ISSN 1407-7493

COMPUTER SCIENCE
DATORZINĀTNE

2008-34

APPLIED COMPUTER SYSTEMS
LIETIŠĶĀS DATORSISTĒMAS

SUPPORT OF A USE-CASE CONTROLLER PATTERN BY TFM4MDA

LIETOŠANAS GADĪJUMU KONTROLLERU ŠABLONA ATBALSTS TFM4MDA PIEEJĀ

Erika Asnina, *Riga Technical University,*
Faculty of Computer Science and Information Technology,
Institute of Applied Computer Systems, Department of Applied Computer Science,
Meza str. 1/3, Riga, LV 1048, Latvia, Lecturer, Dr.sc.ing., erika.asnina@cs.rtu.lv

MDA, use case, controller, pattern, topological functioning model

1. Introduction

Model Driven Architecture (MDA) was developed in 2001 by the Object Management group (OMG). The main principle of MDA is an emphasis that is put on a model not on code. MDA supports the so called *Separation of Concerns* on a problem domain (where the problem domain is business that is planned to be modeled). This means that MDA models describe a system of the real world at different abstraction levels, namely, computation independent, platform independent and platform specific one. Correspondingly, MDA proposes the following models [1]:

- A *Computation Independent Model* (CIM) that describes system's structure and behavior without any computation, i.e., it shows requirements to the system, business vocabulary, etc.;
- A *Platform Independent Model* (PIM) that describes application's structure and behavior, but does not show any platform specific information;
- A *Platform Specific Model* (PSM) that describes application's structure and behavior together with added platform specific constructs.

CIMs are constructed from verbal descriptions of the system, e.g., interviews, business process descriptions, procedure descriptions, experts' knowledge and so on. PIMs are obtained by transformation of CIMs, and, analogously, PSMs are obtained by transformation of PIMs. MDA proposes different kinds of transformations from PIM to PIM, from PIM to

PSM, from PSM to PSM, and from PSM to PIM. A transformation from CIM to PIM is suggested to be intellectual (intuitive, manual) work. Note that a boarder between two models is no quite strict. Therefore, result computation independent models are input for transformation on the PIM level.

Topological Functioning Modeling for Model Driven Architecture (TFM4MDA) is an approach that suggests a solution to construction of the formal CIM [2]. The main results of application of TFM4MDA are input models for the PIMs, i.e. a use-case model and a concept model, as well as formal verification of system requirements both made in compliance with the formal model of the problem domain. Therefore, a transformation from CIM to PIM became more formal than it was before.

Model-driven development, especially the fact that a use of models is a core of such development, leads to build up a collection of both principles and idiomatic solutions that guide software developers in the creation of software. These principles and idioms, if codified in a structured format describing the problem and solution and named, are called *patterns*. As Craig Larman meant in [3] "a pattern is a named description of a problem and solution that can be applied to new contexts". Many patterns, given a specific category of the problem, guide the assignment of responsibilities to objects. The point of patterns is an attempt to codify existing tried-and-true knowledge, idioms, and principles; the more honed, old, and widely used, the better. There are many analysis and design patterns, for example, *General Responsibility Assignment Software Patterns* (GRASP) for basic patterns of assigning responsibilities [3], and *Gang-of-Four* (GoF) for more advanced design ideas [4]. GRASP patterns are used mostly at the level of PIMs and PSMs.

This paper discusses how TFM4MDA supports one of GRASP patterns, namely, Controller Pattern for use cases and makes it in conformity with the formal CIM. Section 2 describes the mentioned pattern and its necessity for modeling. Section 3 considers TFM4MDA in brief. Section 4 illustrates an example of TFM4MDA application and transformation to the GRASP Controller pattern. Section 5 concludes this discussion.

2. GRASP Controller

A popular way of thinking about the design of software objects is in terms of responsibility, roles, and collaborations. This is a part of a larger approach called *responsibility driven design* (RDD). In RDD, software objects are assumed as having responsibilities, i.e. an abstraction of what they have to do. Responsibilities are related to the obligations or behavior of a classifier in terms of its role. Basically, the responsibilities are of the following two types: *doing* and *knowing*.

The GRASP defines nine patterns. They are Controller, Creator, High Cohesion, Indirection, Information Expert, Low Coupling, Polymorphism, Protected Variations, and Pure Fabrications [3]. Let us consider the Controller pattern.

A simple layered architecture has a user interface (UI) layer and a (application's) domain layer, among others. Actors generate UI events. The UI software objects must then react to this event. Corresponding to the Model-View Separation Principle[1], the UI objects should *not* contain application's business logic. Therefore, once the UI objects pick up the event, they need to delegate (forward the task to another object) the request to *domain objects* in the

[1] The Model-View Separation principle states that model (business domain) objects should not have *direct* knowledge of view (UI) objects, at least as view objects [3].

domain layer. The Controller pattern deals with a basic question in object-oriented design: How to connect the UI layer to the application logic layer?

The Controller pattern offers the following advice [3]:

- *Problem:* What first object beyond the UI layer receives and coordinates ("controls") a system operation?
- *Solution (advice):* Assign the responsibility to an object representing one of these choices:
  - Represents the overall "system", a "root object", a device that the software is running within, or a major sub-system (these are all variations of a *facade controller*).
  - Represents a use case scenario within which the system operation occurs (a use case or *session controller*).

A common defect in the design of controllers results from over-assignment of responsibility. A controller then suffers from low cohesion.

3. Topological Functioning Modeling for Model Driven Architecture in a Nutshell

Informal specifications often have ambiguities, hence, it is more difficult to detect errors and subsequently correct them than in case of formal specifications. 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 incomplete specifications become clearer. On the other hand, the formal or executable nature of models makes them benefit from automation.

This section discusses Topological Functioning Modeling for MDA or TFM4MDA in brief. The more detailed description is given in [2], [5], [6]. TFM4MDA suggests consideration of information about the problem in two contexts – the first one is the business (or enterprise system's) context (a problem domain), and the second one is the application context (a solution). These contexts should be analyzed separately (Figure 1). The first idea is that the application context constrains the business context, not vice versa (fully satisfies [7]). And the second one is that functionality determines the structure of the planned system.

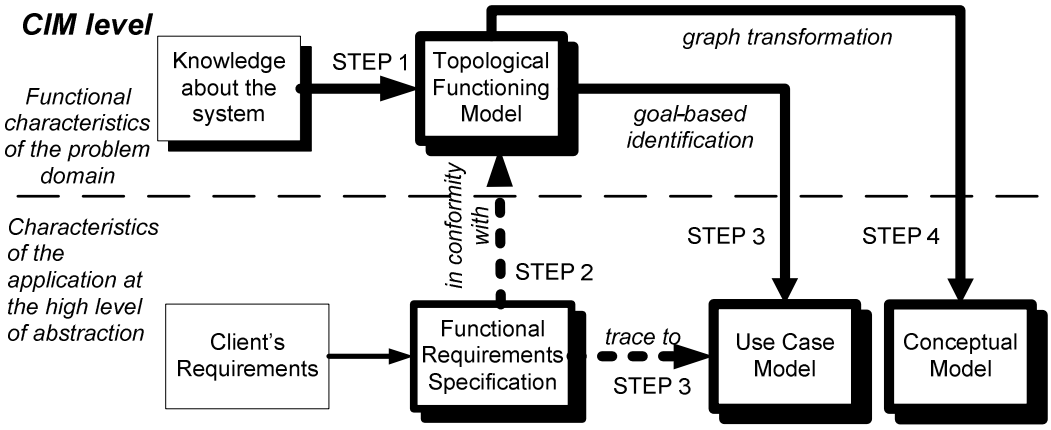


Figure 1. Creation of the CIM using TFM4MDA in object-oriented system analysis

3.1. The Topological Functioning Model in Brief

The TFM has a solid mathematical base. The TFM satisfies the axiom of separation of topological spaces. In the particular application described in the paper, the TFM is represented in the 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 the 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 an abstract object [8], [9].

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 systems' (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 present 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 the TFM construction, because it enables careful analysis of system's operation and communication with the environment.

3.2. TFM4MDA: A General Framework

The four TFM4MDA steps and their sub-steps are illustrated by bold lines in Figure 1 and discussed in this subsection. Having knowledge about a complex system that operates in the real world, a Topological Functioning Model (TFM) of this system can be composed (STEP 1, Figure 1). The TFM of the system affect and is affected by functional requirements (STEP 2, Figure 1). TFM functional features are decomposed into use cases and appropriate action sequence diagrams by means of system's business; this provides identification of business use cases as well as system use cases in compliance with the problem domain context. Besides that, functional requirements become not only in conformity with the business system functionality but can be also traced back to the system use case model (STEP 3, Figure 1).

Problem domain concepts are selected and described in an UML Class Diagram (STEP 4, Figure 1). STEP 4 is omitted in this paper, but it is described in [2] and [6].

STEP 1: Construction of the Topological Functioning Model. Construction of the TFM that reflects the problem domain in the context of business systems consists of the following sub-steps (Figure 2):

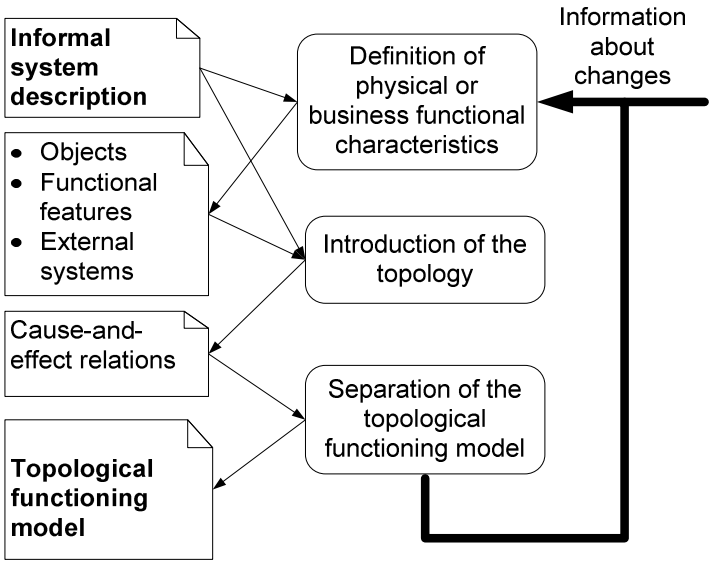


Figure 2. Construction of the TFM

Step 1.1 "Definition of physical or business functional characteristics" consists of the following iterative activities:

1) Definition of objects and their properties from the problem domain description that is performed by noun analysis. This means by establishing meaningful nouns and their direct objects and handling synonyms and homonyms;
2) Identification of external and partially-dependent systems. The former are objects that are not subordinated to the system's rules, and the latter are objects that are partially subordinated to the system's rules, e.g., system workers' roles;
3) Definition of functional features using verb analysis in the problem domain description, i.e., by finding meaningful verbs.

Within TFM4MDA, each TFM 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 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),
- E is an entity responsible for performing actions.

Both precondition and atomic business rule must be either defined as a functional feature or assigned to an already defined functional feature. Two forms of the textual description 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
```

Step 1.2 "Introduction of the topology" means establishing cause and effect relations between TFM 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 particularity of the cause and effect relations is there ability to compose 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's 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.

Step 1.3 "Separation of the topological functioning model" is the same as in the TFM approach [8], [9]. This means that it is performed by applying *the closure operation* of a set of system's *inner* functional features: A topological space is a system represented by Eq. (1). Where *N* is a set of *inner* system functional features and *M* is a set of functional features of other systems. The latter include those interacting with the system or those functional features of the system itself, which affect the external system functionality.

$$Z = N \cup M \tag{1}$$

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

The TFM is separated from the topological space of a problem domain by the closure of the set N as it is shown by Eq. (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 a 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, the closure operation can be applied to chosen subsets of *N* in order to separate the TFM into a series of subsystems.

STEP 2: Functional requirements' conformity to the TFM. This step is the check of degree of functional requirements' conformity to the constructed topological functioning model. On the one side, functional features (hereafter features) specify functionality that exists in the "problem world". On the other side, functional requirements specify functionality that must exist in the "solution domain". Thus, the mapping of functional requirements (hereafter requirements) onto the TFM functional features makes it possible to map and to constrain the "problem domain" by the "solution domain". Such mapping gives two outcomes: first, adequacy of the "solution domain" to the "problem domain" is checked at the very beginning of analysis, and second, the "solution domain's" functionality enhances and/or constrains the "problem domain's" functionality.

Mappings are formally described with arrow predicates. Arrow predicates are constructs borrowed from the universal categorical logic. Universal categorical (arrow diagram) logic for computer science is explored in detail in [10]. TFM4MDA suggests mappings of five types and corresponding arrow predicates. They are as follows:

- *One-to-One* that is used if the requirement A completely specifies what will be implemented in accordance with the functional feature B;
- *Many-to-One* that is used if a set of requirements overlap the specification of what will be implemented in accordance with the functional feature. In case of the covering requirements, their specification should be refined. Otherwise, disjoint requirements together completely specify the functional feature and do not overlap each other.
- *One-to-Many* is used if a part of the requirement incompletely specifies the functional feature or if one requirement completely specifies several functional features. This may

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* is used if 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* is used if 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.

Step 2 results are both checked requirements and the TFM that describes needed, possible functionality of the system and the environment it operates within.

STEP 3: Construction of a use case model. This step is the main subject of the discussion in this paper. This step is transition from initial CIM models (an informal description, a requirements specification) to a CIM "output" model – a use case model. Besides that, this step gives a possibility of more formal tracing of functional requirements to use cases (Figure 1). This activity includes the following sub-steps.

Step 3.1 "Identification of business system users and their goals" is as follows. Business system users can be actors and workers [11]. In the TFM, actors are represented as external systems' 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 direct goals of business system's users is related to the identification of the corresponding set of functional features that are necessary for the goal satisfaction. A goal as the means for identification of use cases has been chosen because a goal can be achieved by performing some process that can be long running. 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. Identification of system (application's) goals helps for additional check of requirements, i.e., for discovering "missing" requirements.

Step 3.2 "Identification and refinement of system use cases" is as follows. Functional features needed to achieve a business goal and *specified by functional requirements* describe the achievement of the corresponding system goal, and, therefore, compose a system use case. A user of the business system who 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. Moreover, this principle also provides additional possibilities for the refinement of system use cases. An inclusion use case is an intersection of functional feature sets needed to achieve different system goals. These shared functional features can be located either in the main flow or alternate flow (i.e., a sub-cycle or a branch of the TFM) 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.

Step 3.3 "Use case scenario reflection". Scenario of the identified use cases can be represented in an UML activity and sequence diagram by transforming corresponding TFM functional features into diagram's activities and corresponding cause-and-effect relations into diagram's control flows.

Besides that, a good style for construction of application systems is separate developing of a user interface layer, and a coordination or controller layer. The user interface should not have responsibility for performing system events. According to GRASP Patterns, a controller is "the first object beyond the user interface layer that is responsible for receiving or handling

a system operation message" [3]. The controller responsibility can be assigned to a class representing the overall "system", a major subsystem or a use case scenario. The latter is the case under discussion.

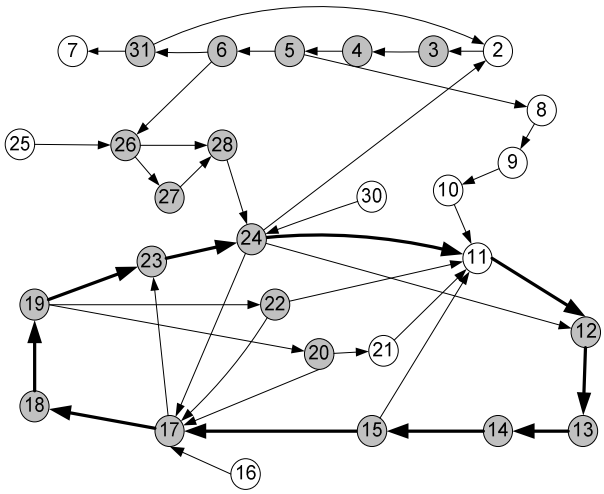
Basically, the elaboration of these objects should be done further during the analysis and design phases, but TFM4MDA enables to define the core of them as an input for the analysis phase. As previously mentioned, after constraining with functional requirements the topological functioning model represents the domain logic that will be implemented in the application. A responsible user initiates a functional feature in accordance with the domain logic. Some functional features can be generated as an effect of the functional feature initiated by a user. If a user initiates this functional feature, this means that he/she interacts using some user interface. Therefore, an action of this functional feature initiated by a user can be assigned to a controller, which then coordinates the fulfillment of this functionality by some responsible class. In case of large systems and used use-case driven techniques, the use-case level is a suitable scope for controller class representation. For example, in the Unified Process, control objects are use-case handlers as described in this Controller pattern. Usually, a controller class is named <UseCaseName>Handler, <UseCaseName>Coordinator, or <UseCaseName>Session [3]. In this work, the first one is used.

Step 3.4 "Use case prioritizing" is as setting priorities to use cases (and hence requirements) that usually is done in accordance with client's desires using some requirement attribute systems, e.g. MoSCoW or GRASP [3]. Within TFM4MDA, implementation priorities to use cases are set in conformity with the main cycle of the system as critical, important, or useful.

4. A Demonstration of the TFM4MDA Use for Identification of the Controller

Direct users of the business system identified for the system reflected by the TFM in Figure 3 (a) are shown in the 1st column of Table 1. The business system users are a Registrar, a Reader and a Librarian. Besides that, the Reader is a business actor, but the Registrar and Librarian are business workers. Their business goals identified in accordance with the functionality represented by the TFM and functional features that implement them and that are planned to be implemented in the application are shown correspondingly in the 4th and 5th column of Table 1. The use case diagram illustrated in Figure 4 then was obtained.

Let us consider the functionality specified by the use case "Take out copy". Descriptions of those functional features are shown in Figure 3 (b). The corresponding UML sequence diagram and its relation to the TFM functional features and cause and effect relations are shown in Figure 5.



a)

Functional features needed to achieve BG5:

11. Taking the request form from a reader, Librarian, inner
12. Checking the availability of a copy, Librarian, inner
13. Calculating the book amount of a reader, [if the copy is available], Librarian, inner
14. Checking the book limit of a reader, Librarian, inner
15. Checking out the copy to a reader, [if the book amount is smaller than the book limit], Librarian, inner

b)

Figure 3. The TFM of the library in the application context (a) and descriptions of the TFM functional features for BG5 (b); functional features in shadowed vertices are those needed to be implemented in the application system

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

User	**Business Goal**		**Functional Features**	**Functional Features to be Realized**
	Label	**Title**		
Registrar	BG1	Register a new reader	2, 3, 4, 5, 6, 31, 7	3, 4, 5, 6, 31
	BG2	Add a new entry	25, 26, 27, 28, 24	26, 27, 28, 24
Reader	BG3	Consult a catalogue	8	none
	BG4	Complete a request	9, 10	none
Librarian	BG5	Take out a copy	11, 12, 13, 14, 15	12, 13, 14, 15
	BG6	Take back a copy	16, 17, 18, 19, 23, 24	17, 18, 19, 23, 24
	BG7	Send a copy to a restoration	20, 21	20
	BG8	Remove a copy	22	22
	BG9	Impose a fine	18, 19	18, 19

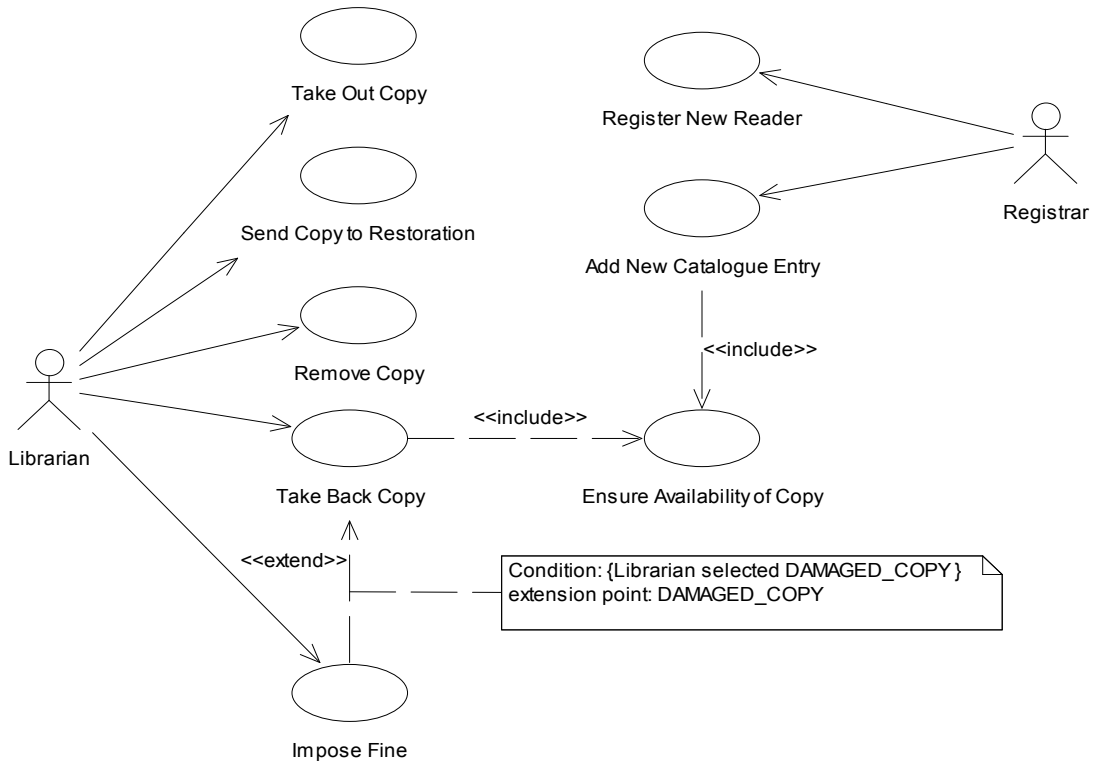


Figure 4. The use case diagram resulted from the TFM by TFM4MDA

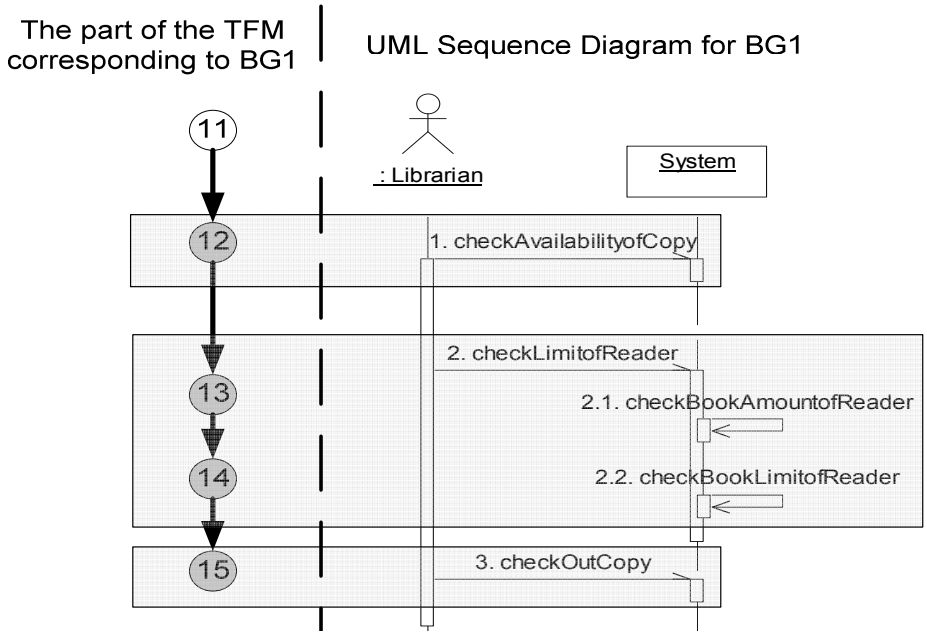


Figure 5. The part of the TFM and corresponding UML sequence diagram

The Controllers for use cases can be defined in accordance with the identified use cases. As mentioned in the preceding subsection, the controller object handles events received from the user interface objects. The defined controller for the use case "Take out copy" is

illustrated in Figure 6. A name of the controller consists of the use case name and the word "Handler". The controller *TakeOutCopyHandler* manages direct user interaction for the functional features 12, 13, and 15.

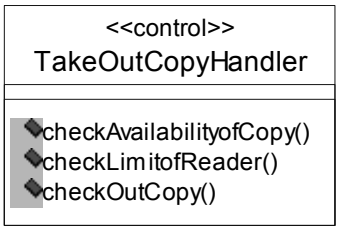


Figure 6. The Controller for the use case "Take out copy"

5. Conclusions

TF4MDA can be used to analysis of the complex business and physical systems that have evident, clear-cut functionality. Generally, TFM4MDA can be applied for any development methodology, because it describes system's functioning on the very high level of abstraction. However, this paper considers application of TFM4MDA in the context of both use-case driven approaches and object-oriented paradigm.

The application of TFM4MDA has the following advantages. Careful analysis of TFM cycles can help 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 their realization. TFM4MDA helps to check completeness and consistency of functional requirements as well as does not limit the use of any requirement gathering techniques. It provides use case completeness, avoids conflicts among use cases, and shows their affect on each other. Use case (requirement) implementation priorities can be ordered not only in accordance with the client's wishes, but also in accordance with the functioning cycles.

TFM4MDA supports the Controller pattern application. The TFM in the application context shows cause-and-effect chains in the functionality of the system. The mapping of the TFM into UML sequence diagrams enables identification of use case controllers. Thus, TFM4MDA supports the Model-View Separation Principle.

The further research is related to investigation of TFM4MDA properties in support of other analysis and design patterns that deals with responsibility delegation.

References

1. OMG *"OMG: MDA Guide Version 1.0.1"*, Miller J., Mukerji J. (eds.), 2003 // Internet. – http://www.omg.org/doc/omg/03-06-01.pdf
2. Asnina E. *"Formalization of Problem Domain Modeling within Model Driven Architecture"* // PhD Thesis, Riga Technical University, RTU Publishing House, Riga, Latvia, 2006
3. Larman Cr. *"Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. "* // Prentice Hall PTR, 2005, 703 pages
4. Gamma E., Helm R., Johnson R. and Vlissides J.M. *"Design Patterns: Elements of Reusable Object-Oriented Software"* // Addison-Wesley, 1995

5. Osis J., Asnina E., Grave A. *"MDA Oriented Computation Independent Modeling of the Problem Domain"* // In: Proceedings of the 2nd International Working Conference on Evaluation of Novel Approaches to Software Engineering, Barcelona, Spain, 2007, p. 66- 71
6. Osis J., Asnina E., Grave A. *"Formal Computation Independent Model of the Problem Domain within the MDA"* // In: Proceedings of the 10th International Conference on Information System Implementation and Modeling, CEUR-WS Vol. 252, Hradec-nad-Moravici, Czech republic, 2007, p. 47-54
7. Jackson M. *"Problem Frames and Software Engineering"* // In: Information and Software Technology, Volume 47, Issue 14, November 2005, p. 903-912
8. Osis J. *"Formal Computation Independent Model within the MDA Life Cycle"* // In: International Transactions on Systems Science and Applications, Vol. 1, Nr. 2, Xiaglow Institute Ltd, Glasgow, UK, 2006, p. 159-166.
9. Osis J. *"Software Development with Topological Model in the Framework of MDA" //* In: Proceedings of the 9th CaiSE/IFIP8.1/EUNO International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'2004) in connection with the CaiSE'2004, Vol. 1, Riga: RTU, Riga, Latvia, 2004, p. 211 – 220.
10. Diskin Z., Kadish B., Piessens F., and Johnson M. *"Universal Arrow Foundations for Visual Modeling" //* In: Proc. Diagramms'2000: 1st Int. Conference on the theory and application of diagrams. Springer LNAI, No. 1889, 2000, p. 345-360.
11. OMG *"UML Extension for Business Modeling, Version 1.1",* 1997 // Internet. - umlcenter.visual-paradigm.com/umlresources/exte_11.pdf

Asņina Ē. Lietošanas gadījumu kontrolleru šablona atbalsts TFM4MDA pieejā

Šajā rakstā tiek apskatīta TFM4MDA (Topological Functioning Modeling for Model-Driven Architecture) pieejas pielietošana lietošanas gadījumu kontrolleru šablonu definēšanai no problēmvides modeļa. TFM4MDA mērķis ir padarīt modeļvadāmu arhitektūru (Model Driven Architecture vai MDA), kurā akcents ir uzlikts uz modeļiem nevis kodu, par formālāku. MDA nodrošina trīs skatījumus uz sistēmu: no skaitļošanas neatkarīgu, platformneatkarīgu un platformai specifisku. TFM4MDA pamatā ir topoloģiskais funkcionēšanas modelis, kas attēlo sistēmas funkcionalitāti neatkarīgi no „skaitļošanas". Tas apraksta sarežģītas sistēmas funkcionalitāti orientētā grafa veidā, kur mezgli ir sistēmas funkcionālas īpašības un loki ir cēloņu seku attiecības starp tām, un nodrošina šīs informācijas kartēšanu uz lietošanas gadījumu specifikācijām. Bez tam, TFM4MDA atbalsta šīs informācijas kartēšanu uz lietošanas gadījumu kontrolleru šablonu. Saskaņā ar GRASP šabloniem, lietošanas gadījuma kontrolleris ir pirmais objekts pēc lietotāja saskarnes slāņa, kas ir atbildīgs par lietotāja ģenerētu notikumu saņemšanu un apstrādi. Dati no topoloģiskā funkcionēšanas modeļa tiek kartēti uz UML secību diagrammām un tad uz kontrolleriem. Tas nozīmē, ka TFM4MDA atbalsta slāņu programmatūras arhitektūras sadalīšanas principu „Modelis-Skats" (Model-View).

Asnina E. Support of a Use-Case Controller Pattern by TFM4MDA

This paper discusses how Topological Functioning Modeling for Model-Driven Architecture or TFM4MDA can be applied for definition of use case controllers from the problem domain model. TFM4MDA aim is to make Model Driven Architecture (MDA), where the main emphasis is put on models not code, more formal. MDA supports separation of concerns and provides three viewpoints on the system: computation independent, platform independent and platform specific. TFM4MDA considers the system from the computation independent viewpoint. There is a topological functioning model in TFM4MDA foundations that represents functionality of a complex system as a directed graph, where nodes are system's functional features and arcs are cause-and-effect relations among them. TFM4MDA provides mapping of this information to use case specifications. Besides that, TFM4MDA supports mapping of this information to a use case controller pattern. According to GRASP Patterns, a use case controller is the first object beyond the user interface layer that is responsible for receiving or handling events generated by the user. Data of the topological functioning model are mapped to UML sequence diagrams to the pattern. This means that TFM4MDA supports the Model-View separation principle of layered software architecture.

Аснина Э. Поддержка шаблона контроллеров прецедентов использования в TFM4MDA
В данной статье рассматривается применение подхода TFM4MDA (Topological Functioning Modeling for Model-Driven Architecture) для определения шаблона контроллеров прецедентов использования, беря за основу модель проблемной среды. Цель TFM4MDA сделать более формальной Управляемую Моделями Архитектуру (Model Driven Architecture или MDA), в которой усилена роль модели, а не кода. MDA предлагает три способа описания систем: независимый от вычислений, независимый от платформ и специфический для платформы. В TFM4MDA система описывается с независимой от вычислений точки зрения. Топологическая модель функционирования, лежащая в основе TFM4MDA, описывает функционирование сложной системы в форме направленного графа, где узлами графа являются функциональные свойства системы, а ориентированными ребрами – причинно-следственные связи между ними. В TFM4MDA предусмотрено отображение данной информации в спецификацию прецедентов использования. Кроме того, в TFM4MDA реализовано возможное отображение данной информации в шаблоны контроллеров прецедентов использования. В соответствии с шаблонами GRASP, контроллер прецедентов использования – это первый объект после слоя интерфейса пользователя, ответственный за получение и обработку событий, сгенерированных пользователем. Данные из топологической модели функционирования сначала отображаются в диграммы последовательностей языка UML, а затем в сами шаблоны. Таким образом в TFM4MDA поддерживается принцип разделения «Модель-Вид» многоуровневой архитектуры программного обеспечения.