Scientific Journal of Riga Technical University
Computer Science. Applied Computer Systems
2011
Volume 47

Automated Derivation of Use Case Model from Topological Functioning Model

Timofejs Murenecs[1], Erika Asnina[2], [1-2]*Riga Technical University*

***Abstract* - The first important step in Model Driven Architecture software development is qualitative analysis and specification of structure and behavior of business and its supporting information system as well as software requirements. We continue the research on achieving a qualitative software requirements model, Use Case Model (UCM), based on a formal business model, Topological Functioning Model (TFM), by using formal model transformations. This paper discusses the results of implementation of the transformation from TFM to UCM by using Query/View/Transformation Relations supported by mediniQvt.**

Keywords: Model Driven Architecture, model transformation, topological functioning model, use case model

I. Introduction

A software system has an impact on organization's work. In most cases, skipping proper analysis of the business due a lack of time, finance, and efforts increases development costs. Thus, understanding of the business, organization's work and structure becomes critically important. Models that specify the business should be clear both for software developers and business people.

Model Driven Engineering (MDE) is based on model development and transformations [1]. Model Driven Architecture (MDA), developed by the Object Management Group (OMG), is an MDE based software development approach, which specifies a model transformation process chain that can be performed using transformation languages. MDA standard provides a possibility of transformation from Computation Independent Model (CIM) to Platform Independent Model (PIM), which can be transformed to Platform Specific Model (PSM) that is a source for automatically generated program code. The CIM is a domain model that should eliminate the gap between business people and software developers. The PIM specifies system structure and behavior, but does not show platform-specific details. The PSM specifies system structure and behavior enhancing the PIM with platform-specific details.

As previously mentioned, the CIM is a domain model that should reflect both problem domain and solution domain. The previous research [3] demonstrated that three parts or layers of the CIM exist. They are CIM – Knowledge Model that reflects an enterprise from the holistic point of view, thus providing the general vision of the enterprise with focus on enterprise knowledge; CIM – Business Model that is focused on the business scope and goals and does not reflect software system considerations; and CIM – Business Requirements for the System that contains the contract between the business and IT about what the business people expect the IS will automate and is built on and refers to the CIM–Business Model.

Different models and diagrams represent the CIM. Most of them also describe the business domain in a fragmentary way, from different aspects, and informally [2]. Because of commonly informal nature of the CIM, MDA proposes *manual* transformation of the CIM.

Formalization of the CIM and automated transformation of the CIM are a challenge for researchers. The previous research on the formalization of the CIM described in [3], [4], [5], [6], [7] and others resulted in the understanding that Topological Functioning Model (TFM) can be used as a *formal holistic* computation independent *domain* model.

The notion "holistic" is explained in [3]: "Holistic representation may be described either as a *single indivisible (or formally refined) model* or as *a view on the system on the whole from different aspects*". Formalism in the definition means that dependencies among elements in different aspects are formally defined. The TFM is a *holistic* model in the former sense. It indivisibly specifies information from three business modeling domains, namely, the functional domain, the organizational domain and the information domain [8]. The TFM is located at the CIM-Business Model level and can specify the problem and the solution in formal conformance [3].

In turn, the CIM- Business Requirements for the System level usually is described by a use case model (UCM), because it specifies functional requirements for the software system (for the solution). Use cases are simple, and this is their main advantage. But at the same time, the informal nature of use cases and a lack of a solid background (i.e., a lack of compliance with a domain model) are causes of core limitations of use cases described in [9], [10].

The TFM is *formal*, it has "simple" mathematics that could be understand also by non IT professionals. This model is a formal base for derivation of qualitative use cases. It allowed getting a solid background while keeping simplicity of use cases. However, such derivation is *manual.* It is described *theoretically* in [7]. It would be much more valuable, if this process is automated.

This paper continues that research and demonstrates first results of *automated* transformation from a business domain model, TFM, to a software requirements model, UCM, in the context of MDA.

The goal of this research is to determine whether it is possible to create complete automated transformation from TFM to UCM by using OMG standard transformation language.

The tasks stated are the following: 1) choose one of the QVT transformation languages and its supporting open-source tool, 2) update/modify and implement the TFM metamodel defined in [16], 3) specify transformation rules in the chosen language, 4) check the completeness of the transformation result on the example, and 5) determine the appropriateness of the chosen language and define future research direction.

This paper is organized as follows. Section II discusses OMG transformation languages. Section III describes a technique which is used to achieve TFM to UCM transformation, and illustrates implementation steps, namely, a developed TFM metamodel and QVT transformation code. Section IV presents an example of transformation from TFM to UCM. Section V discusses research results, and conclusions state directions of further research.

II. OMG Model Transformation Language

Transformation from one model to another is not useful without process automation; therefore usage of special languages and tools is needed.

Query/View/Transformation (QVT) is a standard transformation language used within MDA. It consists of three languages – two declaratives, QVT Relations and QVT Core, and one imperative, QVT Operational Mappings. The declarative languages are characterized by shorter constructs and embedded tracing mechanisms. In turn, the imperative language is more flexible, but its weakness is longer constructs and a lack of embedded tracing mechanisms [15]. There are many model transformation languages that borrow their concept from OMG, namely ATL, MOLA, Kermeta, Tefkat, QVT discussed in the paper, and others. Basically nowadays all transformation languages *are not complete* and are appropriate only for specific field [20]; therefore there is no single solution for every problem. QVT language is not discussed well practically, with realization of nontrivial problems. However, it has the strong theoretical basis that can be used to apply in the solution of a particular problem.

A decision to apply QVT Relations at the beginning of our research was made, because studying the specification of this language showed that it could be appropriate for achievement of our goal.

This paper demonstrates model transformations that mediniQVT tool supports. This tool is developed by "ikv++ technologies". The vendor stated in [12] that "mediniQVT implements OMG's QVT Relations specification in a powerful QVT engine. The standard is designed for model to model transformations to allow fast development, maintenance and customization of process specific transformation rule". It also has advantages in comparison with other QVT Relations tools [13].

III. Implementation of Transformation from TFM to UCM with mediniQVT

The TFM serves as a formal basis for identification of use cases by using goals. We suggest that users of software system and their responsibilities are linked with functional goals. The general method suggested for this purpose includes three steps as mentioned in [6] and described in details in [7]:

1) Identify users, their goals and associated functionality. Identification of goals is identification of a set of functional features necessary for satisfying the concrete goal;

2) Identify and refine use cases. In order to describe steps, conditions and constraints of a use case, a textual specification can be used;

3) The refinement of use cases. It is formal identification of inclusion and extension use cases by using intersections of defined sets of functional features needed for achievement of the goal.

The usual way of model to model transformations is transformation by means of metamodels – by setting transformation mappings from one set of metamodel to another. Therefore, completely elaborated metamodels are needed for transformation.

MediniQVT has a set of needed functions that include metamodel creation in .ecore extension, model creation based on a developed metamodel, transformation creation using the QVT engine, OCL extension support.

Hereafter we describe source and target models in brief, investigate metamodels of the TFM and the UCM, and set model transformation rules.

A. Source Model: Topological Functioning Model

TFM holistically describes system's functionality and can serve as a formal basis for checking of completeness and consistency of requirements, as well for identification of all types of use cases by using several manual model transformations informally defined in [16], refined in [6] and [19].

The TFM has a solid mathematical base. It is represented in a form of a topological space (X, Θ), where X is a finite set of functional features of the system under consideration, and Θ is topology that satisfies axioms of topological structures and is represented in a form of a directed graph [4]. The necessary condition for construction of a topological space is a meaningful and exhaustive verbal, graphical, or mathematical system description. The adequacy of the model can be achieved by analyzing its mathematical properties [4].

The TFM has topological (connectedness, closure, neighborhood, and continuous mapping) and functioning (cause-effect relations, cycle structure, and inputs and outputs) characteristics. The main feedbacks in the system are represented as main functioning cycles. They visualize the "main" functionality that has vital importance for the system life. Proper analysis of cycle structures enables careful analysis of system operation and communication with the environment [5].

The most complete explanation of the TFM is given in [18].

B. Source Metamodel: Metamodel of the TFM

A metamodel of the TFM for MDA was introduced by Asnina in [16]. This metamodel is suitable for creating any TFM model in general scope. The introduced metamodel uses a combination of MOF and UML profile metamodeling strategies in order to provide a complete picture of the TFM

[16]. MediniQVT uses EMF .ecore extension to describe source metamodels so it is not possible to use previously created metamodelis. Therefore, this new metamodel is introduced.

The introduced metamodel is a lightweight version of the metamodel presented by Asnina, where main aspects of TFM remain. However, it is adapted for mediniQVT tool usage. For instance, SharedBlock element is added, due to the fact that one TFM feature can be used in various goals, also GoaltoShared element is added, to combine logic of goal elements. These two elements do not fit the scope of the TFM, but are obligatory for UCM derivation from a TFM model.

The main element of the metamodel is TFMFeature, which is defined as unique tuple <A, R, O, PrCond, PostCond, Pr, Ex> [6]:

- A is an object's action
- R is a result of this action (optional),
- O is an object or objects that receive the result or that is used in this action (for example, a role, a time period, a catalog, etc.),
- PrCond is a set PrCond = f(c1, …, ci), where ci is a precondition or an atomic business rule (it is an optional element),
- PostCond is a set PostCond = f(c1, …, ci), where ci is a post-condition (it is an optional element),
- Pr is a set of entities (systems or subsystems) which provide or suggest an action with a set of certain objects;
- Ex is a set of entities (systems or subsystems) which enact a concrete action.

An association element also is introduced. GoalToShared element is a special association between TFM features that exist in one goal, this association can be transferred to UCM "inclusion" association.

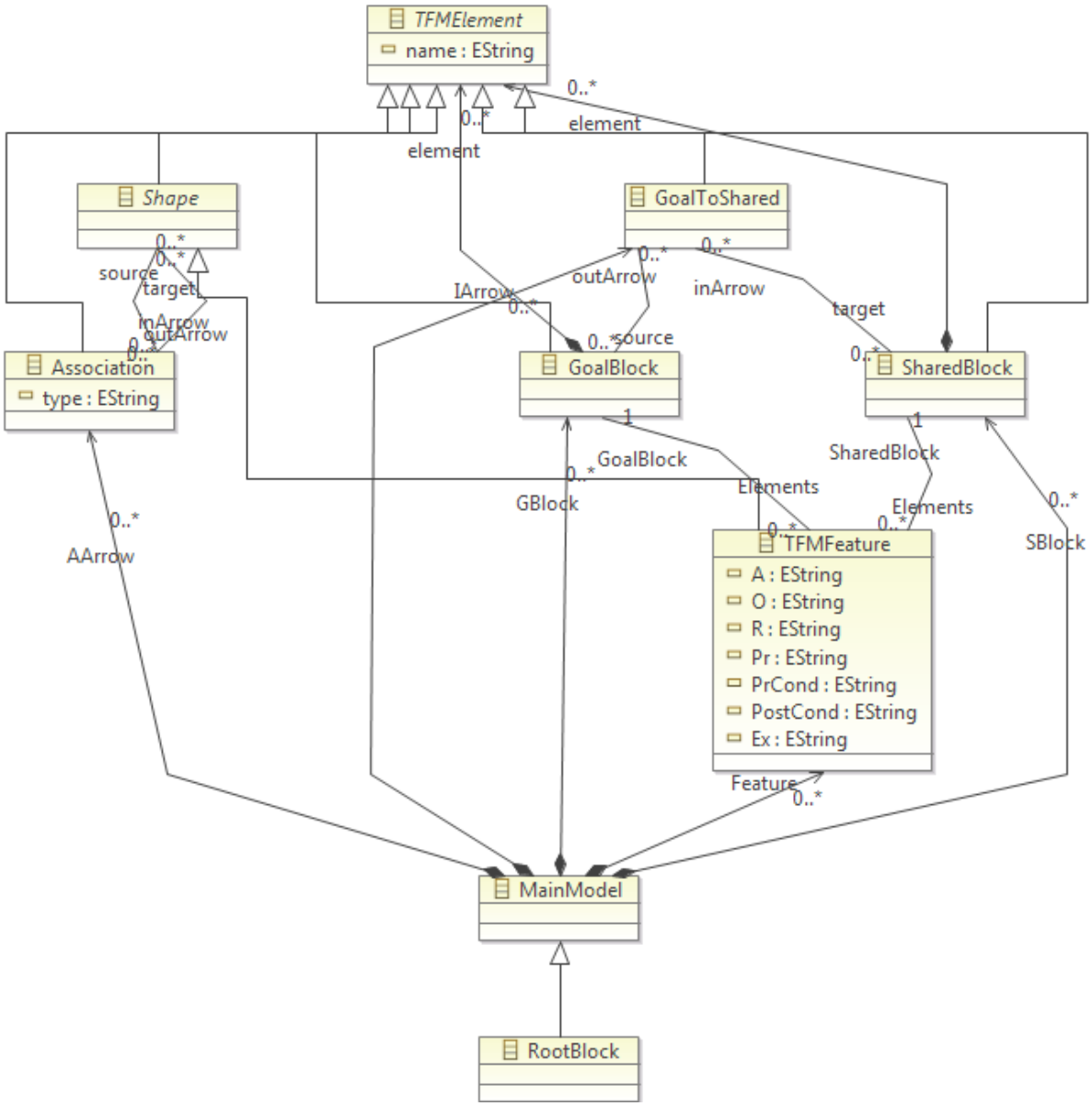

Fig. 1. Metamodel of the Topological Functioning Model in .ecore format

A cause-and-effect relationship cycle element is not defined, due to the fact that using QVT Relation language in MediniQVT, it is not possible to check cycle structure in model. But it is still required, like [16] maintains.

Figure 1 illustrates the created metamodel of the TFM that is based on the model specification described above. The metamodel is stored in a file with .ecore extension, which is an extension of Eclipse Modeling Framework (EMF). According to the Eclipse Foundation, the core EMF framework includes a metamodel (Ecore) for describing models. It describes metamodels and persistence support with default XMI serialization. mediniQVT can be used like a standalone or an Eclipse plugin tool, because of that it supports .ecore extension, but cannot create diagram files. That is why EMF can be used for diagram creation.

The TFM metamodel consists of nine classes:

- RootBlock – a starting point of the model, the basic element that includes all TFM features.
- MainModel – a class that contains references to all existing objects in the model.
- TFMElement – a class that represents a property name of the created model elements.
- Shape – a class that contains a references to Association Class, enabling connectedness of TFMFeatures.
- TFMFeature – it is a TFM functional feature with properties <A, R, O, PrCond, PostCond, Pr, Ex>.
- Association – a class that describes the association model element between TFM features.
- GoalBlock – this kind of blocks contains TFM features that relate to the defined system goal.
- Shared block – this kind of block contains TFM features that relate to the shared functionality of several goals.
- GoalToShared – this class represents an association between a goal and a shared block.

C. Target Model: Use Case Model

Use cases, which were introduced by Ivar Jacobson, have become a very popular technique of the Object-oriented analyses (OOA) from the end of 1990s [11]. After some elaboration, use cases have become one of the fundamental techniques used in the OOA. G. Schneider and J.P. Winters defined use cases as "a behavior of the system that produces a measurable result of value to an actor" [8].

The goal of user's communication with the system is to achieve that "measurable result of value". Besides that, the reason and quality of using goals as criteria for identification of use cases is thoroughly investigated in [16], [19].

There are three concepts associated with use case models – actors, use cases and the subject. The latter one defines the system planned to be built. Actors always represent entities that are outward the subject, i.e. outward the system under consideration. Use cases define offered behavior of the subject without reference to its internal structure. This means that they implement the so-called "black box" approach. The same use case can be associated to several subjects [17].

Use cases are connected with actors using communication associations that show direction of use case initiation and who initiates them – an actor, the system, or both. Use cases can be connected with extend, include and generalization relationships. An extend relationship is used to *conditionally* extend the behavior of the extended use case. An include relationship between use cases defines that the included behavior is *always* required for the including use case to be executed correctly.

D. Target Metamodel: Metamodel of UCM

The OMG provides a metamodel for all UML models [14], which can be used for transformation purposes. This metamodel contains UCM metamodel as well; therefore it is not necessary to create this metamodel manually.

E. Model Transformation Rules

A transformation algorithm for derivation of UCM from TFM in general is provided in Figure 2. It can be used for manual as well as automated transformation [16].

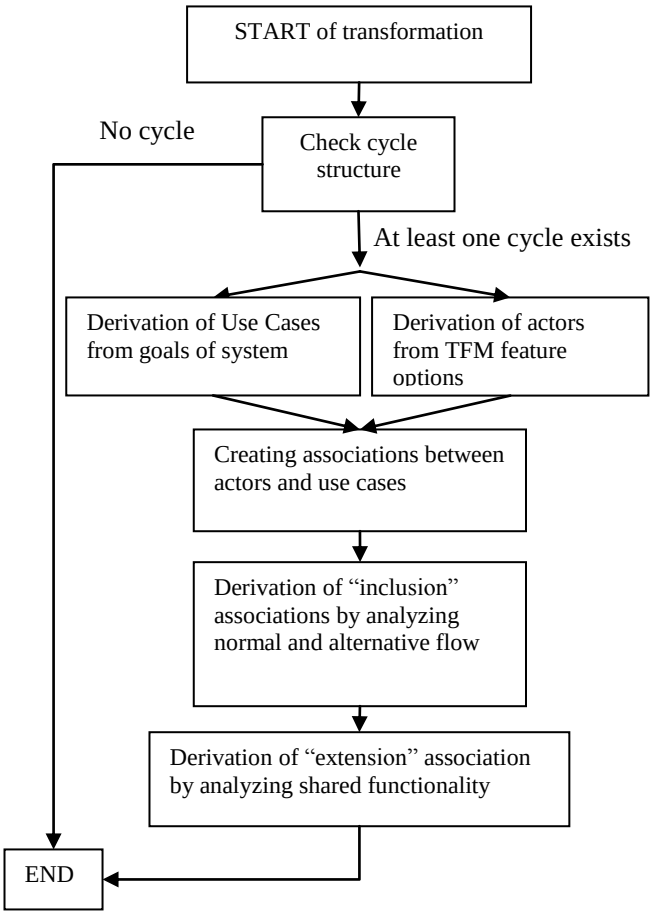

Fig. 2. The transformation algorithm from TFM to UCM

The transformation algorithm implemented with mediniQVT starts with definition of source and target metamodels, in our case, TFM and UCM metamodels:

```
transformation TFM2UseCase(source:TFM, target:uml){
    /*transformation body*/
}
```

Check of the cycle structure of the TFM is not implemented in this particular case, because it is not possible due to restrictions of QVT Relations language syntax.

This transformation consists of 7 model mappings (or mapping transformation rules) – top relations, where one model element is mapped to another model element. It also has one OCL query that can be invoked from any place in the transformation.

The first transformation rule creates an element "Model", where all sub-elements will be stored. It is important, because all .uml files (files of UML diagrams) must have a root element. The root element can be a package, a model or a profile.

```
  top relation MainBlock2UseCaseModel {
    enforce domain source root:TFM::MainModel{
    };

    enforce domain target model:uml::Model {
    };
  }
```

The second transformation rule creates a use case element from goal block element GoalBlock with the same name. In the source domain of transformation variable assignment takes place. Note that 'when' element indicates that this transformation is executed only after rule MainBlock2UseCaseModel(root, model).

```
  top relation GoalBlock2UseCase {
    varName: String; /*goal name, that will be
assigned to use case element name*/
    enforce domain source root:TFM::RootBlock{
      GBlock = field:TFM::GoalBlock{
      name=varName}
    };
    enforce domain target model:uml::Model {
      ownedType = case:uml::UseCase{
      name = varName}
    };
    when{
    MainBlock2UseCaseModel(root, model);
    }
  }
```

The third rule creates actors from all TFM functional features that are placed into the goal block.

```
  top relation Feature2Actor{
    varName: String; /*name of Actor*/
    enforce domain source root:TFM::RootBlock{
      GBlock = Fblock:TFM::GoalBlock{
        Elements=Ffeature:TFM::TFMFeature{
        Ex=varName
        }
      }
    };
    enforce domain target model:uml::Model{
      packagedElement = act:uml::Actor{
        name=varName
        }
    };
    when{
      if Ex.oclIsUndefined() then false
      else true
      endif; /*Actor will be created only if TFM
feature's property Ex is assigned*/
    GoalBlock2UseCase(root, model); /*transformation
will be executed only if this transformation is also
executed */
    }
}
```

The fourth transformation rule creates a use case element that is included by another use case. This element is created from the shared block element, which shows functionality that is included by another goal.

```
  top relation SharedBlock2UseCase{
    varName: String;
    enforce domain source root:TFM::RootBlock{
      SBlock = SherBlock:TFM::SharedBlock{
      name=varName
      }
    };
    enforce domain target model:uml::Model{
      ownedType = usecaseI:uml::UseCase{
      name = varName /*name of included use case*/
      }
    };
    when{
    GoalBlock2UseCase(root, model); /*transformation
will be executed only if this transformation is also
executed */
    }
}
```

The fifth transformation rule creates associations between the created actors and created use case element. Associations must have two properties, the first for the source element and the second one for the destination element.

```
top relation Feature2Association{
    enforce domain source root:TFM::RootBlock{
      GBlock = Fblock:TFM::GoalBlock{
        Elements=Ffeature:TFM::TFMFeature{
        }
      }
    };
    enforce domain target model:uml::Model{
      ownedType = assoc:uml::Association{
        ownedEnd = prop1:uml::Property{
        name = 'src',
        type = getStereotype('Actor') /*type
property defines the source or destination element
type. getStereotype() is a query that gets the
source element: Actor type.*/
        },
        ownedEnd = prop2:uml::Property{
        name = 'dst',
        type = getStereotype('UseCase')
/*getStereotype() is a query that gets the
destination element: UseCase type*/
        },
        navigableOwnedEnd = OrderedSet{prop2}
```

```
        /*property that shows the destination
        element for the association */
      }
    };
    when{
    Feature2Actor(root, model); /*transformation
will be executed only if this transformation is
executed too*/
    }
}
```

The sixth transformation rule creates actors that are placed under the shared functionality block.

```
  top relation Feature2Actor_Shared{
    varName: String;
    enforce domain source root:TFM::RootBlock{
      SBlock = Fblock:TFM::SharedBlock{
        Elements = Ffeature:TFM::TFMFeature{
        Ex=varName
        }
      }
    };
    enforce domain target model:uml::Model{
      packagedElement = act:uml::Actor{
        name=varName
      }
    };
    when{
    GoalBlock2UseCase(root, model);
    }
}
```

The seventh transformation rule creates an association that links actors from the shared functionality block to a use case element. The transformation will be executed if the TFM functional feature defines entities which enact a concrete action, i.e., property Ex.

```
  top relation Feature2Association_Shared{
    enforce domain source root:TFM::RootBlock{
      SBlock = Fblock:TFM::SharedBlock{
        Elements=Ffeature:TFM::TFMFeature{
        }
      }
    };
    enforce domain target model:uml::Model{
      ownedType = assoc:uml::Association{
        ownedEnd = prop1:uml::Property{
        name = 'src',
        type = getStereotype('Actor')
        },
        ownedEnd = prop2:uml::Property{
        name = 'dst',
        type = getStereotype('UseCase')
        },
        navigableOwnedEnd = OrderedSet{prop2}
      }
    };
    when{
    if Ex.oclIsUndefined() then false
    else true
    endif;
    Feature2Actor_Shared(root, model);
}
}
```

The query is a utility expression, which defines some operation. This query passes parameter stName that is a name of a needed stereotype, and returns the first entry found.

```
   query getStereotype ( stName : String ) :
Stereotype
   {Stereotype.allInstances()->select
        (x : uml::Stereotype | x.name = stName)-
>asSequence()->first()
   }
```

IV. Transformation Example

Figure 3 shows an example model that is used for test transformation from TFM to UCM by using the transformation algorithm explained above.

In order to illustrate the chain of model transformation rules (or model transformation process), a simplified model of car rental company (CRC) is used. The CRC rents cars to its clients in order to gain the maximum possible profit. Depending on a situation, a client or a renter enters car parameters to search a possible car list in a car register, from which the client can choose a necessary variant for renting. If any suitable car is not found in the result list, the client can enter other parameters. By selecting a needed car for renting, the renter can complete the rent procedure of the corresponding car and sign a contract with the client. The client can get the car from the renter directly or the car can be delivered to a client destination. When the rent contract is out of date or closed by one of the sides, namely, the client or the renter, a return invoice is created. The renter prints the return invoice to the client, but the client is able to return the rented car. The client can choose one of payment ways or pay for renting directly to the renter. The renter completes payment and creates a rent report. Besides that, the renter can create a general rent report.

The corresponding functional features of the TFM are explained in Table 1. Each functional feature is described accordingly to the tuple <A, R, O, PrCond, PostCond, Pr, Ex> and labeled with a unique number. The analysis stage identified that CRC has three main system goals (achieving of which the software system must implement):

- Goal 1: Rent car. This goal includes functional features 1, 2, 3, 4, 5, 6, and 11.
- Goal 2: Get payment. This goal includes functional features 10, 11, and 12.
- Goal 3: Return rented car. This goal includes functional features 8, 9, and 13.

Cause-and-effect relations denoted by bold arrows form the main functioning cycle “5-6-8-11-5” of the system. The main functioning cycle relates those functional features that are vitally necessary for operation of the system. The main functionality contains rent of cars, return of rented cars and rental payment.

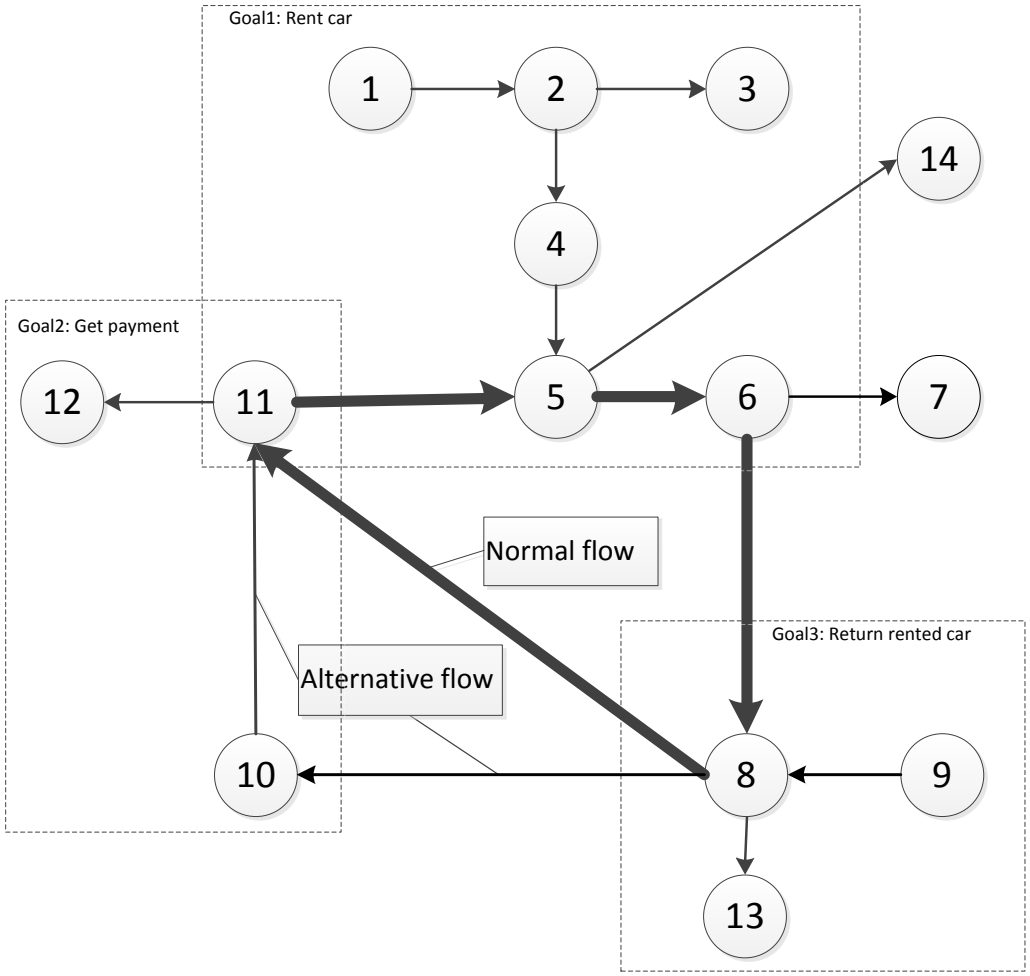

Fig. 3. The TFM of Car Rental Company

TABLE I
FUNCTIONAL FEATURES OF THE TFM OF CAR RENTAL COMPANY

Nr.	TFM feature	A	O	R	PrCond	PostCond	Pr	Ex
1	Entering the car parameters for a request	Enter	Request	Car parameters			CRC	Client, Renter
2	Searching the car list in a car register	Search	Car register	Car list			CRC	Renter
3	Requesting the data about a car	Request	Car	Car data	If the car list is empty		CRC	Renter
4	Selecting the car for a car rent	Select	Car rent	Car	If the car list is not empty		CRC	Client, Renter
5	Completing a car rent	Complete	Car rent				CRC	Renter
6	Signing the contract for a car rent	Sign	Car rent	Contract			CRC	Client, Renter
7	Delivering the car to a client	Delivery	Client	Car			CRC	Renter
8	Creating the return invoice for a car rent	Create	Car rent	Return invoice			CRC	Renter
9	Returning a car	Return	Car				CRC	Client
10	Selecting the way of a payment	Select	Payment	Way			Banking system	Client, Renter
11	Completing a payment	Complete	Payment				Banking system	Renter
12	Creating the report of a payment	Create	Payment	Report			Banking system	Banking system
13	Printing the return invoice for a car rent	Print	Car rent	Return invoice			CRC	Renter
14	Creating a report of car rents	Create	Report of car rents				CRC	Renter

The constructed topological functioning model in Figure 3 satisfies the following topological and functional properties:

• Connectedness. All vertices in the model are interconnected, there are no isolated vertices.

• Cause-and-effect relations. Vertices in the model are linked with arcs that represent cause-and-effect relation semantics.

• Cycle structure. Cause-and-effect relations between vertices form functioning cycles.

• Inputs and outputs. The model has input vertices as well as output vertices. This enables to identify what functionality from the external environment and inside the system generates other functional properties of the system.

• Continuous mapping. This model can be continuously mapped to a more detail model or simplified, while preserving the structure of the model.

The transformation process starts with model creation in mediniQVT tool. The created model of the CRC has .xmi extension and can be transferred to any other tool that supports .xmi extension. Figure 4 shows the model of the CRC created in mediniQVT. Note that only two of nodes are expanded.

Figure 5 shows a derived use case model of the CRC.

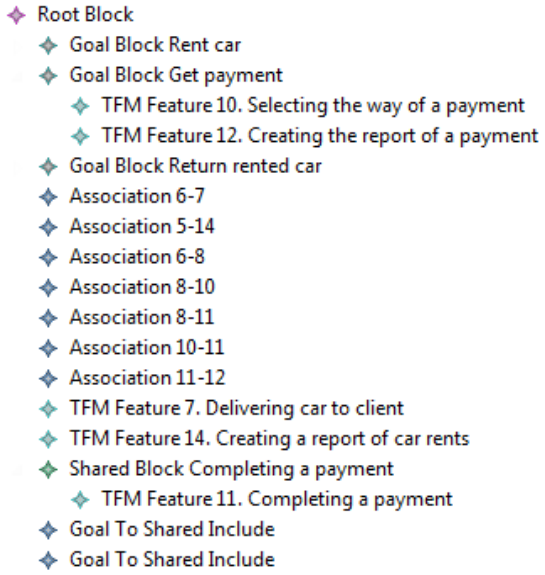

Fig. 4. Model of the CRC created in mediniQVT.

V. Lessons Learned

The obtained (Figure 5) and the desired (Figure 6) use case models are not the same, which means that the transformation is not working like it was desired.

Associations from actors to use cases are not visible. The transformation *does* create associations for every actor and use case, but they cannot be visible in the diagram. Every association has the source and destination property. Properties have types that point to source or destination elements respectively. However, types of properties are not assigned. The reason is that source and destination elements (Actors and Use Cases) are created in another transformation, but QVT Relations syntax realized in mediniQVT has no global variables where pointers on those elements could be kept. It is a reason why types of associations cannot be assigned.

Moreover, inclusion associations cannot be derived. An inclusion association can be created by "Include" property of the use case element, which uses shared functionality. This property needs to contain a type of the included use case. QVT Relations syntax implemented in mediniQVT has no global variables to store included elements, which are created by another transformation rules. Therefore, the inclusion association cannot be created.

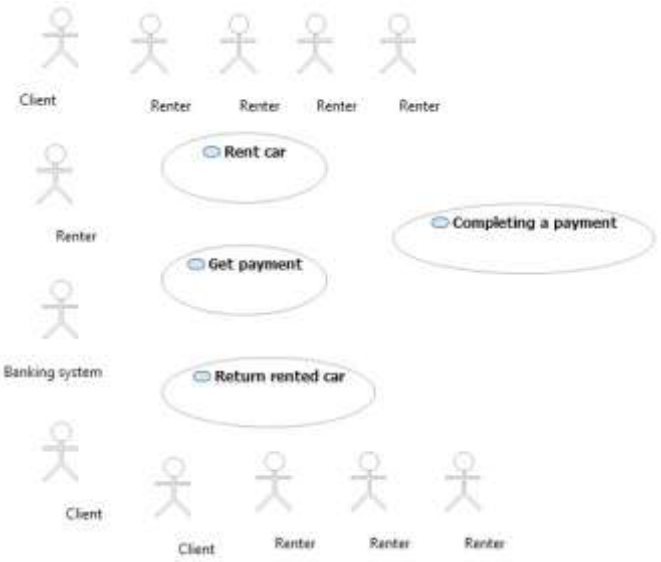

Fig. 5. The UCM of the CRC created by mediniQVT

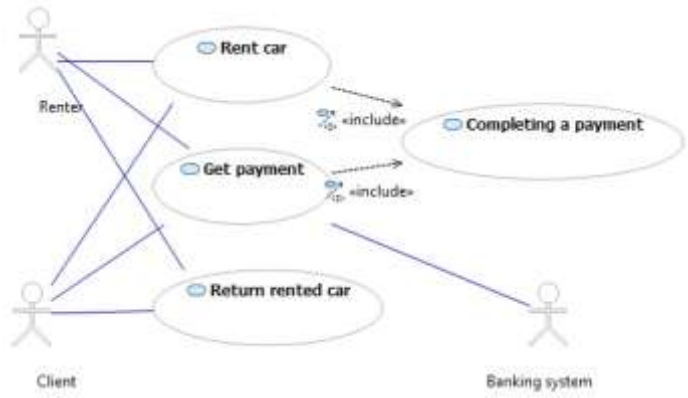

Fig. 6. The desired UCM of the CRC

The same issue also has appeared for use case extensions. Extension associations and use cases that extend functionality cannot be created. The TFM notation defines extension by paths, where normal flow can be outflanked by alternative flow. Standard QVT Relations syntax as well as implemented one in mediniQVT has no any types of loop cycles such as "for", "while" and so on, because it is declarative language. Therefore, there are no ways to check similarities of paths and to derive needed use cases.

VI. Conclusions

This paper discussed a use of QVT Relations language implemented in mediniQVT for deriving the use case model from the topological functioning model. It was determined that mediniQVT and this declarative language itself are not the best way to derive UCM from TFM. The TFM has a complex structure, logical and mathematical background, namely, topological and functioning properties. Its transformation requires careful model analysis that cannot be fully implemented by means of QVT Relations.

Future research directions are related to implementation of use case model and specification derivation from the TFM by means of QVT Operational Mapping. This language is imperative and has additional required constructs. It would be able to describe and realize full analysis of logic of the TFM. Besides that, QVT Operational Mappings supports "Black Box" mechanism in transformations, which means a use of other (programming) languages like Java or C++ within transformations. If problem domain or business logic of the system under discussion is very complicated, TFM model will also be complex. Therefore, a developer must be involved to use case derivation. This can be achieved by using QVT Operational Mappings too. However, the question about the proper implementation of this language is still open, too.

References

[1] D. C. Schmidt, "Model Driven Engineering" *IEEE Computer Society*, February 2006. [Online]. Available: http://www.cs.wustl.edu/~schmidt/GEI.pdf. [Accessed October 3, 2010]

[2] E. Asnina, "A Formal Holistic Outline for Domain Modeling" in *Local Proceedings of Advances in Databases and Information Systems, 13th East-European Conference, ADBIS 2009, Associated Workshops and Doctoral Consortium, Riga, Latvia, September 7-10, 2009*, Riga Technical University, 2009, pp. 400-407.

[3] E. Asnina and J. Osis, "Computation independent models: bridging problem and solution domains." 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 (pp. 23-32). Portugal: SciTePress.

[4] J. Osis, „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) 159-166R.

[5] J. Osis, „Software Development with Topological Model in the Framework of MDA" in *Proceedings of the 9th CaiSE/IFIP8.1/EUNO International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'2004) in connection with the CaiSE'2004*, Vol. 1. Riga: RTU (2004) 211 – 220.

[6] J. Osis and E. Asnina: „A Business Model to Make Software Development Less Intuitive." in *Proceedings of 2008 International Conference on Innovation in Software Engineering ISE 2008*, pp.1240-1245. IEEE Computer Society, Los Alamitos, California, USA (2008)

[7] J. Osis and E. Asnina, "Enterprise Modeling for Information System Development within MDA", *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, IEEE, USA, 2008, p. 490

[8] E. Asnina, J. Osis, "Topological Functioning Model as a CIM-Business Model". *J. Osis, E. Asnina (eds), Model-Driven Domain Analysis and Software Development: Architectures and Functions*, 2011 (pp. 40-64). Hershey, New York, USA: Information Science Reference (an imprint by IGI Global).

[9] S. Ferg, "What's Wrong with Use Cases?" February, 2003. [Online]. Available: http://www.jacksonworkbench.co.uk/stevefergspages/papers/ferg--whats_wrong_with_use_cases.html. [Accessed October 4, 2010]

[10] D. G. Firesmith, "Use Cases: the Pros and Cons." 2003 [Online]. Available: http://www.ksc.com/article7.htm. [Accessed October 4, 2010]

[11] R. Malan and D. Bredemeyer, "Functional Requirements and Use Cases," July 1999. [Online]. Available: http://www.digilife.be/quickreferences/pt/functional%20requirements%20and%20use%20cases.pdf. [Accessed October 3, 2010]

[12] Ikv ++ technologies, "Official ikv++ technologies web site - mediniQVT released" [Online]. Available: http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77 [Accessed October 4, 2010]

[13] T. Murenecs "Review of Query/view/Transformation language realizations" presented at 51th RTU Student conference, Riga, Latvia, 2010.

[14] Object Management Group, "OMG Unified Modeling Language (OMG UML) Infrastructure" OMG standard, May, 2010. [Online] Available: http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/ [Accessed October 4, 2010]

[15] Object Management Group, "Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1.0", OMG standard, April 2008. [Online] Available: http://www.omg.org/spec/QVT/1.0/ [Accessed February 20, 2011]

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

[17] G. Schneider and J. P. Winters, "*Applying Use Cases, 2nd ed. A Practical Guide".* The Addison-Wesley, 2001. – 245 p.

[18] J. Osis, E. Asnina. "Topological Modeling for Model-Driven Domain Analysis and Software Development". *J. Osis, E. Asnina (eds), Model-Driven Domain Analysis and Software Development: Architectures and Functions*, 2011 (pp. 15-39). Hershey, New York, USA: Information Science Reference (an imprint by IGI Global)

[19] J. Osis, E. Asnina, "Derivation of Use Cases from the Topological Computation Independent Business Model". *J. Osis, E. Asnina (eds), Model-Driven Domain Analysis and Software Development: Architectures and Functions*, 2011 (pp. 65-89). Hershey, New York, USA: Information Science Reference (an imprint by IGI Global).

[20] P. Huber, "The Model Transformation Language Jungle - An Evaluation and Extension of Existing Approaches", May 2008, [Online]. Available: http://www.big.tuwien.ac.at/teaching/theses/ma/huber.pdf [Accessed February 23, 2011].

Timofejs Murenecs received B. Sc. in computer control and computer science in 2009 from Riga Technical University.

He is a Scientific Assistant at the Institute of Applied Computer Systems, Riga Technical University, Latvia. He also is a SAP Consultant in the "ecenta Baltic labs", Riga, Latvia. He has one published student conference paper "Review of Query/View/Transformation language realizations", which reflects the first results in his main research interest fields, namely, model-driven software development and model transformation languages.

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

She is a Docent at the Department of Applied Computer Science, Riga Technical University, Latvia. She also worked for 5 years as a Software Developer. She has published 18 conference papers. Her research interests include software quality assurance, business modeling, model-driven software development, model transformation languages and software engineering.

She was awarded as one of the best paper authors at the conferences ISIM'05 and ISIM'06, and a scholarship laureate of the target program "For Education, Science and Culture" of Latvian Education Fund in 2004 and 2005.

Timofejs Murenecs, Ērika Asņina. Automātiska lietošanas gadījumu modeļa iegūšana no topoloģiska funkcionēšanas modeļa

Modeļu vadāmā arhitektūra (Model Driven Architecture, MDA) ir modeļu vadāmas inženierijas (MDE) programmatūras izstrādes pieeja. Tā nosaka modeļu transformācijas procesu, kuru var realizēt, izmantojot transformēšanas valodas, piemēram, Vaicājums/Skats/Transformācijas (Query/View/Transformations, QVT) valodu. Pirmais svarīgais solis MDA pieejā ir kvalitatīvā analīze, kura palīdz noteikt biznesa un atbalstošas sistēmas struktūru, uzvedību un prasības. Raksta autori piedāvā veidot kvalitatīvu prasību modeli, proti, lietošanas gadījumu modeli, pamatojoties uz formālo biznesa modeli, proti, topoloģisku funkcionēšanas modeli (TFM), izmantojot transformāciju. Viena modeļa pārveidošana citā nav lietderīga bez šā procesa automatizācijas, tāpēc ir nepieciešama īpašu transformācijas rīku lietošana. Šajā rakstā tiek demonstrēta modeļu transformācija, izmantojot mediniQVT rīku, kas atbalsta QVT Relations valodu. Transformēšanas procesam tika pielāgots iepriekš definēts TFM metamodelis, kā arī īstenoti transformēšanas likumi, izmantojot QVT Relations valodu. Transformēšanas process tika pārbaudīts, izmantojot automašīnas nomas uzņēmuma vienkāršotu TFM modeli. Transformēšanas rezultātā tika iegūts lietošanas gadījumu modelis, kas tikai daļēji atbilst paredzētajam modelim. Tāpēc tika secināts, ka izmantojot tikai QVT Relations valodu nav iespējams pilnībā un kvalitatīvi pārveidot TFM modeli uz lietošanas gadījumu modelī. Tālākais pētījuma virziens ir saistīts ar QVT Operational Mappings valodas izmantošanu, lai kvalitatīvi un pilnībā transformētu TFM modeli lietošanas gadījuma modelī.

Тимофей Муренец, Эрика Аснина. Автоматическое получение модели прецедентов использования из топологической модели функционирования

Управляемая моделями архитектура (Model Driven Architecture, MDA) - это подход разработки программного обеспечения в управляемой моделями инженерии (Model Driven Engineering, MDE). Данный подход определяет процесс трансформации моделей, используя языки трансформаций, такие как Query/View/Transformations (QVT). Первым важным шагом в MDA является качественный анализ системы, позволяющий определить структуры бизнес- и вспомогательных систем и требования к ним. Авторы статьи предлагают, используя трансформации создавать качественную модель требований, а именно модель прецедентов использования, основываясь на формальной бизнес-модели, а именно топологической модели функционирования (TFM). Так как процесс преобразования моделей без автоматизации не имеет практической ценности, возникает необходимость использовать специальные программы для автоматизации трансформаций. В данной статье для трансформаций было использовано mediniQVT - программное средство, поддерживающее язык QVT Relations. Для процесса трансформации была расширена метамодель TFM, а также были реализованы правила трансформации, используя язык трансформаций QVT Relations. Процесс трансформации моделей из TFM в модель прецедентов использования был проверен, используя упрощенную модель компании по аренде автомобилей. В результате трансформации была получена модель прецедентов использования, которая только частично соответствует ожидаемой модели. Был сделан вывод, что ограничения языка QVT Relations препятствуют качественной трансформации модели TFM в модель прецедентов использования. Поэтому в дальнейшем планируется создать трансформацию из TFM в модель прецедентов, используя QVT Operational Mappings.