The Formal Approach to Problem Domain Modeling Within Model Driven Architecture

Erika Asnina*
Erika.Asnina@cs.rtu.lv

Abstract: This paper represents a formal approach that introduces more formalism into problem domain modeling within OMG Model Driven Architecture and bridges a gap between models of the business system and the application. The basis of this approach is Topological Functioning Modeling (TFM) that allows representing knowledge of problem domain functionality from a computation independent viewpoint. This paper discusses a formal method for TFM construction, TFM capabilities for functional requirement validation and traceability at the beginning of problem domain analysis, the formal goal-based use case identification and refinement.

Key Words: problem domain, topological functioning model, MDA, CIM, use case

1 Introduction

Problem domain modeling in software development raises the question about what domain to be modeled at first: the domain of today reality ("as is") or the domain of customer's expected reality ("to be"). The first step should be evaluating client's current situation, and only after that, developers should define what the new product will be able to do. System requirements themselves are constraints hung on real world phenomena, not vice versa [1], [2].

Object Management Group (OMG) suggested Model Driven Architecture (MDA) that has precise confines among viewpoints on a system, and a framework for model handling, transformation and management [3]. MDA suggests a Computation Independent Model (CIM) for representation of a problem domain and system requirements. Unfortunately, it has a gap in formal (manual or automated) transition from a CIM (a problem domain model) to a Platform Independent Model (an application model). MDA suggests using of Unified Modeling Language that applies use cases for requirement specification. Use cases are driven primarily by application domain analysis, where a problem domain is regarded almost as a black box describing a number of aspects of the system. Even when one applies use cases for business modeling, relationships between a business system and a planned computerized one remain informal as well as identification of business and system use cases themselves.

Therefore, there *is* a lack of formal ways of the application domain conformance to problem domain knowledge. This paper discusses a way, which formalizes this connection, using mostly Topological Functioning Modeling (TFM) and some properties of universal categorical logic.

2 Problem Domain Modeling With the TFM

The TFM was introduced at Riga Technical University by Janis Osis [4]. The most recent researches are related to the TFM application for more formalism introducing in the MDA

* Riga Technical University, Faculty of Computer Science and Information Technology, Institute of Applied Computer Systems, Department of Applied Computer Science, Meza 1/3, LV-1048, Riga, Latvia

framework [2], [5]. The TFM is based on assumption that a complex system can be described in abstract terms as a topological space ($X<\Theta$), where X is a finite set of physical or business functional characteristics, and Θ is a topology in the form of a digraph. It satisfies two Kolmogorov's axioms [2]. The necessary condition of the construction of a topological space in a form of a directed graph $G(X, U)$ is a meaningful and exhaustive verbal, graphical, or mathematical description of the system. The adequacy of the model, describing functionality of some system, can be achieved by analyzing mathematical properties of such an abstract object [4]. A topological functioning model has topological (connectedness, closure, neighborhood, continuous mapping) and functional (cause-effect relations, cyclic structure, inputs and outputs) properties. The detailed information can be found in [2], [4], [5].

In turn, universal categorical logic for computer science was explored in Zinovy Diskin's, Boris Kadish's *et al.* work [6]. It is based on category theory's principles and suitable for representation of complex structural relations. The main idea is specifying any problem domain under construction as a collection of objects and their morphisms.

2.1 A General Description of the Approach

The suggested approach formally connects functionality of a planned system with functionality of the problem domain it works within.

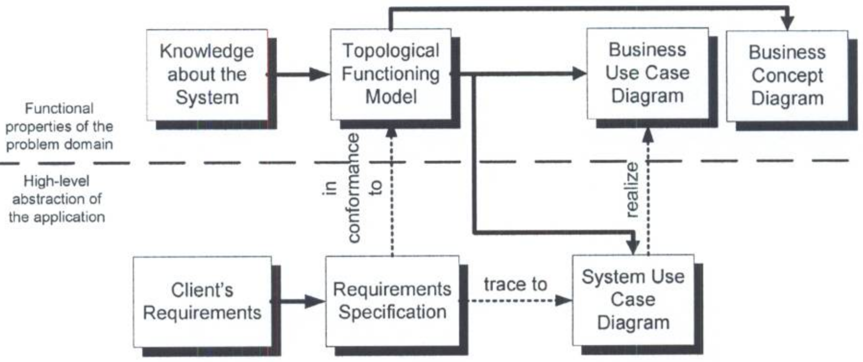
Knowledge about the System
Topological Functioning Model
Business Use Case Diagram
Business Concept Diagram
Functional properties of the problem domain
High-level abstraction of the application
in conformance to
realize
Client's Requirements
Requirements Specification
trace to
System Use Case Diagram

Figure 1. The suggested formalism for problem domain modeling

Figure 1 illustrates two parallel branches: The first one is analysis at the real world system level (a business system), and the second one is analysis at the application level. Having knowledge about a business system, a TFM of this is composed. This model holistically represents complete system functionality from the computation independent viewpoint. At the other branch, client's requirement acquisition concludes with requirements specification drafting. The suggested approach combines these branches and produces problem domain concepts, business use cases and/or system use cases. This means that a TFM of a business system, constrained by the functional requirements and associated to the business goals, provide identification of business and system use cases accordingly to the problem domain realities. Besides that, functional requirements can be easily traced to the created use case diagrams.

The main idea of this paper is to consider modeling of the system functional aspect. Therefore, the suggested here method includes the following: 1) topological functioning model construction, 2) functional requirements mapping onto a TFM (the categorical diagram predicates are used for formal specification of such mapping properties as completeness, overlapping, etc.; although, other techniques also can be used), and 3) system use cases separation from a TFM, constrained by functional requirements.

2.1 A Formal Method for Topological Functioning Model Construction

A formal method for construction of a TFM of a complex technical or business system from its informal verbal description consists of the following: 1) identifying physical or business functional characteristics, 2) introducing topology into a topological space of system characteristics, and 3) separating a TFM of the system from the topological space.

2.1.1 Identification of Physical and Business Functional Characteristics

As a result of definition of physical or business functional characteristics from an informal system description, a list of business objects, a list of inner and external functional features must be found:

- *Identification of business objects.* Every meaningful noun together with a direct object that is expressed as a numeral or a pronoun is to be separated. It is an object or a property.
- *External and self-dependent system identification.* Objects that act as self-dependent entities must be emphasized. They can be objects that affect business system functionality from the external environment, or get something from the system work (e.g. human roles, services, products, organizations, etc.).
- *Functional features identification.* It must be done in accordance with the defined actions that help the system in realizing its goals. This means that meaningful verbs, action preconditions and business rules are to be emphasized. Business rules must be refined to the *atomic*, i.e. until the level, where they do not have sub-rules. Afterwards, each action, precondition and business rule has to be either transformed into an appropriate functional feature (hereafter "FF") or attached to the already identified one. Each FF has the following elements: a) an object's action, b) a result of this action, and c) an object that gets the result of the action or an object that is used in this action (e.g. a role, a time period, a moment, catalogues, etc.). Each FF must be expressed in the following form:

  <action>-ing the <result> [to, into, in, by, of, from] a(n) <object>

  A set of FFs represents a certain business process that is a chain of structured business activities. The result of execution of a FF set must be some useful or valuable output. A FF set must be written down as follows:

  <action>-ing a(n) object

  Continuous mapping holds between FFs and a corresponding FF set. Such description is useful for later on transforming each FF into a corresponding object that represents a business concept, i.e. a class candidate.

2.1.2 Topology Introduction

After defining a list of FFs, cause-effect relations between them must be defined, i.e. *topology* Θ must be set. Cause-effect relations are represented as digraph's arcs that are oriented from a cause to an effect vertex. The connection between a cause and an effect is the *causal implication* (not logical). In causal relations *"something is allowed to go wrong"* whereas logical statements do not allow exceptions. The main properties of cause-effect relations are as follows [7]: a) a cause chronologically precedes an effect; b) a cause *generates* and is *condition* on an effect, c) a cause may be sufficient or necessary (in TFM is assumed that a deal is always with necessary causes), d) a causality is universal. Cause-effect relations can form a causal chain, where all relations independently from their importance are necessary for reaching a final effect. The adequately set causality is a main condition of closuring of an adequate TFM.

2.1.3 TFM Separation

The topological space is a system represented by the expression $Z = N \cup M$, where N is a set of inner system's FFs and M is a set of FFs of other systems that interact with the system or *vice versa*, constituting an interaction area of the external environment and the system itself. A TFM (X, Θ) is separated from the topological space of a problem domain by the closure operation over the set N, i.e. $X = [N] = \bigcup_{\eta=1}^{n} X_\eta$, where X_η is an adherence point of the set N and capacity of X is the number n of adherence points of N. An adherence point of the set N is a point, each neighborhood of which includes at least one point from the set N. The neighborhood of a vertex x in a digraph is the set of all vertices adjacent to x and the vertex x itself. It is assumed here that all vertices adjacent to x lie at the distance $d=1$ from x (here: output arcs from x).

Studying a complex system, its model can be separated into a series of subsystems by the closures of chosen subsets of N.

2.2 Refinement of Functionality, Which Must be Implemented in the Application

The requirement gathering is a complex process. In our case, this approach assumes that functional requirements (hereafter "requirements") are gathered using some proper existing technique and drafted in a complete as possible requirements specification. The given approach does not handle non-functional requirements.

Having identified FFs and business processes, the next step is to determine the functionality that will benefit from automation. This means that requirements must be mapped onto a TFM. This makes us able to validate requirements in conformance to the business logic of the real world system as early as possible. Between requirements and a TFM the following mappings and their visualization by categorical diagram predicates [6] exist:

- *One to One.* One requirement maps onto one FF. This relationship is visualized by inclusion diagram predicate that specifies complete, non-covering correspondence.
- *Many to One.* Several requirements map onto the same FF (complete by default). This is possible when: a) they are covering (overlapping) requirements and, hence, they must be refined; this mapping visualization is a covering predicate; b) the target FF is an abstracted FF set, it is represented by a disjoint predicate (non-overlapping).
- *One to Many.* One requirement maps onto several FFs (a simple projection). The cause is: a) this requirement joins several requirements, hence, it must be split up; or b) FFs in a TFM are at the more detailed level then the requirement and must be abstracted.
- *One to Zero.* One requirement describes some new (or undefined) system functionality. It is necessary to define how this affects the existing system functionality. The TFM must be supplemented by all defined cause-effect relations, and all possible changes in the business logic must be discussed with the customer.
- *Zero to One, Zero to Many.* A requirements specification does not contain any requirement corresponding to the defined FF(s): a) because this FF(s) will not be implemented, or b) a requirement(s) is (are) missed.

2.3 From a TFM to Use Cases

According to [8], a business use case model describes: 1) the business modeled as use cases, 2) external organizations, systems, etc. modeled as actors, 3) the relationships between the external participants and the business process. Both TFM and use cases have functional character. Moreover, a TFM allow formal identification of both business and system use cases that consists of the following:

- *Actors, workers and their business and system goals identification.* Actors in a TFM can be represented as functionality of external systems. Usually they are TFM inputs or outputs. Workers interact with actors and are executors (or initiators) of inner FFs.
- *Goals association with FFs.* Actors and workers set business goals (use cases). For each goal a set of FFs of a TFM that help to reach it and form a goal area must be defined.
- *Use case separation and identification of inclusion and extension use cases.* FFs that have been mapped from requirements and are in the business goal area specify the corresponding system goal and form the system use case. An inclusion use case contains FFs of intersections of goal areas. An extension use case contains FFs in a goal area that are in sub-cycles or a branch and a branching FF is its extension point.
- *Use case prioritizing.* Use cases priorities are set in accordance with client's desires or functional cycles in a TFM of the system, i.e. the closer to the main cycle are use case's FFs the higher is its priority of implementation.

3 A Case Study: Library Functioning

Let's assume that the following simplified informal description of library functioning exists:

"A librarian registers a new person. The librarian fills in a new reader's card with person's data. After that, the librarian issues a library ticket to the new registered reader. Reader's data and the reader's card number are indicated in a library ticket. A reader searches a book in a book catalogue. If there is this book, then he fills in a request for the book. The librarian receives a filled request from a reader, and searches a reader's card by its number. If it is found, the librarian looks for reader's unpaid fines. If there are reader's unpaid fines, then the request cannot be satisfied. If the reader doesn't have fines, then the librarian searches the book in the book fund. If there is a free book instance in the fund, then the librarian writes the code, taking out date and return date of each book, issued from the fund, in the reader's card. The librarian fixes the number of the taken out books and return date in the reader's library ticket and gives out the books to the reader. When a reader returns a book, the librarian searches a reader's card and writes the actual taking back date in it. If the reader returns a book with delay, the librarian imposes a fine to him. After the fine is repaid, the librarian writes out a receipt to the reader. The book is returned into the fund."

3.1 Construction of a TFM of the Library

From this description an *object list* was identified: Librarian, Person (person's data), Reader's Card (number), Book Taking Out (actual taking back date, taking out date, return date, book count, and delay), Library Ticket (book number, return date), Reader (reader's data), Book (code), Catalogue, Request (for the book), (reader's) Fine, Book Fund, and Receipt. *Self-dependent objects* are Person, Reader and Librarian. Person and Reader are *external objects.*

The identified *functional features* are as follows: 1) Registering a reader, 2) Filling in a reader's card, 3) Filling in a library ticket, 4) Issuing the library ticket to a reader, 5) Searching the book in a catalogue, 6) Filling in the request by a reader, 7) Receiving a request, 8) Searching a reader's card, 9) Looking for an unpaid fine, 10) Searching a free book in a book fund, 11) Filling in the book taking out in a reader's card, 12) Filling in the book taking out in a library ticket, 13) Issuing the book to a reader, 14) Returning a book, 15) Filling in the book tacking back in a reader's card, 16) Calculating a fine, 17) Imposing a fine, 18) Repaying the fine by a reader, 19) Writing out the receipt of a fine, 20) Returning the book into a book fund, 21) Coming of a person, 22) Coming of a reader.

In order to introduce topology, cause-effect relations were defined between these functional features as shown in Figure 2(a). Besides that, the FFs were divided into the set $M = \{4, 6, 13, 14, 19, 21, 22\}$ and the set $N = \{1, 2, 3, 5, 7, 8, 9, 10, 11, 12, 15, 16, 17, 18, 20\}$ (see Section 2.1.3). A TFM of the library is got applying the closuring operation over the set N as follows:

$X_1 = \{1, 2\}$ $X_2 = \{2, 3\}$ $X_3 = \{3, 4, 7\}$ $X_5 = \{5, 6\}$

$X_7 = \{7, 8\}$	$X_{10} = \{10, 11\}$	$X_{15} = \{15, 16\}$	$X_{18} = \{18,19,8,10, 7\}$
$X_8 = \{8, 9, 15\}$	$X_{11} = \{11, 12\}$	$X_{16} = \{16, 17, 20\}$	$X_{20} = \{7, 8, 20\}$
$X_9 = \{9, 10, 18\}$	$X_{12} = \{12, 13, 7, 15\}$	$X_{17} = \{17, 18\}$	

FFs of the set $X = [N] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20\}$ forms a TFM of library functioning (the white nodes and relations between them in Figure 2.(a)).

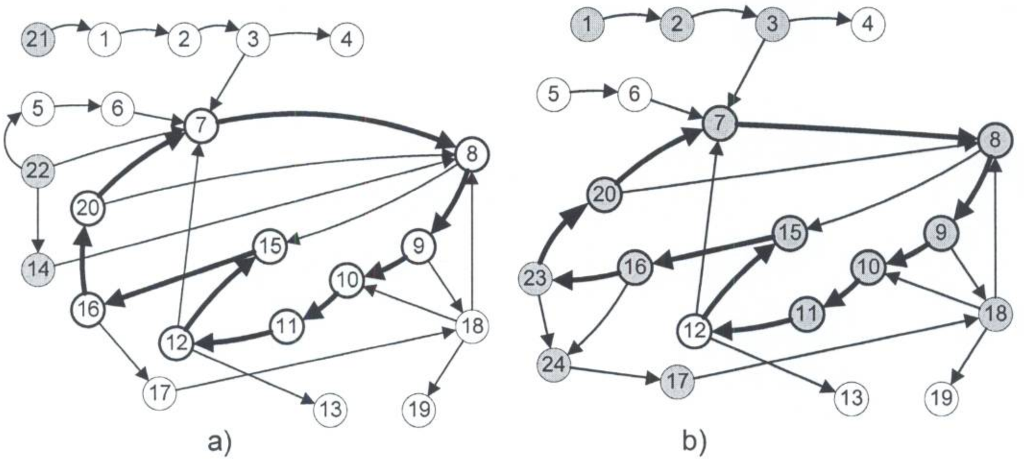

Figure 2. A topological space of library functioning (a) and the TFM constrained by requirements (b)

The main functioning cycle of the TFM of the library is "7-8-9-10-11-12-15-16-20-7" (bold lines in Figure 2.a), which describes main library activities, i.e. book taking out and return.

3.2 Refinement of Functional Requirements

In order to illustrate the approach, let's assume that the following (*ad hoc*) functional requirements are established by the customer:

- *FR1*. The system shall perform registration of a new reader: For each new reader, a new reader's card and a library ticket is written down. It handles information about the reader's name, surname, and personal code, address, contact phone and date of registration. Reader's card has a unique number.
- *FR2*. The system shall perform library ticket printing on the special letterhead: It includes the reader's card number, reader's name and surname, registration date.
- *FR3*. The system shall perform book loan registration: In the registration form, the book code, reader's card number, loan period. The loan is restricted if the reader has one or more unpaid fines.
- *FR4*. The system shall perform book return registration: In the registration form, the return date and book condition are indicated.
- *FR5*. The system shall perform fine calculation and imposing: If the overdue book is returned, then the fine is \$0.02 per day. If the book is damaged, the fine is ½ of the book's price. The calculated fine is recorded in the reader's card. A status of a fine is paid or unpaid.

The correspondence between these requirements and the defined FFs (their numbers are given in parentheses) is set as follows: FR1 (1, 2, 3), FR2 (3), FR3 (7, 8, 9, 10, 11), FR4 (15, 20, and a new FF 23 "Registering the condition of a book"), FR5 (16, 17, 18, and a new FF 24 "Calculating a fine"). Functional features 23 and 24 modify cause-effect relations between functional features 16, 17 and 20; the modified TFM is shown in Figure 2(b). The main cycle in the modified TFM is "7-8-9-10-11-12-15-16-***23***-20-7" (bold lines) with a new added functionality. Functional features {4, 5, 6, 12, 13, 19} will not be implemented (white nodes).

3.3 Use Case Separation from the TFM of the Library

There are three entities that directly interact with the defined system in the TFM: *business actors* - a person and a reader, and *a business worker* - a librarian. Each of them has his business goals. They can be gathered using some suitable technique. Here, a person's goal is to be registered as a reader; reader's goals are book searching, book taking out and tacking back, and fine paying; librarian's goals are to perform registration of a new reader (designated as *BGoal1*), to handle book tacking out (designates as *BGoal2*), to handle book taking back (designates as *BGoal3*), and to register a fine payment (designates as *BGoal4*). Let's assume that only a librarian will be a direct user of the application, i.e. an actor for all use cases. Therefore, his goals considered above help in separating the relevant system use cases. The following functional features help the librarian to reach them: {1, 2, 3, 4} for *BGoal1*, {7, 8, 9, 10, 11, 12, 13, 18, 19} for *BGoal2*, {8, 15, 16, 23, 24, 17, 20, 18, 19} for *BGoal3*, and {8, 9, 18, 19} for *BGoal4*. In accordance with the requirement mappings (Section 3.2), the functional features 4, 12, 13, and 19 will not be implemented. Each goal can be specified as a *business* use case that has the behavior specified by the defined FFs.

The following *system* use cases are defined for each business goal/use case: "Register a new reader" for *BGoal1* as *SGoal1*={1, 2, 3}, "Handle book taking out" for *BGoal2* as *SGoal2*={7, 8, 9, 10, 11, 18}, "Handle book taking back" for *BGoal3* as *SGoal3*={8, 15, 16, 23, 24, 17, 20, 18}, and "Execute a fine payment" for *BGoal4* as *SGoal4*={8, 9, 18}.

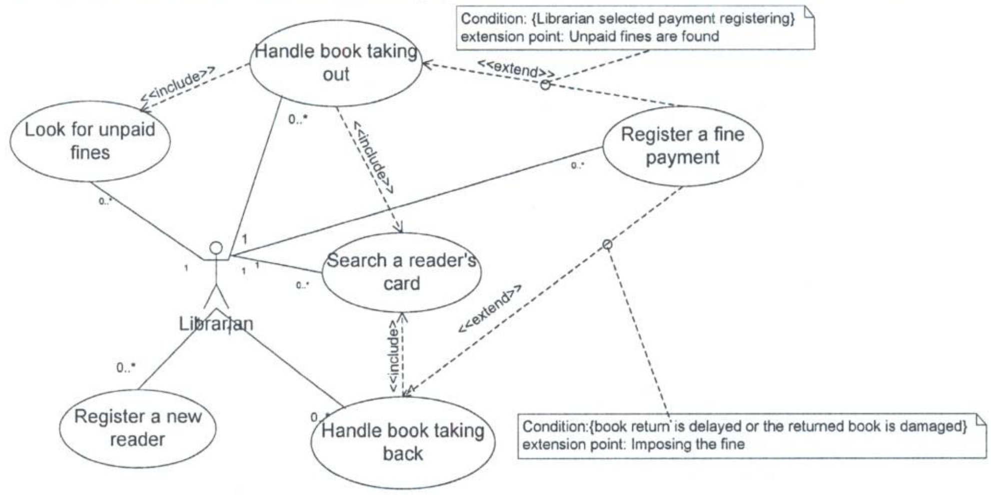

Figure 3. Refined Use Cases Diagram separated from a TFM

As previously mentioned, use case execution can share common functionality. It is defined by intersection of FFs sets, the non-empty are $SGoal2 \cap SGoal3 = SGoal3 \cap SGoal4 = \{8, 18\}$, $SGoal2 \cap SGoal4 = SGoal4 = \{8, 9, 18\}$. In order to define whether it is an inclusion or extension use case, cause- effect relations between these FFs and the others must be analyzed. After that, the following additional use cases were defined from a TFM of the library (Figure 3): "Search a reader's cards"- the FF 8, "Look for unpaid fines"- the FF 9, and "Register a fine payment"- the FF 18. "Search a reader's cards" is *included* into two use cases: "Handle book taking out" and "Handle book taking back". "Look for unpaid fines" is *included* into the use case "Handle book taking out". "Register a fine payment" *extends* two use cases, i.e. "Handle book taking out" with the extension point "Unpaid Fines are Found", and "Handle book taking back" with the extension point "Imposing the Fine". The sequential execution of "Search a reader's cards", "Look for unpaid fines" and "Register a fine payment" provide realizing the *SGoal4* (the unrefined use case "Execute a fine payment").

Requirements are traced to the use cases through the FFs: FR1 and FR2 affect the use case "Register a new reader", FR3 affects "Handle book taking out", "Look for unpaid fines", "Search a reader's cards" and can affect "Handle book taking back" execution, FR4 affects "Handle book taking back", and FR5 affects "Handle book taking back", "Register a fine payment" and, hence, also "Handle book taking out".

4 Conclusions

The represented approach is dedicated to the composing of a formal problem domain model from a computation independent viewpoint and to connecting it with functional requirements to the application. It uses the use cases and topological and functional properties of the TFM. This approach uses also categorical logic, but mainly for formal representation of structural relations between TFM functional features and functional requirements, and provides functional requirement tracing to the use cases. This approach is formal, expressive and intuitively understandable. Models used in this approach help in bridging knowledge experts and software developers. The main bottlenecks of it are cause-effect relations discovering and use case refinement, because it requires experience and human participation in decision making. Approach application is illustrated by the example of library functioning.

The future research objective is creating a tool for this method. Despite necessary human participation in decision making, mathematics used in the approach can be automated as well as formal model verification, and checking satisfaction of topological and functional properties of a TFM.

This work has been partially supported by the European Social Fund within the National Programme "Support for the carrying out doctoral study program's and post-doctoral researches" project "Support for the development of doctoral studies at Riga Technical University".

References

1. Michael Jackson: *Problem Frames and Software Engineering.* The Open University, 2005 - http://mcs.open.ac.uk/mj665/PFrame7.pdf
2. Janis Osis: *Software Development with Topological Model in the Framework of MDA.* 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, RTU, Riga, Latvia 2004, pp. 211 – 220
3. Joaquin Miller, Jishnu Mukerji (eds.): *Model Driven Architecture (MDA).* Architecture Board ORMSC, ormsc/2001-07-01 - www.omg.org/mda/
4. Janis Osis: *Topological Model of System Functioning.* Automatics and Computer Science, J. of Acad. Of Sc., Riga, Latvia #6, 1969, pp. 44-50 (in Russian)
5. Gundars Alksnis, Erika Asnina, Janis Osis, Janis Silins: *Formalization of Software Development: Problems and Solutions,* Scientific Proceedings of Riga Technical University, Series, Computer Science (5), Applied Computer Systems, Volume 22, Riga, RTU, 2005, pp. 204 – 216
6. Zinovy Diskin, Boris Kadish, Frank Piessens, Michael Johnson: *Universal Arrow Foundations for Visual Modeling.* Proc. Diagramms'2000: 1st Int. Conference on the theory and application of diagrams, Springer LNAI, 2000, No. 1889, pp. 345-360
7. The Free Dictionary Com: *Causality,* http://encyclopedia.thefreedictionary.com/ causality
8. OMG: *UML Extension for Business Modeling,* version 1.1, http://umlcenter.visual-paradigm.com/umlresources/exte_11.pdf