Notion of Causal Relations of the Topological Functioning Model

Erika Asnina, *Riga Technical University*

***Abstract* – The paper discusses application of the topological functioning model (TFM) of the system for its automated transformation to behavioural specifications such as UML Activity Diagram, BPMN diagrams, scenarios, etc. The paper addresses a lack of formal specification of causal relations between functional features of the TFM by using inference means suggested by classical logic. The result is reduced human participation in the transformation as well as additional check of analysis and specification of the system.**

***Keywords* – analytical models, system analysis and design, topological functional model**

I. Introduction

Software engineering means analysis and modelling of systems and corresponding software. In turn, software development used to limit these activities and put the main focus on analysis and modelling of the software. The system analysis is quite superficial. The result is a gap between the systems and implementation of its supporting software [1].

The ideas and implementation approaches for overcoming this issue, e.g. [2], [3], [4], [5], [6] and [7], are not widely used in industry, since they require more efforts and time resources at the very beginning of software development process, namely, before or simultaneously with requirements gathering. Additional disadvantage is that results of such analysis are not considered as contractual deliverable items to the client.

A principle of architectural separation of concerns in specifications proposed in OMG's Model Driven Architecture (MDA) could be a solution for wide adoption of system analysis and modelling in industry [8]. MDA suggests three viewpoints on the system, namely, a computation independent, a platform independent and a platform specific one. It is evident that the platform specific viewpoint considers the software.

Historically, a border between the computation independent viewpoint and platform independent one is fuzzy. The computation independent viewpoint should consider both software and system, thus providing the correspondence of the software model to the system model [9], [10]. A topological functioning model (TFM) provides such correspondence by means of mathematical continuous mapping between graphs on different level of abstractions as well as similar graphs [11]. Thanks to holistic formal nature of the TFM, it is a means for verification of requirements completeness [12], determination of shared functionality and derivation of use cases [13], integration of system knowledge that usually are expressed as a set of interrelated fragments [14], and derivation of system's structure [15].

Research on applications of the topological functioning model gave the theoretical definition of TFM functional feature, this will be discussed below. However, in author's opinion, determination of cause-and-effect (causal) relations between functional features of the TFM of business is still an issue. There are open questions on the scope of relations, their necessity and sufficiency as well as temporal characteristics. Besides it is not clear enough how causality of the TFM of the *system* should be reflected in *software* models. Therefore, there is a need for more formal definition of semantics of causal relations of the TFM.

This paper discusses theoretical foundations of cause-and-effect relations of the TFM by considering logical and mathematical characteristics of causal implication. The paper is organized as follows. Section II describes mathematical foundations of the TFM and gives a short introduction to causal relations. Section III investigates notion of the TFM cause-and-effect relation and gives its formal definition. Section IV illustrates application of the proposed definition. Section V concludes the paper by discussing results and future research directions.

II. Topological Functioning Model in Brief

Mathematically, the TFM is represented in the form of a topological space (X, Θ), where X is a finite set of functional features (characteristics) of the system under consideration, and Θ is the topology that satisfies axioms of topological structures and is represented in the form of a directed graph [1]. Properties of topological spaces are described in details in [16], [17]. The process of construction of the TFM consists of definition of system's functional features, cause-and-effect relations among them, and separation of the TFM from the topological space of the system. The details are described in [11], [12], and [15]. The stage we consider here is related to determination of cause-and-effect relations.

A metamodel of the TFM was defined in [18]. Its fragment is illustrated in Fig. 1. The metamodel is described at the MOF (Meta Object Facility) metalevel M2, and represents the TFM as an instance of the metaclass *TFMTopological-FunctioningModel* that includes *at least two* functional features of the metaclass *TFMFunctionalFeature*. They can be joined in functional feature sets, the metaclass *TFMFunctionalFeatureSet*. This means that a functional feature represented in a TFM can visualize a functional feature set. One functional feature can contain only one set and one functional feature can belong only to one set. Functional features can form functioning cycles of different order, the

metaclass *TFMCycle*. For every particular system (or a subsystem), only one cycle can be the main one.

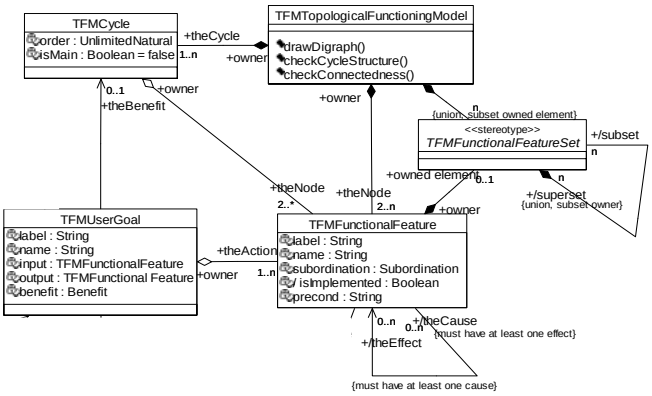

Fig. 1. The fragment of TFM metamodel [18]

Cause-and-effect relations connect functional features. A *cause* functional feature must have at least one effect. An *effect* functional feature must have at least one cause. The functional features can be associated with several goals, the metaclass *TFMUserGoal*, which are established by direct users of the business or software system [18].

A. Definition of TFM Functional Features

The functional feature is defined in [10] as **a unique tuple <A, R, O, PrCond, PostCond, Pr, Ex>**, where:

- **A** is an action linked with an object;
- **R** is a result of that action (it is an optional element);
- **O** is an object (objects) that get the result of the action or an object (objects) that is used in this action; it could be a role, a time period or a moment, catalogues etc.;
- **PrCond** is a set PrCond = $\{c_1, \ldots, c_i\}$, where c_i is a precondition or an atomic business rule (it is an optional element);
- **PostCond** is a set PostCond = $\{c_1, \ldots, c_i\}$, where c_i is a post-condition or an atomic business rule (it is an optional element);
- **Pr** is a set of responsible entities (systems or subsystems) which provide or suggest an action with a set of certain objects;
- **Ex** is a set of responsible entities (systems or subsystems) which enact a concrete action.

B. Informal Definition of TFM Cause-and-Effect Relations

Identification of cause-and-effect relations is intuitive work based on modeller's knowledge and understanding of system's operation. As stated in [10] "it is assumed in topological functioning modelling that a cause-and-effect relation between two functional features of the system exists if the appearance of one feature is caused by the appearance of the other feature without participation of any third (intermediary) feature."

Advice on text analysis borrowed from writing discipline can be used here in order to identify a certain conditional expression, the causal implication. What do we know about causal implication? The knowledge about causality can be summarized as follows [10]:

- It has *a time dimension*, since a cause chronologically precedes an effect;
- In causal connections *"something is allowed to go wrong"*, whereas logical statements allow no exceptions;
- Causes may be *sufficient* or *necessary* (in other words, complete or partial) for generating an effect;
- Cause-and-effect relations involve *multiple factors*. Sometimes there are factors in series. Sometimes there are factors in parallel.
- The causality is *universal*. This means that there is no such problem domain without causes and effects.

Indeed, identification of cause-and-effect relations is and probably will remain intuitive work. But results of this intuitive work must be specified in the form that could be used in automated transformations from the TFM to other, more detailed, models.

C. Particularities of TFM Cause-and-Effect Relations

By now, it is assumed that all causes are sufficient in application of the TFM for business modelling. Additionally, there are no explicit means for determination and specification of multiplicity of factors. Therefore, it is hard to automatically handle logical branching in case of transformation from the TFM to behavioural specifications (such as Unified Modelling Language (UML) activity diagrams, use case models, and diagrams in Business Process Modelling Notation). Research on transforming the TFM to such diagrams summarized in [19] has demonstrated the following:

- From TFM to BPMN diagrams. Cause-and-effect relations are used to specify *control* and *message flows* between activities – control flows between activities in the same pool, and message flows between activities from different pools. Due to different interpretation of chronology, BPMN shows the fragment of chronology as a sequence of events or a message flow from the start event to the end event, while the TFM shows it as a sequence of causes and effects that repeats all the time while the system works. Post-conditions and preconditions are transformed to annotations.
- From TFM to UML activity diagrams. A cause-and-effect relation is transformed into a *control flow* between corresponding activities. However, it is impossible to create fork and join nodes automatically, because the TFM does not hold information of concurrency. Functional feature's precondition is transformed into activity's precondition and may indicate at the necessity of creation of a *decision node*. Functional feature's post-condition is transformed into a post-condition of the activity and may indicate at necessity of creation of a *decision* or *merge node*. Thus, the TFM can be transformed to a simple activity diagram that should be refined by a developer if necessary.
- From TFM to Use Case Model. Cause-and-effect relations are used to define *logical flows* in the use case scenario. Optional execution of the step (IF condition), and cyclic execution of the steps (WHILE or FOR

cycles) can be defined from pre- and post-conditions of functional features. However, it is impossible to define synchronous or asynchronous executions of these steps without human participation.

Summarizing, the TFM does not hold information about concurrent (synchronous and asynchronous) execution of functional features. In behavioural diagrams of software, cause-and-effect relations are reflected as *logical sequences* of functional parts and *control flows* between functional parts. It is necessary to note, that a logical sequence should be a specialization of a control flow that complies with business logic of the system.

III. Formal Definition of Cause-and-Effect Relations

As it can be concluded from Section II, automated mappings of cause-and-effect relations to control flows of software require human participation in case of branching, since some knowledge of system functionality still is kept informally in textual descriptions or implicitly in experts' minds. Therefore, the open task is to move this implicitly or informally expressed knowledge to formal specification in form of TFM elements. This task has one important constraint, namely, this formal specification should not make it harder presenting and understanding complex graph structures. Complex graphs have multiple arcs among multiple vertices, and additional multiple graphical constructs on these arcs complicates human operation on such graphs.

A. Formal Definition

One of possible formalizations is obligatory determination and specification of all pre- and post-conditions of every TFM functional feature. Then it would be possible "to connect" a post-condition of one functional feature with an equal precondition of another functional feature. Thus *sequence* of functional parts would be defined. However, the question about *logical (control) relations* between those sequences within a behavioural scenario and among behavioural scenarios cannot be solved without introducing some logical operations in the textual or visual specifications of pre- and post-conditions.

The other way, suggested in this paper, is giving formal specification of cause-and-effect relations similarly to formal definition of TFM functional feature in order to automate their handling.

The formal specification of a cause-and-effect relation is a unique tuple <Id, C, E, G, N, S, Refs>, where:

- **Id** is a unique identifier of this cause-and-effect relation;
- **C** (cause) is a functional feature that generates functional feature E, this could not be empty;
- **E** (effect) is a functional feature that is generated by functional feature C, this could not be empty;
- **G** (goal) is system user's goal, which achievement requires execution of functional features C and E;
- **N** is necessity of the functional feature C for achievement of goal G; the values are *true* or *false*;
- **S** is sufficiency of the functional feature C for achievement of goal G; the values are *true* or *false*;
- **Refs** (references) is a set of unique tuples <*Ref_Ids*, *LOp*>, where *Ref_Ids* is a set of identifiers (ref_1, … ref_m) of cause-and-effect relations that participate in logical operation *LOp* in order to achieve goal G together. The set *Ref_Ids* must not include the identifier of the relation Id.

Necessity and sufficiency are concepts of classical logic; they induce substantial and consistent effects on conditional reasoning performance. There are four classical combinations possible [20]:

- *Modus ponens*. IF cause THEN effect. Cause occurs. Thus, the effect follows.
- *Modus Tollens*. IF cause THEN effect. Effect does not occur. Thus, the cause did not preceded.
- *Affirmation of the Consequent*. IF cause THEN effect. Effect occurs, thus the cause preceded.
- *Denial of the Antecedent*. IF cause THEN effect. Cause did not occur. Thus, the effect does not follow.

The *necessity* of the cause is determined when the occurrence of the effect indicates the occurrence of the cause. The *sufficiency* of the cause is determined when the occurrence of the cause indicates the occurrence of the effect. The *necessary and sufficient* cause is when the occurrence of the effect is possible if and only if the cause occurred, and occurrence of the effect indicates the obligatory occurrence of the cause.

Logical operators are operators from classical logic such as conjunction (AND) and disjunction (OR). Conjunction indicates synchronous occurrence of referenced causes. Disjunction indicates asynchronous occurrence of referenced causes.

IV. Illustrating Example

For a better illustration let us consider the following small fragment of an informal description from the project, in which a library application is developed. The process of constructing the TFM of this example is described in detail in [21].

"When an unregistered *person* **arrives**, the *librarian* **creates** a new *reader account* and a *reader card*. The *librarian* **gives out** the *card* to the *reader*. When the *reader* **completes** the *request for a book*, he **gives** it to the *librarian*. The *librarian* **checks out** the requested *book* from a *book fund* to a *reader*, if the *book copy* is available in a *book fund*. When the *reader* **returns** the *book copy*, the *librarian* **takes** it **back** and **returns** the *book* to the *book fund*. He **imposes** the *fine*, if the *term of the loan* is exceeded, the *book* is lost, or is damaged. When the *reader* **pays** the *fine*, the *librarian* **closes** the *fine*. If the *book copy* is hardly damaged, the *librarian* **completes** the *statement of utilization*, and **sends** the *book copy* to the *Utilizer*."

In the fragment nouns are denoted by *italic*, verbs are denoted by **bold**, and action pre- (or post-) conditions are underlined.

A. Functional Features

Functional features identified from the fragment are given in the form "identifier: feature_description, precondition,

responsible_entity (where, “Lib” denotes “librarian”, and “R” denotes “reader”) and they are as follows [21]:

f1: Arriving [of] a person, {}, person;

f2: Creating a reader account, {unregistered person}, Lib;

f3: Creating a reader card, {}, Lib;

f4: Giving out the card to a reader, {}, Lib;

f5: Getting the status of a reader, {}, R;

f6: Completing a request_for_book, {}, R;

f7: Sending a request_for_book, {}, R;

f8: Taking out the book copy from a book fund, {}, Lib;

f9: Checking out a book copy, {completed request, book copy is available}, Lib;

f10: Giving out a book copy, {}, Lib;

f11: Getting a book copy [by a registered reader], {}, R;

f12: Returning a book copy [by a registered person], {}, R;

f13: Taking back a book copy, {}, Lib;

f14: Checking the term of loan of a book copy, {}, Lib;

f15: Evaluating the condition of a book copy, {}, Lib;

f16: Imposing a fine, { loan term is exceeded, lost book, or damaged book}, Lib;

f17: Returning the book copy to a book fund, {}, Lib;

f18: Paying a fine, {imposed fine}, R;

f19: Closing a fine, {paid fine}, Lib;

f20: Completing a statement_of_utilization, {hardly damaged book copy}, Lib;

f21: Sending the book copy to a Utilizer, {}, Lib;

f22: Utilizing a book copy, {}, utilizer;

f23: Adding the request_for_book in a wait list, {unavailable book}, Lib;

f24: Checking the request_for_book in a wait list, {a book copy is returned to the book fund}, software system;

f25: Informing the reader via SMS, {a request in the wait list can be satisfied}, software system;

f26: Avoiding a request for a book, {book copy is not available}, software system.

B. Users and Goals

There are two responsible entities that belong to set Ex, namely, the Librarian and the Software System. The Librarian sets the following goals, which correspond to the following functional features:

- BG1 “Register a reader”, {f2, f3, f4};
- BG2 “Check out a book”, {f5, f6, f7, f8, f9, f23, f26};
- BG3 “Return a book”, {f5, f12, f13, f14, f15, f16, f17, f24, f25};
- BG4 “Pay a fine”, {f18, f19};
- BG5 “Impose a fine”, {f16};
- BG6 “Close a fine”, {f19}.

Software System sets the goal BG7 “Inform of available book”, {f24, f25}.

C. Cause and Effect Relations

Cause-and-effect relations identified from the fragment are represented as arcs of a digraph that are oriented from a cause vertex to an effect vertex and they are illustrated by the means of the TFM in Fig. 2. However, in order to demonstrate the presented formalism, each cause-and-effect relation is extended with the tuple <Id, C, E, G, N, S, Refs>. The descriptions are the following:

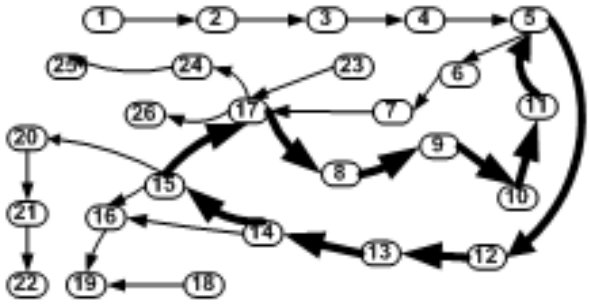

Fig. 2. The TFM of library functionality [21]

- **r1-2:** C= f1, E= f2, G= BG1, N=true, S=true, Refs = empty set;
- **r2-3:** C= f2, E= f3, G= BG1, N=true, S=true, Refs = empty set;
- **r3-4:** C= f3, E= f4, G= BG1, N=true, S=true, Refs = empty set;
- **r5-6:** C= f5, E= f6, G= BG2, N=true, S=true, Refs = empty set;
- **r6-7:** C= f6, E= f7, G= BG2, N=true, S=true, Refs = empty set;
- **r17-8:** C= f17, E= f8, G= BG2, N=true, S=true, Refs = empty set;
- **r8-9:** C= f8, E= f9, G= BG2, N=true, S=true, Refs = empty set;
- **r9-10:** C= f9, E= f10, G= None, N=true, S=true, Refs = empty set;
- **r10-11:** C = f10, E= f11, G= None, N=true, S=true, Refs = empty set;
- **r5-12:** C= f5, E = f12, G= BG3, N=true, S=true, Refs = empty set;
- **r12-13:** C= f12, E= f13, G= BG3, N=true, S=true, Refs = empty set;
- **r13-14:** C= f13, E= f14, G= BG3, N=true, S=true, Refs = empty set;
- **r14-15:** C= f14, E= f15, G= BG3, N=true, S=true, Refs = empty set;
- **r17-24-I:** C= f17, E= f24, G= BG3, N=true, S=true, Refs = empty set;
- **r17-24-II:** C= f17, E= f24, G= BG7, N=true, S=true, Refs = empty set;
- **r17-26:** C= f17, E= f26, G= BG2, N=true, S=true, Refs = empty set;
- **r24-25-I:** C= f24, E= f25, G= BG3, N=true, S=true, Refs = empty set;
- **r24-25-II:** C= f24, E= f25, G= BG7, N=true, S=true, Refs = empty set;
- **r15-20:** C= f15, E= f20, G= None, N=true, S=true, Refs = empty set;
- **r20-21:** C= f20, E= f21, G= None, N=true, S=true, Refs = empty set;

- **r21-22:** C= f21, E= f22, G= None, N=true, S=true, Refs = empty set;
- ***r4-5:*** *C= f4, E= f5, G=BG2, N=true, S=false, Refs = {r11-5, OR};*
- ***r11-5:*** *C= f11, E= f5, G=BG2, N=true, S=false, Refs = {r4-5, OR};*

- ***r14-16-I:*** *C= f14, E= f16, G= BG3, N= false, S=true, Refs = {r15-16-I; OR}*
- ***r14-16-II:*** *C= f14,E= f16, G= BG5, N= false, S= true, Refs = {r15-16-II; OR};*
- ***r15-16-I:*** *C= f15, E= f16, G=BG3, N= false, S= true, Refs = {r14-16-I; OR};*
- ***r15-16-II:*** *C= f15, E= f16, G=BG5, N=false, S= true, Refs ={r14-16-II; OR};*

- ***r7-17:*** *C= f7, E= f17, G=BG3, N= false, S= false, Refs = empty set;*
- ***r15-17:*** *C= f15, E= f17, G=BG3, N= true, S= false, Refs = empty set;*
- ***r23-17:*** *C= f23, E= f17, G= BG3, N=false, S=false, Refs =empty set;*

- ***r16-19-I:*** *C= f16, E= f19, G=BG4, N= true, S=false, Refs = {r18-19-I, AND};*
- ***r16-19-II:*** *C= f16, E= f19, G=BG6, N=true, S= false, Refs ={r18-19-II, AND};*
- ***r18-19-I:*** *C= f18, E= f19, G=BG4, N=true, S= false, Refs ={r16-19-I, AND};*
- ***r18-19-II:*** *C= f18, E= f19, G=BG6, N=true, S= false, Refs = {r16-19-II, AND}.*

Let us consider cause-and-effect relations *r4-5* and *r11-5*. They both are necessary and not sufficient. This means that the occurrence of functional feature f5 “Getting the status of a reader” was preceded by either functional feature f4 “Giving out the card to a reader” or functional feature f11 “Getting a book copy [by a registered reader]”.

Cause-and-effect relations *r15-16-I* and *r14-16-I* as well as *r15-16-II* and *r14-16-II* both are sufficient but not necessary. This means that the occurrence of functional feature f16 “Imposing a fine” can be caused either by functional feature f14 “Checking the term of loan of a book copy” or by functional feature f15 “Evaluating the condition of a book copy” or both.

The description of cause-and-effect relations *r7-17*, *r15-17* and *r23-17* indicates **a mistake** in understanding the causality of the system made before. Both *r7-17* and *r23-17* are not necessary and not sufficient. This means that the occurrence of functional feature f17 does not depend on functional features f7 and f23. Therefore these cause-and-effect relations are set incorrectly and the TFM must be corrected in accordance with the business logic.

Cause-and-effect relations *r16-19-I* and *r18-19-I as* well as *r16-19-II* and *r18-19-II* are necessary but not sufficient. The logical operator AND between functional features f16 “Imposing a fine” and f18 “Paying a fine” indicates that the occurrence of the functional feature f19 “Closing a fine” is possible only after occurrence of both functional features f16 and f18. This means that only an imposed and paid fine can be closed. If the fine was imposed and not paid or a fine that was not imposed was paid then the closing of the fine is not possible because of a lack of payment or because of its absence in the reality. Additionally, the textual description and precondition of functional feature 19 does not take into account the latter possibility.

All the other cause-and-effect relations are both necessary and sufficient. This means that occurrence of the effect functional feature E is possible if and only if the cause functional feature C occurred.

D. Mapping to UML Activity Diagram

Fig. 3 shows how a part of the use case model borrowed from [21] can be described in an UML Activity Diagram using information from the TFM, where functional features are transformed into activities, but cause-and-effect relations into control flows in conformity with their formal specifications. The logic of control flows is kept in accordance with the TFM and sufficiency of functional features f14 and f15 as well as their logical relationships are taken explicitly from the TFM specification and this does not require any human participation.

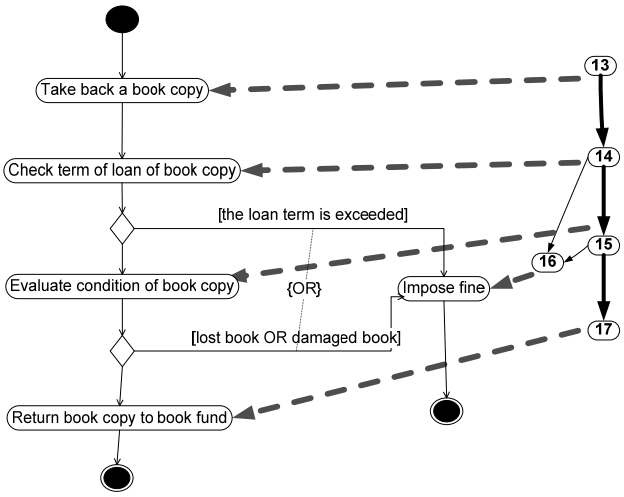

Fig. 3. Use cases “Return book” and “Impose fine” in modified UML activity diagram

V. Conclusion

The aim of this paper was to demonstrate that formalization of knowledge about the system in form of mathematical model is a step towards better understanding of the system’s functionality and automated model-to-model transformation starting from the very beginning of the software development process.

The suggested formal specification answers the question about necessity and sufficiency of causal relations in the TFM. It allows discovering misunderstandings or mistakes in the already constructed TFM when necessity and sufficiency of causes are analyzed.

Besides that, this specification allows inferring chronology of occurrences of causes and effects. The proposed specification of the fact that logical relations among causes are necessary and/or sufficient for generating an effect allows automatic definition of synchronous and asynchronous occurrences of the causes in order to achieve a particular system (or business) goal.

Assigning causal relations to the goals of the system helps in understanding meaning of each relation in a particular case of system's operation, thus the scope of relations is defined.

However, the question of how causality of the TFM of the *system* should be reflected in *software* models still is not addressed completely, because a control flow in the software models can be expressed in different ways.

The future research directions are related to model checking. Model checking is a field where results of analysis and modelling of system and software behaviour are formally and automatically verified.

REFERENCES

[1] J. Osis, "Formal Computation Independent Model within the MDA Life Cycle", *International Transactions on Systems Science and Applications, V. 1, Nr. 2*, Xiaglow Institute Ltd, Glasgow (UK), 2006, pp. 159 – 166.

[2] A. van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", *Proceedings RE'01, 5th IEEE International Symposium on Requirements Engineering*, Toronto, 2001, pp. 249-263.

[3] A. Dardenne, A. van Lamsweerde, S. Fickas, "Goal-Directed Requirements Acquisition", *The Science of Computer Programming*, 20(November), 1993, pp. 3-50.

[4] E.S.K. Yu, "Towards Modeling And Reasoning Support For Early-Phase Requirements Engineering", *Proceedings of International Symposium on Requirements Engineering*, Annapolis, Maryland, USA, 1997, pp. 226-235.

[5] T. Gorschek, C. Wohlin, "Requirements Abstraction Model", *Requirements Engineering, vol. 11*, 2006, pp. 79-101.

[6] H. Kaindl, "A Design Process Based on a Model Combining Scenarios with Goals and Functions", *IEEE Trans. on Systems, Man and Cybernetics, Vol. 30, No.5*, 2000, pp.537-551.

[7] M. Jackson, "Problem Frames and Software Engineering", *Information and Software Technology*, 47(November), 2005, pp. 903-912.

[8] The Object Management Group. (2003). MDA guide version 1.0.1. (J. Miller, & J. Mukerji, Eds.). [Online]. Available: http://www.omg.org/. [Accessed: Nov 20, 2010]

[9] J. Osis, E. Asnina: A Business Model to Make Software Development Less Intuitive. In: Proceedings of 2008 International Conference on Innovation in Sofware Engineering (ISE 2008). December 10-12, 2008, Vienna, Austria. IEEE Computer Society Publishing, 2008, pp. 1240-1245.

[10] Asnina, E., Osis, J. (2010). Computation independent models: bridging problem and solution domains. In J. Osis, & O. Nikiforova (Ed.), Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development MDA & MTDD 2010, In conjunction with ENASE 2010, Athens, Greece, July 2010 (pp. 23-32). Portugal: SciTePress.

[11] J. Osis, E. Asnina. Chapter 2: Topological Modeling for Model-Driven Domain Analysis and Software Development. In J. Osis, E. Asnina, Model-Driven Domain Analysis and Software Development: Architectures and Functions, 2011 (pp. 15-39). Hershey, New York, USA: IGI Global.

[12] J. Osis, E. Asnina, A. Grave. Formal Problem Domain Modeling within MDA. Communications in Computer and Information Science (CCIS), Springer Verlag Berlin Heidelberg, Volume 22, Part III, 2008, pp. 387-398.

[13] J. Osis, E. Asnina. Chapter 4: Derivation of Use Cases from the Topological Computation Independent Business Model. In J. Osis, E. Asnina, Model-Driven Domain Analysis and Software Development: Architectures and Functions, 2011 (pp. 65-89). Hershey, New York, USA: IGI Global.

[14] A. Slihte, J. Osis, U. Donins. Knowledge Integration for Domain Modeling. In: J. Osis, O. Nikiforova (Eds.). Model-Driven Architecture and Modeling-Driven Software Development: ENASE 2011, 3rd Whs. MDA&MDSD, SciTePress, Portugal, 2011, pp. 46 - 56.

[15] U. Donins, J. Osis, A. Slihte, E. Asnina, B. Gulbis. Towards the Refinement of Topological Class Diagram as a Platform Independent Model. In: J. Osis, O. Nikiforova (Eds.). Model-Driven Architecture and Modeling-Driven Software Development: ENASE 2011, 3rd Whs. MDA&MDSD, SciTePress, Portugal, 2011. pp. 79 - 88.

[16] Osis, J.: Software Development with Topological Model in the Framework of MDA. In: Proceedings of the 9th CaiSE/IFIP8.1/EUNO International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'2004) in connection with the CaiSE'2004, Vol.1, pp. 211--220. RTU, Riga, Latvia (2004)

[17] 7. Basener, W.F.: Topology and Its Applications. John Wiley and Sons, Inc., New Jersey, USA (2006)

[18] E.Asnina, "Formalization of Problem Domain Modeling within Model Driven Architecture", Ph.D. thesis, Riga Technical University, RTU Publishing House, Riga, Latvia, 2006.

[19] E. Asnina. A Formal Holistic Outline for Domain Modeling. In: Local Proceedings of Advances in Databases and Information Systems, 13th East-European Conference, ADBIS 2009, Associated Workshops and Doctoral Consortium, Riga, Latvia, September 7-10, 2009.- Riga Technical University, 2009.- pp. 400-407.

[20] Cummins, D. D. Naïve theories and causal deduction. Memory and Cognition, 23, 1995, pp. 646-658.

[21] Osis J., Asnina E., Grave A. "Computation Independent Modeling within the MDA" // Proceedings of IEEE International Conference on Software, Science, Technology & Engineering (SwSTE07), 30-31 October 2007, Herzlia, Israel. – IEEE Computer Society, Conference Publishing Services (CPS), 2007. – 22-34 p.

Erika Asnina received M.Sc. in computer systems with specialization in applied computer science in 2003 and Doctoral degree in engineering science (Dr.sc.ing.) in information technology with specialization in system analysis, modelling and design in 2006 from Riga Technical University.

She is Assistant Professor at Department of Applied Computer Science at Riga Technical University, Latvia. She also worked 5 years as a Software Developer. She is an author of 25 published conference papers, 4 book chapters and 1 monograph. Her research interests include software quality assurance, business modelling, model-driven software development, model transformation languages and software engineering.

She was awarded as one of the best paper authors in the conferences ISIM'05 and ISIM'06, and a scholarship laureate of the target program "For Education, Science and Culture" of Latvian Education Fund in 2004 and 2005.

Contact address is 1/3 Meža Str., Room 504, Riga Technical University, Riga, LV 1048, Latvia; e-mail: erika.asnina@rtu.lv.