e-Informatica Software Engineering Journal, Volume 2, Issue 1, 2008

Computation Independent Representation of the Problem Domain in MDA

Janis Osis*, Erika Asnina*, Andrejs Grave*

*Faculty of Computer Science and Information Technology, Institute of Applied Computer Systems, Riga Technical University

janis.osis@cs.rtu.lv, erika.asnina@cs.rtu.lv, andrejs.grave@cs.rtu.lv

Abstract

The object-oriented analysis suggests semiformal use-case driven techniques for problem domain modeling from a computation independent viewpoint. The proposed approach called Topological Functioning Modeling for Model Driven Architecture (TFMfMDA) increases the degree of formalization. It uses formal mathematical foundations of Topological Functioning Model (TFM). TFMfMDA introduces more formal analysis of the problem domain, enables defining what the client needs, verifying textual functional requirements, and checking missing requirements in conformity with the domain model. A use case model of the application to be build is defined from the TFM using a goal-based method. Graph transformation from the TFM to a conceptual model enables definition of domain concepts and their interrelation. This paper also outlines requirements to the tool to support TFMfMDA.

1 Introduction

The purpose of this work is to introduce more formalism into the problem domain modeling within OMG *Model Driven Architecture*® (*MDA*®) [19] in object-oriented software development. The main idea is to introduce a more formal definition of consistency between real world phenomena and an application that will work within these phenomena without introducing complex, hard to understand mathematics used while composing *Computation Independent Models (CIMs)*. For that purpose, formalism of a *Topological Functioning Model (TFM)* is used [22]. A TFM provides a **holistical** representation of system's **complete functionality** from the computation-independent viewpoint.

This paper is organized as follows. Section 2 describes related work. Section 3 describes key principles of *MDA*, and discusses suggested solutions of computation independent modeling and their weaknesses in the object-oriented analysis within MDA. Section 4 discusses a developed approach, i.e. *Topological Functioning Modeling for Model Driven Architecture (TFMfMDA)*, that makes it possible to use a formal model, i.e. a TFM, as a computation independent one without introducing complex mathematics. Besides that, it allows verifying of functional requirements at the beginning of analysis. TFMfMDA is illustrated by an application example in Section 5. Section 6 shows TFMfMDA conformity to the MDA Foundation Model. Section 7 describes requirements to the tool that should partially support automation of TFMfMDA. Conclusions state further directions of the research.

2 Related Work

Our work completely supports Jackson's work, which states that "...the principal parts of a software development problem are the machine, the problem world, and the requirements..." [15]. We also assume that the first step in the requirements gathering should be analysis of the "problem world" or "business" [10]. Therefore within TFMfMDA, the TFM describes functionality of the "problem world", while requirements describe functionality of the solution.

Analysis of the "business" context is also understood in goal-oriented requirements gathering approaches. Unfortunately, most of them are solution-orientated. Successful exceptions are KAOS methodology that analyzes the "problem world" and deals with conflicts by global representation of goals and agents [7], the i* modeling framework that investigates agents that are assumed to be strategic and whose intentionality are only partially revealed [24], and, in some degree, the Requirement Abstraction Model [13] that links product requirements to organization's strategies. However, all these approaches operate rather with organization's strategic goals than with organization's functionality.

3 Construction of the CIM within MDA

Within MDA, the CIM usually includes several distinct models that describe system requirements, business processes and objects, an environment the system will work within, etc. Object-oriented analysis (OOA) is a semiformal specification technique that contains three steps: a) use case modeling; b) class modeling, and c) dynamic modeling. Use case usage is not systematic in comparison with systematic approaches that enable identifying of system requirement majority. Creation of use case models and determination of concepts and concept relations usually are rather *informal* than semiformal. Figure 1 shows several of the existing approaches of creating the mentioned models. Some approaches apply assisting questions [16, 18], category lists of concepts and concept relations (or noun-verb analysis) [17], or goals [6, 18] in order to identify use cases and concepts from the description of the system (in the form of informal description, expert interviewing, etc.). Other approaches draft a system requirements specification using classical requirements gathering techniques. Then these requirements are used for identification of use cases and creation of conceptual models. The most complete way is identification of use cases and concepts having knowledge of the problem world as well as a system requirements specification [2].

Use case modeling starts with some initial estimation (a tentative idea) about where the system boundary lies. For example, in the Unified Process [2], use cases are driven by requirements to the solution (but the business model is underestimated, and, thus, system boundaries are being identified intuitively), any requirement gathering technique can be applied, and requirements traceability to use cases is *ad hoc* defined. The B.O.O.M. approach [23] uses business-scope and system-scope use cases to make the solution more consistent with the problem world. The business-scope use cases are used as a requirements gathering technique. Unfortunately, they are IT project driven not business

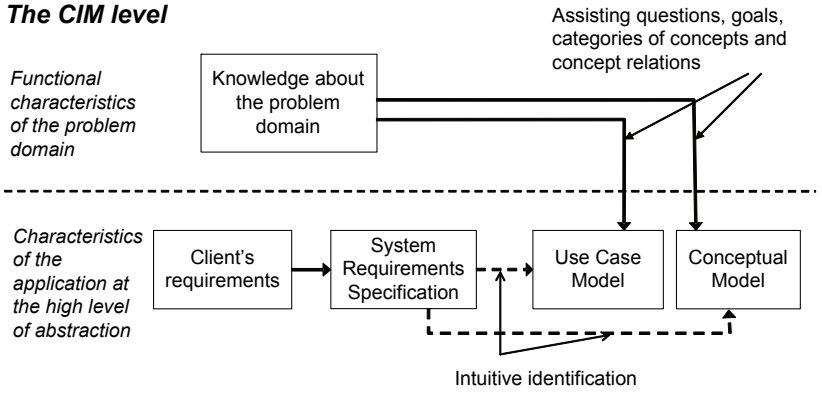

Figure 1: The current state of creation of the CIM in OOA

driven. This means that analysis of the existing and planned business logic is also solution-oriented. Besides that, the traceability between system-scope use cases and business-scope use cases is captured with use-case packages that have their bottlenecks (intuitive and ad hoc creation; changes in business processes cannot be traceable in a natural way, etc.). Alistair Cockburn's approach [6] structures use cases with goals at different abstraction levels: system scope, goal specification, and interaction details. Despite benefits of such structuring, this approach also does not have proper problem domain analysis, and the multilevel character of the technique is not easy for everyone.

This means that the priority of problem domain modeling is very low. Thus, system functioning and its structure are based on intuitive understanding of the environment the system will work within. Until now use cases relate to the narrow area, where the real world interacts directly with the system (*the solution*), and, hence, focuses requirement analyst's attention on events that happen within *the solution* boundaries, but the properties of the surrounding real world can remain underestimated, e.g., software system requirements can conflict with rules that exist in the organization. Besides that, fragmentary nature of use cases does not give any answer on questions about: a) identifying all of the use cases for the system; b) conflicts among use cases; c) gaps that can be left in system requirements; d) how changes can affect behavior that other use cases describe [10, 11]. Use case checklists cannot completely help here, because reviews of lists of use cases are made only based on knowledge of the solution domain without formal connection to system's functionality in the problem world.

We consider that understanding and modeling the problem domain should be the primary stage in the software development, especially in case of embedded and complex business systems, which failure can lead to huge losses. This means that use cases must be applied as *a part of* a technique, whose first activity is construction of a well-defined problem domain model. Such an approach - *Topological Functioning Modeling for Model Driven Architecture (TFMfMDA)* is suggested in this paper. This research can be considered as a step towards MDA completeness and, therefore, towards MDA maturity.

4 Topological Functioning Modeling for MDA

This section discusses the proposed TFMfMDA approach. TFMfMDA main steps illustrated by bold lines in Figure 2 are discussed further in the paper. The approach is based on the formalism of a Topological Functioning Model and uses some capabilities of universal category logic [4, 3, 22].

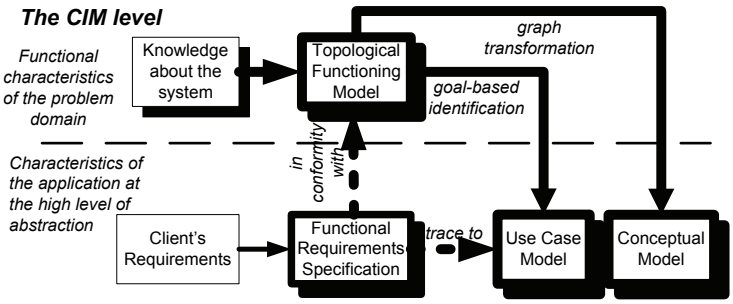

Figure 2: Creation of the CIM using TFMfMDA

As previously discussed, there are two interrelated branches at the beginning of system analysis: The first one is analysis of the problem world (the business or enterprise level), and the second one is analysis of the possible solution (the application level). Having knowledge about the complex system that operates in the real world, a topological functioning model of this system could be composed (Figure 2). This composed TFM is used to verify functional requirements and may be partially changed by them. TFM functional features are associated with business goals of the system; this provides identification of business-scope use cases as well as system-scope use cases in conformity with problem world's actualities. As a result, functional requirements are not only in conformity with the business-scope system's functionality but also can be traceable to the system-scope use case model. Problem domain concepts are selected and described in UML Class Diagram.

The TFM has a rigor mathematical base. It is represented in the form of 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 the form of a directed graph. "In combinatorial topology, the goal is to represent a topological space as an union of simple pieces. The word 'combinatorial' is used to suggest that the properties of the topological space rely on how the simple pieces are arranged. A graph is a simple combinatorial topological space." [5]. The necessary condition for construction of the topological space is a meaningful exhaustive verbal, graphical, or mathematical description of the system. The adequacy of the model describing functioning of a system can be achieved by analyzing mathematical properties of such an abstract object [22].

A TFM has as topological properties, namely, *connectedness, closure, neighborhood*, and *continuous mapping*, as functional properties, namely, *cause-effect relations, cycle structure, inputs* and *outputs*. These properties set model capabilities such as formal separation of subsystems, formal abstraction and refinement of the TFM, and analysis of similarities and differences of functioning systems. The last point relates to the structure

of cycles in the TFM. It is proved that every business and technical system is a subsystem of its environment. The common characteristic of functionality of all systems (technical, business, or biological) is a **main feedback circuit**, whose visualization is **an oriented cycle**. Therefore, topological modeling states that at least one directed closed loop must be in every topological model of system functioning. This cycle visualizes the "main" functionality that has vital importance to the system's life. Usually feedback is expressed as an expanded hierarchy of cycles. Therefore, proper analysis of cycles is mandatory in composing the TFM, because it supports careful analysis of system's operation and interaction with its environment [21]. Composition of the TFM is discussed in Section 4.1.

4.1 Construction of the Topological Functioning Model

This section discusses construction of the TFM that represents the problem world in business context (Figure 3). Its steps illustrated in Figure 4 are the following: a) Definition of physical or business functional characteristics, b) Introduction of the topology, and c) Separation of the TFM.

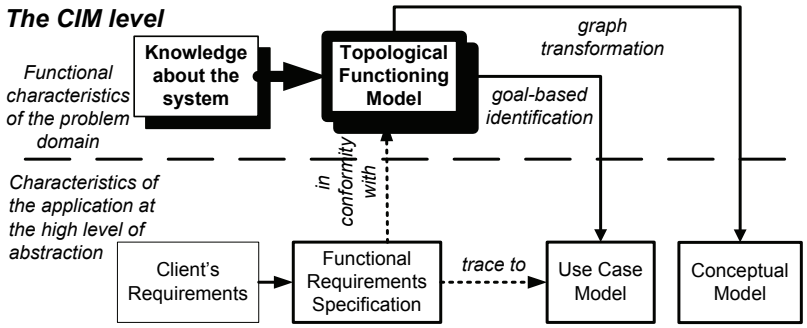

Figure 3: Construction of the TFM within TFMfMDA

Definition of physical or business functional characteristics consists of the following activities: 1) Definition of objects and their properties from the description of the problem world is performed by noun analysis, i.e. by establishing as meaningful nouns and their direct objects as handling synonyms and homonyms; 2) Identification of external systems (objects that are not subordinated to the system rules) and partially-dependent systems (objects that are partially subordinated to the system rules, e.g. workers' roles); and 3) Definition of functional features is performed by verb analysis, i.e. by founding meaningful verbs in the description. Each functional feature is a unique tuple ¡*A, R, O, PrCond, E*¿, where A is an object action, R is a result of this action, O is an object (objects) that receives the result or that is used in this action (for example, a role, a time period, a catalog, etc.), $PrCond$ is a set $PrCond = \{c_1 \dots c_i\}$, where c_i is a precondition or an atomic business rule (optional), and E is an entity responsible for action performing. Each precondition and atomic business rule must be either defined as a functional feature or assigned to the already defined functional feature. Two forms of textual

descriptions are defined. The first is the more detailed form: *¡action¿-ing the ¡result¿ [to,into,in,by,of,from] a(n) ¡object¿, [PrCond,] E.* An example is "*Check*-ing *out* the *availability* of a *copy*, PrCond= {a valid reader account}, E= a librarian". The latter is the more abstract form: *¡action¿-ing a(n) ¡object¿, [PrCond,] E.* An example is "*Check*-ing *out* a *copy*, PrCond={a copy is available}, E= a librarian".

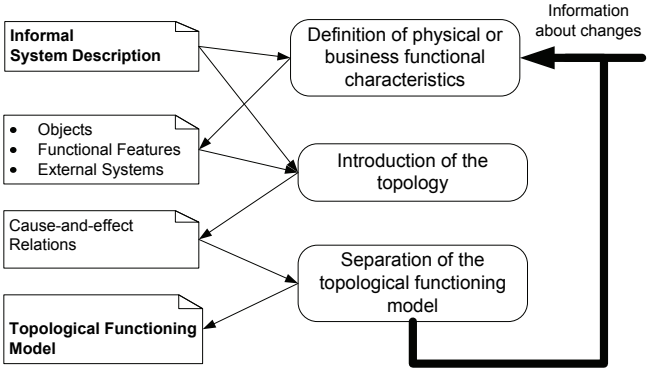

Figure 4: The method of construction of the TFM

Introduction of the topology Θ is the establishing of cause-effect relations between functional features. Cause-effect relations are represented as arcs of a digraph that are oriented from a cause vertex to an effect vertex. A structure of such relations can form a causal chain, wherein each relation is important.

Morevoer, cause-effect relations can form cycles. Therefore, cause-effect relations should be carefully checked whether they form **cycles** or **subcycles** in order to completely identify existing functionality of the system. The main cycle (cycles) of system functioning (i.e. functionality that is vitally necessary for system life) must be found and analyzed before starting further analysis. In case of studying a complex system, a TFM can be separated into a series of subsystems according to identified cycles.

Separation of the topological functioning model is performed by applying the closure operation over a set of system's inner functional features [22]. A *topological space* is a system represented by $Z = N \bigcup M$. Where N is a set of system's inner functional features, and M is a set of functional features of other systems interacting with the system or those of the system itself, which affect external systems. The TFM (X, Θ) is separated from the topological space of the problem world by the closure operation over the set N as it is shown by the equation $X = [N] = \bigcup_{\eta=1}^{n} X_\eta$. 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, whose each neighborhood 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 on ends of output arcs from x. Moreover, a TFM can be separated into a series of subsystems by the closures of chosen subsets of N. The closure is illustrated in Section 5.

4.2 Functional Requirements Conformity to the TFM

The next step is verification of functional requirements (hereafter: requirements) whether they are in conformity with the constructed TFM. TFM functional features specify functionality that *exists in the problem world*, and functional requirements specify functionality that *must exist in the solution* [14]. Thus, it is possible to map requirements onto TFM functional features (Figure 5).

Mappings are specifyed using arrow predicates. An arrow predicate is a construct borrowed from the universal categorical logic. Universal categorical (arrow diagram) logic for computer science was explored in detail in Zinovy Diskin's et al. work [8].

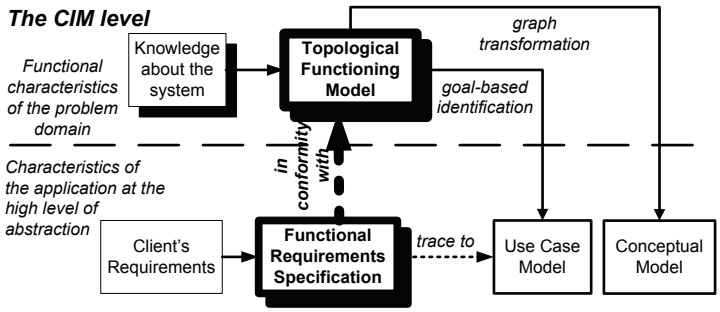

Figure 5: Making functional requirements in conformity with the TFM

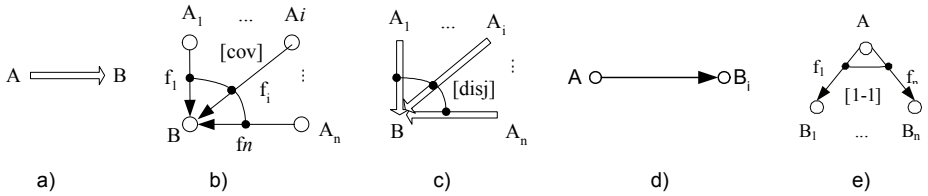

Figure 6: Functional requirements mapping onto TFM functional features

Within TFMfMDA, five types of mappings together with corresponding arrow predicates are defined. **One to One.** *Inclusion predicate* (Figure 6a) is used if the requirement A completely specifies what will be implemented in accordance with the functional feature B. **Many to One.** *Covering predicate* (Figure 6b) is used if the requirements $A_1, A_2, \ldots, A_n$ overlap the specification of what will be implemented in accordance with the functional feature B. In case of the covering requirements, their specification should be precised. *Disjoint (component) predicate* (Figure 6c) is used if the requirements $A_1, A_2, \ldots, A_n$ together completely specify the functional feature B and do not overlap each other. **One to Many.** *Projection* (Figure 6d) is used if some part of the functional requirement A incompletely specifies the functional feature B_i. *Separating family of functions* (Figure 6e) is used if one requirement A completely specifies several functional features $B_1, \ldots, B_n$. It can be because: a) the requirement joins several ones and can be split up, or b) the functional features are more detailed than the requirement. **One to**

Zero. One requirement specifies new or undefined functionality. In this particular case it is necessary to define possible changes of the problem domain's functioning (see Figure 4 "Information about changes"). **Zero to One.** The requirements specification does not contain any requirement related to the defined functional feature. This means that it can be a missed requirement and, hence, it could be not implemented in the application. Thus, it is mandatory to take a decision about implementation of the discovered functionality together with the client.

The result of this activity are both verified requirements and the TFM, which describes needed (and possible) functionality of the system and its environment.

4.3 Construction of the Use Case Model

The next step is transition from the model of the problem world constrained by the requirements to the use case model, supporting the possibility of more formal tracing of requirements to use cases (Figure 7).

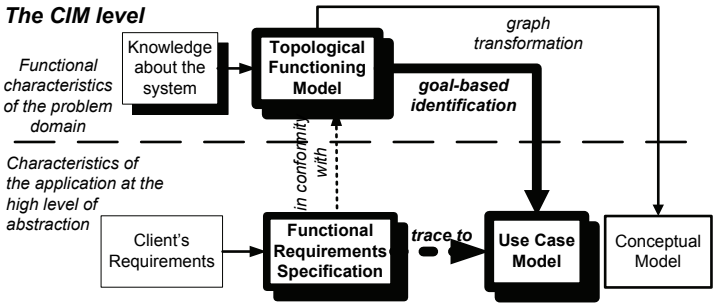

Figure 7: Construction of the use case model within TFMfMDA

This activity includes the following steps: a) Identification of system's users and their goals, b) Identification and refinement of system use cases, and c) Prioritization of use cases (and requirements).

Identification of system's users and their goals. At this stage, the TFM represents functionality of the problem world constrained by the requirements. System's users can be those, who interacts within the business system (workers) and with the business system (actors). *Actors* are external companies, clients, etc. *Workers* are system's *inner* entities (humans, roles, etc.) Identification of system users' *direct* **goals** is related to the identification of the corresponding set of functional features that are necessary for satisfaction of these goals. A goal as a means for identification of use case has been chosen because it can be achieved performing some process that can be long running. The time gap cannot do this. For each goal, an input functional feature (input transaction), an output functional feature (output transaction), and a functional feature chain between them can be defined. Both actors and workers can be users of the application. Identification of system-scope goals helps in verifying additional requirements, e.g., for discovering "missing" requirements.

Identification and refinement of system use cases. Functional features mapped by functional requirements that are grouped together by a goal describe functionality necessary for ***achievement of this goal***, and, hence, describe ***a system-scope use case***. System's users that establish the goal are (UML) actors that communicates with such use cases. This principle enables formal identification of a use case model from the TFM. However, this principle provides also additional possibilities for refinement of the system use cases. An *inclusion use case* is some common sequence for several use cases. In the TFM, it is an intersection of sets of functional features that belongs to more than one system goals. Each common functional feature must be analyzed. The common functional feature in the main flow of a use case is a candidate to an inclusion use case. An *extension use case* shows an alternative way of the scenarios execution. In the TFM, it is functional features in a sub-cycle or a branch, existing within the system goal. The point of branch beginning is an extending point. Identified use cases can be represented in UML Activity Diagram by transforming functional features into diagram's activities, and cause-effect relations into diagram's control flows.

Prioritization of use cases. Prioritization of use cases and, thus, functional requirements can be done in accordance with client's desires or using requirements attribute systems, e.g. MoSCoW or GRASP [2]. Within TFMfMDA, priorities of implementation of use cases are defined in conformity with the TFM main cycle as follows (in accordance with the Rational Unified Process): a) *critical (must be implemented otherwise the application will not be acceptable)* - if a use case implements any functional feature that belongs to the main functional cycle; b) *important (it would significantly affect the usability of the application)* - if a use case implements any functional feature that is a cause or an effect of a functional feature that belongs to the main cycle; and c) *useful (it has a low impact on the acceptability of the application)* - if a use case does not implement any functional feature of the main cycle or functional feature that affects or is affected by a functional feature that belongs to the main cycle.

4.4 Construction of the Conceptual Model

The last step of TFMfMDA is identification of the conceptual model. After requirements mapping, the TFM represents functionality that must be implemented in the application, and includes all concepts that are necessary for proper system's functioning (Figure 8a).

In order to obtain a conceptual model, it is necessary to detail each TFM functional feature to the level when it describes only objects of one type. This more precise model must be transformed one-to-one into a graph of domain objects. Then vertices with objects of the same type must be merged keeping all cause-effect relationships to graph vertices, which contain objects of other types (this is illustrated by the example in Section 5). The result is a graph of domain objects with indirect associations (Figure 8b). In order to make these relations more precise, the graph can be transformed into a sketch [8], then refined, and represented as a refined conceptual model. This transformation also indicates possible inheritance relations among types, and common operations, which can further be transformed into use case interfaces.

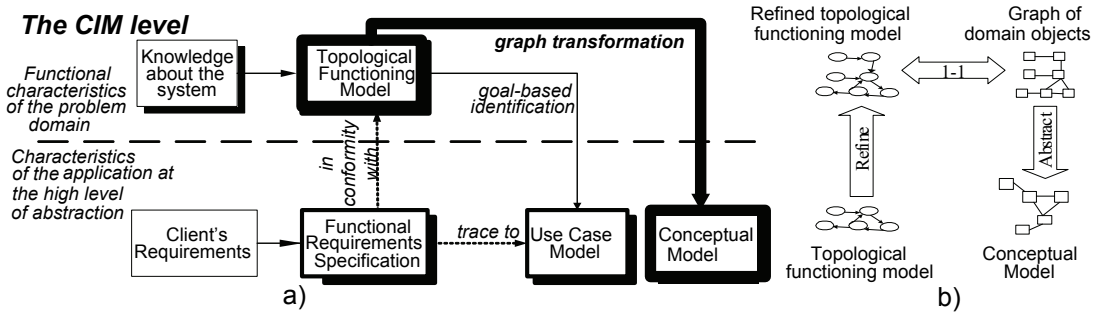

Figure 8: The step (a) and the process (b) of construction of the conceptual model

5 An Example of Application

This section gives an example of apllying TFMfMDA. Let us consider the *small fragment of an informal description* of the system from the project, within which the application for a library was developed. In this fragment, nouns are denoted by *italic*, verbs are denoted by **bold**, and action pre- (or post-) conditions are underlined.

"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*, hi **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*."

Construction of the TFM. The identified objects (or concepts) are the following: a) inner objects are a librarian (L), a book copy (a synonym is a book), a reader account, a reader card, a request for a book, a fine, a loan term, a statement of utilization, book fund, and b) external objects are a person (P), a reader (R), and an utilizer (U).

The identified functional features are represented as *¡number: a description of the functional feature, a precondition, a repsonsible entity and subordination¿*, where "In" denotes "inner", and "Ex" denotes "external" subordination. They are the following: 1: Arriving a person, {}, P, Ex; 2: Creating a reader account, {unergistered person}, L, In; 3: Creating a reader card, {}, L, In; 4: Giving out the reader card to a reader, {}, L, In; 5: Getting a reader status, {}, R, Ex; 6: Completing a request for a book, {}, R, In; 7: Sending a request for a book, {}, L, In; 8: Checking out the book copy from a book fund, {}, L, In; 9: Checking out the book copy to a reader, {completed request AND book copy is available}, L, In; 10: Giving out a book copy, {}, L, In; 11: Getting a book copy, {}, R, Ex; 12: Returning a book copy, {}, R, Ex; 13: Tacking back a book copy, {}, L, In; 14: Checking the term of loan of a book copy, {}, L, In; 15: Evaluating the condition of a book copy, {}, L, In; 16: Imposing a fine, {the loan term is exceeded OR the lost book

OR the damaged book}, L, In; 17: Returning the book copy to a book fund, {}, L, In; 18: Paying a fine, {imposed fine}, R, In; 19: Closing a fine, {paid fine}, L, In; 20: Completing a statement of utilization, {hardly damaged book copy}, L, In; 21: Sending the book copy to Utilizer, {}, L, In; 22: Utilizing a book copy, {}, U, Ex.

In order to define system's functionality - the set X, we perform the closuring operation over the set of system's inner functional features N = {2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20}. The set of external functional features and system's functional features that affect the external systems $M = \{1, 4, 5, 18, 21, 22\}$. The neighborhood of each element of the set N is as follows: $X_2 = \{2, 3\}$, $X_3 = \{3, 4\}$, $X_6 = \{6, 7\}$, $X_7 = \{7, 17\}$, $X_8 = \{8, 9\}$, $X_9 = \{9, 10\}$, $X_{10} = \{10, 11\}$, $X_{11} = \{11, 5\}$, $X_{12} = \{12, 13\}$, $X_{13} = \{13, 14\}$, $X_{14} = \{14, 15, 16\}$, $X_{15} = \{15, 16, 17, 20\}$, $X_{16} = \{16, 19\}$, $X_{17} = \{17, 8\}$, $X_{19} = \{19\}$, $X_{20} = \{20, 21\}$. The obtained set is X ={2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21}.

The identified cause-effect relations between the functional features are illustrated in Figure 9a. The main functional cycle is defined by an expert and includes the following functional features "17-8-9-10-11-5-12-13-14-15-17". It is denoted by bold lines in Figure 9a. These functional features describe checking out and taking back a book. They are assumed to be main, because have a major impact on business system's operation. The example of the first order subcycle is "5-6-7-17-8-9-10-11-5".

Functional requirements conformity to the TFM. Let us assume that the drafted functional requirements (FR) are as follows. **FR1:** The system shall perform registration of a new reader; **FR2:** The system shall perform check out of a book copy; **FR3:** The system shall perform check in of a book copy; **FR4:** The system shall perform imposing of a fine to a reader; and **FR5:** The system shall perform handling of an unsatisfied request (the description: the unsatisfied request should be added to the wait list; when a book copy is returned to the book fund, the system checks what request can be satisfied and, in success, informs the readers by SMS).

FR1 maps onto the functional features 2, 3, and 4, i.e. FR1={2, 3, 4}; FR2={7, 8, 9}, FR3={13, 14, 15, 17}, FR4={16}. The functional requirement FR5 describes new functionality that must be implemented in the application and introduced in the business activities of the system. System's functionality described in the TFM by the functional features 18, 19, 20, and 21 is not specified by requirements. This means that more careful analysis of the requirements and problem world is needed, because they can be missed. The better way in this situation is to specify these features in the requirements specification (and as use cases). The final decision must be taken together with the client that is warned beforehand about possible negative aftereffects. In this context, the interesting one is the functional feature 19, which describes closing of an imposed fine. It should be implemented. Therefore, FR4 is modified as *"The system shall perform imposing and closing of a fine to a reader"*. Hence, FR4={16, 19}.

The new functionality introduced by FR5 can be described by new identified objects (the system, a wait list and SMS), and the following functional features - **23:** Adding the request_for_a_book in a wait list, {unavailable book}, L, In; **24:** Checking the request_for_a_book in a wait list, {a book copy is returned to the book fund}, system, In;

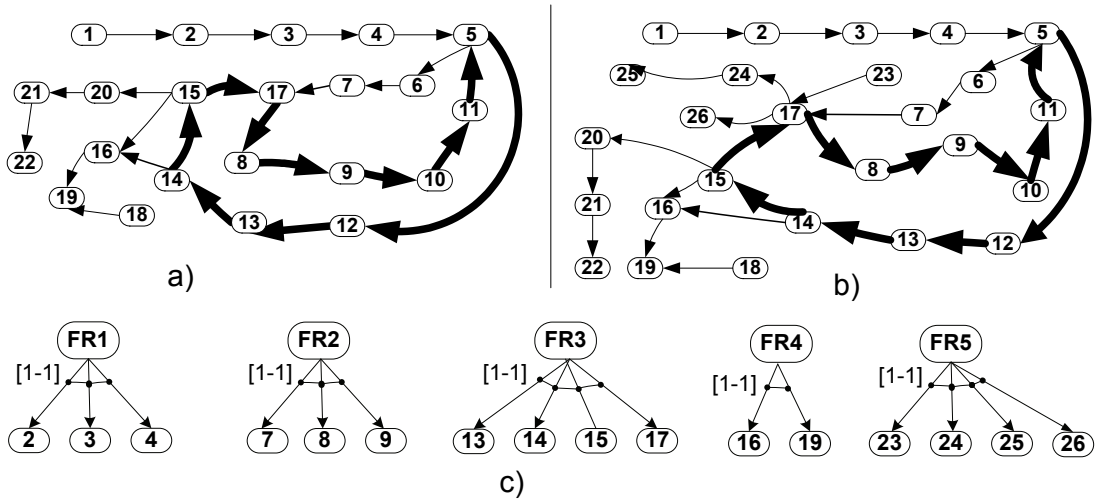

Figure 9: The topological space (a) and the modified topological space (b) of the library functioning; the correspondence between requirements and TFM functional features (c)

25: Informing the reader by SMS, {a request in the wait list can be satisfied}, system, In;
26: Avoiding a request for a book, {book copy is not available}, system, In.

Introducing this functionality into the TFM, we must recheck all the existing cause-effect relations between the previously identified functional features taking into account possible changes in causes and effects. The set $N = \{2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 23, 24\}$. The set $M = \{1, 4, 5, 18, 21, 22, 25, 26\}$. After the closuring, the set $X = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 23, 24, 25, 26\}$. The result is the model represented in Figure 9b. The final correspondence between the functional features and requirements is illustrated in Figure 9c. All the identified mappings of the requirements onto the functional features have the type "one-to-many".

Construction of the use case model. In order to define use cases, system's users and their goals together with necessary functional features are identified. System's users (Librarian, and System) are transformed into UML actors, goal names into use case names, and functional features into steps of the corresponding use cases. The resulting use case model, functional features to be implemented, and implementation priorities of use cases defined accordingly to TFM functioning cycles are illustrated in Figure 10a. Figure 10b shows how two of the use cases can be described in UML Activity Diagram using information from the TFM, where functional features are transformed into activities, but cause-effect relations into control flows.

Construction of the conceptual model. The step of the TFM refinement is skipped, because each functional feature takes a deal with objects of the only one type. Figure 11 shows transformation of the TFM to the graph of domain objects. Additionally, Figure 12*a* reflects this graph after the gluing all graph vertices that represent functional features with objects of the same types. This reflects the idea proposed in [20, 21, 22] that the holistic representation of the domain by means of the TFM enables identifying

of all necessary domain concepts, and, even, enables defining their necessity for successful implementation of the system.

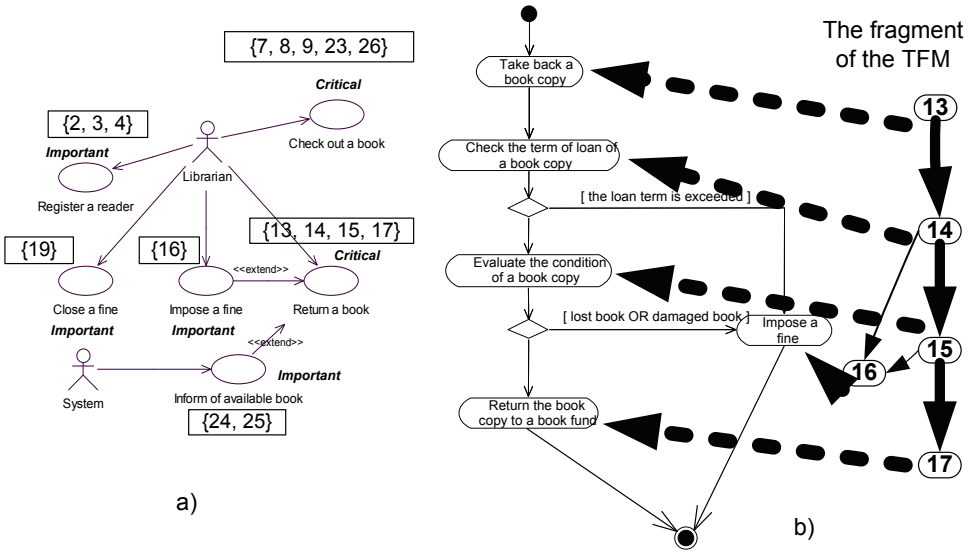

Figure 10: The use case model (a), and the fragment of the TFM described in UML Activity Diagram that specifies functionality of use cases "Return a book" and "Impose a fine" (b)

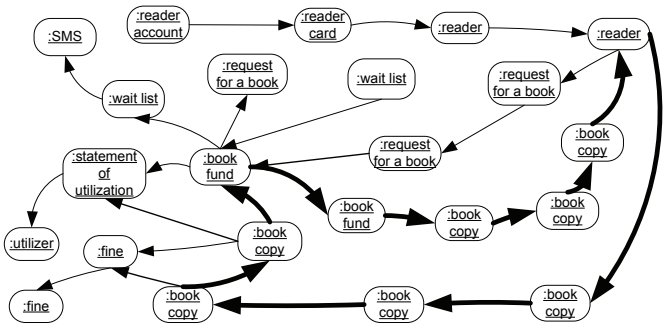

Figure 11: The graph of types of domain objects

6 MOF-based Metamodel of TFMfMDA

In compliance with [1], the Foundation Model of MDA requires that a metamodel of each modeling language used within MDA must be defined in Meta Object Facility (MOF) terms for conformance purposes. Therefore, a metamodel of TFMfMDA concepts was defined as well as an UML profile for TFMfMDA [4].

The MOF is a core standard of MDA. Its architecture has four *metalevels*. They are named M3, M2, M1 and M0 [12]. Conceptually the level M3 is the MOF itself, i.e. a set of constructs used to define metamodels. M2 describes instances of constructs from M3.

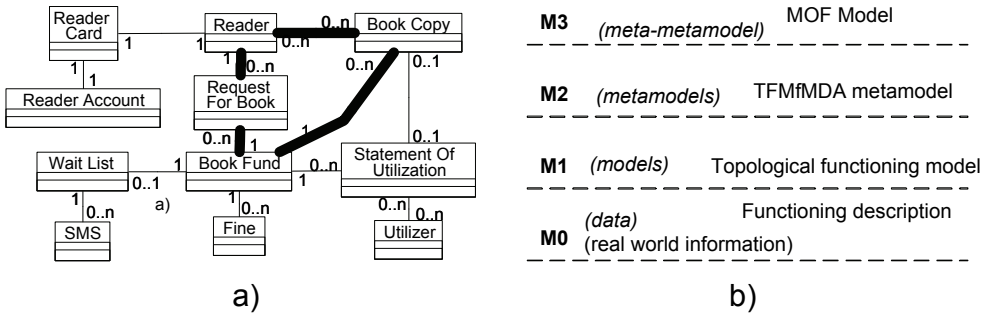

Figure 12: The initial conceptual model (a), TFMfMDA at the MOF metalevels (b)

M1 includes instances of metamodel constructs from M2. Finally, the level M0 describes objects and data that are instances of elements from M1. TFMfMDA constructs are made in conformity with these metalevels as illustrated in Figure 12*b*. The metamodel for TFMfMDA is described at the level M2 [22]. These metamodel illustrated in Figure 13 specifies how TFMfMDA concepts related to each other.

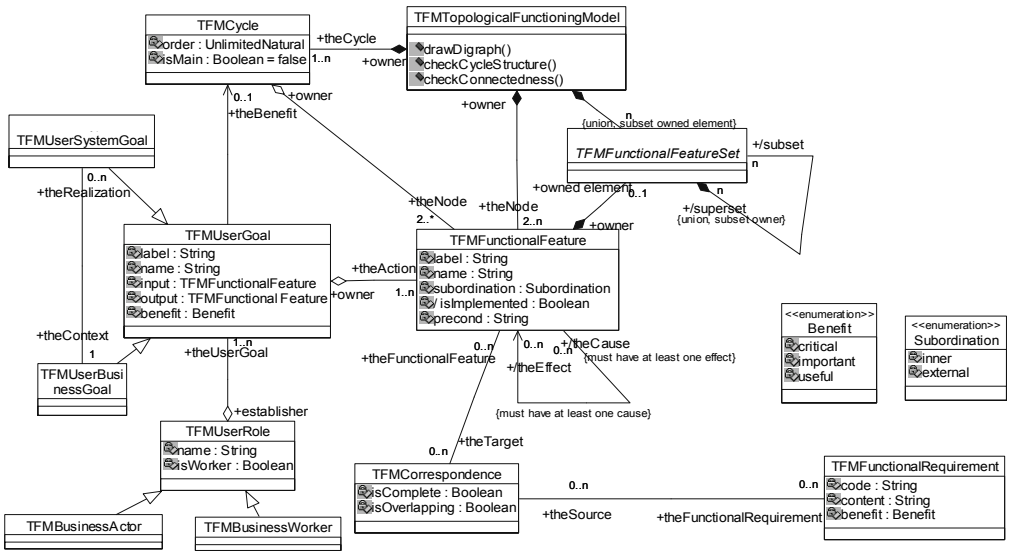

Figure 13: The MOF-based metamodel of TFMfMDA

A topological functioning model is an instance of the type *TFMTopologicalFunctioningModel* that includes at least two functional features of the type *TFMFunctionalFeature*. They can be united in functional feature sets (*TFMFunctionalFeatureSet*). This means that a functional feature represented in the TFM can visualize a functional feature set. One functional feature can contain only one set and one functional feature can belong only to the one set. A functional feature can be subordinated to a business system itself or to an external system (*Subordination*). Functional features can form functioning

cycles (*TFMCycle*) of different order. Functional features are connected by cause-effect relations. A causal functional feature must have at least one effect. An effect functional feature must have at list one cause. Functional features are mapped by functional requirements (*TFMFunctionalRequirement*) via the correspondence (*TFMCorrespondance*). The correspondence is many to many in general. It can be complete or incomplete, overlapping or disjoint. Functional features can be associated with several goals (*TFMUserGoal*) that are established by direct users (*TFMUserRole*) of the business system. The users can be external entities that interact with the business system (*TFMBusinessActor*) or workers that interact within the business system (*TFMBusinessWorker*). A user goal can be specialized to a business goal (*TFMUserBusinessGoal*) and to a system goal (*TFMUserSystemGoal*). The latter includes functional features to be implemented. This means that it includes functionality that is specified in the functional requirements specification. A user goal and, thus, corresponding functional requirements, are associated with functioning cycles, whose order affect a benefit value (*Benefit*) of implementing requirements.

7 Requirements to the Tool to Support TFMfMDA

As previously mentioned, TFMfMDA introduces certain formalism into the problem domain modeling from the computation independent viewpoint. Unfortunately, a use of complex graph-based constructs requires additional efforts. Therefore, the main purpose of the TFMfMDA tool is model management, which relates to model verification, traceability handling, automation of TFMfMDA steps, etc. This section discusses the requirements to the tool for TFMfMDA support.

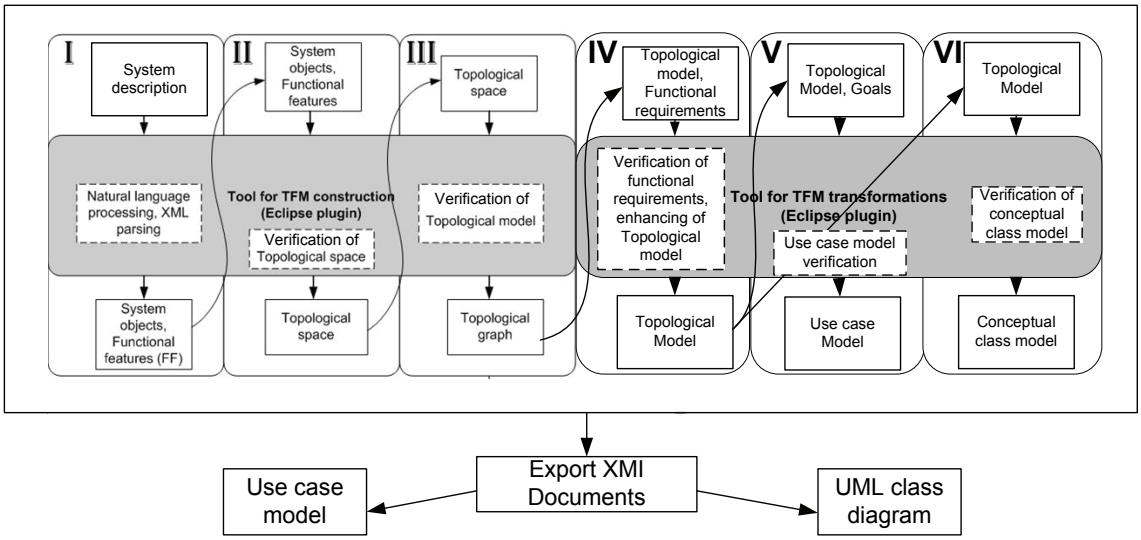

Figure 14: The general scheme of the tool supporting TFMfMDA

The tool should support the client-server architecture. In case of the client-server architecture, the server should keep information of models; the client part should enable the connection with the server and use of the kept information. The tool should be realized as an Eclipse plug-in [9]. Eclipse is an open development platform that consists of different components, which helps in developing Integrated Development Environments (IDEs). For implementation of the tool the following Eclipse components can be used: Workbench UI, Help system, and Plug-in Development Environment (PDE). The Workbench UI is a component that is responsible for plug-in integration with Eclipse User Interface (UI). It defines extension points, using which a plug-in can communicate with the Eclipse UI. Help System is a component that provides complete integration of help information into the Eclipse help system. PDE is the environment that enables automation of activities related to the plug-in development.

The tool should enable work with textual information (an informal description of the system, a description of functional requirements) and graph-based constructs (a TFM, a conceptual model, and a use case model). All changes must be propagated automatically to all the related models. A general scheme of tool's activities is illustrated in Figure 14. The scheme describes TFMfMDA steps considered above in this paper. The first three steps reflects construction of the TFM. The fourth step reflects check of functional requirements and activities of enhancing the TFM. The fifth step illustrates creation of the use case model. Additionally, the sixth step shows composing of the conceptual model.

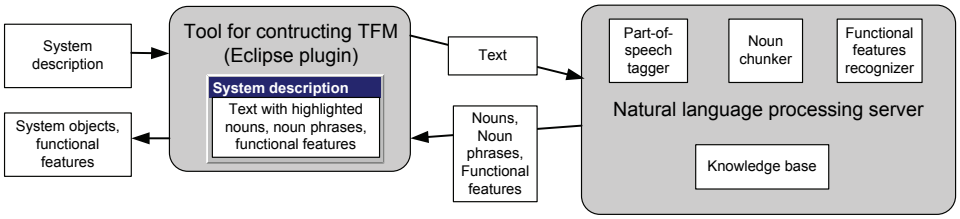

Figure 15: Handling the informal description of the system

The challenge is realization of work with informal descriptions (Figure 15). The informal text should be handled on the server side because of several causes, namely, using of the knowledge base, the multi-user environment, and "learning" possibilities of the tool. The server side should support detection of nouns, noun phrases, and verbs. The detected information should be sent to the client side in XML file form. On the client side, it can be highlighted to the user in different ways (different colors, fonts, etc.). The tool must provide convenient interface for handling this information and creating TFM functional features.

Introduction of the topology between TFM functional features should be realized as a mix of graphical and textual representations of the functional features. The tool should offer a user to union or split up functional features, and to define cause-effect relations among them using tabular representations, but the result should be also represented in the graph form.

The TFMfMDA tool must provide a separate editor for each step. Each editor should have related views that help to represent information actual in this step for a user. All automated steps that require human participation should be realized as wizards.

8 Conclusions

The paper discusses about TFMfMDA and its application to certain formalism introducing in the process of creation of the CIM. TFMfMDA specifies complex systems using graph constructs and their transformations. Note that formal transformations of graphs are not limited with the number of vertices in graphs. The number of graph vertices can be decreased using formal abstraction of the graph. The primary goal of TFMfMDA is to specify functionality of the system in the problem domain. Certainly, the careful modeling of the problem domain requires additional expenses, but further it will be worthwhile, because it gives the formal CIM, decreases further expenses as decrease the number of development iterations, and facilitates change implementation.

TFMfMDA application has the following advantages. First, careful cycle analysis can help in identifying all (possible at that moment) functional and causal relations between objects in complex business systems. Implementation priorities of requirements can be set not only in accordance with client's whishes, but also in accordance with functioning cycles of the TFM. The latter makes it possible to take a decision about change acceptability in functionality of the problem domain before implementation of the changes in the application, and helps to check completeness of functional requirements. Second, TFMfMDA solves some use case limitations using formal mathematical means, e.g., it provides use case completeness, avoids conflicts among use cases, and shows their affect on each other. Besides that it does not limit a use of any requirements gathering techniques.

The tool built accordingly to the requirements would partially automate TFMfMDA steps described above. However, TFMfMDA requires human participation, thus, the further research is related to enhancing TFMfMDA with capabilities of natural language handling in order to make it possible to automate more steps of TFMfMDA and to decrease effect of human participation in decision making.

References

[1] A proposal for an MDA foundation model. ORMSC White Paper ormsc/05-04-01, OMG, www.omg.org/docs/ormsc/05-04-01.pdf, Apr 2005. V00-02.

[2] J. Arlow and I. Neustadt. *UML2 and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley, Pearson Education, second edition, 2005.

[3] E. Asnina. Formalization aspects of problem domain modeling within model driven architecture. In O. Vasilecas, editor, *Databases and Information Systems. 7th International Baltic Conference on Databases and Information Systems. Communications, Materials of Doctoral Consortium*, pages 93–104, Vilnius, Lithuania, 2006. Vilnius Gediminas Technical University, Technika.

[4] E. Asnina. *Formalization of Problem Domain Modeling within Model Driven Architecture*. PhD thesis, Riga Technical University, RTU Publishing House, Riga, Latvia, 2006.

[5] W. F. Basener. *Topology and Its Applications*. John Wiley and Sons, Inc., New Jersey, USA, 2006. p. 339.

[6] A. Cockburn. Structuring use cases with goals. http://alistair.cockburn.us/crystal/articles/sucwg/.

[7] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *The Science of Computer Programming*, 20(November):3–50, 1993.

[8] Z. Diskin, B. Kadish, F. Piessens, and M. Johnson. Universal arrow foundations for visual modeling. In *Proc. Diagramms'2000: 1st Int. Conference on the theory and application of diagrams*, pages 345–360. Springer LNAI, 2000. No. 1889.

[9] Eclipse. Eclipse - an open development platform. http://www.eclipse.org.

[10] S. Ferg. What's wrong with use cases? http://www.ferg.org/papers/, Feb 2003.

[11] D. Firesmith. Use cases: the pros and cons. http://www.ksc.com/article7.htm.

[12] D. Frankel. *Model Driven Architecture : Applying MDA to Enterprise Computing*. Wiley Publishing, Inc., Indiana, 2003.

[13] T. Gorschek and C. Wohlin. Requirements abstraction model. *Requirements Engineering*, 11:79–101, 2006.

[14] M. Jackson. The real world. http://www.ferg.org/papers/, Jul 2003.

[15] M. Jackson. Problem frames and software engineering. *Information and Software Technology*, 47(November):903–912, 2005.

[16] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.

[17] C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall PTR, 3rd edition, 2005.

[18] D. Leffingwell and D. Widrig. *Managing Software Requirements: a use case approach*. Addison-Wesley, 2nd edition, 2003.

[19] OMG, http://www.omg.org/. *MDA Guide Version 1.0.1*, Jun 2003.

[20] J. Osis. Extension of software development process for mechatronic and embedded systems. In *Proceeding of the 32nd International Conference on Computer and Industrial Engineering*, pages 305–310. University of Limerick, Limerick, Ireland, Aug 2003.

[21] J. Osis. 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*, pages 211–220, Riga, Latvia, 2004. Riga Technical University, RTU.

[22] J. Osis. Formal computation independent model within the MDA life cycle. In *International Transactions on Systems Science and Applications*, pages 159–166. Xiaglow Institute Ltd, Glasgow, UK, 2006. ISSN 1751-1461, V. 1, Nr. 2.

[23] H. Podeswa. *UML for the IT Business Analyst: A practical Guide to Object-Oriented Requirements Gathering*. Thomson Course Technology PTR, Boston, 2005.

[24] E. S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *International Symposium on Requirements Engineering*, pages 226–235, Annapolis, Maryland, 1997.