Using Functional Characteristics to Analyze State Changes of Objects

Uldis DONINS, Janis OSIS[1], Erika ASNINA and Asnate JANSONE
Department of Applied Computer Science, Riga Technical University, Latvia

Abstract. Event-driven software systems continuously wait for occurrence of some external or internal events. When such event is received and recognized, the system reacts by performing corresponding computations which may include generation of events that trigger computation in other components. After the event handling operation is complete the system returns to the waiting state for the next event occurrence. The response to the received event depends on the current state of the system and underlying objects and can include a change of state leading to a state transition. The state changes and transitions within a system can be formally analyzed by using functional characteristics of Topological Functioning Model (TFM). TFM captures system functioning specification in the form of topological space consisting of functional features and cause-and-effect (i.e. topological) relations among them and is represented in a form of directed graph. The functional features together with topological relationships contain the necessary information to create State diagram which reflects the state changes within system.

Keywords. Topological functioning modeling, functional characteristics, objects, states

Introduction

The behavior of an object over time could be surmised by analyzing system use-case descriptions, activity diagrams, or other software design artifact. To avoid surmising the state change of objects in system, a State diagram is used [1, 2]. The state diagram is part of the Unified Modeling Language (UML) [3]. The application of design models provide better understanding of proposed solution and allows making better decisions concerning the implementation details. Additionally, the model driven development has been put forward to enable development, validation and transformation of syntactically and semantically complete models, thus allowing source code generation automation. In such way models are promoted as the core and main artifact of software design and development.

Despite the presence of UML and a number of software development methods, the way the software is built still remains surprisingly primitive (by meaning that major software applications are cancelled, overrun their budgets and schedules, and often have hazardously bad quality levels when released) [4]. This is due that the very beginning of software development lifecycle is too fuzzy and lacking a good structure since the software developers has limited analysis and modeling of systems [5]. Instead

[1] Corresponding Author: Janis Osis, Department of Applied Computer Science, Riga Technical University, Meza iela 1/3, Riga, LV 1048, Latvia; e-mail: Janis.Osis@cs.rtu.lv

of analyzing the system software developers set the main focus on analysis and modeling of software thus leading to a gap between the system and its supporting software [6]. This issue can be overcome by formalizing the very beginning of the software development lifecycle. By adding more efforts at the very beginning of lifecycle it is possible to build better quality software systems [7, 8].

By having too fuzzy beginning of the software development and lacking a good structure of it, the elimination of gap between computation independent viewpoint and the platform independent viewpoint depends much on designers' personal experience and knowledge (both viewpoints mentioned in the context of Model Driven Architecture – MDA [9]). Thus the quality of software system design models cannot be well controlled [10, 11]. There are a number of researches which try to enforce the initial phase in software development by strengthening it with various models like use cases [12], goal based models [13], behavioral models (like Activity and Sequence diagrams) [14], and structural models [15]. Previous researches in the field of formalizing very beginning of software development lifecycle propose TopUML modeling that enables modeling the functioning of both the problem and solution domains [16, 17]. Additionally it supports early solution domain model validation against functioning of the problem domain. TopUML modeling is a model-driven approach which combines Topological Functioning Model (TFM) [18] and its formalism with elements and diagrams of TopUML [19] (a profile based on UML). The TFM holistically represents a complete functionality of the system from the computation independent viewpoint [20]. It considers problem domain information separate from the solution domain information.

The purpose of this research is to strengthen the TopUML modeling with formal development of State diagram thus enabling transformation from TFM to it and eliminating the gap between problem domain model and software design (solution) model. Thus the paper is organized into following sections. Section 1 discusses the UML modeling driven methods that supports analysis of object state transitions and composition of corresponding State diagrams. Section 2 explores TopUML modeling and the prerequisites that should be satisfied in order to formally develop State diagrams in strong relevance with the problem domain. This section gives the formal method of developing State diagram based on TFM, i.e., the TFM to State diagram transformation pattern. Section 3 shows an example of using functional characteristics to analyze state changes of objects based on enterprise data synchronization system. Paper is concluded with conclusions of the performed research.

1. Related Works

UML is a notation and as such its specification does not contain any guidelines of software development process (e.g., which diagrams to use in which order). In fact this is pointed out as one of the UML weaknesses [21]. Despite that UML is independent of particular methods and approaches, most of the UML modeling driven methods use Use Case driven approach [22]. This might be caused by the originators (Booch, Rumbaugh, and Jacobson) of the UML since they recommend a Use Case driven process in their book "The Unified Modeling Language User Guide" [23].

According to [24] a successful software development project can be measured against the deliverables that satisfy and possibly exceed expectations of customer, the

delivery schedule that has occurred in a timely and economical fashion, and the created result is resilient to change and adaptation. For software development project to be successful by means of given measurements, it should satisfy the following two characteristics:

- Solution should have a strong architectural vision, and
- A well-managed development lifecycle should be used.

This section discusses the current state of the art of UML based software development approaches by paying attention on one aspect – support of analysis for object state changes and transitions:

- State diagrams within Unified software development process [2] are developed during elaboration and construction phases. The use of state diagrams is emphasized for showing system events in use cases, but they may additionally be applied to any class.
- Business Object-Oriented Modeling (B.O.O.M.) developed by Podeswa [1] states that at least for every key business object a state diagram should be created.
- According to GRASP patterns (General Responsibility Assignment Software Pattern) introduced by Larman in [25] the State diagrams are used to describe allowed sequence of external system events that are recognized and handled by a system in the context of a use case. Additionally state diagrams can be applied to any class.
- Conceptual modeling described in [26] states that each entity type may be associated with zero, one, or more State diagrams. Conceptual modeling can be viewed as an activity related to capturing the knowledge about the desired system functionality.
- State diagrams within Component based development are used to determine the threads of control within the system [27].

The reviewed methods share common viewpoint of the application of State diagrams within software development process:

- State diagrams are developed by analyzing Use cases (more precisely: the scenario described by it),
- One state diagram per class or object, and
- State diagram should be developed for each most important object within the system.

Above mentioned three statements regarding application of State diagrams raise a set of ambiguousness and questions. The Use cases cannot be considered as a complete problem domain representation and a formal connection between problem domain and the solution [28]. The application of Use cases to develop diagrams of other types (such as State diagram) depends much on the designers' personal experience and knowledge, thus leaving the following question open:

- How to formally eliminate and overcome the gap between problem domain model and the design models?, and
- What are "most important objects" and how to formally identify them?

To overcome these issues the TopUML modeling is applied within software development as described in the next section.

2. Object State Change and Transition Analysis by using Functional Characteristics of Problem Domain

The object state change and transition analysis by using functional characteristics of problem domain is based on TopUML modeling, which is a model-driven approach intended to model problem domain and design software systems [19]. It combines TFM and its formalism with elements and diagrams of TopUML. The TFM considers problem domain information separate from the solution domain information and holistically represents a complete functionality of the system from the computation independent viewpoint while TopUML has elements of representing system design at the platform independent viewpoint and platform specific viewpoint. TFM has strong mathematical basis and is represented in a form of a topological space (X, Θ), where X is a finite set of functional features of the system under consideration, and Θ is the topology that satisfies axioms of topological structures and is represented in a form of a directed graph [18].

The application of TopUML modeling ensures proper analysis of system functioning by identifying and analyzing the functioning cycles. By using TopUML the information of system functioning from TFM can be transferred to design models thus allowing marking and evaluating the most important objects and components within system and to assign proper responsibilities to the right objects in a formal way. The most important objects are the ones that are participating in the main functioning cycle of the system. The main functional cycle is a directed closed loop that shows the functionality of system which is essential to its existence. By interrupting the main functional cycle the system cannot function or even exist. [19, 29]

In the context of state change analysis of objects the following TopUML modeling activities should be performed:

- TFM development (see Section 2.1 below),
- Domain model analysis and design (see Section 2.2), and
- Object state change and transition analysis (see Section 2.3).

2.1. Topological Functioning Model Development

The development of TFM consists of four steps. By completing these steps a TFM representing complete functioning of the problem domain gets developed. Afterwards the TFM is used as a source for development of other diagrams thus overcoming the gap between problem and solution domains. The four steps of TFM development are as follows:

Step 1: Definition of physical or business functional characteristics which consists of the following activities [30]:

1. Definition of objects and their properties from the problem domain description;
2. Identification of external systems and partially-dependent systems; and
3. Definition of functional features using verb analysis in the problem domain description, i.e., by finding meaningful verbs.

As a result a set of functional features are defined. Each functional feature X_{id} is a unique tuple specifying an action in problem and solution domains [18]. Equation (1)

defines tuple of functional feature together with elements required to transform TFM into State diagram:

$$X_{id} = <\text{Id, A, Op, R, O, Cl, St, PrCond, PostCond, E, Es, S}>, \text{ where} \quad (1)$$

- *Id* – identifier of functional feature,
- *A* – action of object *O*,
- *Op* – operation which will provide functionality defined by action *A* (can be acquired when the class diagram is synthesized),
- *R* – result of action *A* (optional),
- *O* – object that receives the result or that is used in action *A* (for example, a role, a time period, a catalogue, etc.),
- *Cl* – class which will represent object *O* in static viewpoint of system (can be acquired when the class diagram is synthesized),
- *St* – new state of object *O* after performing action *A* (optional),
- *PrCond* – is a set of preconditions (optional),
- *PostCond* – is a set of postconditions (optional),
- *E* – entity responsible for performing action *A*,
- *Es* – indicates if execution of action *A* could be automated (i.e., performed without human interaction), and
- *S* – subordination of functional feature (can be internal or external).

At the lowest abstraction level one functional feature describes only one atomic action. Atomic action means that it cannot be further divided into a set of business actions. The functional features are represented as vertices in a directed graph of TFM.

Step 2: Introduction of topology Θ (in other words – creation of topological space) which involves establishing cause-and-effect relations between identified functional features. Cause-and-effect relations are represented as arcs of a directed graph that are oriented from a cause vertex to an effect vertex. Topological space represents the system under consideration together with the environment in which this system exists.

Step 3: Separation of TFM from topological space which is done by applying the closure operation over a set of system's inner functional features [31]. Initial TFM can be called "*TFM as-is*" where "*as-is*" means that the TFM represents the functioning of the problem domain without the impact of planned software system. Construction of initial TFM can be iterative. Iterations are needed if the information collected for TFM development is incomplete or inconsistent or there have been introduced changes in system functioning or in software requirements. The TFM development steps 1 to 3 can be partly automated as shown in [32] where the business use cases are automatically transformed into TFM.

Step 4: Identification of logical relations between cause-and-effect relationships consists of two steps since there are two kinds of logical relationships – one kind is between arcs that are outgoing from functional features and the other kind is between arcs that are incoming to functional features. Thus the identification of logical relations consists of two actions:

1. Identification of logical relations between cause-and-effect relationships that are outgoing from functional feature, and
2. Identification of logical relations between cause-and-effect relationships that are incoming to functional feature.

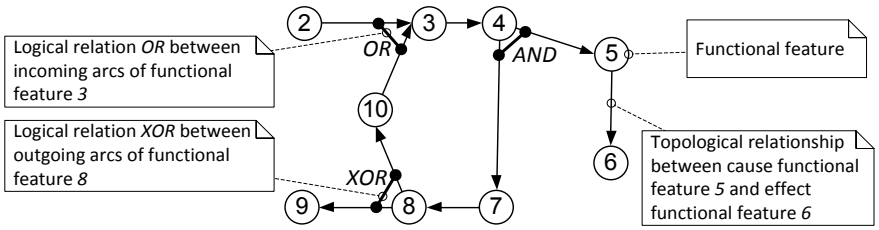

Figure 1. Example of TFM

Each logical relation consists of two or more cause-and-effect relationships and a relation type. Within TFM can be defined three types of logical relations:

- Conjunction (*and*),
- Disjunction (*or*), and
- Exclusive disjunction (*xor*).

An example of TFM consisting of nine functional features, nine topological (i.e., cause-and-effect) relationships and three logical relations is given below in Figure 1.

2.1.1. Mappings between TFM and State Diagram

This section discusses the mappings between elements of TFM and State diagram. The mappings between standard UML diagrams can be found in various books and researches, like [1, 23, 27, 33] and [34]. Mappings between elements of TFM and State diagram are described in the form of table (see Table 1) by giving element of TFM and corresponding element in State diagram. For better understanding in addition a description of each mapping is given.

Table 1. Mappings between elements of TFM and elements of State diagram

TFM element	State diagram element	Description
Object state (*St* (1) specified by functional feature)	State	Each functional feature specifies an object performing certain operation. If during execution of this action changes the state of object performing this action, functional feature specifies the new state of the object. Object state from functional feature is transformed into state in State diagram.
	Initial state	When information from input functional feature is transformed into a state, an initial state is added before this state.
	Final state	When information from output functional feature is transformed into a state, a final state is added after this state.
Topological relationship	Transition	If during execution of action specified by functional feature is changed the state of object performing this action then incoming topological relationship defines transition from previous state to the new state.
Operation (*Op* (1) specified by functional feature)	Event	Each functional feature specifies an atomic business action which later is specified by topological operation in TFM. If functional feature specifies the new state of object, the operation is transformed into the event triggering transition from one state to another.
	Entry effect	If current functional feature specifies the new state of object, the operation is transformed into the entry effect of this new state.

TFM element	State diagram element	Description
	Exit effect	If descendant functional feature specifies the new state of object, the operation of this descendant functional feature is transformed into the exit effect of current state.
Preconditions of functional features (*PrCond* and *PostCond* (1))	Guard condition	If current functional feature specifies the new state of object, the preconditions of this functional feature are transformed into the guard conditions.
Logical relationship with type "*and*" (and partially "*or*")	Fork and Join	A logical relation in TFM give additional information about execution concurrency of functional features, thus conjunction (and) within State diagram is represented with fork and corresponding join. Disjunction (or) indicates of possible fork and join.

2.2. Domain Model Analysis and Design

Domain model analysis and design within TopUML modeling is based on the Topological class diagram and consists of the following two steps:

Step 1: Analysis of objects and their communication is based on the TFM transformation into Communication diagram (in previous researches the Problem domain objects graph was use instead of Communication diagram [19]). This transformation can be done automatically since TFM has all the information that is necessary for Communication diagram. When transforming TFM into Communication diagram the following are used:

- Functional features – source for lifeline identification and message sending from object to object,
- Topological relationships – determines the message sender and receiver as well as the message sending sequence, and
- Logical relations – shows the message sending concurrency.

In order to obtain a Communication diagram, it is necessary to check if each functional feature of the TFM reflects only one type of object. If some of functional feature reflects more than one type of object then it is needed to decompose it to the level where one functional feature uses only one type of objects. If TFM has been successfully checked it can be transformed into Communication diagram. The first step in transformation is to merge functional features with objects of the same type in one lifeline (the lifeline represents the class attribute of the functional feature). While merging functional features into lifelines the relationships with other lifelines should be retained (if there is more than one topological relationship then only one link is added between lifelines). Actors to Communication diagram are added from the input functional features.

For a better understanding of TFM to Communication diagram transformation, a small fragment of TFM consisting of two functional features A and B is used (see Figure 2), where A is an input functional feature of TFM and dashed arrows show mappings between elements of TFM and Communication diagram.

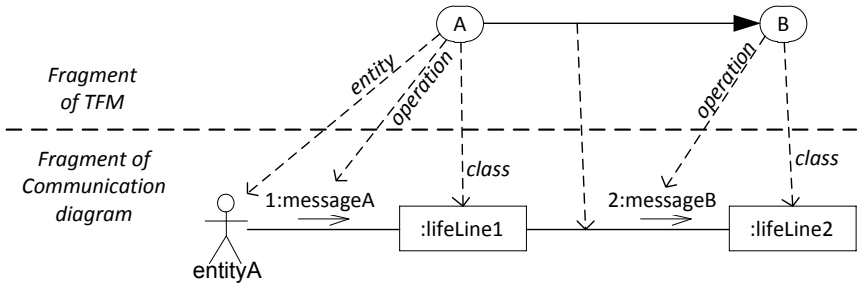

Figure 2. Example of TFM to Communication diagram transformation

Step 2: Domain model development by means of Topological class diagram consists of four activities:

1. Adding classes and operations,
2. Adding topological relationships between classes,
3. Identifying attributes, and
4. Refining initial Topological class diagram.

At first the Communication diagram is used for adding classes and operations to the Topological class diagram – lifelines are transformed into classes and messages into operations. The next step is adding topological relationships between classes. Since the notation of Topological class diagram allows variations of topological relationship graphical representation, it is advised to draw only one directed arrow in the same direction between classes (the arrow will show the cause and the effect operations).

After the classes and topological relationships between them have been established the next step is identification of attributes. This can be achieved by taking into consideration attributes of the object represented by functional feature. If the functional feature is well specified the class attribute of it is determined. If the class attribute is not determined, it can be specified in several ways (e.g. by analyzing functioning description of the system and searching nouns that represents attributes of the object [35], performing expert interviews [1], or by using ontology [36]).

By transforming Communication diagram an initial Topological class diagram is obtained (with attributes, operations, and topological relations between classes). A topological relation shows the control flow within the system. If static relations should be included (such as associations, generalization, etc.) then initial topological class diagram should be refined [37].

2.3. Object State Change and Transition Analysis

Object state change and transition analysis is based on the TFM transformation into a set of State diagrams (see Figure 3). The input of this activity is refined TFM and classes (either from Topological class diagram or lifelines from Communication diagram) and the output of this activity is one State diagram for each class.

Each functional feature specifies an object performing certain action. The count of obtained State diagrams is denoted by count of distinct objects specified by functional features. It is advised to analyze state changes of complex or most important objects in the system [1]. The most important objects are denoted by TFM – the functional features that are included into main functional cycle denote them, thus the identification of most important objects are done in a formal way.

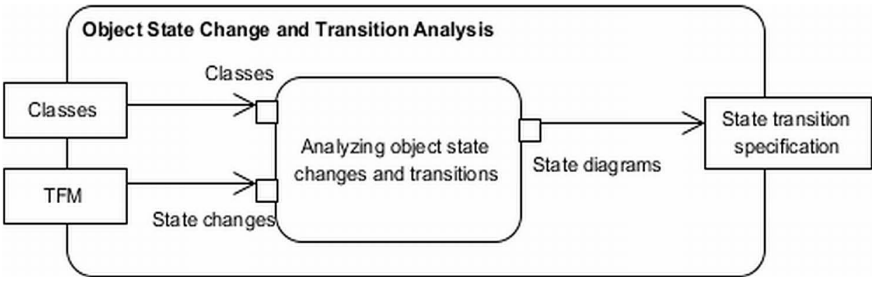

Figure 3. Analyzing object state changes and transitions

The first action is to scale down TFM which is performed by removing functional features which does not represent the object under consideration but in the same time retaining cause-and-effect relations. For example, assume that TFM consists of three functional features A, B, and C and are in the following causal chain: A→B→C. The A and C represent the same object while B represents another object. The resulting (scaled down) TFM is as follows: A→C.

States for each class are obtained from the functional features of refined TFM (functional feature has an attribute that defines the new state of the object). If the execution of functional feature involves the change of the corresponding object's state, then the state attribute has value, otherwise the value is not set. State transitions are obtained by transforming cause-and-effect relationship between functional features.

The special states (initial state and final state) are added to the obtained State diagram as follows:

- The initial state is added before the states that are obtained from the functional features which are the inputs of the downscaled TFM (the ones which has no predecessors), and
- The final state is added after the states that are obtained from the functional features which are the outputs of the downscaled TFM (the ones which has no descendants).

The example of transforming generic example of TFM into State diagram is given in Figure 4.

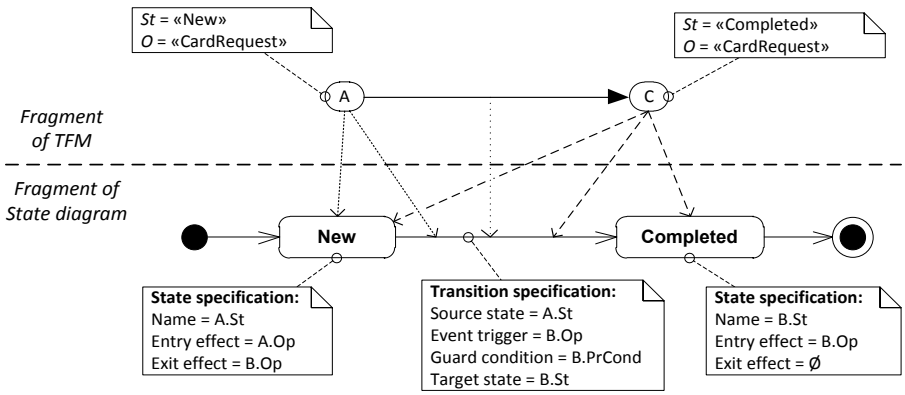

Figure 4. Example of TFM to State diagram transformation

3. Example of Object State Change and Transition Analysis

Example of object state change and transition analysis by using functional characteristics of problem domain is based on a case study in which TFM is developed for enterprise data synchronization system. The enterprise data synchronization system is developed by applying TopUML modeling and involves creation of TFM, Use case diagram, Problem domain objects graph (applied instead of Communication diagram), Topological class diagrams, and Sequence diagrams [16].

Within the case study have been defined 30 functional features by analyzing functioning of enterprise data synchronization system. Part of defined functional features is given in Table 2 where are included features that specify the new state for object named *"Scheduler"*. After definition of functional features the topology Θ (cause-and-effect relationships) is identified between those functional features thus creating topological space. In order to get the TFM the closuring operation is applied over the set of internal system functional features. The developed TFM after applying closuring operation is as follows: X={2, 3, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 19, 20, 22, 24, 25, 26, 27, 28, 29}. The resulting graph is given in Figure 5 (a) which shows functional features (vertices), cause-and-effect relationships (arcs between vertices).

Table 2. Part of functional features defined for enterprise data synchronization system

ID	Object action (*A*)	Precondition (*PrCond*)	Object (*O*)	New state (*St*)
5	Reading all data from source data base	If import should be performed from source data base	Scheduler	Reading data
6	Checking if read data structure is according to specification		Scheduler	Checking data
7	Putting the read data into temporal internal table	If data structure is according to specification	Scheduler	Importing
9	Checking import folder		Scheduler	Reading data
12	Checking if import file data structure is according to specification		Scheduler	Checking data
13	Converting the read data from import file into temporal internal table	If import file structure is according to specification	Scheduler	Importing
15	Moving import file to processed files folder		Scheduler	Completing import
19	Checking if data from a particular row already exists in target data base		Scheduler	Importing
25	Logging data row from temporal internal table		Scheduler	Logging status
29	Archiving log file	If data import is completed	Scheduler	Completing import

The example of object state change analysis in the context of enterprise data synchronization system development case study is performed for the object name *"Scheduler"*. The functional features specification in Table 2 shows that this object in total has five different states: 1) *Reading data*, 2) *Checking data*, 3) *Importing*, 4) *Logging status*, and 5) *Completing import*. The resulting State diagram is given in Figure 5 (b).

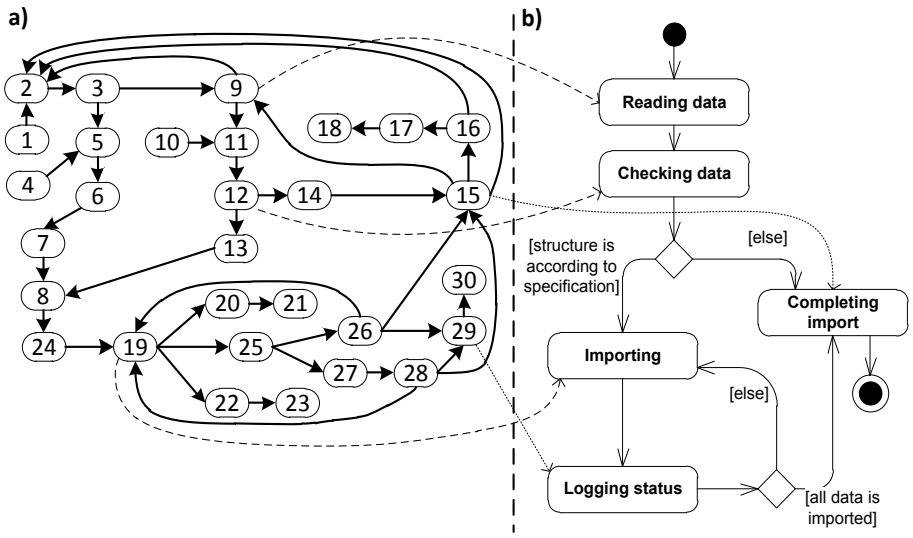

Figure 5. TFM of enterprise data synchronization system functioning (a) and State diagram for object *"Scheduler"* (b)

4. Conclusions

The main goal of this research is to do formal development of State diagram by analyzing functional characteristics of a problem domain. The result of research is method for transforming TFM into State diagram thus eliminating the gap between problem domain model and software design (solution) model.

UML modeling driven methods (like Unified process, B.O.O.M. and patterns based software development) manifests that the State diagrams are developed by analyzing Use cases (more precisely: the scenario described by it), one state diagram per class or object. In fact they say that State diagram should be developed for each most important object within the system. These statements raise a set of ambiguousness and questions. The Use cases cannot be considered as a complete problem domain representation and a formal connection between problem domain and the proposed solution. The application of Use cases to develop diagrams of other types (such as State diagram) depends much on the designers' personal experience and knowledge.

The elaborated TopUML modeling (including the State diagram development) proposes a way on how to formally overcome the gap between problem domain and solution domain – the first one is represented by TFM which shows the complete functioning of a problem domain and the latter one is obtained by transforming TFM of a problem domain. Moreover the TopUML enables formal identification of the most important objects and classes within system – they are denoted by TFM: functional features that are included into main functional cycle specify these objects and classes. In contrast, the reviewed UML modeling driven methods relies that the designers' personal experience and knowledge is sufficient to identify most important objects within system. In addition the example described in paper shows State diagram

development for the case study in which enterprise data synchronization system has been developed by using TopUML modeling.

This research shows that by adding additional efforts at the very beginning of software development life cycle it is possible to create a model that contains sufficient and accurate information of problem domain. By "sufficient" meaning that this model can be transformed into other diagrams without major re-analysis of problem domain and by "accurate" meaning that the model precisely reflects the functioning and structure of the system.

Acknowledgement

This work has been supported by the European Social Fund within the project "Support for the implementation of doctoral studies at Riga Technical University".

References

[1] H. Podeswa, *UML for the IT Business Analyst*. 2nd ed. Course Technology PTR, 2009.
[2] K. Scott, *The Unified Process Explained*. Addison-Wesley, USA, 2001.
[3] *Unified Modeling Language Superstructure version 2.3*. OMG, May 2010. Available from: http://www.omg.org/spec/UML/2.3/Superstructure/PDF/.
[4] C. Jones, Positive and negative innovations in Software Engineering, *International Journal of Software Science and Computational Intelligence* **1**(2) (2009), 20-30.
[5] J.Osis and E. Asnina, A business model to make software development less intuitive. In: *Proceedings of the 2008 International Conference on Innovation in Software Engineering*, Vienna, Austria. IEEE Computer Society CPS, Los Alamitos, USA, 2008, 1240-1246.
[6] J. Osis, Formal computation independent model within the MDA life cycle, *International Transactions on Systems Science and Applications* **1**(2) (2006), 159- 166.
[7] J. Osis and E. Asnina, Topological modeling for model-driven domain analysis and software development: functions and architectures. In: *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, IGI Global, Hershey - New York, 2011, 15-39.
[8] J. Osis and E.Asnina, Is modeling a treatment for the weakness of Software Engineering? In: *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, IGI Global, Hershey - New York, 2011, 1-14
[9] J. Miller and J. Mukerji, editors, *MDA Guide Version 1.0.1*. OMG, 2003.
[10] J. Osis, E. Asnina, and A. Grave, 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)*, Barcelona, Spain, 2007, 66 -71.
[11] W. Zhang, H. Mei, H. Zhao, and J. Yang, Transformation from CIM to PIM: A feature-oriented component-based approach. In: *Model Driven Engineering Languages and Systems*, Lecture Notes in Computer Science **3713** (2005), Springer-Verlag, Berlin, 248-263.
[12] T. Yue, L. Briand, and Y. Labiche, A use case modeling approach to facilitate the transition towards analysis models: concepts and empirical evaluation. In: *Model Driven Engineering Languages & Systems*, Notes in Computer Science **5795** (2009), Springer-Verlag, Heidelberg, 484-498.
[13] E. Letier and A. van Lamsweerde, Deriving Operational Software Specifications from System Goals. In: *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*, ACM, New York, 2002, 119-128.
[14] I. Diaz, O. Pastor, and A. Matteo, Modeling interactions using role-driven patterns. In: *Proceedings of 13th IEEE International Conference on Requirements Engineering (RE 2005)*, Paris, France. IEEE Computer Society, 2005, 209-220.
[15] E. Insfran, O. Pastor, and R. Wieringa, Requirements engineering-based conceptual modeling, *Requirements Engineering* **7**(2) (2002), 61-72.
[16] U. Donins, and J. Osis, Topological modeling for enterprise data synchronization system: a case study of topological model-driven software development. In: *Proceedings of the 13th International Conference on Enterprise Information Systems (ICEIS 2011)* **3** (2011), SciTePress, 87-96.

[17] U. Donins, Software development with the emphasis on topology. In: *Advances in Databases and Information Systems*, Notes in Computer Science **5968** (2010), Springer-Verlag, Berlin, 220-228.
[18] J. Osis and E. Asnina, *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, IGI Global, Hershey-New York, USA, 2011.
[19] J. Osis and U. Donins, Platform independent model development by means of Topological Class Diagrams. In: *Model-Driven Architecture and Modeling Theory-Driven Development*, 2nd international MDA & MTDD workshop in conjunction with ENASE 2010, Athens, Greece, SciTePress, Portugal, 2010, 13-22.
[20] J. Osis and E. Asnina, Enterprise modeling for information system development within MDA. In: *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008),* USA, 2008, 490.
[21] S. Kent, The Unified Modeling Language. In: *Formal Methods for Distributed Processing: A Survey of Object-Oriented Approaches*, 1st edition (October 22, 2001), Cambridge University Press, 126-151.
[22] B. Dobing and J. Parsons, Dimensions of UML diagram use: practitioner survey and research agenda. In: *Principle Advancements in Database Management Technologies: New Applications and Frameworks*, IGI Global, 2010, 271-290.
[23] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. 2nd ed. Addison-Wesley, 2005.
[24] G. Booch, R. Maksimchuk, M. Engel, B. Young, J. Conallen, and K. Houston, *Object-Oriented Analysis and Design with Applications*. 3rd ed. Addison-Wesley, 2007.
[25] C. Larman, Applying UML and Patterns: *An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3rd ed. Prentice Hall, 2005.
[26] A. Olive, *Conceptual Modeling of Information Systems*, Springer, 2007.
[27] P. Stevens and R. Pooley, *Using UML: Software Engineering with Objects and Components*. 2nd ed. Addison-Wesley, (2005).
[28] J. Osis and E. Asnina, Derivation of use cases from the topological computation independent business model. In: *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, IGI Global, Hershey - New York, 2011, 65-89.
[29] J. Osis, E. Asnina, and A. Grave, Formal problem domain modeling within MDA. In: *Evaluation of Novel Approaches to Software Engineering*. Communications in Computer and Information Science **22** (2008), Springer-Verlag, Berlin, 387-398.
[30] J. Osis, E. Asnina, and A. Grave, Formal computation independent model of the problem domain within the MDA. In: *Proceedings of the 10th International Conference on Information Systems and Formal Models (ISIM'07)*, Silesian University in Opava, Czech Republic, 2007, 47-54.
[31] E. Asnina and J. Osis, Topological Functioning Model as a CIM-business model. In: *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, IGI Global, Hershey - New York, 2011, 40-64.
[32] J. Osis and A. Šlihte Transforming textual use cases to a computation independent model. In: *Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development*, Athens, Greece, July 22-24, 2010, 33-42.
[33] O. Nikiforova, Object interaction as a central component of object-oriented system analysis. In: *Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development*, Greece, Athens, July 22-24, 2010, 3-12.
[34] R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers, Using description logic to maintain consistency between UML models. In: *UML 2003 - The Unified Modeling Language, Modeling Languages and Applications*. Lecture Notes in Computer Science **2863** (2003), Springer-Verlag, Berlin, 326-340.
[35] J. Osis and U. Donins, Formalization of the UML class diagrams. In: *Evaluation of Novel Approaches to Software Engineering.* Communications in Computer and Information Science **69** (2010), Springer-Verlag, Berlin, 180-192.
[36] X. Li and J. Parsons, Ontological semantics for the use of UML in conceptual modeling. In: *Challenges in Conceptual Modelling. Tutorials, posters, panels and industrial contributions at the 26th International Conference on Conceptual Modeling*, Auckland, New Zealand, November 5-9, 2007, 179-184.
[37] U. Donins, J. Osis, A. Šlihte, E. Asnina, and B. Gulbis, 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*, China, Beijing, June 8-11, 2011, 79-88.