VERSITA
Applied Computer Systems
doi: 10.2478/acss-2013-0005
2013/ 14

Ensuring Domain Conformity by Means of the Topological Functioning Model

Erika Asnina, *Riga Technical University*

***Abstract* – The paper focuses on an open question about ensuring conformity among a domain of system knowledge, an analysis domain, a software design domain, and code. Principles of Model Driven Engineering are used in the research, namely, formal specifications of domains and support of conformity among these specifications with the help of traceability mechanism. Topological Functioning Model (TFM) has a mathematical mechanism for supporting traceability, and, thus, conformity. The main results demonstrate that the TFM holds the central place in a chain of trace links between specifications, and analysis of these links enables verification of domain conformity. The results are theoretical and require additional practical experiments.**

***Keywords* – domain modeling, topological functioning model, traceability**

I. Introduction

The common case in software development projects is a refinement of software specifications after the end product is ready to be delivered. Specifications are used as an aid at the beginning of the development process and in case of significant changes in customer requirements. Ensuring permanent conformity between system specifications, software specifications and the code is still a challenge. Conformity of specifications is meant as a description showing which constructs of the system specification have corresponding constructs in the software specification and code, which constructs are optional, and which constraints are kept within the specifications. Traceability of constructs in specifications is the key mechanism for analysis of conformity.

In order to solve this challenge, Model-Driven Engineering (MDE) suggests using specifications as a source and a base for production of code. This means that transformations of composed specifications into code should be automated. Automated transformations require formally defined specifications and mappings (correspondence of constructs) between them. The main challenge is formalism of computation independent models (CIMs), because these models deal with informal knowledge sources and are semi-formal at most. CIMs specify the problem domain, i.e., system and software requirements.

Besides the transformation and composition of the CIM specifications, traceability of requirements and their sources from code to specifications and vice versa is an open question. This research illustrates an attempt to answer this question by means of mathematical formalism and MDE. This paper discusses application of a Topological Functioning Model (TFM) as a formal CIM of both systems "as-is" (problem) and "to-be" (solution) and tracing of conformity from this CIM to platform independent model to platform specific model to code (Section III).

The paper is organized as follows. Section II briefly describes a TFM and continuous mapping. Section III discusses and illustrates the mechanism for ensuring domain conformity and storage of tracing data during transformations. Section IV illustrates this mechanism by an example. Section V discusses related work. Conclusions discuss the obtained results and directions of future research.

II. The Topological Functioning Model in Brief

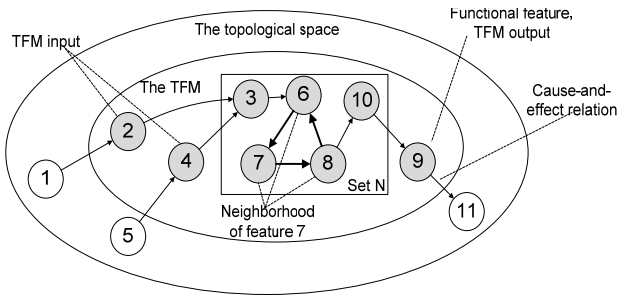
First, let us describe in brief the TFM. The Topological Functioning Model was developed in Riga Technical University by Janis Osis in 1969. It is a mathematical model that describes a topology $\mathcal{O}$ among system's functional features $\boldsymbol{X}$ from the computation independent viewpoint. The model is presented as a directed graph. Topology $\mathcal{O}$ is presented by a set of directed arcs from cause functional features to effect functional features (Fig. 1).

The TFM is independent of any modeling and implementation techniques. It comes about through the acquisition of the experts' knowledge of the complex system, verbal descriptions, and other documents concerning structure and functioning of the system.

A functional feature represents a part of system's functioning and is defined as a unique tuple <A, R, **O**, **PrCond**, **PostCond**, **Pr**, **Ex**> [1], where:

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

The TFM has topological and functional properties illustrated in Fig. 1. The topological properties are connectedness, closure, neighborhood and continuous mapping.



The set of all functional features in the topological space Z {1,2,3,4,5,6,7,8,9,10,11}.
The set of inner system functional features N = {3,6,7,8,10}.
The set of external functional features M = {1,2,4,5,9,11}.
The set of system functional features X, which is obtained after the *closure* of N and contains of features of the topological functioning model (TFM), is {2,3,4,6,7,8,9,10}.
Functional features 1, 5, and 11 represents activities inside the external environment.

Correctness of the structure of the TFM is characterized by the following:
- All features are connected by *cause-and-effect relations* and therein no isolated vertices (*connectedness*).
- The functioning *cycle* exists and it is 6->7->8->6.
- *Input* functional features of the TFM exist and they are 2 and 4.
- An *output* functional feature exists and it is 9.

Fig. 1. An abstract topological functioning model and its properties.

The functional properties are cause-effect relations (causal dependency between functional features, where one feature, cause, generates another feature, effect), cycle structure (causal dependencies between causes and effects that form a cycle), inputs and outputs. Properties and application of the TFM within the Model Driven Architecture and MDE are described in detail in [2].

The closure of the set of selected functional features of the system is a formal mathematical mechanism for separation of the subsystems from the system. Using the closure it becomes possible to formally define the system or subsystem boundaries.

Continuous mapping is a formal mathematical means for providing mappings between graphs. According to the statement and corollaries of continuous mapping described in [2], the model with any complexity can be abstracted to a simpler one and vice verse. Continuous mapping states that direction of arcs in the topological model must be kept in a refined model the same as in a simplified one. **Error! Reference source not found.** illustrates a refined TFM that is continuously mapped to the simplified one. The refined TFM includes four specified functional features: *i3.1* and *i3.2* are refinements of the functional feature *3*, and *i10.1* and *i10.2* refine functional feature *10*.

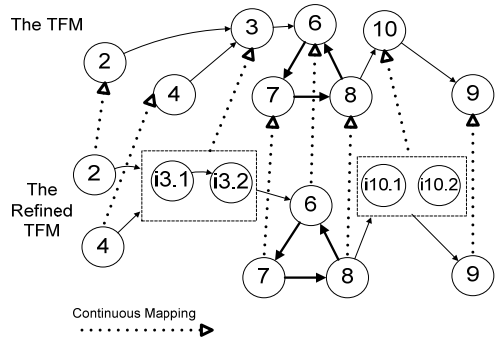


Fig. 2. Continuous mapping between two topological functioning models.

Cause-and-effect relations are kept as they are in the simplified TFM, although new cause-and-effect relations may also be between the specified functional features (as, for example, the cause-and-effect relation from *i3.1* to *i3.2*).

Lack of knowledge about the system can be filled with the knowledge that is obtained from the continuous mapping of the same type model to the system model under consideration.

III. Mechanism for Ensuring Domain Conformity

The mathematical formalism of the TFM can be extrapolated to models, which represent the problem and the solution in MDE.

The main MDE approach is Model Driven Architecture (MDA) suggested by the Object Management Group in 2001. MDA provides three models for system and software specifications, namely, a Computation Independent Model (CIM), a Platform Independent Model (PIM), and a Platform Specific Model (PSM) [3].

PIMs and PSMs specify a solution, i.e., requirements implementation in software. The CIM specifies *a domain*, i.e., system and software requirements, as well as the way the system works within its environment. Details of the software structure and realization are assumed to be hidden or as yet undetermined. The CIM is sometimes called a *domain model* and a *domain vocabulary*. However, domains that CIMs reflect are not clearly stated in the specification of [3].

Later investigations [1], [2], [4], [5], [6] showed that the CIM specifies both the problem domain and the solution domain. The problem domain, the system, is represented by knowledge models and *system* business models. The solution domain, software, is represented by *software* business models, which are supplemented with business requirements for the system.

Software business models must be in conformity with the system business models. The following chains of domain conformity must be ensured (Fig. 3):

1) From code to PSM to PIM: Business entities and execution logic in *code* and *data storage* must conform to business entities and execution logic in *software design models (structural and behavioral).*

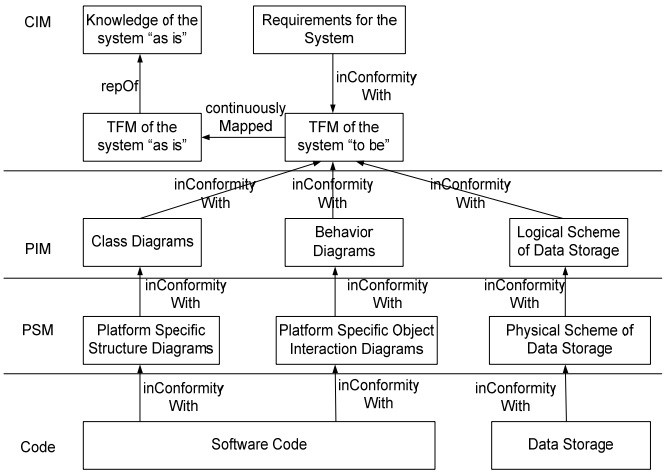


Fig. 3. Conformity among models in MDA terms.

Their conformance to business entities and business logic in a *software requirements specification* is provided be the CIM business models;

2) From PIM to CIM "to-be" to CIM "as-is": *The software requirements specification* must conform to *Business Requirements for the System* (the system "to-be"). Usually business requirements are merged with the software requirements in specifications. *Requirements for the System* must conform to *system "to-be" business models*, which in their turn must conform to *system "as-is" business models*. *System "as-is" business models* must conform to *Knowledge Models* about the System "as-is" that usually is informal, i.e., a set of documents and persons' knowledge. MDE transformations in accordance with this conformity chain would ensure that all specifications are up-to-date with code, and code origins are clear and traceable.

As Gotel and Finkelstein wrote in [7] "*requirements traceability is an ability to follow the life of a requirement*". Tracing the conformity links illustrated in Fig. 3 would facilitate implementation of changes during iterative software development and maintenance.

There should be two mechanisms for requirements traceability within the chains:

1) *Backward traceability* from the requirements for the system via the TFM of the system "to be" and the TFM of the system "as-is" to knowledge models, and

2) *Forward traceability* from the TFM of the system "to be" via PIMs and via PSMs to code. The traceability mechanism should be implemented at the metamodel level in order to support automation of model transformations.

A traceability mechanism from PIMs to PSMs to code is quite understandable, because transformation mapping languages support them in different ways. Tracing data should be kept in a separate traceability model that usually is created for each conducted transformation. This aspect is discussed in Section V. Here a less clear subject is considered, namely, backward traceability and traceability from the TFM of the system "to be" to some elements of structural and behavioral elements.

Table I demonstrates key mappings between the models, which are identical to the trace links from elements of the TFM of the system "to be" backward to elements of other computation independent models and forward to elements of some structural and behavioral diagrams:

1. *Mappings from "TFM of the System TO BE" to "Knowledge Model"*. The key elements of TFM of the system "as is", i.e. functional features and cause-and-effect relations between them, are mapped to textual, audio or video fragments of the description of the system. These trace links allow identifying and keeping the origin of the phenomena represented by the business model [2], [5].

2. *Mappings from "TFM of the System TO BE" to "TFM of the System AS IS"*. The same key elements of the TFM of the system "to be" are mapped to functional features and cause-and-effect relations of the TFM of the system "as is". Functional features of the system "to be" are "implementation" of the knowledge about the system "as is" in the system "to be", thus mapping can be "many to one" [1].

3. *Mappings from "Requirements for the System" to "TFM of the System TO BE"*. These mappings allow tracing functional requirements to functional features in the TFM of the system "to be", i.e. tracing from requirements to a business model [6], [8]. Analysis of trace links enables identification of missing, incorrect, and new functionality that is specified by requirements.

It should be noted that each functional feature of the TFM of the system "to be" must be mapped either to a corresponding functional feature of the TFM of the system "as is", or to corresponding one or several functional software requirements. Otherwise, such inconformity indicates incompleteness of functional software requirements or unreliable knowledge about the system "to be".

4. *Mappings from "Class Diagrams" to "TFM of the System TO BE"*.

Classes in PIMs, which represent phenomena of the problem domain and the solution domain, map to objects ***O*** of TFM functional features (see Section II).

Associations between classes map to cause-and-effect relations between functional features, which own objects ***O*** of the corresponding types.

Derived attributes of classes map to results *R* of TFM functional features. Nested attributes come from object ***O*** properties.

Operations of classes map to actions *A* of TFM functional features.

5. *Mappings from "Behavior Diagrams" to "TFM of the System TO BE"*.

An activity of a behavior diagram maps to the name of the TFM functional feature, which joins action *A*, set of object ***O*** and result *R*. Guard conditions of activities map to preconditions of TFM functional features.

TABLE I

KEY MAPPINGS BETWEEN ELEMENTS OF COMPUTATION INDEPENDENT MODELS AND PLATFORM INDEPENDENT MODELS

Computation Independent Model				Platform Independent Models		
Knowledge Model	TFM of the system "as is"	TFM of the system "to be"	Requirements for the System	Class Diagrams	Behavior Diagrams	Logical Scheme of Data Storage
Fragments of textual descriptions, audio and video	Functional Feature	Functional Feature	Software Functional Requirement	Class Attribute Operation	Activity Guard Condition Data Object	Table
	Cause-and-Effect Relation	Cause-and-Effect Relation		Association	Control Flow	

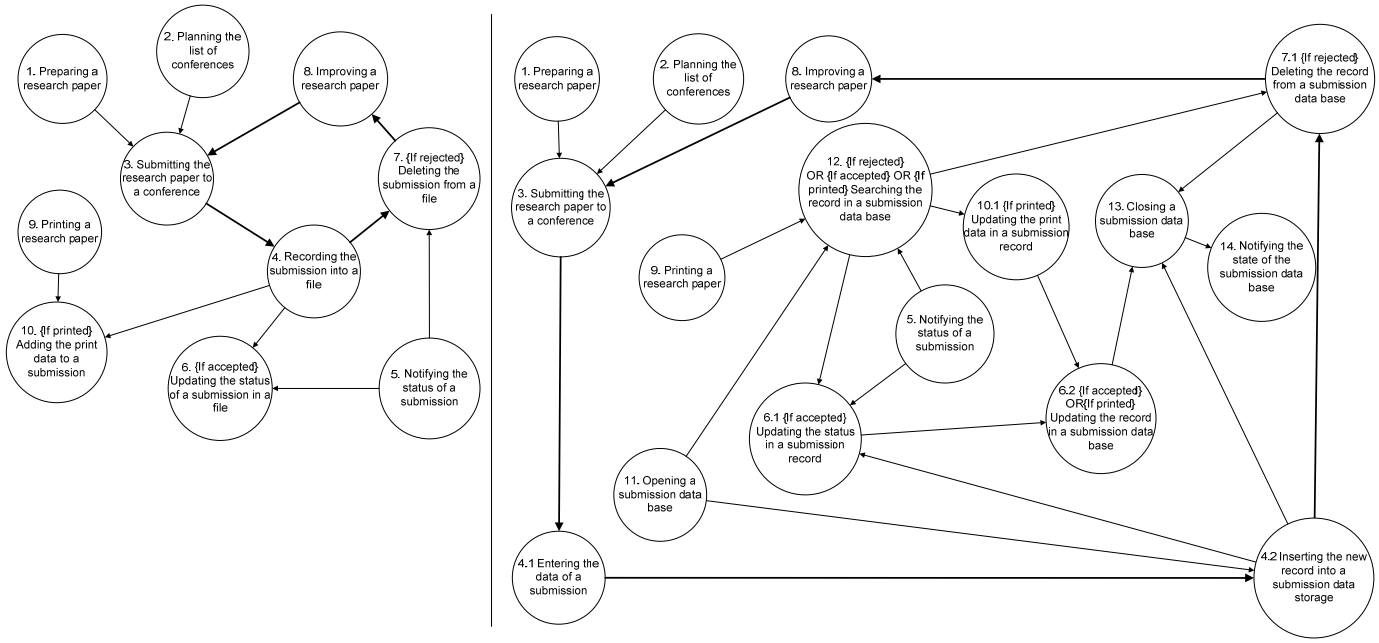


Fig. 4. The TFM of the system "AS IS" (on the left side) and the TFM of the system "TO BE" (on the right side).

Data objects map to objects ***O*** or results *R* of TFM functional features.

Control flows between the activities map to cause-and-effect relations between TFM functional features, and keep the same directions and logical relations.

6. *Mappings from "Logical Scheme of Data Storage" to "TFM of the System TO BE"*.

Tables are mapped to persistent types of objects ***O*** of those TFM functional features, which represents functionality for recording, updating, and deleting data in data storages. Table columns include persistent properties of objects ***O***.

To sum up, TFM functional features and cause-and-effect relations are sources for tracing static and dynamic properties of software. Data storages are traced to persistent classes, which in their turn are traced to static properties of the TFM functional features – a set of objects ***O*** and result *R*. Classes (and data objects of behavioral diagrams), which represent problem domain phenomena, are also traced to these static properties. Software functions (activities) are traced to dynamic properties of the TFM functional features – action *A*, and preconditions ***PreCond***. Interaction between software functions (or business logic of software) is traced to cause-and-effect relations between TFM functional features. The modularization principle could be kept here, since the TFM supports formal abstraction and refinement of functionality.

Traceability matrices [9] have been usually used to show dependencies between domain phenomena especially in early phases, because they show the relationships between source and target elements both forward and backward. Traceability matrices are used for pairs of models.

IV. Illustration of the Domain Conformity

The suggested traceability mechanism is demonstrated by an example of the simplified system that manages a part of research group activities.

A. The Problem Domain

The knowledge model as a textual description: When members of a group have prepared a research paper, they submit it to one of the planned conferences. A responsible author records the submission data (authors' names, title, conference name) into a file of group activities. If the paper is accepted, then he updates the status of the submission. Otherwise, the record is deleted from the file, and the paper is improved by its authors and resubmitted to another conference. After printing, the responsible author adds additional data (a print name, page numbers, a publisher name, a publishing year) to the record of submission and updates the status of the paper.

Requirements for the System: FR1. The system shall record, update, and delete records of paper submissions in a database. Data of submission are authors' first and last names, title of the submission, name of the conference, print data (title of the print where the submission is printed, publisher name, publishing year), pages from-to, the status (submitted, accepted, printed).

The TFM of the system "as is": The model is a representation of the description given above (Fig. 4, on the left side). It consists of 10 functional features and has a functioning cycle "3-4-7-8-3" for improvement of rejected papers.

B. The Solution Domain

The TFM of the system "to be": The model is a representation of the problem domain, which was modified in accordance with *Requirements for the System*. It has four new and six refined functional features. All cause-and-effect relations kept their directions. The functioning cycle is extended – "3-4.1- 4.2-7.1-8-3".

TABLE II
THE TRACEABILITY MATRIX BETWEEN THE TFM OF THE SYSTEM "AS IS" AND THE TFM OF THE SYSTEM "TO BE"

	1	2	3	4.1	4.2	5	6.1	6.2	7.1	8	9	10.1	11	12	13	14
1	1															
2		1														
3			1													
4				1	1											
5						1										
6							1	1								
7									1							
8										1						
9											1					
10								1				1				

TABLE III
THE TRACEABILITY MATRIX BETWEEN THE CLASSES AND THE TFM OF THE SYSTEM "TO BE"

	1	2	3	4.1	4.2	5	6.1	6.2	7.1	8	9	10.1	11	12	13	14
SubmissionRecord				1	1		1	1	1			1		1		
SubmissionDataBase					1			1	1				1	1	1	1

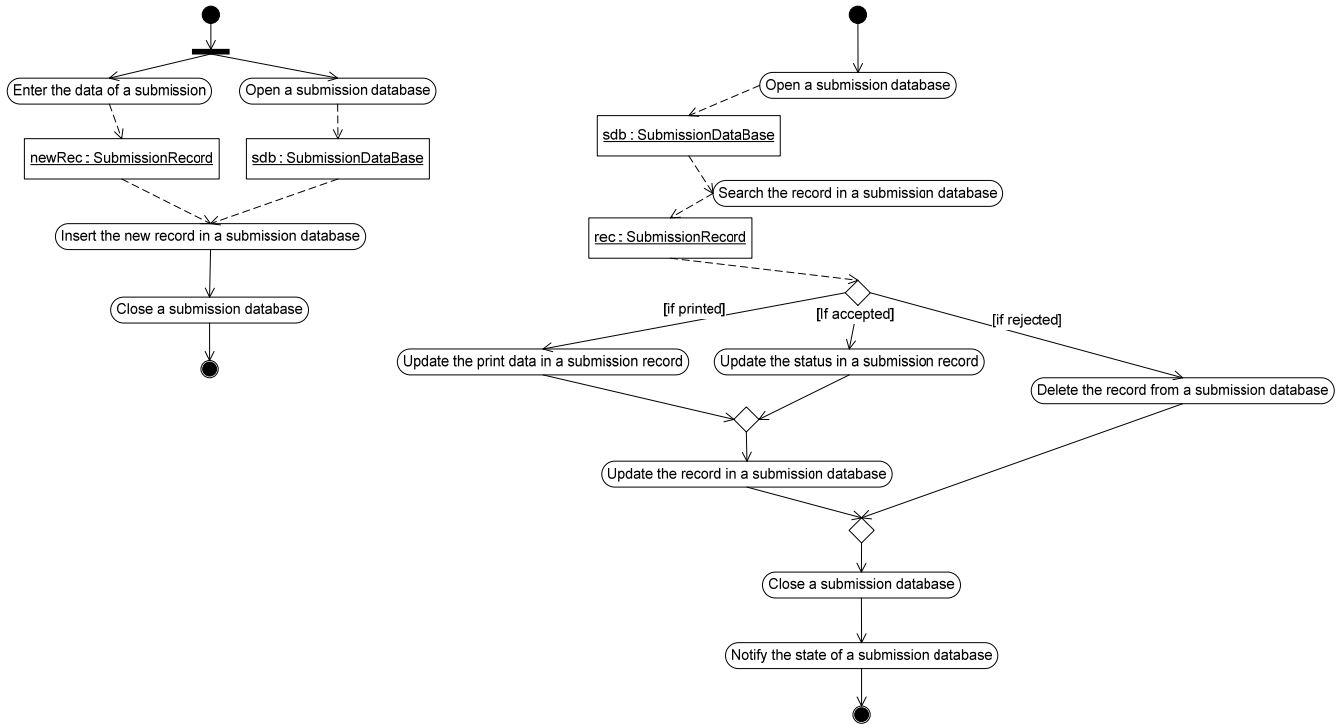


Fig. 5. Behavior diagrams (UML Activity diagrams) created in conformity with the TFM of the system "to be".

New functional features are "11. Opening a submission database", "12. If {rejected OR accepted OR printed} Searching for the record in a submission database", "13. Closing a submission database", and "14. Notifying the state of a submission database". They are originated by requirement FR1. The following TFM functional features are traced to FR1: 11, 12, 13, 14 as well as "4.1 Entering the data of a submission", "4.2 Inserting the new record into a submission database", "6.1 {IF accepted} Updating the status in a submission record", "6.2 {If accepted or printed} Updating the record in a submission database", and "10.1 {If printed} Updating the print data in a submission record".

Table II illustrates trace links between the TFM of the system "as is" (the problem domain) and the TFM of the system "to be" (the solution domain). The rows represent functional feature of the TFM of the system "as is", while the columns – of the TFM of the system "to be". The digit 1 indicates the existence of a trace link between functional features in two models.

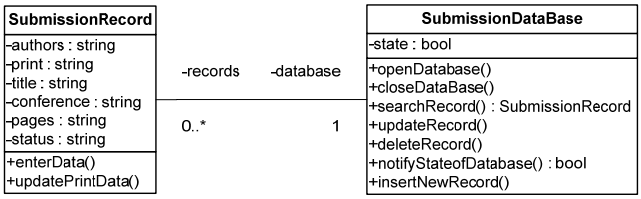


Fig. 6. Classes obtained from the TFM of the system "to be".

Class Diagram: The classes *SubmissionRecord* and *SubmissionDataBase* (Fig. 6) are obtained by transforming those functional features of the TFM of the system "to be", which have trace links to FR1, into types of the objects ***O***.

The traceability matrix from these classes (table rows) to the functional features of the TFM of the system "to be" (table columns) is given in Table III.

Logical Database Scheme: In the current example the only class that is persistent is *SubmissionRecord*. Its trace links to the TFM of the system "to be" were discussed in the previous paragraph (Table III).

Behavior Diagrams: The behavior diagrams are represented as two UML Activity Diagrams (Fig. 5), which are in conformity with the TFM of the system "to be" and FR1. The traceability matrix (Table IV) contains trace links from the activities to those functional features, which are traced to FR1. Control flows between activities and object flows "activity - data object – activity" are traceable to cause-and-effect relations among those functional features. Data objects to the objects ***O*** of the functional features. Decision nodes and forks illustrate OR and AND logical relations among cause-and-effect relations correspondingly.

To conclude, the trace links forward and backward from FR1 are explicitly specified by a means of TFM, which serve as a business model that joins the problem domain and the solution domain.

TABLE IV

THE TRACEABILITY MATRIX BETWEEN ACTIVITY DIAGRAMS AND THE TFM OF THE SYSTEM "TO BE"

	4.1	4.2	6.1	6.2	7.1	10.1	11	12	13	14
Activities										
1.Enter the data of a submission	1									
2. Open a submission database							1			
3. Insert the new record in a submission database		1								
4. Close a submission database									1	
5. Search the record in a submission database								1		
6. Update the print data in a submission record						1				
7. Update the status in a submission record			1							
8. Delete the record from a submission database					1					
9. Update the record in a submission database				1						
10.Notify the state of a submission database										1
Quards										
[if printed]				1		1		1		
[if accepted]			1	1				1		
[if rejected]					1			1		
Data objects										
newRec, rec of SubmissionRecord	1	1	1	1	1	1		1		
sdb of SubmissionDataBase		1		1	1		1	1	1	1
Control flows										
1-to-newRec-to-3	1	1								
2-to -sdb-to-3		1					1			
3-to-4		1							1	
2-to-sdb-5							1	1		
5-to-rec-to-6						1		1		
5-to-rec-to-7			1					1		
5-to-rec-to-8					1			1		
6-to-9				1		1				
7-to-9			1	1						
9-to-4				1					1	
8-to-4					1				1	
4-to-10									1	1

V. Related Work

As mentioned in Section III, tracing in transformations from model to model (from PIMs to PSMs, from PSMs to code) is implemented in transformation languages such as Query/View/Transformation (QVT) [10]. Usually, it is implemented as trace classes that are implicitly and automatically derived from transformation rules. The trace contains a tuple that stores a reference of the mapping operation - or, which is equivalent, a reference to the corresponding implicit transformation rule - and then the value for each parameter, including the context variables and the result variables.

ATLAS Transformation Language (ATL) implements the tracing mechanism by Traceability Generation Code (TGC) [11]. TGC is loosely coupled with transformation logic and is used in higher-order transformations, thus does not require any language support or ATL engine modification.

As you see, the support of traceability in transformation languages is well-elaborated, because of the formal specifications of a source model, a target model, and a transformation rule. The worst case with traceability is in transformations between CIMs and from CIMs to PIMs because of the informal nature of CIMs.

Authors in [12] solved this challenge by tracing responsibilities, which are assigned to components in a target model by features in a source model. Thus, the authors provide loose coupling between the elements and realization of their responsibilities. This is an example of forward traceability of requirements. Contrary to the authors, in the present research responsibilities are not traced so explicitly, rather the implementation of functionality in software units is represented here by trace links.

The authors in [13] used trace links for identification of and dealing with crosscutting concerns in early stages of software development in the field of Aspect-Oriented Software Development. They used inter-level dependencies matrices for specification of trace links between levels of CIMs, PIMs, and PSMs. The authors' idea is similar to the one forwarded in this paper, namely, backward and forward requirements traceability starting from the CIM is vital for qualitative product development.

Authors in [14] suggested using MDE principles in transformations from CIM business models via PIMs and PSMs to code in Web Engineering. The important role is devoted to trace links starting from the requirements in CIMs to code. They use Navigation Development Techniques (NDTs) for the creation of derivation relations between the models by means of formal models. The authors also use inter-level traceability matrices between requirements and basic analysis models, and QVT transformations for checking the conformity between tables in a database and persistence classes in code.

VI. Conclusions

Identification of trace links between the elements of source and target models is a means for better understanding of the conformity between the models. The result of the research is verifiable conformity of the solution domain models to problem domain models that is ensured by means of a formal model, TFM and trace links between elements of the models. The trace links are defined and supported during transformations of system and software specifications. It could make evaluation and implementation of possible changes easier.

Since the TFM plays a leading role in model transformations and traceability analysis, directions of future research are related to TFM verification with the help of formal mechanisms, e.g., Colored Petri Nets.

VII. References

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

[2] J. Osis, E. Asnina, Model-Driven Domain Analysis and Software Development: Architectures and Functions. Hershey, New York, USA: IGI Global, 2011.

[3] J. Miller, and J. Mukerji, Model Driven Architecture (MDA). Architecture Board ORMSC, ormsc/2001-07-01. *The OMG.* 2001. Retrieved from: www.omg.org/mda/

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

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

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

[7] O.C. Gotel, A. C. Finkelstein, An Analysis of the Requirements Traceability Problem. Proceedings of the First International Conference on Requirements Engineering, 1994. Colorado Springs, CO, USA: IEEE, 1994, pp. 94 – 101.

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

[9] Davis, A. Software Requirements: Objects, Functions and States. Prentice–Hall, Second Edition, 1993.

[10] Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1 - January 2011. *The OMG.* 2011. Retrieved from: http://www.omg.org/spec/QVT/index.htm

[11] Jouault F. Loosely Coupled Traceability for ATL. In: ECMDA Traceability Workshop (ECMDA-TW) 2005 Proceedings, November 8th 2005, Nuremberg, Germany. SINTEF, 2005, pp. 29-37.

[12] Zhang W., Mei H., Zhao H. and Yang J. Transformation from CIM to PIM: A Feature-Oriented Component-Based Approach. In: Lecture Notes in Computer Science, Volume 3713, Model Driven Engineering Languages and Systems. Springer, 2005, pp. 248-263.

[13] van den Berg K., Conejero J.M., Hernández J. Analysis of crosscutting across software development phases based on traceability. In: Proceedings of the 2006 international workshop on Early aspects at ICSE (EA'06). ACM, May 2006, pp. 43-49.

[14] Escalona M. J., Gutiérrez J. J., Rodríguez-Catalán L., and Guevara A. Model-driven in reverse: the practical experience of the AQUA project. In: Proceedings of the 2009 Euro American Conference on Telematics and Information Systems: New Opportunities to increase Digital Citizenship (EATIS '09). ACM, New York, NY, USA, 2009, Article 17 , 6 pages.

Erika Asnina received M.Sc. in computer systems in 2003 and a doctor's degree (Dr.sc.ing.) in information technology with specialization in system analysis, modeling and design from Riga Technical University in 2006.

She is Assistant Professor in the Department of Applied Computer Science in Riga Technical University since 2008. She also worked 5 years as a Software Developer. She is author of 28 conference papers, four book chapters and one book. Her research interests include software quality assurance, model-driven and object-oriented software development, and software engineering.

Latvian Academy of Sciences has awarded her and her co-author, Janis Osis, for the book “Model-Driven Software Development: Architectures and Functions”, which was recognized as one of the most significant theoretical achievements of Latvian Science in 2011. She was also awarded as a scholarship laureate of the target program “For Education, Science and Culture” of Latvian Education Fund in 2004 and 2005.

Contact address is the Department of Applied Computer Science, Riga Technical University, Meža Street 1.-k.3, Riga, LV-1048, Latvia; e-mail: erika.asnina(at)rtu.lv.