DE GRUYTER OPEN

Applied Computer Systems

doi: 10.1515/acss-2014-0007

2014/16

Dependencies among Architectural Views Got from Software Requirements Based on a Formal Model

Janis Osis[1], Erika Asnina[2], Uldis Donins[3], Vicente García-Díaz[4],
[1–3]*Riga Technical University, Latvia*, [4]*University of Oviedo, Spain*

***Abstract* – A system architect has software requirements and some unspecified knowledge about a problem domain (e.g., an enterprise) as source information for assessment and evaluation of possible solutions and getting the target point, a preliminary software design. The solving factor is architect's experience and expertise in the problem domain ("AS-IS"). A proposed approach is dedicated to assist a system architect in making an appropriate decision on the solution ("TO-BE"). It is based on a formal mathematical model, Topological Functioning Model (TFM). Compliant TFMs can be transformed into software architectural views. The paper demonstrates and discusses tracing dependency links from the requirements to and between the architectural views.**

***Keywords* – Analytical models, object oriented modeling, software architecture, software quality, topology.**

I. Introduction

We would like to begin our paper with Jones' words – "The way the software is built remains surprisingly primitive" [1]. Jones' 30-year research has shown that major software applications are cancelled, overrun their budgets and schedules, and often have hazardously bad quality levels when released. Jones, as well as we and other scientists, explains this phenomenon by the fact that the problem domain is wider and more complex than the solution domain, and, due to its complexity, in particular cases, the problem domain is not considered by developers at all, or more often it is considered only in part that is the closest one to the suggested solution.

The solution at the design level consists of a set of architectural views, e.g., structural, static, behavioral, deployment and so on. This set should be integrated by following some traceable structure, since "decentralization without structure is chaos" [2]. The structure of decentralization should be clear by tracing correspondence or dependency links among views. By reducing or even avoiding proper analysis of the problem domain, the traceability of artifacts of problem and solution domains (i.e., between requirements, behavioral models, structural models, and code) is weak. This makes the acceptance process of developed software meaningless since the customer as well as developers themselves cannot fully validate the produced solution.

The UML (Unified Modeling Language) is widely used by developers for object-oriented architectural descriptions of software. Although both system static and dynamic viewpoints can be described by UML diagrams, the language itself is not sufficient for successful design of software. It should be combined together with an appropriate software modeling technique that allows negotiating and integrating architectural descriptions, and motivates software developers to pay more attention to the analysis and understanding of the problem domain structure and functioning. Thus, appropriate models and their application methods should be provided [3].

This research extends previous work on the topological modeling of system functioning, which has proved itself as a powerful means in the domain of embedded systems where the analysis is done by applying the TFM in the architecture co-design method [4], in business process modeling [5], in software development for mechatronic and embedded systems [6], in introduction of more formalism in a problem domain analysis ([5], [7], [8]) and Model Driven Architecture (MDA) [9]. Within the MDA, the research on the formal analysis of Computation Independent Model (CIM) ([10] – [15]), formal specification of Platform Independent Model (PIM) and conduction of transformation CIM-to-PIM within the MDA ([16], [17]) is conducted. It summarizes and refines results of the research on TopUML as well as explores traceability from a domain model to and between architectural views in more detail (that was not done before).

The aim is to demonstrate and to trace dependency links from software functional requirements (we do not consider non-functional requirements) to and between structural and behavioral software views. We suggest using the formal approach, TopUML modeling, which includes formal modeling of the problem domain, analysis and verification of the domain model – TFM, and formal transformation of this model into structural (communication, object, and topological class diagrams) and behavioral (topological use case, sequence, activity, and interaction overview diagrams) views.

The paper is organized as follows. Section II discusses background, and Section III – related work, on the links in UML-driven and goal-oriented modeling approaches correspondingly. Section IV explains the suggested TopUML approach in brief. Section V gives an illustration of the approach capabilities. Conclusions discuss main results and further research directions.

II. UML-driven Modeling Methods

First, the UML is a semi-formal language system for the expression of knowledge. Besides, it is independent of particular methods and approaches. However, most of the UML-driven modeling methods make a use of use-case-driven approaches [18], probably following recommendations on a use-case-driven process given in [19]. A majority of UML-driven modeling approaches have endorsed this view, and

most contain at least some further prescriptions for applying the UML in modeling (e.g., [20]–[22]), which sometimes differ. However, not only use cases may be used to verify requirements with users, but also other diagrams, e.g., activity diagrams [18]. There is also a difference in applying use case narratives across various methods due to the lack of guidance on the narrative format in the UML specification. The UML specification [23] only states that "use cases are typically specified in various idiosyncratic formats such as natural language, tables, trees, etc. Therefore, it is not easy to capture its structure accurately or generally by a formal model".

The success of the software development project can be measured against the deliverables which satisfy and even exceed customer's expectations, the delivery schedule that occurred in a timely and economical fashion, and the created result that is resilient to change and adaptation. Strong architectural vision of the solution is one of the two characteristics [24] that the successful (in the given measurements) project should satisfy. In addition to the conceptual integrity, well-designed software architectures tend to have several attributes in common [25]: 1) constructs in well-defined layers of abstraction; 2) a clear separation of concerns between the interface and implementation; and 3) the architecture itself is simple, i.e., a common behavior is achieved through common abstractions and mechanisms.

Since the UML itself is a language system and it does not contain guidelines on how it can be applied in practice, the largest benefit of presence of UML-driven modeling approaches (UDMAs) is that the use of the UML is more or less systematized. The wide-known applications of the UML are together with software development lifecycles [25], use case driven methods [21], MDA [9], pattern based development [20], component based development [22], and conceptual modeling [26]. The current state of the art of these approaches includes those that are well known in software development industry (like in [18]), formalize the development process and problem domain, and are used in conjunction with software development tools (like in [27]).

The UML in combination with one of the systematical approaches is a powerful tool for the analysis and understanding of systems as well as for software design. Despite the fact that the approaches provide the systematic use of UML diagrams, they do cover different parts of software development lifecycles. The whole software development lifecycle is covered only by the Unified Process (UP) [25] and Microsoft Solution Framework (MSF) [28]. Some methods focus more on an analysis (e.g., B.O.O.M. [21], TFM4MDA [10], and Conceptual modeling [28]), while others – more on a design and less on an analysis (e.g., Pattern based design [30], Component based development [22]). This impacts the number of types of UML diagrams used by each approach. The summary of the use of UML diagrams by each approach shows that not every type of UML diagrams is used.

The overview of UDMAs leads to the following conclusions: 1) Modeling approaches determine the application of UML diagrams and not the UML itself (the review of UML application in industry [18] and UDMAs shows that the top five most applied UML diagrams are the same); and 2) Due to the partial coverage of the UML and software development lifecycle and the fragmentary application of UML diagrams, software developers are forced to combine the UML with several modeling methods and techniques (instead of taking the UML and one UDMA). Thus, application of the UML becomes more complicated and incomprehensible.

III. Related Work: Solutions Provided by Goal-based Modeling Approaches

Although our research is focused on approaches that apply use cases as well as other UML diagrams, and where evidence of relationships among software development artifacts is weak and insufficiently supported, it is worth considering approaches, which exist in goal-based requirements engineering, where evidence of these relationships is the key concept. We believe that the same evidence must be provided for software development with the UML and UDMAs.

Michael Jackson's problem frames are applied as a general scheme that visualizes the overall relationship between the world and the machine (multiple information sources are presented in [31]). This approach suggests considering the machine (i.e., software and hardware) as part of multiple domains, which have impact on it. The idea (that is also elaborated in other requirements engineering approaches) is the same – starting development from the problem domain structuring and analysis to design the solution and then its implementation. Thus, trace (dependency) links are also evident and could be specified and traced.

KAOS (it stands for "Keep All Objectives Satisfied") is a requirements engineering approach, where the problem is analyzed and the solution is developed by using several complementary views [32]. The scope of the problem domain is defined by goals in a goal model; an active element, agent, is responsible for each goal satisfaction; the agents need to perform the operations in order to operationalize the goal; the goals and operations refer to the objects. Each view is described by dedicated diagrams. Trace links from agents to goals are evident and could be specified and traced.

The i* approach suggests structuring the problem domain into agents, which may depend on another for a goal to be satisfied, a task to be performed, or a resource to be made available [33]. The agents are interconnected through dependency links of various types. Thus, elements of the problem domain and the solution domain are traceable.

IV. TopUML: Obtaining Architectural Views by Analyzing Functional Characteristics of the Problem Domain

A. TopUML Relations to the TFM, UML and NLP

In order to strengthen software development with the UML, we suggest using a TopUML modeling approach (an UML profile based on UML version 2.4.1 [23] and its supporting model-driven method for problem domain modeling and software design). It combines the TFM and its formalism with elements and diagrams of the UML. The TFM considers

information about the problem domain separately from information of the solution domain and holistically represents a complete functionality of the system at the high level of abstraction, while the UML has more specialized elements for representing a system design. The application of the TFM ensures proper analysis of system functioning by identifying and analyzing functioning cycles. A functioning cycle is a common thing of all systems' (technical, business, biological, etc.) functioning. It represents the vitally important functionality of the system (so by destroying the main functioning cycle the system cannot longer function or it is seriously malfunctioning) based on income and handling of resources, producing output of this handling, and getting new resources for "a life". Therefore, at least one directed closed loop (main functional cycle) must be present in every TFM of the system [34]. Usually the main functionality is even an expanded hierarchy of cycles [35]. Therefore, a proper cycle analysis is necessary at the very beginning of the software development lifecycle, because it enables a careful analysis of system's operation and communication with the environment [36].

By using TopUML, the information of system functioning kept in the TFM is transferred to design models (and diagrams), thus allowing marking and evaluating the most important objects and components within the system and assigning proper responsibilities to the objects in a formal way [37]. It is possible to automate transitions between TopUML diagrams while the validation of the acquired diagrams is needed to be conducted by the domain experts. Since the text analysis is a linguistic problem and fact finding identifies any domain concept by analyzing terms, the TopUML, namely, development of the TFM, can be partly automated as shown in [38], where business use cases are transformed into the TFM by using Natural Language Processing (NLP) facilities. Thus, in order to obtain functions, external entities (actors), and internal entities (classes) we do not need to apply the human reasoning.

TopUML modeling does not include the development of UML Profile, Timing and Composite structure diagrams. The last two are not included, because the TFM shows timing within functioning of a system, but Composite structure diagram specifies structure of components. In its turn, Profile diagram is not addressed, because it is intended to specify a new profile of UML (not a software design).

B. TopUML in General

Problem domain analysis and software system design with the TopUML modeling method [39] consist of the following six activities as given below in Fig. 1 (notes assigned to each activity outline the diagrams developed within this activity).

Fig. 1 represents the recommended order of the application of the activities. However, within the software development project they can be applied in any order, and only part of the activities can be used. There is one restriction, namely, inputs of each activity should be provided before conducting the activity. The TopUML modeling method is only guidelines of TopUML profile application in software development; it does not restrict the use and application of TopUML diagrams.

The analysis of functionality of the problem domain is conducted during development of the TFM, i.e., during problem domain modeling. TFM development consists of three activities: 1) Development of the topological space with the purpose to classify functional characteristics of the problem domain and causal dependencies among them [8], [40], [34]; 2) Development of the initial TFM ("AS-IS") [40], in which determination of boundaries of the problem domain is conducted by separation of the TFM from the topological space. Along with the causal dependencies, logical relations among them should be analyzed as well (they show the behavior of the control flow within the system, e.g., decision making, parallel activities, branching, and joining) [41]; 3) Development of the refined TFM ("TO-BE") with the aim to conform functional requirements with the initial TFM in order to bridge solution and problem domains. As a result of requirements validation, both TFM and requirements are checked [11]–[15].

The outputs of the activity of analysis of the problem domain functionality are two TFMs (i.e., one representing functioning of the problem domain and another representing functionality of the desired software system), mappings between refined functional features (hereinafter, FFs) and refined functional requirements.

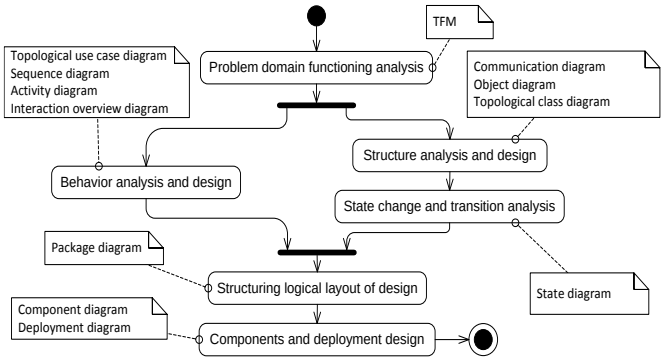

Fig. 1. TopUML modeling activities (borrowed and enhanced from [39]).

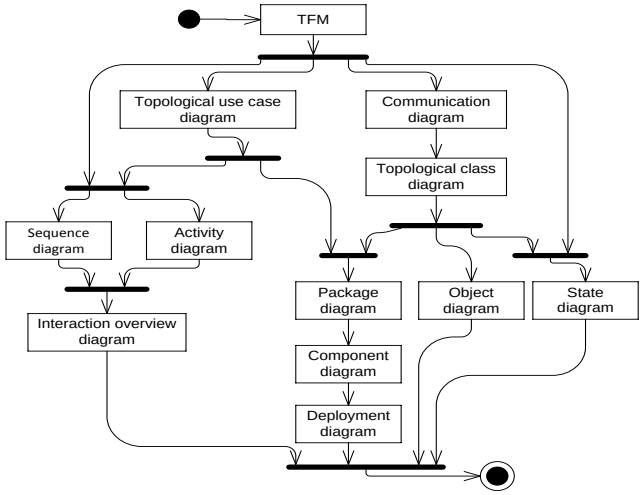

Fig. 2. Transitions between TopUML diagrams (borrowed from [39]).

When the TFM is used as part of TopUML [8], it can be transformed into other diagrams (Fig. 2), such as UML Sequence diagram, UML Activity diagram, UML Communication diagram, Topological use case diagram and Topological class diagram. Transformations of diagrams are crucial, e.g., the topological class diagram should contain classes with attributes and operations (responsibilities of classes) after transformation. As underlined by Larman in [20]: "deciding what operations belong where, and how the objects should interact, is terribly important and anything but trivial. This is a critical step – this is at the heart of what it means to develop an object-oriented system, not drawing domain model diagrams, package diagrams, and so forth."

The trace links are established in top-down direction – from functioning properties and functional requirements of the problem domain to the software design and development artifacts. Suggested transitions between TopUML diagrams are given in Fig. 2, where the diagrams are shown as object nodes and the edges between them as object flows within the activity diagram.

Two UML diagrams are formalized and called *topological*. A topological use case diagram is *topological*, as it is formally defined from and corresponds to the TFM functional features and directions (flows) of cause-effect relationships. In its turn, an UML use case diagram is constructed in the *ad-hoc* manner, without clear understanding what a use case is and where are its borders. A topological class diagram is *topological*, as it holds not only standard types of relationships of the UML class diagram but also topological relationships between classes kept from the TFM.

V. An Example of Getting Architectural Views

The main idea of this section is to explore an example of applying TopUML modeling in a real software development project, in which a service application is developed to synchronize enterprise employee data by taking data from multiple data sources and placing them into one central data storage. The example covers the full software development life cycle (the application is at the maintenance phase now, the example is covered till the implementation phase) and includes development of different architectural views by using the following TopUML diagrams (for more details see [39]):

- The initial and refined TFMs in accordance with the informal system description and functional requirements;
- The topological use case diagram defined in accordance with the developed TFMs and mappings between FFs and determined functional requirements;
- The sequence diagrams and the activity diagram in accordance with the TFM and use cases;
- The communication diagram obtained by performing transformations of the TFM;
- The initial and refined topological class diagrams obtained in accordance with the communication diagram and the TFM;
- The state diagram in the example is prepared for the main object of the system by applying transformations on the TFM. The main object is determined by its membership in the main functioning cycle.

Thus, the trace links are set as mentioned in Section IV (Fig. 2).

A. Requirements Specification for the Example

The following functional requirements (FRs) are defined for the data synchronization system – FR1: Employee data synchronization should be done between input data sources and the target data source:

- FR1/1: By starting the synchronization process, configuration information should be taken from the configuration file;
- FR1/2: If needed, data from the source database should be taken;
- FR1/3: Data should be taken from import files in CSV format;
- FR1/4: If the import CSV file is with the wrong data structure, the processing of the particular file should be skipped and the faulty import file should be logged;
- FR1/5: All data obtained from either the source database or import files should be placed in the target database;
- FR1/6: When importing data in the target database all rows from source data should be logged together with the import status for each particular data row.

B. Analysis of the Data Synchronization System Functionality

Construction of the TFM. Within the example, 30 FFs have been defined by analyzing functionality of the enterprise data synchronization system. Then, the topology Θ (cause-effect relationships) was defined between them. The resulting TFM (Fig. 3) contains 29 FFs (vertices), cause-effect relationships (arcs between vertices), and logical relations between cause-effect relationships (AND, OR, and XOR). Part of FFs used further for diagram transformations is explained in Table 1.

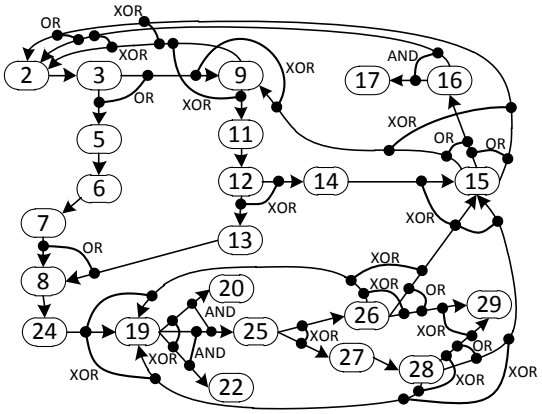

Fig. 3. TFM representing enterprise data synchronization system functioning (borrowed from [39]).

Refinement of the TFM. The result of requirements validation is that both TFM and FRs are checked for completeness, inconsistence, etc. In order to validate functional requirements and the constructed TFM, mappings from them to FFs should be established. Since in this example use cases are used to model requirements, the set of mappings

of FRs includes both FFs and functional requirements. The established mappings are as follows: FR1 = {FR1/[1–6]}; FR1/1 = {2, 3}; FR1/2 = {5, 6, 7, FR1/5}; FR1/3={9, 11, 12, 13, 14, FR1/[4-5]}; FR1/4 = {15, 16, 17}; FR1/5 = {8, 24, 19, 20, 22, FR1/6}; and FR1/6 = {25, 26, 27, 28, 29}. The identified mappings show that there are no missing FRs and no missing FFs, thus the refined TFM is equal to the initial TFM.

C. Developing the Behavioral View

The behavioral view of the design starts from use cases. According to the mappings defined between FFs, FRs and logical relations in the TFM, the «include» and «extend» relationships are automatically established between use cases. Each requirement is modeled with a use case: FR1 = "Employee data synchronization", FR1/1 = "Obtaining configuration information", FR1/2 = "Obtaining data from source data base", FR1/3 = "Obtaining data from import files", FR1/4 = "Logging faulty import file", FR1/5 = "Importing data in target data base", and FR1/6 = "Logging import status".

Since actors in a use case diagram show interaction between the system and external systems or entities, they are explicitly obtained from the TFM – actors are entities from FFs and the set of all actors involved in use cases is obtained from all the FFs that build up the TFM. The developed topological use case diagram is given in Fig. 4.

In this example, use cases are applied to model FRs; therefore, the use cases define the number and the scope of sequence diagrams. The scope of sequence diagrams defines a set of FFs, which are included in each sequence diagram. A total set of 7 sequence diagrams is created. The sequence diagram for use case "Importing data in target data base" (correspondingly to FR1/5) is given in Fig. 6. As FR1/5 mappings include also FR1/6, the corresponding sequence diagram contains a use of *ref* interaction to sequence diagram "Logging import status". FFs, the diagram maps (see Section V. B), are explained in TableI.

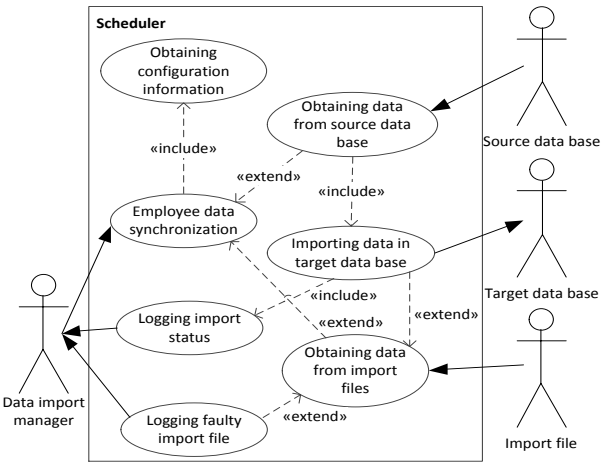

Fig. 4. Use case diagram of the enterprise data synchronization system (borrowed from [39]).

D. Developing the Structural View

The main goal of developing the structural view is to get a topological class diagram, which contains classes with their responsibilities, and, in essence, is a frame of software architecture. To identify classes and to assign the correct responsibilities to each of them, the TFM is transformed into the communication diagram, which then is transformed into the topological class diagram. To obtain a communication diagram, it is necessary to detail each functional feature of the TFM to a level, where it uses only one type of objects. The developed communication diagram that represents data synchronization with the source database is given in Fig. 5.

The resulting topological class diagram is considered initial, since it contains classes (with attributes and responsibilities) and topological relations between them. The operations are obtained during TFM transformation to the communication diagram, and the attributes are added from the TFM while transforming the communication diagram into the topological class diagram. Responsibilities of classes are assigned as operations. To add other relationship types (e.g., associations, generalizations, and dependencies), as well as required and provided interfaces in accordance with inputs and outputs of the TFM, to the initial topological class diagram, it should be refined in accordance with the steps given in [17].

The refined topological class diagram of the enterprise data synchronization system is given in Fig. 7, where the main functioning cycle is denoted with topological relationships (bolder arrows).

VI. Conclusion

This paper discusses a question on development of the qualitative solution, i.e., well-designed software architecture. TopUML makes achievement of architectural views and their interdependencies more formal, clear and traceable back- and for-ward.

The well-designed software architecture has several common quality attributes described in Section II. TopUML forces architectural views to have them. First, TopUML leads architectural views to be constructed in well-defined layers of abstraction. It clearly separates specific layers of abstraction. Second, TopUML allows separating the interface and implementation. The use cases are explicitly obtained from the domain TFM and clearly states actors and their communication points with software. And third, TopUML provides formal general mechanisms for determination of common and different behavioral patterns and structure, as well as responsibilities, starting from transformation of the TFM to the topological use case and communication diagrams. Simultaneously, common functionality and classes (or components), which are responsible for that, are defined.

Further research is related to implementation of the proposed solution by means of the modeling and transformation tools or a tool chain.

TABLE I

THE PART OF FFS DEFINED FOR THE ENTERPRISE DATA SYNCHRONIZATION SYSTEM

ID	Object action	Precondition
8	Importing every row from an internal table into the target data base	
19	Checking if data from a particular row already exists in the target data base	
20	Updating existing data in the target data base	If data from the particular row exist
22	Inserting new data in the target data base	If data from the particular row do not exist
24	Creating a log file in a log file folder for import file processing	If data are read from an import file
25	Logging data row from a temporal internal table	
26	Logging successful status	If import is successful for a particular row
27	Logging error status	If import is not successful for a particular row
28	Logging error description	If an error is logged
29	Archiving a log file	If data import is completed

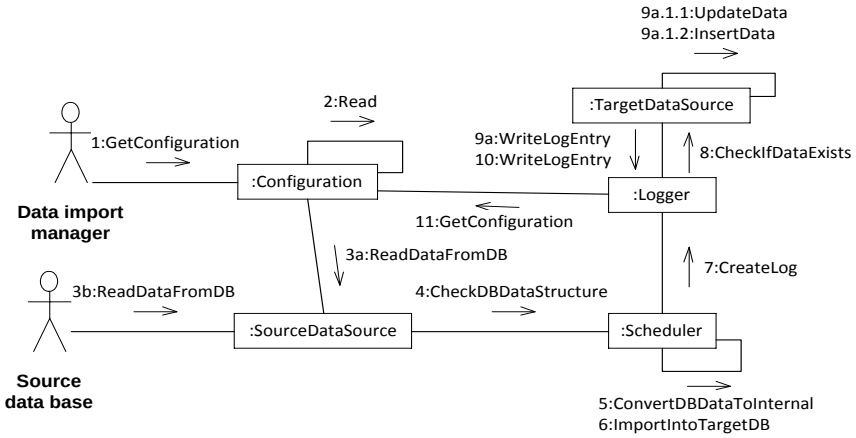

Fig. 5. Communication diagram representing data synchronization with the source data base (borrowed from [39]).

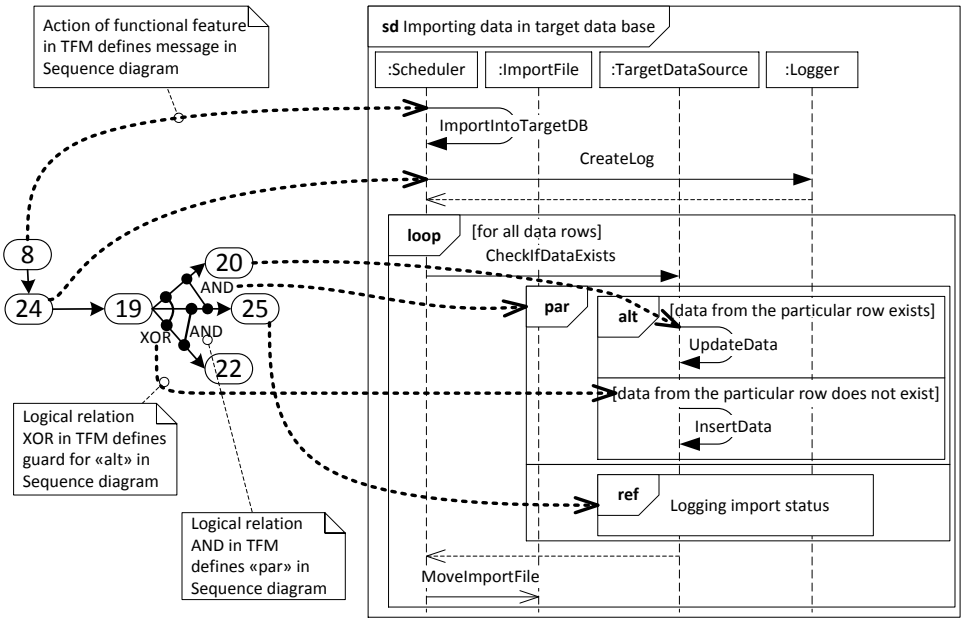

Fig. 6. Fragment of TFM (on the left side), Sequence diagram "Importing data in target data base" and trace links from TFM elements to the Sequence diagram elements (borrowed from [39]).

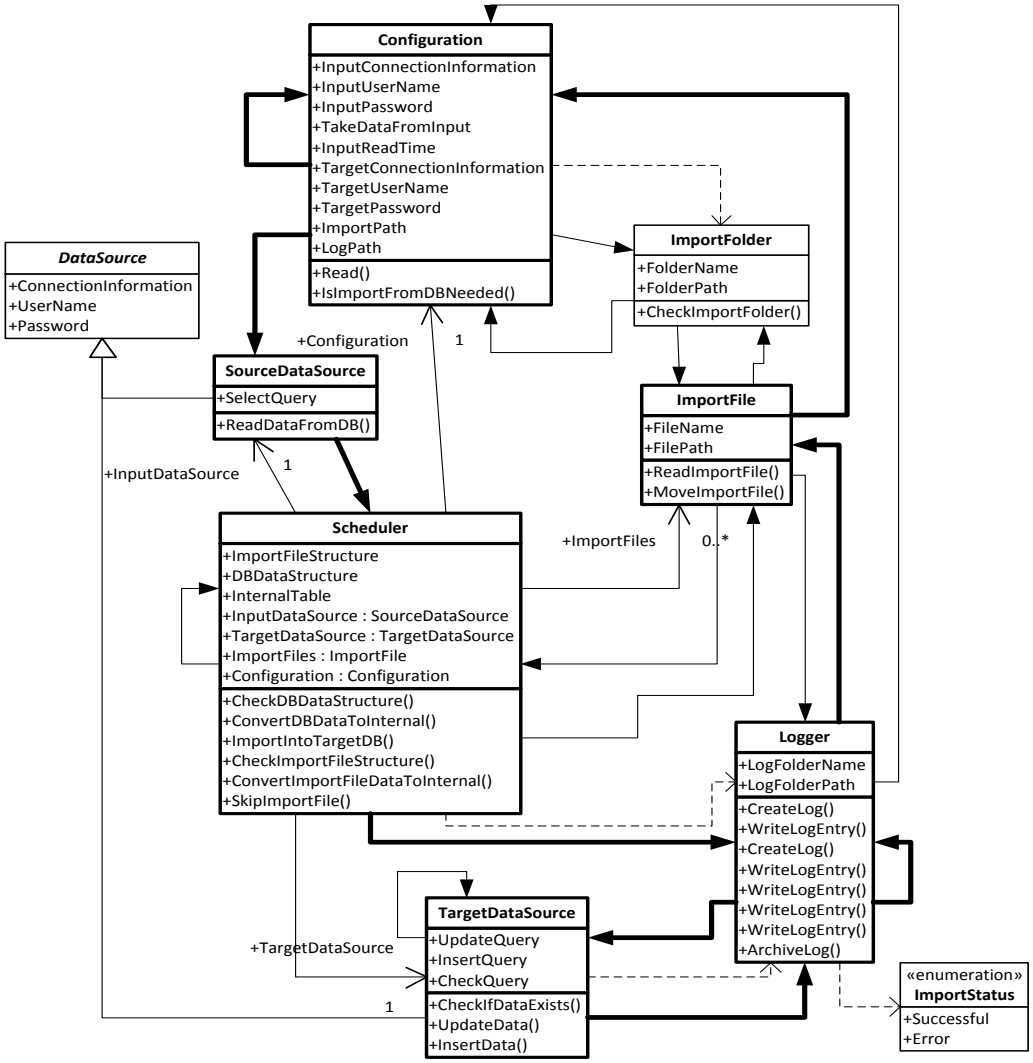

Fig. 7. Topological class diagram of the enterprise data synchronization system (borrowed from [39]).

REFERENCES

[1] C. Jones, "Positive and Negative Innovations in Software Engineering," *International Journal of Software Science and Computational Intelligence*, vol. 1, no. 2, pp. 20–30, April–June 2009. http://dx.doi.org/10.4018/jssci.2009040102

[2] J. A. Zachman, "A framework for information systems architecture," *IBM SYSTEMS JOURNAL*, vol. 26, no. 3, pp. 276–292, 1987. http://dx.doi.org/10.1147/sj.263.0276

[3] J. Osis and E. Asnina, "Is Modeling a Treatment for the Weakness of Software Engineering?" in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, Hershey – New York, IGI Global, 2011, pp. 1–14. http://dx.doi.org/10.4018/978-1-61692-874-2.ch001

[4] J. Osis and J. Silins, "Topological Function-Architecture Co-Design of Embedded Systems," in *Advances in Databases and Information Systems. 13th East-European Conference, ADBIS 2009: Associated Workshops and Doctoral Consortium, Local Proceedings*. Riga, Latvia, 2009.

[5] J. Osis and E. Asnina, "A Business Model to Make Software Development Less Intuitive," in *Proceedings of 2008 International Conference on Innovation in Sofware Engineering (ISE 2008)*. December 10–12, 2008, Vienna, Austria, 2008.

[6] J. Osis, "Extension of Software Development Process for Mechatronic and Embedded Systems," in *Proceeding of the 32nd International Conference on Computer and Industrial Engineering*, Limerick, Ireland, 2003.

[7] E. Asnina, "The Formal Approach to Problem Domain Modelling Within Model Driven Architecture," in *Proceedings of the 9th International Conference "Information Systems Implementation and Modelling" (ISIM'06)*, Přerov, Czech Republic, 2006.

[8] U. Donins, "Software Development with the Emphasis on Topology," in *Advances in Databases and Information Systems. Lecture Notes in Computer Science*, vol. 5968, Berlin, Germany, Springer-Verlag, 2010, pp. 220–228. http://dx.doi.org/10.1007/978-3-642-12082-4_28

[9] A. Kleppe, J. Warmer and W. Bust, *MDA Explained. The Model Driven Architecture: Practice and Promise*, Upper Saddle River, NJ, USA: Addison-Wesley, 2003.

[10] J. Osis, E. Asnina and A. Grave, "Formal Computation Independent Model of the Problem Domain within the MDA," in *Information Systems and Formal Models, Proceedings of the 10th International Conference ISIM'07*, Opava, Czech Republic, 2007.

[11] J. Osis, E. Asnina and A. Grave, "Computation Independent Modeling within the MDA," in *Proceedings of IEEE International Conference on Software, Science, Technology & Engineering (SwSTE07)*, 30–31 October, 2007, Herzlia, Israel, 2007.

[12] J. Osis, E. Asnina, A. Grave, "Formal Problem Domain Modeling within MDA," *Communications in Computer and Information Science (CCIS)* vol. 22, Software and Data Technologies, Springer-Verlag Berlin Heidelberg, 2008, pp. 387–398. http://dx.doi.org/10.1007/978-3-540-88655-6_29

[13] J. Osis, E. Asnina and A. Grave, "MDA Oriented Computation Independent Modeling of the Problem Domain," in *Proceedings of the 2nd International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2007)*, Barcelona, Spain, 2007.

[14] E. Asnina and J. Osis, "Topological Functioning Model as a CIM-Business Model," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*. Hershey, New York, USA, IGI Global, 2011, pp. 40–64. http://dx.doi.org/10.4018/978-1-61692-874-2.ch003

[15] J. Osis and E. Asnina, "Derivation of Use Cases from the Topological Computation Independent Business Model". In *Model–Driven Domain Analysis and Software Development: Architectures and Functions*. Hershey, New York, USA, IGI Global, 2011, pp. 65–89. http://dx.doi.org/10.4018/978-1-61692-874-2.ch004

[16] U. Donins and J. Osis, "Topological Modeling for Enterprise Data Synchronization System: A Case Study of Topological Model-Driven Software Development," in *Proceedings of the 13th International Conference on Enterprise*, Beijing, China, 2011.

[17] U. Donins, J. Osis, A. Slihte, E. Asnina and B. Gulbis, "Towards the Refinement of Topological Class Diagram as a Platform Independent Model," in *Model-Driven Architecture and Modeling-Driven Software Development: ENASE 2011, 3rd Whs. MDA&MDSD*, Beijing, China, 2011.

[18] B. Dobing and J. Parsons, "Dimensions of UML Diagram Use: Practitioner Survey and Research Agenda," in *Principle Advancements in Database Management Technologies: New Applications and Frameworks*, Hershey, New York, USA, Information Science Reference, 2010, pp. 271–290.

[19] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, 2nd ed., Upper Saddle River, NJ, USA: Addison-Wesley, 2005.

[20] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed., Upper Saddle River, NJ, USA: Prentice Hall, 2005.

[21] H. Podeswa, *UML for the IT Business Analyst*, 2nd ed., Boston, MA, USA: Course Technology PTR, 2009.

[22] P. Stevens and R. Pooley, *Using UML: Software Engineering with Objects and Components*, 2nd ed., Harlow, England: Addison-Wesley, 2005.

[23] OMG, "Unified Modeling Language Superstructure Version 2.4.1," [Online]. Available: http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/.

[24] G. Booch, R. Maksimchuk, M. Engel, B. Young, J. Conallen and K. Houston, *Object-oriented analysis and design with applications*, 3rd ed., Upper Saddle River, NJ, USA: Addison-Wesley, 2007.

[25] K. Scott, *The Unified Process Explained*, Upper Saddle River, NJ, USA: Addison-Wesley, 2001.

[26] O. Nikiforova, "System Modeling in UML with Two-Hemisphere Model Driven Approach," *Scientific Proceedings of Riga Technical University, Computer Science (Series 5), Applied Computer Systems*, vol. 43, pp. 37–44, 2010.

[27] T. Loton, *UML Software Design with Visual Studio 2010*, Breinigsville, PA, USA: LOTONtech Limited, 2010.

[28] M. Turner, *Microsoft Solutions Framework Essentials: Building Successful Technology Solutions*, Redmond, Washington, USA: Microsoft Press, 2006.

[29] A. Olive, *Conceptual Modeling of Information Systems*, Berlin, Germany: Springer, 2007.

[30] M. Fowler, *Patterns of Enterprise Application Architecture*, Upper Saddle River, NJ, USA: Addison-Wesley, 2002.

[31] Keywood Computer Services Limited, "Problem Analysis and the Problem Frames Approach," 2005. [Online]. Available: http://www.jacksonworkbench.co.uk/stevefergspages/pfa/index.html. [Accessed June 2013].

[32] A. van Lamsweerde, "Requirements Engineering: From Craft to Discipline," in *Proc. FSE'2008: 16th ACMSigsoft Intl. Symposium on the Foundations of Software Engineering*, Atlanta, November 2008, 2008.

[33] E. Yu, "Modelling Organizations for Information Systems Requirements Engineering," in *Proc. RE'93 – 1st Intl Symp. on Requirements Engineering*, 1993.

[34] J. Osis, "Topological Model of System Functioning," in *Automatics and Computer Science*, J. of Acad. of Sc., no. 6, pp. 44–50, 1969.

[35] J. Grundspenkis, "Fault Localisation Based on Topological Feature Analysis of Complex System Model," in Diagnostics and Identification, pp. 38–48, 1974.

[36] J. Osis and E. Asnina, *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, Hershey, New York, USA: IGI Global, 2011. http://dx.doi.org/10.4018/978-1-61692-874-2

[37] J. Osis and U. Donins, "Formalization of the UML Class Diagrams," in *Evaluation of Novel Approaches to Software Engineering, Berlin*, Springer-Verlag, 2010, pp. 180–192. http://dx.doi.org/10.1007/978-3-642-14819-4_13

[38] J. Osis and A. Slihte, "Transforming Textual Use Cases to a Computation Independent Model," in *Proceedings of the 2nd InternationalWorkshop on Model-Driven Architecture and Modeling Theory-Driven Development MDA & MTDD 2010, In conjunction with ENASE 2010*, Athens, Greece, July 2010, Portugal, 2010.

[39] U. Doniņš, "Topological Unified Modeling Language: Development and Application," Doctoral Thesis. Thesis. Rīga: RTU, 2012.

[40] J. Osis and E. Asnina, "Topological Modeling for Model-Driven Domain Analysis and Software Development," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, Hershey, New York, USA, IGI Global, 2011 d, pp. 15–39. http://dx.doi.org/10.4018/978-1-61692-874-2.ch002

[41] U. Donins, "Semantics of Logical Relations in Topological Functioning Model," Portugal: SciTePress, 2012.

Janis Osis graduated from the Latvian State University cum lauda and received a diploma of Electrical Engineering in Electrical Systems (1953); Dr. sc. ing. in Automatics from Kaunas Technological University, Lithuania (1961); Dr. habil. sc. ing. in System Analysis from the Latvian Academy of Sciences (1972).

He has been a Professor at the Faculty of Computer Science and Information Technology of Riga Technical University, Latvia since 2001. He is an honorary member of the Latvian Academy of Sciences. The list of publications contains more than 250 titles, including 15 books.

Since 1965 his research interests have been topological modeling of complex systems. Recent fields of interest are object-oriented system development, formal methods of software engineering, software development within the framework of MDA by means of topological functioning model support.

Address: Meža Str. 1/3, Riga, LV-1048, Latvia; E-mail: Janis.Osis@rtu.lv

Erika Asnina received a Doctoral Degree (Dr. sc. ing.) in Information Technology with specialization in system analysis, modeling and design in 2006 from Riga Technical University.

She has been an Associate Professor at the Department of Applied Computer Science of Riga Technical University since 2013. She also worked as a Software Developer. She is the author of 37 conference papers, 4 book chapters and 1 book. Her research interests include software quality assurance, model-driven and object-oriented software engineering.

The Latvian Academy of Sciences has awarded her and a co-author, Janis Osis, for the book "Model-Driven Software Development: Architectures and Functions" in 2011.

Address: Meža Str. 1/3, Riga, LV-1048, Latvia;
E-mail: erika.asnina@rtu.lv

Uldis Donins received a Doctoral Degree (Dr. sc. ing.) in Information Technology with specialization in system analysis, modeling and design in 2012 from Riga Technical University.

He is a Leading Researcher at the Department of Applied Computer Science of Riga Technical University and a Leading Architect at Software Development Division of Lattelecom Technology Ltd. He has also worked as a Software Developer. The overview of his research is published by Springer-Verlag in series of LNCS: U. Donins, "Software Development with the Emphasis on Topology," in *Lecture Notes in Computer Science*, Vol. 5968, Advances in Databases and Information Systems, J. Grundspenkis, M. Kirikova, Y. Manolopoulos, L. Novickis, Berlin: Springer-Verlag, 2010, pp. 220–228.

He was awarded the "Werner Von Siemens Excellence Award 2008" for his Master Thesis "Reconciling Software Requirements and Architectures within MDA by means of Topological Functioning".

Address: Meža Str. 1/3, Riga, LV-1048, Latvia;
E-mail: uldis.donins@rtu.lv

Vicente García-Díaz is an Associate Professor at the Computer Science Department of the University of Oviedo. He has a PhD from the University of Oviedo in Computer Engineering. His research interests include model-driven engineering, domain specific languages, technology for learning and entertainment, project risk management, software development processes and practices. He has graduated in Prevention of Occupational Risks and is a Certified Associate in Project Management through the Project Management Institute.

Address: Edificio de la Facultad de Ciencias. C/ Calvo Sotelo s/n. 33007 Oviedo (Asturias, España); E-mail: garciavicente@uniovi.es