. et al., 2014. Technology. [Online] Available at: http://www.technology.org/2014/05/05/collaboration-simulated-model-external-system-controlling-lego-mindstorms-cameo-simulation-toolkit/ [Accessed 23 January 2016].

Slihte, A., 2015. The integrated domain modeling: an approach & toolset for acquiring a topological functioning model. PhD Thesis. Riga: RTU.

Slihte, A. and Osis, J., 2014. The Integrated Domain Modeling: A Case Study. In Databases and Information Systems: Proceedings of the 11th International Baltic Conference (DB&IS 2014), Estonia, Tallinn, 8-11 June, 2014. Tallinn: Tallinn University of Technology Press. pp. 465-470.

Slihte, A., Osis, J. and Donins, U., 2011. Knowledge Integration for Domain Modeling. In Osis, J. and Nikiforova, O., eds. Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development (MDA & MDSD 2011). Lisbon, 2011. SciTePress. pp. 46-56.

Spangelo, S., Kim, H. and Soremekun, G., 2013. Simulation of CubeSat Mission. [Online] Available at: http://www.omgsysml.org/Modeling-and-Simulation_of_CubeSat_Mission_v15-May_2013-Spangelo-Kim_Soremekun.pdf [Accessed January 23 2016].

W3C, 2015. State Chart XML: State Machine Notation for Control Abstraction. [Online] Available at: http://www.w3.org/TR/scxml/ [Accessed 30 November 2015].

xtUML, 2012. Executable and Translatable UML Summary. [Online] Available at: https://xtuml.org/wp-content/uploads/2012/09/xtUML_Summary.pdf [Accessed 17 January 2016].

xtUML, 2015. What is xUML? [Online] Available at: http://xtuml.nrt.se/index.php?section=17 [Accessed 17 January 2016].

xtUML, 2016. Action Language (OAL) Tutorial. [Online] Available at: https://xtuml.org/learn/action-language-tutorial/ [Accessed 17 January 2016].