

# Designing of Software Model to Model Transformation Language

OLENA V. CHEBANYUK<sup>1</sup>  
Software Engineering Department  
National Aviation University,  
KYIV, UKRAINE,  
chebanyuk.elen@ithea.org

**Abstract:** Software model transformation is a central operation in Model-Driven Development approach. In order to represent software models, graphical modeling notations, for example UML, are used. Quality of software model, obtained after transformation, influences on further operations with it. Many papers, proposing strong contribution in model to model transformational approach, consider transformational tasks, relating to concrete transformational languages or environments. Respectively, transformational results are visualized in concrete modeling environments (for example Eclipse or Microsoft Visual Studio) and software models are represented in concrete formats XML (XML, 2015) or XMI. Such approaches depend on possibilities of concrete tools, formats or model transformation languages (QVT [9], ATLAS or other). Variety of transformational operations is limited by supported features of chosen practical tools. From other side, development of analytical approaches for model to model transformations permit avoiding transformation environment limitations and composing transformational rules with different levels of complexity. This paper is devoted to designing of modeling language for Model to Model Transformation. The language designing is started from abstract syntax development. Then concrete syntax based on software model graph representation is described. Software model representation is proposed on two levels: at metalevel and representation considering details. Transformation operations are described by means of transformation grammar. Designing of software tool architecture for software model to model transformation is presented. Case study, containing description of transformation process, is outlined.

**Key-Words:** Software Model, Modeling Languages, Syntax and Semantics of Modeling Languages, Software Model Transformation, Graph Transformation, Model-Driven Development, Transformation Rules, UML, Use Case Diagram, Collaboration Diagram.

## 1 INTRODUCTION

Software models, represented as UML diagram, are central development artifacts in Agile approach.

Direction of Model-Driven Engineering is development of code generation approach [6]. Necessary condition to develop investigations in this field is to involve new fundamentals and analytical approaches to improve existing software model processing techniques. New proposed approaches should be related to all 4-layer metamodeling stack [1]. To achieve this goal software engineering standards and formal approaches touching foundation of software model processing are developed. One of the important tasks in this field is to modeling develop standards on metalevel. For example, formal aspects of representation objects and interconnections between them covers OMG MOF standard [8].

But standards, related to transformation languages consider model to model transformation, are developed on third level of 4-layer metamodeling stack. These standards are QVT

(group of standards) [9], ATLAS, ATL [5], and others. Mathematical transformation grammar, introduced by Chomsky considers transformation process on the first layer of 4-layer metamodeling stack (meta-metalevel) [4]. Contribution of this paper is proposing *Model to Model Transformation Language* (M2MTL) that describes model transformation operation on the metalevel. For supporting model to model transformation process corresponding software tool is proposed.

## 2 Related papers

Tasks, needed to be automated, for increasing effectiveness of different software development operations are summarized in paper [12]. Authors formulate requirements for software model transformation language, which supports model-driven software development are formulated.

This paper makes strong contribution to systematic review of model transformation requirements. But some requirement to software model transformation languages are remains still not clear. For example requirement “b”, namely “be implementable in an efficient way” or difference between model selection rules and rules for producing target model.

In the paper [10] an idea to combine representation of abstract and concrete syntax to support graph transformation operations is proposed. Then the transformation rule language for hierarchical automata was introduced. This introduction is informal in the sense that it points out the style of transformation rules and what happens at execution time of these rules. Syntax and semantics of these rules is not completely defined. That is why goal, formulated by authors, is to generate transformation languages from the grammars of DSLs is not archived completely.

Also, it is interesting to explore such questions:

- How to archive precise representation of transformation rules when transformational grammar will be spread?
- How to represent analytically transformation rules, if rules in metalevel and concrete syntax are represented together?

Touched questions make difficult reusing of proposed language.

In the paper [11] is proposed to proceed use-case templates, that are composed for requirements description. Authors present a systematic mapping study about the software product line variability description. From this mapping twelve Software Product Line (SPL) use case templates were defined. Also classification of these templates is proposed.

Use-case evaluation is divided into three main phases: Research Directives, Data Collection, and Results. In the first phase, the protocol and the research questions are established. The second phase, Data Collection, comprises the execution of the Systematic Mapping, and the inclusion/exclusion criteria are used in order to select relevant studies according to the research questions. Finally, the third phase, “Results”, is responsible for reporting the study outcomes based on a classification scheme.

This classification can be used as entire information for successfully performing many activities in requirement engineering, especially in model-driven approaches. Analyzing templates can be a starting point to make model execution more

precise. Also systematic representation of templates can help to design and precise profile constrains.

Represented review shows that model transformation techniques are actively investigated from fundamental points of view. Also there are many approaches, solving important tasks in software development life cycle, that can be performed more effectively implementing software model transformation techniques. Thus, designing language for software model to model transformation is a very important task.

### 3 Task and challenges

**Task:** to design a Model To Model Transformation Language (M2MTL) following approach for modeling language designing proposed in [1]. Language should be defined by abstract syntax, metamodel with constraints, and concrete syntax.

Challenges to the abstract syntax of M2MTL:

- Describe all entities and processes related to software model to model transformation process.
- Define them uniquely.

Challenges to the metamodel of language

- Facilitate process of acquainting with designed language from the cognitive point of view [2].
- Represent interconnection between main language entities.
- Provide extensible language constraints [7].

Challenges to the concrete syntax of M2MTL:

- Support both compact and detailed software model representations.
- Allow flexible choosing of graphical notations that participate in transformation (UML or other modeling languages).
- Be convenient for model software proceeding (analysis of structure, comparing, merging and so on).
- Be convenient for human cognitive perception [2].

Challenges to transformational rules of M2MTL:

- Be compatible with analytical approaches representing software models both in compact and detailed notations [7].
- Allow matching software model elements of compact and detailed view.
- Be compatible with representation of rules in natural language. Namely reflect all transformational conditions and details of transformational process.

#### 4 Designing of model to model transformation language

Book [1] proposes the procedure of designing your own modeling language. The first procedure of modeling language designing is to define its abstract syntax. Following [1] in Table 1, the elements of

abstract syntax for M2MTL are represented. The aim of abstract syntax is define main meaning of the designed language for the further graphical metamodel designing.

Table 1. Elements of M2MTL abstract syntax

Language element	Description of M2MTL element
Software model	According to standard UML 2.5 Software Model (SM) is UML diagram.
Initial software model	It is a software model from which transformation is started. This model contains initial information for transformation.
Resulting software model	Software model designed as a result from the transformation.
Software model representation	Graph representation [4].
Software model elements	Structural components from which software model is consists. They are objects and links.
Software models objects	Software model nodes.
Software model links	Elements of software model that connect two (or more) nodes.
Initial sub-graph	Part of software model chosen for the further transformation.
Elementary sub-graph	Sub-graph consisting from two linked vertexes.
Initial selecting rules	Rules for selecting sub-graphs from initial software model for using them in further transformation.
Resulting sub-graph	Sub-graphs of resulting software model designed during transformation.
Transformation	Process of obtaining a set of resulting sub-graphs by means of performing all transformational operations.
Transformation operation	Operation for obtaining one resulting sub-graph from the initial one.
Transformation rules	A set of recommendations for performing transformation operations.
Mappings	Operation which defines correspondence between initial and resulting software model elements.
One to one mapping	Mapping of an object or a link of initial software model to an object or a link of resulting software model.
One to many mapping	Mapping of an object or a link of initial diagram to sub-graph of resulting software model.
Many to one mapping	Mapping of sub-graph of initial model to an object or a link of resulting software model.
Many to many mapping	Mapping of sub-graph of initial model to sub-graph of resulting model.

Metamodel of M2MTL is represented on the Figure 1. It is designed using MOF syntax [8].

Metamodel constrains

- For performing transformation operation one initial and one resulting software model should be used (Initial SM, resulting SM).

- After transformation a set of resulting sub-graphs are appeared.
- Transformation operation uses at least one transformation rule.

Concrete M2MTL syntax is represented in Table 2.

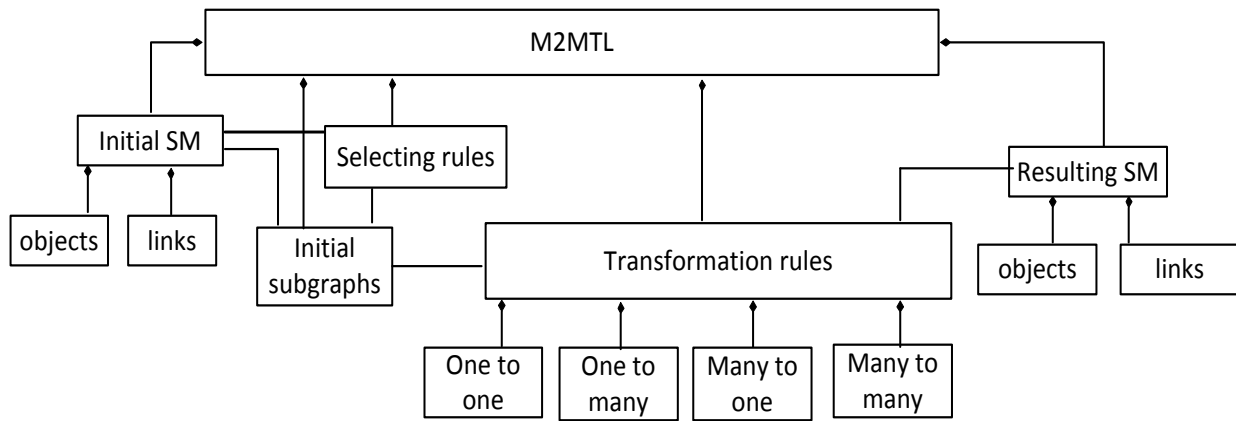


Fig. 1 Metamodel of M2MTL

According to sequence of action proposed by [1] the next step after designing of modeling language metamodel is to choose way of language concepts representation in terms of chosen analytical apparatus. The ground of choosing mathematical tools for performing different operation in

transformation approach is represented in the paper [3]. The concrete syntax is formed in graph representation of software model and using set theory for description of the language elements. Details are represented in the table 2.

Table 2. Concrete M2MTL syntax

Language element	Description of M2MTL element
Software model	Denote software model as SM and SM of some type as $SM_{type}$ . Where type=use case, type=class. The note if the name of UML diagram is long one several first letters denoting UML diagram type can be used. If the name of UML diagram is long type is denoted by several first letters with further explanation. For example $SM_{com}$ where type=communication.
Initial software model *	Initial software model is a software model of $SM_{type}$ . Denote it as $SMI_{type}$
Resulting software model *	Resulting software model is a software model of $SM_{type}$ . Denote it as $SMR_{type}$
* Example transformation from use case diagram to class one is denoted as: transform $SMI_{use\ case}$ to $SMR_{class}$	
Software model representation and its elements, namely objects and links	To represent software model graph representation is used. Denote software model(SM) as: $SM_{type} = (O_{type}, L_{type}) \quad (1)$ where $O_{type}$ – a set of software model objects that are used in $SM_{type}$ notation. Objects are elements of software model (SM) notations that can be expressed as graph vertexes. $L_{type}$ – a set of software model links that are used in $SM_{type}$ notation. Links are elements of software model notation that can be expressed as graph edges.
Representation of initial software model	$SMI_{type} = (OI_{type}, LI_{type}) \quad (2)$ Where $OI_{type}$ - a set of objects in notation of $SMI_{type}$ , $LI_{type}$ - a set of links in notation of $SMI_{type}$
Representation of resulting software	$SMR_{type} = (OR_{type}, LR_{type}) \quad (3)$ Where

Language element	Description of M2MTL element
model	$OR_{type}$ - a set of objects in notation of $SMR_{type}$ , $LR_{type}$ - a set of links in notation of $SMR_{type}$
Elementary sub-graph	Denote elementary sub-graph of software model as $SME$ $SME = (o_1, l, o_2) \quad (4)$ where $o_1$ and $o_2$ are software model objects (graph vertexes) $l$ - connection between objects $o_1$ and $o_2$ (graph edge).
Software model sub-graph	Part of software model, consisting from linked chain of elementary sub-graphs. Denote sub-graph of software model as $SM_{sub}$ . Using (4) this chain is denoted by the following: $SM_{sub} = (o_1, l_1, o_2), (o_2, l_2, o_3), \dots, (o_{n-1}, l_{n-1}, o_n) \quad (5)$ $SM_{sub} = SME_1, SME_2, \dots, SME_n$ where $n$ – is a number of elementary sub-graphs in sub-graph
Initial sub-graph	Part of software model chosen for the further transformation It is denoted as $SMI_{sub}$ . This part consist from several linked sub-graphs of initial software model.
Resulting sub-graph	Denote resulting sub-graph as $SMR_{sub}$ It is designed as a result of transformation of $SMI_{sub}$ to $SMR_{sub}$
Initial selecting rule	Initial selecting rules define how to choose $SMI_{sub}$ from $SMI_{type}$ for performing transformational operation. Denote initial selecting rule as $R(SMI_{type})$ . Thus, operation of selecting $SMI_{sub}$ applying $R$ , namely rule, on $SMI_{type}$ is written as follows: $R(SMI_{type}) = SMI_{sub} \quad (6)$ Usually initial selecting rules are composed as conditional statements, defining which elementary sub-graphs of $SMI_{type}$ compose $SMI_{sub}$ . Denote sub-graph for selecting $SMI_{sub}$ from $SM_{initial}$ as $S$ . Thus: $S = (OS, LS) = (os_1, ls_1, os_2), \dots, (os_{n-1}, ls_{n-1}, os_n) \quad (7)$ $OS$ - set of objects in $S$ , $LS$ – set of links in $S$ , $m$ – number of elementary sub-graphs in $S$ . Thus statement (4) can be written as follows: $select\ S\ from\ (SMI_{type}) = SMI_{sub} \quad (8)$ $S$ is a mask which is applied to every elementary sub-graph of $SMI_{type}$ . Graph $SMI_{sub}$ is formed by the next: every elementary sub-graph of $SMI_{type}$ $(oI_1, lI_1, oI_2); oI_1, oI_2 \in OI, lI_1 \in LI$ is compared with the first elementary sub-graph of $S$ , $(oS_1, lS_1, oS_2); oS_1, oS_2 \in OS, lS_1 \in LS$ . If they are the same, then the next elementary sub-graphs of $S$ and $SMI_{type}$ are compared consequently. If $SMI_{type}$ contains $S$ then $SMI_{sub}$ is formed.
Initial selecting rules	A set of initial selecting rules is denoted as RULES. Using (6), RULES for selecting set of $SMI_{sub}$ from $SMI_{type}$ are written as follows: $RULES = \{R_i(SMI_{type}) \mid i = 1, \dots, q\}; \quad q =  RULES  \quad (9)$
Transformation	To represent transformation rules, the transformational grammar [4] is used.

Language element	Description of M2MTL element
grammar	Transformation rules are syntax of this grammar [3]. Second order logic is used for representation of transformation rules in high level [3]. Also, such representation can be described in details using first-order logic [3].
Transformation operation	Transformation operation from $SMI_{sub}$ to $SMR_{sub}$ is written as follows: $SMI_{sub} \rightarrow SMR_{sub} \quad (10)$ where $\rightarrow$ transformation operation [4] According to (2) and (3) transformation operation using second order logic is represented by the following $(OI_{type}, LI_{type}) \rightarrow (OR_{type}, LR_{type}); \quad (11)$ According to (11) represent transformation operation in details using first-order logic $((oI_1, lI_1, oI_2), \dots, (oI_{n-1}, lI_{n-1}, oI_n)) \rightarrow ((oR_1, lR_1, oR_2), \dots, (oR_{m-1}, lR_{m-1}, oR_m));$ $oI_i \in OI, lI_i \in LI, oR_j \in OR, lR_j \in LR; i = 1, \dots, n; j = 1, \dots, m; n =  OI , m =  OR  \quad (12)$
Transformation rules	A set of rules for performing transformation operations Denote a set of transformation rules as <b>TRANS</b> . According to (10): $TRANS = \{SMI_{sub,i} \rightarrow SMR_{sub,i} \mid i = 1, \dots, t\}; t =  TRANS  \quad (13)$
Model to Model transformation process	Every $SMI_{type}$ is transformed to set of sub-graphs in $SMR_{type}$ notation applying all transformation rules. Denote all obtained sub-graphs $SMR_{sub}$ in notation as <b>SMR</b> . According (9) and (13): $TRANS(RULES(SMI_{type})) = SMR_{sub} \quad (14)$

## 5 Case study

Consider process of transformation Use Case diagram to Collaboration one. Denote this process as  $SMI_{uc}$  (uc = use case) to  $SMR_{col}$  (col = collaboration).

1. Prepare analytical representation of initial and resulting software model using graph representation based on (1)-(3)

$$SMI_{uc} = (O_{uc}, L_{uc})$$

$$O_{uc} = \{A_{uc}, P_{uc}\}$$

$$L_{uc} = \{L_{uc}, L(include)_{uc}, L(extends)_{uc}\}$$

where  $A_{uc}$  - a set of Use Case diagram actors.

$P_{uc}$  - a set Use Case diagram precedents.

$L_{uc}$  - a set of Use Case diagram links.

$L(include)_{uc}$  - a set of Use Case diagram links with mark <<include>>.

$L(extends)_{uc}$  - a set of Use Case diagram links with mark <<extends>>.

$$SMR_{col} = (O_{col}, L_{col})$$

$$O_{col} = \{A_{col}, O_{col}, C_{col}\} \quad L_{col} = \{M_{col}\}$$

where  $A_{col}$  - a set of Collaboration Diagram actors.

$O_{col}$  - a set of Collaboration Diagram objects.

$C_{col}$  - a set of Collaboration Diagram conditions.

$M_{col}$  - a set Collaboration Diagram messages.

2. Design transformation rules for transformation of  $SMI_{uc}$  to  $SMR_{col}$

### 2.1. One to one transformation

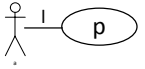
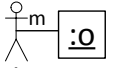
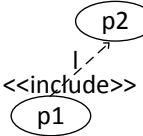
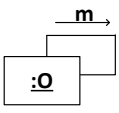
Use case diagram precedents are transformed to collaboration diagram messages.

$$SMI_{uc} \rightarrow SMR_{coll}$$

$$P_{uc} \rightarrow M_{col}$$

2.2. Many to many transformation rules are represented in the table 3.

Table 3. Many to many transformation rules for transforming use case diagram to collaboration one

Sub-graphs S formed for initial selection rules	Graphical representation of transformation rule	Analytical representation of transformation rule
Consider elementary sub-graph of use-case diagram $(a_{ucase}, l_{ucase}, p_{ucase})$  It is transformed to the next fragment of collaboration diagram: actor and outgoing message from it $(a_{col}, m_{col}, \emptyset)$	a) Use Case diagram fragment   b) collaboration diagram fragment 	$SMI_{uc} \rightarrow SMR_{coll}$  $(A_{uc}, L_{uc}, P_{uc}) \rightarrow (A_{col}, M_{col}, \emptyset)$  $(a_{uc}, l_{uc}, p_{uc}) \rightarrow (a_{col}, m_{col}, \emptyset)$
Consider elementary sub-graph of use-case diagram $(p1_{us}, L(include)_{uc}, p2_{uc})$  It is transformed to the next fragment of collaboration diagram: actor and outgoing message from it $(o_{col}, m_{col}, o_{col})$	a) Use Case diagram fragment   b) collaboration diagram fragment 	$SMI_{uc} \rightarrow SMR_{coll}$  $(P1_{uc}, L_{uc}, P2_{uc}) \rightarrow (O_{col}, M_{col}, O_{col})$  $(p1_{uc}, l_{uc}, p2_{uc}) \rightarrow (o_{col}, m_{col}, o_{col})$

3. Designing initial software model. Consider use case diagram for solving square equation (Figure 2):

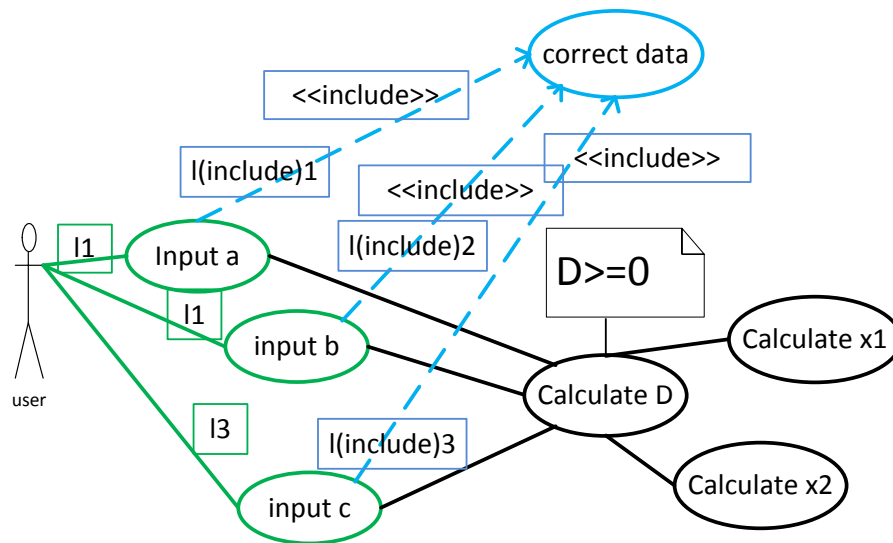


Fig. 2. Use Case diagram for solving square questions

4. Prepare initial sub-graphs of use case diagram for further transformation using (6) and (7).

Denote a set of sub-graphs compose according to transformation rules as  $SMI_{uc,i}, i = 1, \dots, n$

Where n- is a number of transformation rules.

4.1. Fragments of Use Case diagram that correspond to the first transformation rule (table 3) are marked by green color.

$$SMI_{uc,1} = R_1(SM_{uc}) =$$

$$= select(A_{uc}, L_{uc}, P_{uc}) from SM_{uc} =$$

$$= ((user, l_1, input a),$$

$$(user, l_2, input b), (user, l_3, input c))$$

4.2. Fragments of Use Case diagram that correspond to the second transformation rule (table 3) are marked by blue color.

$$SMI_{uc,2} = R_2(SM_{uc}) = \text{select}(P_{1,uc}, L(\text{include})_{uc}, P_{2,uc}) \text{ from } SMI_{uc} = ((\text{input } a, l(\text{include})_1, cd), (\text{input } b, l(\text{include})_2, cd), (\text{input } c, l(\text{include})_3, cd))$$

*cd* is an acronym for name of precedent “correct data”.

4.3. Prepare  $SMI_{uc,3}$ . It will consists from all precedents that were not included to first and the second transformation rules. They are colored in use case diagram (Figure 2) in black.

$$SMI_{uc,3} = \text{Select}(P, \emptyset, \emptyset) \text{ and } (\emptyset, \emptyset, P) \text{ from } SM_{uc} = ((\text{calculate } D, \emptyset, \emptyset), (\emptyset, \emptyset, \text{calculate } x1), (\emptyset, \emptyset, \text{calculate } x2))$$

sign  $\emptyset$  picks elements of elementary sub-graphs are not important for the transformation.

5. Perform transformation operation using (11) and (12).

$$SMI_{uc,1} \rightarrow SMR_{col,1} \\
 (user, l_1, \text{input } a) \rightarrow (user, \text{input } a, \emptyset) \\
 (user, l_2, \text{input } b) \rightarrow (user, \text{input } b, \emptyset) \\
 (user, l_3, \text{input } c) \rightarrow (user, \text{input } c, \emptyset)$$

Elements of collaboration diagram that are designed using the first transformation rule are marked by green

$$SMI_{uc,2} \rightarrow SMR_{col,2} \\
 (\text{input } a, l(\text{include})_1, cd) \rightarrow (a, \text{correct } a, a) \\
 (\text{input } b, l(\text{include})_2, cd) \rightarrow (b, \text{correct } b, b) \\
 (\text{input } c, l(\text{include})_3, cd) \rightarrow (c, \text{correct } c, c)$$

Elements of collaboration diagram that are designed using the second transformation rule are marked by blue

$$SMI_{use\ case,3} \rightarrow SMR_{col,3}$$

Precedents with names “calculate D”, “calculate x1”, “calculate x2” become messages of collaboration diagram with the same names.

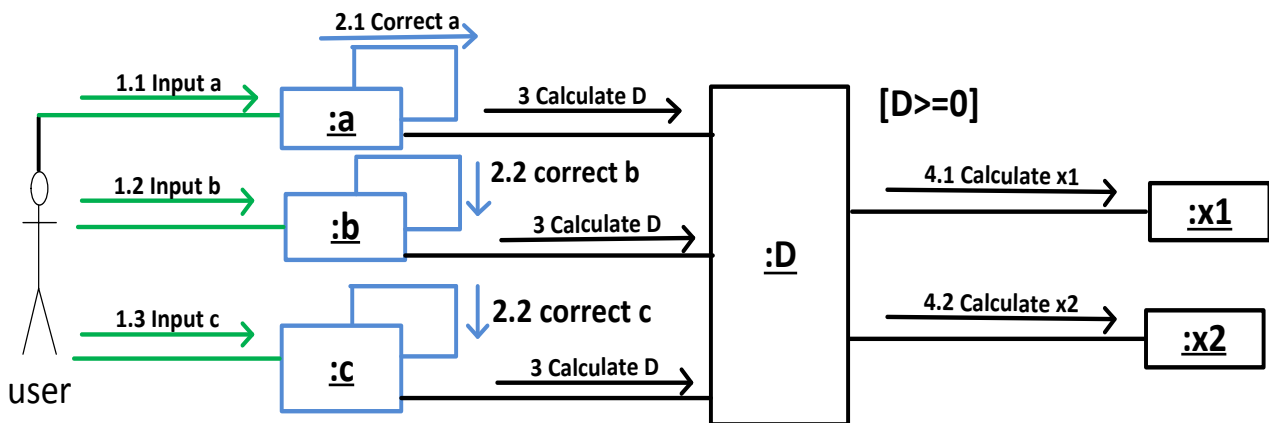


Fig. 3. Collaboration diagram for solving square questions

## 6 Designing an architecture of model to model transformation framework

Consider an example of model to model transformation designing framework to transform Use Case to Collaboration diagram.

Class diagram of model to model transformation framework is represented on Figure 4

Information about software models is stored in the classes *Use\_Case* and *Collaboration* from *UML\_model* package.

Diagram notation is stored in separate list, for instance list of actors, precedents, and comments.

Link between elements is characterized by two linked objects in UML diagram. Information about links is stored in class *Link*.

Information about Collaboration diagram entities, obtained after transformation is stored in class *Collaboration*.

Lists of Use Case diagram entities are populated in constructor *Use\_Case(string path)*. It obtains path to Use Case diagram, stored in XML based format [13]. Usually in this format environments for software models designing store information about UML diagrams. Examples of such environments are Microsoft Visual Studio, IBM Rational Software Architect, Integrated Development Environment Eclipse, ect.



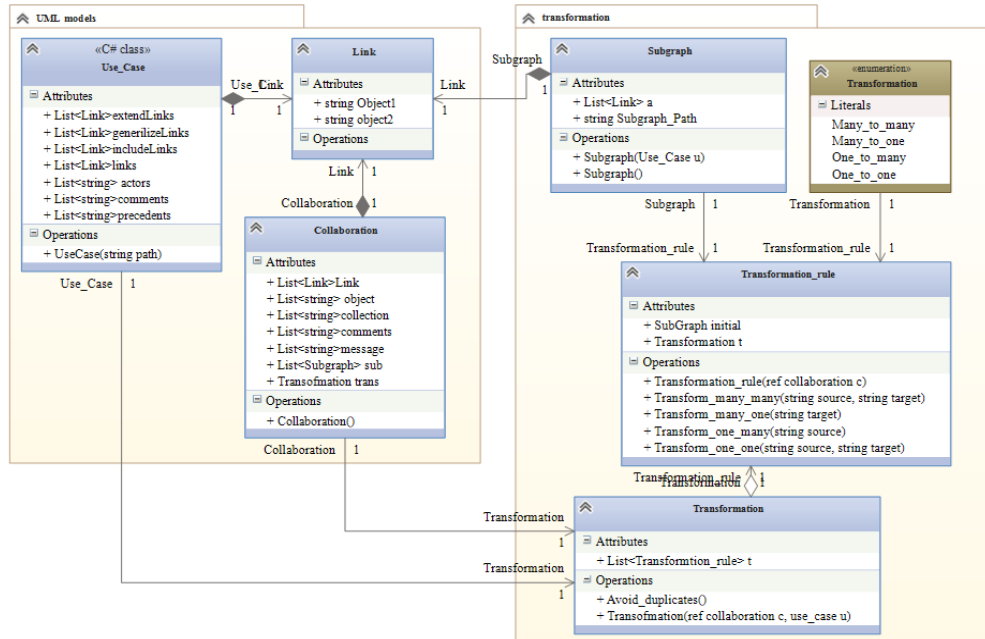


Figure 4. Class diagram of framework for transforming Use Case to Collaboration diagram

Consider use case diagram, created in Microsoft Visual Studio format.

The example of XML tags, corresponded to actor entity is given below.

```
<actor Id="f32af9db-669a-486d-a5a2-c66ebab0af85" name="Actor1" isAbstract="false" isLeaf="false">
```

Then actor is linked with precedent “a”

```
<association Id="000dd500-e774-4754-b526-9fea979b34a1" isDerived="false" sourceEndRoleName="Actor1" targetEndRoleName="a" isLeaf="false" isAbstract="false">
```

Precedent itself is stored in the next XML string

```
<memberEnd Id="9e1f21b1-9d2f-4e56-b61e-304d50b96f35" name="a" isLeaf="false" isStatic="false" isReadOnly="false" isDerived="false" isDerivedUnion="false" aggregation="None" isComposite="false">
```

Implementing LINQ queries XML tags can be proceeded. The aim of proceeding UML model file is to obtain lists of all its components and interconnections between them.

The next step is to compose sub-graphs according to initial selection rules. Information about every sub-

graph is stored in class Subgraph from “Transformation” package. Every sub-graph serves information to perform one transformation rule. Transformation rules are divided to several types, namely support one to one transformation, one to many, and many to many. Types of transformation rules are defined in enumeration Transformation from transformation package.

Consider realization of different types of transformation rules.

Transformation rule one to one realized in the method *Transform\_one\_one(string source, string target)* in the next way: Every entity from list, named as source is added to list of entities, named target. Transformation many to one is realized by method *Transform\_many\_one(string target)*.

Entities, represented in target parameter are chosen from an initial sub-graph that is a property of Transformation\_rule class. Then, these entities are added to proper list of Collaboration Diagram.

Class “Transformation” stores list of transformation rules. Method *Transofmation(ref collaboration c, use\_case u)* populates Collaboration Diagram entities. Collaboration Diagram is transmitted as a parameter to method.

And the last step is to delete duplicated entities from lists of collaboration diagram entities.

As a result of transformation, user obtains a list of sub-graphs and lists of Collaboration Diagram entities. They help user to verify designed collaboration diagram.

## 7 Conclusions

Language of software model to model transformation is presented in this paper. The motivation of performing such a research is to obtain extensible model to model transformation language supporting several concrete syntaxes, allowing representation of software models and transformation rules both in compact and detail view. Graph representation of software models allows providing a bridge between UML (or other modeling language with graphical concrete syntax) and analytical tools for software models processing and analyzing (1)-(3). Proposed representation of transformation rules is also compatible with graph representation of software models (10)-(13). Such representation allows considering complex expression for performing transformations, including several preconditions, or software model elements that are not linked each other directly.

Concrete syntax of the M2MTM, proposed in this paper, allows considering transformation process both on metalevel and model level [6]. General transformation ideas and software models notations can be analyzed on metalevel.

Considering of sub-graphs and software models at level of elements permits analyzing transformations in details. Doing this existing transformation rules can be refined and new transformation rules also can be designed.

## 8 FURTHER WORK

Propose an approach of resulting software model designing grounded on problem domain ontology analysis. Visualized resulting software model should consider possibilities of human cognitive abilities for perception (Chebanyuk and Markov, 2015).

Define operations that are used for analysis of software model before and after transformation (for example refinement or merging). Extend M2MTL abstract and concrete syntaxes for performing these operations and propose corresponded analytical tools.

Develop a software tool for extracting information from initial software model designed in different modeling environments.

### References

[1] Brambilla M., Cabot J., Wimmer M., 2012. *The book Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering. Morgan & ClayPool publishers. Pages 1-182

[2] Chebanyuk E., Markov K., 2015. Software model cognitive value. *International Journal "Information Theories and*

*Applications"*, Vol. 22, Number 4, ITHEA 2015  
<http://www.foibg.com/ijta/vol22/ijta22-04-p04.pdf>

[3] Chebanyuk E., Markov K., 2016. Model of problem domain "Model-driven architecture formal methods and approaches" *International Journal "Information Content and Processing"*, Vol. 22, Number 4, ITHEA 2016, p.202-222.  
<http://www.foibg.com/ijcp/vol03/ijcp03-03-p01.pdf>

[4] Chomsky, N. 1957. *The book*. Syntactic Structures. Mouton publishers, Eilenberg: Mac Lane The, Hague, 1945 - 1957. ISBN 90 279 3385 5. p.107.

[5] ATL 2016 . ATL Transformation Language, 2016 .  
<http://www.eclipse.org/atl/>

[6] IBM, 2016.  
<http://researcher.ibm.com/researcher/files/zurich-iku/mdse-07.pdf>

[7] OCL, 2014. Object Constraint Language Version 2.4 OMG standard.  
<http://www.omg.org/spec/OCL/2.4/PDF>

[8] OMG, 2016. Meta Object Facility™ (MOF™) Core  
<http://www.omg.org/spec/MOF/>

[9] QVT, 2016. Meta Object Facility Query/View/Transformation, v1.3  
<http://www.omg.org/cgi-bin/doc?formal/2016-06-03>

[10] Rumpe B., Weisemöller I., 2011. A Domain Specific Transformation Language. *In: ME 2011 - Models and Evolution*, Wellington, New Zealand. Ed: B. Schätz, D. Deridder, A. Pierantonio, J. Sprinkle, D. Tamzalit, Wellington, New Zealand, Okt. 2011.  
<https://arxiv.org/fip/arxiv/papers/1409/1409.2309.pdf>

[11] Santo I.S., MC Andrade R., Santos P.A., 2015. Templates for textual use cases of software product lines: results from a systematic mapping study and a controlled experiment. *Journal of Software Engineering Research and Development* (2015) 3:5 DOI 10.1186/s40411-015-0020-3  
<https://jserd.springeropen.com/articles/10.1186/s40411-015-0020-3>

[12] Sendall, S., Kozaczynski. W., 2003. "Model transformation: The heart and soul of model-driven software development." *IEEE Software* 20.5 (2003): 42-45.  
<http://ieeexplore.ieee.org/document/1231150/>

[13] XMI, 2015. XML Metadata Interchange. *Access mode*  
<http://www.omg.org/spec/XMI/>