Enterprise Modeling for Information System Development within MDA

Janis Osis Erika Asnina
Faculty of Computer Science and Information Technology, Institute of Applied Computer Systems, Riga Technical University, Latvia
janis.osis@cs.rtu.lv erika.asnina@cs.rtu.lv

Abstract

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 mathematical foundations of Topological Functioning Model (TFM) that holistically represents complete functionality of the system. TFMfMDA introduces more formal analysis of a business system, namely enables defining not what the client wants, but what the client needs. Additionally, it also enables verification of textual functional requirements, checking of missing requirements in conformance with the "as is" model of the problem domain. A use case model of the application is defined with the help of a goal-based method. A model of domain concepts is defined by graph transformation of the TFM. TFMfMDA satisfies the main feature of MDA – Separation of Concerns.

1. Introduction

This work continues research on computation independent modeling started in [1], [2], [3], and [4]. The main idea of the given work is to introduce more formalism into problem domain modeling from a computation independent viewpoint within OMG Model Driven Architecture (MDA) [5] in object-oriented development. For this purpose formalism of a Topological Functioning Model (TFM) is used [1]. The TFM is an expressive and powerful instrument for a clear presentation and formal analysis of functionality of a system and the environment the system works within. The TFM is a base of the suggested approach, i.e., *Topological Functioning Modeling for Model Driven Architecture* (TFMfMDA). TFMfMDA satisfies the main feature of MDA – Separation of Concerns. It distinguishes two kinds of computation-independent information, i.e. information of a problem domain captured in experts' knowledge and information of the corresponding application captured in requirements. TFMfMDA provides functional requirements to be in compliance with functionality of the system under consideration at the very beginning of analysis. It makes it possible to use a formal TFM as a computation independent model without introducing complex mathematics.

This paper is organized as follows. Section 2 describes key principles and suggested solutions of computation independent modeling and its weaknesses in the object oriented analysis (OOA) within the MDA. Section 3 discusses steps of TFMfMDA. This section draws attention to the TFMfMDA activities described in brief in [4] using a detailed example, in which functionality of a library is modeled. Subsection 3.1 described how experts' knowledge about functionality of the problem domain can be represented and analyzed by a TFM. Subsection 3.2 considers a way to check how functional requirements to the system reflect functionality of the problem domain. Subsection 3.3 and Subsection 3.4 illustrated how the obtained knowledge can be analyzed and represented in user-friendly notation – a use case model and an initial model of classes. Moreover, those activities help in solving some limitations of use cases. Subsection 3.5 describes in brief appropriateness of TFMfMDA to MDA. Conclusions establish further directions of the research.

2. Construction of the CIM within the Model Driven Architecture

The main purpose of the MDA is to separate viewpoints in specifications and strengthen the role of analysis and design in the project development. MDA authors foresee three specification viewpoints and their corresponding models. First, a Computation Independent Model (CIM) represents system requirements and the way in which the system works

1530-1605/08 $25.00 © 2008 IEEE

within the environment without details of the system's structure and application implementation. Second, a Platform Independent Model (PIM) describes a system at such level of abstraction that it renders this model to be suitable for use with different platforms of similar type. And third, a Platform Specific Model (PSM) provides a set of technical concepts, representing different kinds of parts that make up a platform and its services to be used by an application, and, hence, does change transferring system functioning from one platform to another. The conception of the MDA supports abstraction and refinement in models [5]. *Models can be described in any modeling language defined in accordance with the MDA Foundation Model* [6]. Models are obtained by transformations: PIM-to-PIM, PIM-to-PSM, PSM-to-PSM, and PSM-to-PIM. Each transformation must be recorded in a record of transformation [5].

However, lack of traceability within the CIM, and lack of transformation from the computation independent viewpoint to the platform independent one are underestimated. This does not promote elimination of the gap between business and application domains.

There are two fundamental aspects for system modeling that need to be distinguished: analysis, which defines what an application has to do with a problem domain to fit customer's requirements, and design, which defines how the application will be built. The analysis implies that a software developer should analyze client's current situation in as much detail as possible, and gather requirements of the system planned to build. Usually, requirement analysis is assumed to be determination of what software the client wants. Indeed, *requirements must determine what software the client needs*. Moreover, they must determine not just what software the client needs, but also within what environment this software will work.

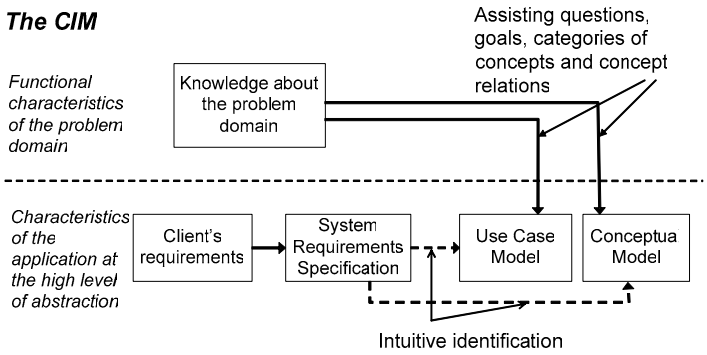

Figure 1. The current state of creation of the CIM in OOA

Within the MDA, the CIM usually includes several distinct models that describe system requirements, business processes and objects, the 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. Traditional OOA approaches define a CIM using some mixture of information from both sides of the dashed line (fig.1). Use cases are rather weak formalized approach that fragmentary describes the application domain. Their usage is not systematic in comparison with systematic approaches that allow identifying all system requirements. Creation of use case models and determination of concepts and relations between them usually is rather informal than semiformal. Figure 1 shows several existing ways of creating these models. One way is to apply assisting questions [7], [8], categories of concepts and concept relations [9], or goals [10], [8] in order to identify use cases and concepts from the description of the system (in the form of informal description, expert interviewing, etc.). Another way is to draft a system requirement specification using some requirement gathering technique. Then these requirements are used for identification of use cases and creation of a conceptual model [11]. The most complete way is identification of use cases and concepts while having knowledge of the problem domain as well as a system requirement specification [12].

Use cases describe the problem domain as a "black box"; moreover, the priority of problem domain modeling is very low. Thus, system functioning and its structure are based on an intuitive understanding of the environment the system will work within. Until now use cases relate to a narrow area, where the real world interacts directly with the system (the application), and, hence, focuses requirement analyst's attention on events that happen within the application boundaries, but the properties of the surrounding real world can remain underestimated. Besides that, their fragmentary nature does not give any answers to the questions

about: a) identifying all of the use cases to 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 [13], [14].

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

This research can be considered to be a step towards the completeness of the MDA.

3. Topological Functioning Modeling for Model Driven Architecture

Since an informal specification often has ambiguous parts, it is more difficult to detect errors and subsequently correct them. Precision of a formal specification means that even if such a specification is not what the customer wanted, it is easier to tell where it is incorrect and improve it. Additionally, missing parts of an incomplete specification become clearer. Besides that, the formal or executable nature of models makes the models benefit from automation.

This section discusses the proposed TFMfMDA approach [15], [2], and [1]; the main steps discussed further in the paper are illustrated by bold lines in Figure 2. The TFMfMDA uses some capabilities of the universal category logic and is based on the formalism of the Topological Functioning Model.

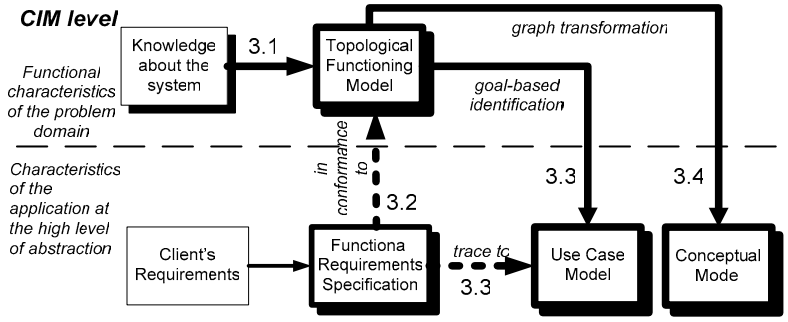

Figure 2. Creation of the CIM using TFMfMDA in the OOA with Separation of Concerns

As previously discussed, there are two kinds of information at the beginning of the problem analysis (the dashed line in Figure 2 shows Separation of Concerns): The first one is at the (business or enterprise) system level, and the second one is at the application level. The main idea is that functionality determines the structure of the planned system (Figure 2). Having knowledge about a complex system that operates in the real world, a topological functioning model of this system can be composed (see Section 3.1). This means that a TFM of the system is tested and can be partially changed by functional requirements (see Section 3.2). Then TFM functional features are associated to business goals of the system; this provides business use case as well as system use case identification according to the problem domain realities. Moreover, after those activities functional requirements are not only in conformance with the business system functionality but can also be traced back to the system use case model (see Section 3.3). Problem domain concepts are selected and described in an UML Class Diagram (see Section 3.4). All these steps are illustrated by the example given further.

The TFM has a solid mathematical base. It is represented in a form of a topological space (X,Θ), where X is a finite set of functional features of the system under consideration, and Θ is the topology that satisfies axioms of topological structures and is represented in a form of a directed graph. The necessary condition for constructing a topological space is a meaningful and exhaustive verbal, graphical, or mathematical system description. The adequacy of a model describing the functioning of a concrete system can be achieved by analyzing mathematical properties of such abstract object [1].

A TFM has topological (connectedness, closure, neighborhood, and continuous mapping) and functional (cause-effect relations, cycle structure, and inputs and outputs) characteristics. It is acknowledged that every business and technical system is a subsystem of the environment. Besides that a common thing for all system (technical, business, or biological) functioning should be the main feedback, visualization of which is

an oriented cycle. Therefore, it is stated that at least one directed closed loop must be presented in every topological model of system functioning. It shows the "main" functionality that has a vital importance in the system's life. Usually it is even an expanded hierarchy of cycles. Therefore, a proper cycle analysis is necessary in construction of the TFM, because it enables careful analysis of system's operation and communication with the environment [3].

3.1. Construction of the topological functioning model

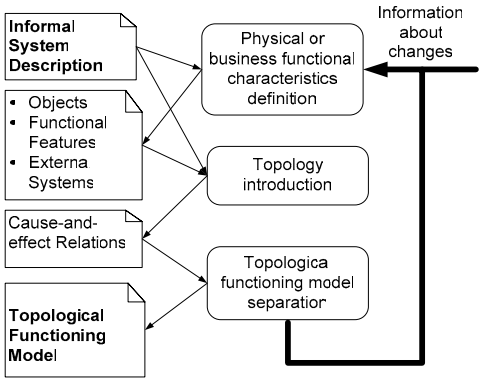

Figure 3. Construction of the TFM

This section discusses the construction of the topological functioning model for problem domain modeling in business system context. It consists of the following steps (Figure 3): 1) Definition of physical or business functional characteristics; 2) Introduction of topology; and 3) Separation of the topological functioning model.

For a better understanding let us consider the following small fragment of an informal description from the project, in which a library application is developed.

"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*."

Step 1: Definition of physical or business functional characteristics consists of the following activities: 1) Definition of objects and their properties from the problem domain description that is performed by noun analysis, i.e., by establishing meaningful nouns and their direct objects and handling synonyms and homonyms; 2) Identification of external systems (objects that are not subordinated to the system's rules) and partially-dependent systems (objects that are partially subordinated to the system's rules, e.g., system workers' roles); and 3) Definition of functional features using verb analysis in the problem domain description, i.e., by finding meaningful verbs.

In the fragment nouns are denoted by *italic*, verbs are denoted by **bold**, and action pre- (or post-) conditions are underlined. The identified objects (or concepts) are the following: a) inner objects are a librarian, a book copy (the synonym is a book), a reader account, a reader card, a request for a book, a fine, a loan term, a statement of utilization, a book fund, and b) external objects are a person, a reader, and an utilizer.

Within the TFMfMDA each functional feature is a 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 $PrCond = \{c_1, ..., c_i\}$, where c_i is a precondition or an atomic business rule (it is an optional parameter), and E is an entity responsible for performing actions. Each precondition and atomic business rule must be either defined as a functional feature or assigned to an already defined functional feature. Two textual description forms are defined. The more detailed form is as follows:

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

And the more abstract form is the following:

<action>-ing a(n) <object>, [PrCond,] E

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"), subordination (where "in" – inner, "ex" – external)" and they are as follows: **1:** Arriving [of] a person, {}, person, ex; **2:** Creating a reader account, {unregistered person}, Lib, in; **3:** Creating a reader card, {}, Lib, in; **4:** Giving out the card to a reader, {}, Lib, in; **5:** Getting the status of a reader, {}, R, ex; **6:** Completing a request_for_book, {}, R, in; **7:** Sending a request_for_book, {}, R, in; **8:** Taking out the book copy from a book fund, {}, Lib, in; **9:** Checking out a book copy, {completed request, book copy is available}, Lib, in; **10:** Giving out a book

copy, {}, Lib, in; **11:** Getting a book copy [by a registered reader], {}, R, ex; **12:** Returning a book copy [by a registered person], {}, R, ex; **13:** Taking back a book copy, {}, Lib, in; **14:** Checking the term of loan of a book copy, {}, Lib, in; **15:** Evaluating the condition of a book copy, {}, Lib, in; **16:** Imposing a fine, {the loan term is exceeded, the lost book, or the damaged book}, Lib, in; **17:** Returning the book copy to a book fund, {}, Lib, in; **18:** Paying a fine, {imposed fine}, R, in; **19:** Closing a fine, {paid fine}, Lib, in; **20:** Completing a statement_of_utilization, {hardly damaged book copy}, Lib, in; **21:** Sending the book copy to a Utilizer, {}, Lib, in; **22:** Utilizing a book copy, {}, utilizer, ex.

Step 2: Introduction of topology Θ means establishing cause and effect relations between functional features. Cause-and-effect relations are represented as arcs of a digraph that are oriented from a cause vertex to an effect vertex.

The identified cause-and-effect relations between the functional features are illustrated by the means of the TFM in Figure 4(a). Figure 5(a) clearly shows that cause-and-effect relations form functioning cycles. All cycles and subcycles should be carefully analyzed in order to completely identify existing functionality of the system. *The main cycle (cycles)* of system functioning (i.e. functionality that is vital for system's life) must be found and analyzed before starting further analysis. In the case of studying a complex system, a TFM can be divided into a series of subsystems according to the identified cycles.

The example represents the *main functional cycle* defined by the expert, which includes the following functional features *"17-8-9-10-11-5-12-13-14-15-17"* and is denoted by bold lines in Figure 4(a). These functional features describe checking out and taking back a book. These are assumed to be the main, because they have a major impact on the operation of the business system. A cycle that includes the functional features "5-6-7-17-8-9-10-11-5" illustrates an example of the first-order subcycle.

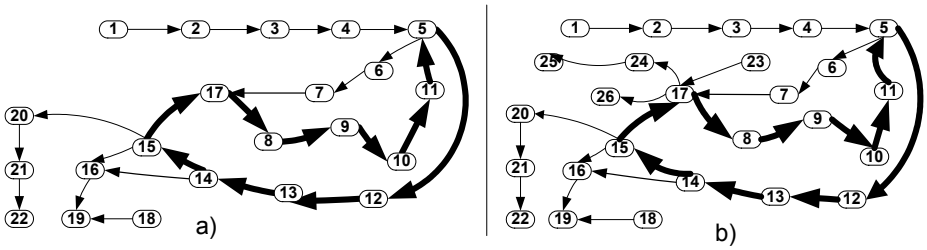

Figure 4. Both topological space (a) and modified one (b) of the library functioning

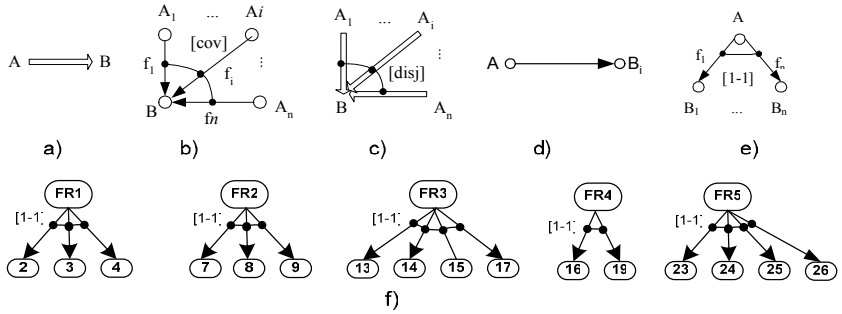

Figure 5. Functional requirement mappings onto the TFM functional features

Step 3: Separation of the topological functioning model is the same as in the TFM approach, i.e., it is performed by applying the closure operation over a set of system's inner functional features [1]: A topological space is a system represented by Equation (1). Where N is a set of inner system functional features and M is a set of functional features of other systems that interact with the system or of the system itself, which affect the external ones.

$$Z = N \cup M \quad (1)$$

$$X = [N] = \bigcup_{\eta=1}^{n} X_\eta \quad (2)$$

A TFM ($X \in \Theta$) is separated from the topological space of a problem domain by the closure operation over the set N as it is shown by Equation (2). 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 example illustrates how we perform the closuring operation (2) over the set *N* in order to get all of the system's functionality – the set *X*. The set of the 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 functional features that affect the external environment *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}, and 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}.

3.2. Functional requirement conformity to the TFM

The next step is the verification of functional requirements (hereafter requirements) in conformance with the constructed topological functioning model. Functional features (hereafter features) specify functionality that exists in the problem domain, and requirements specify functionality that must exist in the application. Thus, it is possible to map requirements onto functional features of a TFM.

Mappings are described with arrow predicates. An arrow predicate is a construct borrowed from the universal categorical logic. Universal categorical (arrow diagram) logic for computer science is explored in detail in [16].

Within TFMfMDA, five types of mappings and corresponding arrow predicates are defined. **One to One:** *An inclusion predicate* in Figure 5(a) is used if the requirement *A* completely specifies what will be implemented in accordance with the feature *B*. **Many to One:** *A covering predicate* in Figure 5(b) is used if requirements (A_1, A_2,..., A_n) overlap the specification of what will be implemented in accordance with the feature *B*. In case of the covering requirements, their specification should be refined. *A disjoint (component) predicate* in Figure 5(c) is used if requirements (A_1, A_2,..., A_n) together completely specify the feature *B* and do not overlap each other. **One to Many:** *Projection* in Figure 5(d) is used if some part of the requirement *A* incompletely specifies some feature B_i. *Separating family of functions* in Figure 5(e) is used if one requirement *A* completely specifies several features (B_1, ..., B_n). It can be because: a) the requirement joins several requirements and can be split up or b) features are more detailed than the requirement. **One to Zero:** One requirement specifies some new or undefined functionality. In this particular case it is necessary to define possible changes in the problem domain functioning. **Zero to One:** Specification does not contain any requirement corresponding to the defined feature. This means that it could be a missed requirement, and therefore it could be left unimplemented in the application. Thus, it is mandatory to take a decision about the implementation of the discovered functionality together with the client. As a result of this activity, both checked requirements and TFM, which describes a needed, possible functionality of the system and the environment it operates within, are obtained.

For instance, let us assume that the drafted requirements in our example are as follows. **FR1:** The system shall perform a registration of a new reader; **FR2**: The system shall perform a check-out of a book copy; **FR3**: The system shall perform a check-in of a book copy; **FR4**: The system shall perform an imposition of a fine to a reader; and **FR5**: The system shall perform a 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 which requests can be satisfied and if successful informs the readers via an SMS).

The requirements map onto the features as follows: "FR1"= {2, 3, 4}; "FR2"= {7, 8, 9}; "FR3"= {13, 14, 15, 17}; "FR4"= {16}; but "FR5" describes new functionality that must be implemented in the application and introduced in the business activities of the system. Existing system functionality described in the topological functioning model by the functional features 18, 19, 20, and 21 is not specified by functional requirements. *This means that a more careful analysis is needed, because they can be missing requirements. A better way in this situation is to specify them in the requirement specification (and as a use case). The final decision must be taken together with the client who is warned beforehand about possible negative aftereffects*. In this context, the interesting one is the functional feature 19, which describes closure of an imposed fine. It should be implemented. Therefore, the functional requirement *FR4 is modified as "The system shall perform an imposition and closure of a fine to a reader"*. Then, "FR4"= {16, 19}.

The new functionality introduced by the functional requirement FR5 can be described by new identified objects (the system, the wait list and an SMS), and the

following functional features: **23:** Adding the request for a book in a wait list, {unavailable book}, Lib, in; **24:** Checking the request for a book in a wait list, {a book copy is returned to the book fund}, system, in; **25:** Informing the reader via an 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.

When introducing this functionality into the topological functioning model, we must recheck all the existing cause-and-effect relations between the previously identified functional features taking into account new functional features and possible changed ones 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 closure, 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 resulting model is presented in Figure 4(b).

The final correspondence between functional features and functional requirements is illustrated in Figure 5(f). All identified mappings of functional requirements onto the functional features have the type "one–to–many".

3.3. Construction of a use case model

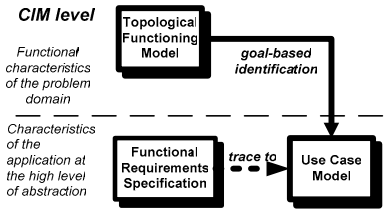

Figure 6. Construction of a use case model

The next step is transition from an initial problem domain model to a CIM "output" model, i.e., obtaining a use case model, and possibility of a more formal tracing of functional requirements to use cases (Figure 6). This activity includes the following steps: 1) Identification of business system users and their goals; 2) Identification and refinement of system's use cases; and 3) Use case (and requirement) prioritizing.

Step 1: Identification of business system users and their goals is as follows. Business system users can be actors and workers [17]. In the TFM, actors are represented as an external system functionality or functional properties of the system under consideration that interact with external systems (in this case, their identification is necessary), e.g., external companies, clients, etc. Workers are system's inner entities, e.g., humans, roles, etc.

Identification of business system users' direct goals is related to the identification of the corresponding set of functional features that are necessary for the goal satisfaction. 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. Business actors as well as business workers can be users of the application. System (application) goal identification helps for additional requirement checking, i.e., for discovering "missing" requirements. A goal as the means for use case identification has been chosen because a goal can be achieved by performing some process that can be long running. The "time gap" concept cannot do it.

The example illustrates that business users who communicate with the business system are entities responsible for execution of functional features. By looking up functional feature descriptions, the following business users are defined: a person (P), a librarian (L), a reader (R), a system (S), and a utilizer (U). The utilizer only gets the result of the library functioning. However, if we consider those of them who are responsible for execution of functional features specified by the functional requirements (i.e., those who are system users), then they are a librarian (L) and the application (S) itself. Table 1 shows their business goals and functional features they use for reaching goals.

Step 2: Identification and refinement of system's use cases is as follows. Functional features that are specified by functional requirements and that are needed to achieve a business goal describe the achievement of the corresponding system goal, and, therefore, a system use case. A business system user that established this goal is an (UML) actor that communicates with this use case. This principle enables formal identification of a use case model from the topological functioning model.

However, this principle also provides additional possibilities for the refinement of the system's use cases.

Table 1. Business users, their goals, and TFM functional features

Business actor	Goal name	Business worker	Business goal	Functional features	System goal	Functional features
P	Register a reader	L	BG1	2, 3, 4	SG1	2, 3, 4
R	Check out a book	L	BG2	6, 7, 8, 9, 23, 26	SG2	7, 8, 9, 23, 26
R	Return a book	L	BG3	13, 14, 15, 16, 17, 24, 25	SG3	13, 14, 15, 16, 17, 24, 25
R	Pay a fine	L	BG4	18, 19	SG4	19
	Impose a fine	L	BG5	16	SG5	16
	Close a fine	L	BG6	19	SG6	19
	Utilize a book	L	BG7	20, 21	SG7	----------
R	Inform of available book	S	BG8	24, 25	SG8	24, 25

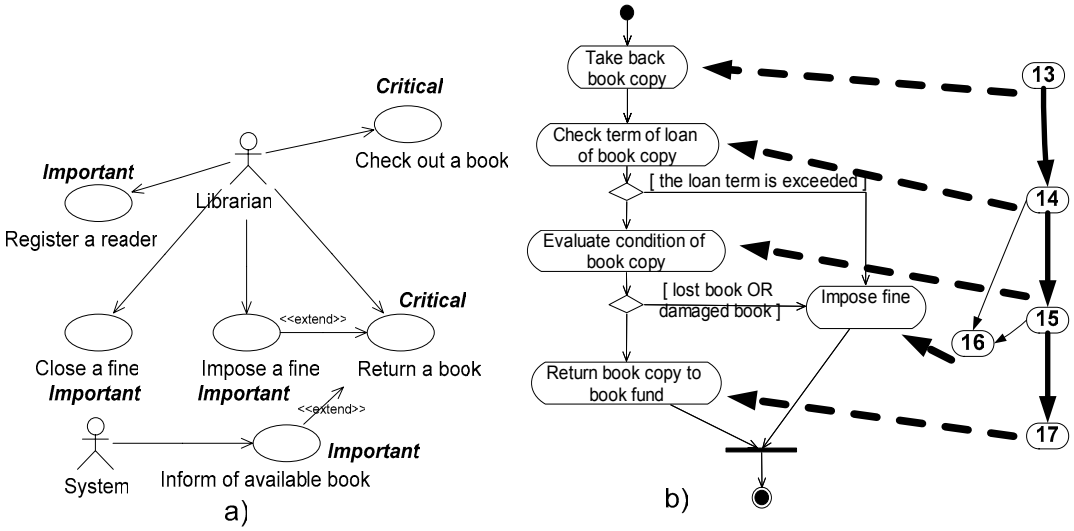

Figure 7. A use case model (a); use cases "Return book" and "Impose fine" in Activity Diagram (b)

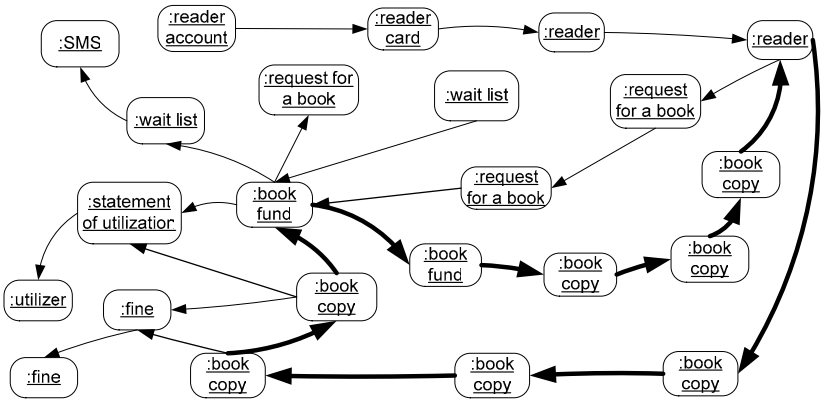

Figure 8. The graph of domain object types

An inclusion use case is an intersection of functional feature sets of several system goal areas in the TFM. The common functional features can be located either in the main flow or alternate flow (i.e., a TFM sub-cycle or a branch) of the use case. In the TFM, a sub-cycle or a branch, existing within the

system goal is *an extension use case* where an extending point is a start point of the branch. Identified use cases can be represented in an UML activity diagram by transforming functional properties into diagram activities and cause-and-effect relations into its control flows.

In our example, Table 1 illustrates that the system's goal SG4 is equal to the system goal SG6. Besides that, the intersections are SG3∩SG5 = SG5 = {16} and SG3∩SG8 = SG8 = {24, 25}. Hence, these common functional features are candidates for extension or inclusion use cases.

Business workers defined in Table 1 are transformed into actors in a use case model. A business goal name is a use case name. Functional features from the last column, i.e., those that must be implemented in the application in accordance with the requirements, are transformed into steps of the corresponding use case.

The resulting use case model is illustrated in Figure 7(a). This shows how a use case 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.

Step 3: Use case prioritizing is as follows. Use case and, thus, requirement prioritizing can be done in accordance with client's requirements or can be defined using some requirement attribute systems, e.g. MoSCoW or GRASP [12]. Within TFMfMDA, use case implementation priorities are defined in conformance with the main cycle as follows: a) critical (must be implemented otherwise the application will not be acceptable) – if a use case will implement 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 will implement 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).

A use case is useful if it will not implement any main cycle functional feature or a functional feature that affects or is affected by some functional feature that belongs to the main cycle. Figure 7(a) illustrates priorities defined for use cases of our example.

3.4. Construction of the initial conceptual class model

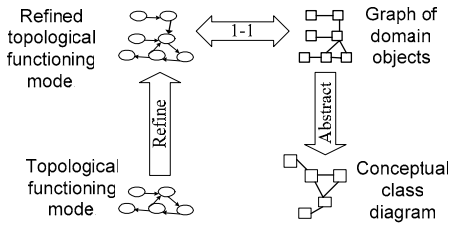

Figure 9. The process of construction of the conceptual model

The last step of TFMfMDA is the identification of a conceptual class model. After functional requirement mapping the topological functioning model also represents functionality that must be implemented in the application; in addition, it includes all concepts that are necessary for a proper functioning.

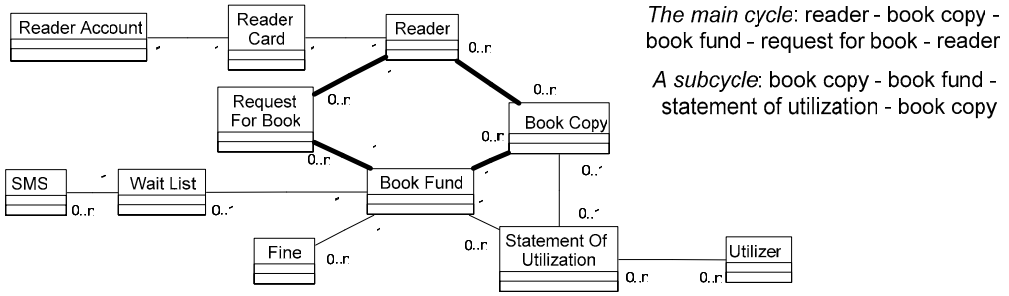

Figure 10. The initial model of conceptual classes

In order to obtain a conceptual class model, it is necessary to detail each functional feature of the TFM to a level where it uses only one type of objects. After that this more accurate model must be transformed one-to-one to a problem domain object graph and then the vertices with the same type of objects must be merged, while keeping all relations with other graph vertices. As a result, a conceptual class graph with indirect associations is defined (Figure 9). In order to make these relations more accurate, the graph can be transformed into a sketch, then refined, and represented as a refined conceptual class diagram. Such transformation also indicates possible inheritance

relations among types and common operations, which can be further transformed into use case interfaces.
Figure 8 shows the transformation of the topological functioning model to the graph of domain object types. Additionally, Figure 10 presents this graph after its abstraction, i.e., after gluing all graph vertices with the same object types. This reflects the idea proposed in [18], [3], and [1] that the holistic domain representation by the means of the TFM enables identification of all necessary domain concepts and, even, enables to define their necessity for a successful implementation of the system.

3.5. Appropriateness of TFMfMDA to MDA

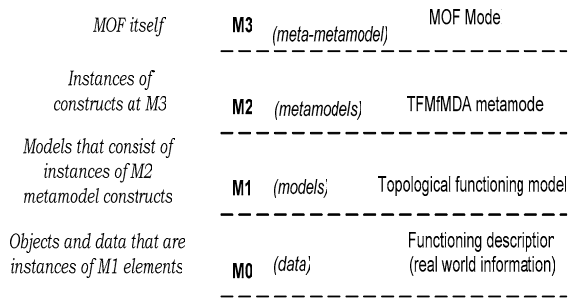

Figure 11. TFMfMDA at the MOF metalevels

In accordance with [6], a metamodel of TFMfMDA is described in terms of OMG Meta-Object Facility (MOF). The metamodel of TFMfMDA is situated at M2 metalevel as illustrated in Figure 11. The complete specification of the TFMfMDA metamodel and a UML profile is described in [15].

In brief, a metamodel of TFMfMDA describes classes, associations and attributes as follows. TFMTopologicalFunctioningModel is the class, whose instance is a topological functioning model that includes at least two functional features – instances of the class TFMFunctionalFeature. They can be united in functional feature sets (TFMFunctionalFeatureSet). 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 (a data type Subordination). Functional features can form functioning cycles (TFMCycle) of different order.

Functional features are connected by cause-and-effect relations. A cause functional feature must have at least one effect. An effect functional feature must have at list one cause. Functional features are related to functional requirements (TFMFunctionalRequirement) via some correspondence (TFMCorrespondance). The correspondence is many to many in general. The correspondence can be complete or incomplete, overlapping or disjoint.

The functional features can be associated with several goals (TFMUserGoal) that are established by direct users (TFMBusinessUser) 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 one (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, corresponded functional requirements are associated with a functioning cycle, whose order establishes their benefit value (a data type Benefit).

4. Conclusions

The application of TFMfMDA has the following advantages. It heightens the traceability within the CIM. Careful cycle analysis helps to identify all (at that moment possible) functional and causal relations between objects in complex business systems. Therefore, this makes it possible to make a decision about acceptability of changes in the problem domain functioning before realization of them. TFMfMDA helps to check completeness of functional requirements and does not limit the use of any requirement gathering techniques. It solves some limitations of use cases, namely, information capturing, thinking limitation, and completeness checking; this provides use case completeness, avoids conflicts among use cases, and shows their affect on each other. Implementation priorities of use cases (requirements) can be ordered not only in accordance with the client's wishes, but also in accordance with the functioning cycles, i.e. with the functionality that is vital necessary for basic operating of the (business) system in the real world.

The further research is related to enhancing of TFMfMDA with the capabilities of natural language handling in order to make it possible to automate more steps of this approach and to decrease human participation in decision making.

5. References

1. J. Osis, "Formal computation independent model within the MDA life cycle", *International Transactions on Systems Science and Applications*, Vol. 1, Nr. 2, Xiaglow Institute Ltd, Glasgow, UK, 2006, pp. 159-166.
2. E. Asnina, "Formalization Aspects of Problem Domain Modeling within Model Driven Architecture", *Databases and Information Systems, 7th International Baltic*

Conference on Databases and Information Systems, Communications, Materials of Doctoral Consortium, Vilnius: Technika, Vilnius, Lithuania, 2006, pp. 93-104.

3. J. 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*, Riga: RTU, Riga, Latvia, 2004, pp. 211 – 220.
4. J. Osis, E. Asnina, A. Grave, "Formal Computation Independent Model of the Problem Domain within the MDA", *Proceedings of the 10th International Conference on Information System Implementation and Modeling*, Hradec nad Moravicí, Czech Republic, April 23-25, 2007, pp. 47-54.
5. J. Miller, J. Mukerji (eds), *OMG: MDA Guide Version 1.0.1*, 2003, available at http://www.omg.org/docs/omg/03-06-01.pdf
6. *A Proposal for an MDA Foundation Model, ORMSC White Paper, V00-02, ormsc/05-04-01,* OMG, 2005, available at www.omg.org/docs/ormsc/05-04-01.pdf
7. I. Jacobson, M. Christerson, and P. Jonsson, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.
8. D. Leffingwell, D. Widrig, *Managing Software Requirements: a use case approach, 2nd ed.*, Addison-Wesley, 2003.
9. Cr. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd ed.,* Prentice Hall PTR, 2005.
10. A. Cockburn, "Structuring Use Cases with Goals", 2001, available at http://alistair.cockburn.us/crystal/articles/sucwg/structuringucwithgoals.htm
11. H. Podeswa, *UML for the IT Business Analyst: A practical Guide to Object-Oriented Requirements Gathering,* Thomson Course Technology PTR, Boston, 2005.
12. J. Arlow, I. Neustadt, *UML2 and the Unified Process: Practical Object-Oriented Analysis and Design,* Addison-Wesley, Pearson Education, 2005.
13. S. Ferg, "What's Wrong with Use Cases?", available at http://www.ferg.org/papers/ferg--whats_wrong_with_use_cases.html
14. Knowledge Systems Corporation, "Use Cases: the Pros and Cons", available at http://www.ksc.com/article7.htm
15. E. Asnina, "Formalization of Problem Domain Modeling within Model Driven Architecture", *PhD thesis*, RTU Publishing House, Riga, Latvia, 2006.
16. Z. Diskin, B. Kadish, F. Piessens, and M. Johnson, "Universal Arrow Foundations for Visual Modeling", *Proc. Diagramms'2000: 1st Int. Conference on the theory and application of diagrams. Springer LNAI, No. 1889*, 2000, pp. 345-360.
17. *UML Extension for Business Modeling, Version 1.1*, 1997, available at umlcenter.visual-paradigm.com/umlresources/exte_11.pdf
18. J. Osis, "Extension of Software Development Process for Mechatronic and Embedded Systems", *Proceeding of the 32nd International Conference on Computer and Industrial Engineering,* University of Limerick, Limerick, Ireland, 2003, pp. 305-310.