Visual Models Transformation in MetaLanguage System

Alexander O. Sukhov, Lyudmila N. Lyadova

Abstract — At the process of creation and maintenance of information systems the model-based approach to the software development is increasingly used. This approach allows to move the focus from writing of the program code with using general purpose language to the models development with automatic generation of data structures and source code of applications. However at usage of this approach it is necessary to transform models constructed by various categories of users at different stages of system creation with usage of various modeling languages. An approach to models transformation in DSM platform MetaLanguage is considered. This approach allows fulfilling vertical and horizontal transformations of the designed models. The Metalanguage system support “model-text” and “model-model” types of transformations. The component of transformations is based on graph grammars described by production rules. Transformations of model in Entity-Relationship notation are presented as example.

Keywords — Domain-specific languages, visual language, DSM-platform, language workbench, model-based approach, model transformation, vertical transformation, horizontal transformation.

I. INTRODUCTION

Development of information systems with usage of the modern tools is based on the design of the various models describing the domain of the information system, defining data structures and algorithms of system functioning. The main idea of such model-driven approach is the systematic usage of models at various stages of software development that allows to shift the focus from writing code in general purpose programming language to building models and automatic generation of the source code and other necessary artifacts. At modeling developer abstracts from concrete technologies of implementations. It facilitates the creation, understanding and maintenance of models. This approach is intended to increase productivity and to reduce development time.

There are implementations of model-driven approach which use general purpose modeling languages for describing of information systems. So, the modeling language UML with the standard MOF (Meta-Object Facility) forms a basis of the concept MDA (Model-Driven Architecture) [1]. Other implementations of the model-driven approach are based on use of the visual domain-specific modeling languages (DSMLs, DSLs), intended to solve a particular class of problems in the specific domain. Unlike general purpose modeling languages, DSMLs are more expressive, simple in applying and easy to understand for different categories of users as they operate with domain terms. To support the process of development and maintenance of DSMLs the special type of software — language workbench (DSM-platform) – is used.

The various categories of specialists (programmers, system analysts, database designers, domain experts, business analysts, etc.) are involved in the process of information systems creation and maintenance. Often they need modification of modeling language description to customize and adapt DSML to new conditions, requests of business and possibilities of users. The transformations of models constructed by various users at different stages of information system creation with usage of various DSMLs are necessary for the models adjustment and integration [2].

For implementation of these possibilities it is necessary, that the language workbench allowed to build the whole hierarchy of models: model, metamodel, meta-metamodel, etc., where model is an abstract description on some formal language of system characteristics that are important from the point of view of the modeling purpose, a metamodel is a model of the language, which is used for models development, and a meta-metamodel (metallanguage) is a language for the metamodels description. Furthermore, the language workbench should contains the tools allowing to fulfill conversion of models between various levels of hierarchy (vertical transformations) and in one hierarchy level (horizontal transformations).

The MetaLanguage system is a language workbench for creating of visual dynamic adaptable domain-specific modeling languages. This system allows to fulfill multilevel and multi-language modeling of domain [3]. The basic elements of the metalanguage are entity, relationship and constraint.

Usage of domain-specific languages and tools for the system development also affects a transformation problem as there is a need of export of the models created with DSML to external systems which, as a rule, use one of the standard modeling languages that is different from used DSL. That is why one of the main components of the MetaLanguage system is the transformer. This component uses graph grammars for models transformations description. Implementation of graph grammars in the MetaLanguage system is defined by appointment of this language workbench.

This work was supported by Russian Foundation for Basic Research (grant 14-07-31330).

A. O. Sukhov is with the National Research University Higher School of Economics, Perm, Russia (phone: (+7) 912-589-0986; e-mail: Sukhov.psu@gmail.com).

L. N. Lyadova is with the National Research University Higher School of Economics, Perm, Russia (e-mail: LNYladova@gmail.com).
II. BASIC CONCEPTS

The basic concept of transformation definition is a production rule which looks like \( p : L \rightarrow R \), where \( p \) is a rule name, \( L \) is a left-hand side of the rule, also called the pattern, and \( R \) is a right-hand side of the rule, which is called the replacement graph. Rules are applied to the starting graph named the host-graph.

Let’s suppose that four labeled graphs \( G, H, L, R \) are given, and graph \( L \) is a subgraph of graph \( G \). Applying of the rule \( p : L \rightarrow R \) to the starting graph \( G \) is called the replacement in graph \( G \) of subgraph \( L \) on graph \( R \), which is a subgraph of graph \( H \). The graph \( H \) is the result of this replacement [4].

Graph transformation is a sequenced applying to the starting labeled graph \( G_0 \) of finite set of rules \( P = (p_1, p_2, \ldots, p_n) : G_0 \rightarrow G_1 \rightarrow \ldots \rightarrow G_n \).

Transformations can be classified as horizontal and vertical according to direction. The horizontal transformation is the conversion, in which the source and target models belong to one hierarchy level. An example of a horizontal transformation is a conversion of model description from one notation to another (see Fig. 1). The vertical transformation converts the models which belong to various hierarchy levels, for example, at mapping of the metamodel objects to domain model objects.

III. RELATED WORKS

There are various approaches to model transformations. Some of them have the formal basis, so the systems AGG, GReAT, VIATRA use graph rewriting rules to perform transformations, and others apply technologies from other areas of software engineering, for example the technique of programming by example.

Various modifications of the algebraic approach [9] are implemented in systems AGG, GReAT, VIATRA. In AGG (Attributed Graph Grammar) [10], [11] the left- and right-hand sides of the production rule are the typed attribute graphs, both sides of a rule should be described in one notation, i.e. this system allows to fulfill only endogenous transformations that does impossible its usage in MetaLanguage system. Besides, this tool does not allow to make transformation of a “model-text” type. However the usage as the formal basis of the algebraic approach to graph transformations allows to produce graph parsing, to verify graph models, and the extension of graphs of Java possibilities makes transformations more powerful.

The GReAT (Graph REwriting And Transformation) system [12], [13] is based on the algebraic approach with double-pushout, therefore for transformation description it is necessary to create the domain that contains both the left- and right-hand sides of the production rule simultaneously with instructions of what element it is necessary to add, and what to remove. This form of rule is unusual for the user and a bit tangled. However it provides a possibility of execution the transformation of several source metamodels at once, which is significant advantage in comparison with other approaches. For metamodels definition the GReAT uses UML and OCL, it does not allow the user to choose the language of metamodels specification or to change its description. It makes this
approach unsuitable for usage in MetaLanguage.

The QVT (Query/View/Transformation) is the proposed by OMG approach to models transformation, which provides the user with declarative and imperative languages [14], [15]. Conversion is defined at the level of metamodels, which is described on MOF. The advantage of this approach is the existence of standard of its description, and also usage of standard languages OCL and MOF at the models transformation definition. But usage of MOF as a meta-metamodeling language, does not allow the user to choose a metalanguage convenient for him, or to change description of the metalanguage which is integrated in the QVT.

VIATRA (VIual Automated model TRansformations) [16], [17] is a transformation language, based on rules and patterns, which combines two approaches into a single specification paradigm: the algebraic approach for models description and the abstract state machines intended for exposition of control flow. Thanks to constructions of state machines the developers significantly raised the semantics of standard languages of patterns definition and graph transformation. Besides, powerful metalanguage constructions allow to make multilevel modeling of domains. One of shortcomings of the VIATRA is an inexpressive textual language of metamodels description. VIATRA is not intended for execution of horizontal model transformations. Its main purpose is a verification and validation of the constructed models by their transformation.

The ATL (ATLAS Transformation Language) is the language, allowing to describe transformations of any source model to a target model [18], [19]. Transformation is performed at the level of the metamodels. The disadvantage of this language is high requirements to the developer of transformation. Since ATL in most cases uses only textual definition of transformation, then in addition to knowledge of source and target metamodels the developer needs to know language of transformation definition. The ATL is a dialect of QVT language and therefore inherits all its shortcomings.

MTBE (Model Transformation By-Example) approach [20], [21] is quite non-standard and unusual. The main purpose of MTBE is automatic generation of transformation rules on a basis of an initial set of learning examples. However implementations of this approach do not guarantee that the generation of model transformation rules is correct and complete. Moreover, the generated transformation rules strongly depend on an initial set of learning examples. Current implementations of MTBE approach allow to fulfill only full equivalent mappings of attributes, disregarding the complex conversions.

In summary, it is possible to say that all considered systems have some disadvantages which restrict their applicability for transformation definitions in the MetaLanguage system. But the most appropriate and perspective, from the author’s point of view, is the algebraic approach.

IV. MODEL TRANSFORMATIONS

**Horizontal transformation** is the conversion, in which the source and target models belong to one hierarchy level.

All horizontal transformations in MetaLanguage system are described at level of metamodels that allows to specify conversions which can be applied to all models created on basis of this metamodels. For a transformation definition it is necessary to select a source and target metamodels and to define production rules that are describing conversion.

To define the rule it is necessary to select objects (entities and relationships) in a source metamodel, to set constraints on pattern occurrence and to define the right-hand side of the rule. Depending on a type of transformation a right-hand side will be a text template for code generation, or a fragment of a target visual metamodel.

Transformation rules are applied according to their order. At first all occurrences of a first rule pattern will be found, for each of them the system will replace it by the right-hand side of the production rule, then the system will pass to the second rule and will begin to execute it, etc.

Let’s assume that the system has selected next production rule of transformation and trying to execute it. For implementation of rule application it is necessary to describe two algorithms: the algorithm of the pattern search in the source host-graph and the algorithm of replacement of the left-hand side of the rule by the right-hand side.

There are various algorithms of search of subgraph isomorphic to the given pattern: Ullmann algorithm [22], Schmidt and Druffel algorithm [23], Vento and Foggia algorithm [24], Nauty-algorithm [25], etc. These algorithms are the most elaborated and often used in practice.

However difference of the proposed approach from the classical task of graph matching is that in this case it is necessary to find a pattern in the metamodel graph, i.e. it is required to lead matching of graphs which belong to various hierarchy levels, thus it is necessary to consider type of nodes and arcs, as between two nodes of the metamodel graph the several arcs of various type can be led.

The offered algorithm for finding a pattern in the graph model is a kind of backtracking algorithm that takes exponential time.

Since the amount of arcs in the model graph is less than amount of the nodes usually, each arc uniquely identifies nodes, that are incident to it, and the degree of node can be more than two, that does not allow to select the following node of the model graph, entering into a pattern. It was decided to start search of subgraph in a model graph on the basis of search of particular type arcs.

At the first step of algorithm all instances of some arbitrary relationship of the pattern will be found, i.e. search of an initial arc with which execution of the second step of algorithm will begin is carried out. At the second stage it is necessary to find one of possible occurrence of all relationships instances of the pattern-graph $G_p$ in the source model graph $G_s$. At the third step necessary nodes will be add to target graph $G_t$ and
replace the left-hand side of the rule by the right-hand side. Then it is necessary to replace the left-hand side of the production rule by the right-hand side after the subgraph of left-hand side has been found in the source graph. The algorithm of replacement will depend on a type of transformation: whether transformation is “model-text” or “model-model”.

Transformation “model-text”. The transformation of this type allows to generate the source code on any target programming language on the basis of the constructed models as well as any other textual representation of model, for example, its description on XML. In this case the right-hand side of production rule contains some template consisting of as static elements, which are independent of the found pattern, and dynamic parts, i.e. elements which vary depending of the found fragment of model.

For transformation fulfillment it is necessary to find all occurrences of a pattern in a source graph and to produce an insertion of an appropriate text fragment with a replacement of a dynamic part by appropriate names of entities, relationships, values of their attributes, etc.

The template is described on the target language. For selection of a dynamic part of a template the special metasymbols are used: “<<” (double opening angle brackets) to indicate the beginning of a dynamic part, “>>” (double closing angle brackets) to indicate the end of a dynamic part. As entities and relationships can have the same name, then for entity describing before its name the prefix “E.” is specified, and for relationship describing before its name the prefix “R.” is specified.

At the transformation specifying it is possible to set constraints on pattern occurrence. These constraints allow to define the context of the rule. They contain conditions with which found fragment of model should satisfy.

Let’s consider an example: define the transformation that allows on the basis of Entity-Relationship Diagrams (ERD) to generate a SQL-query, building the schema of a corresponding database.

At the first step it is necessary to choose the metamodel of Entity-Relationship Diagrams (see Fig. 2) and to set the transformation rules.

The metamodel contains the entities “Abstract”, “Attribute”, “Entity”, “Relationship”. Attributes of the entity “Abstract” are “Name” that identifies an entity instance, and “Description”, containing the additional information about the entity. The entity “Abstract” is abstract, i.e. it is impossible to create instances of this entity in the model. “Abstract” acts as a parent for entities “Entity” and “Relationship” (in the figure it is shown by an arrow with a triangular end). Both child entities inherit all parent attributes, relationships, constraints. “Entity” does not have own attributes and constraints. “Relationship” has the own attribute “Multiplicity”. The entity “Attribute” has following attributes: “Name”, “Type” and “Description”.

The bidirectional association “Linked_Links” connects entities “Relationship” and “Entity”. It means that it is possible to draw equivalent relationship between these entity instances in ERD-models. The second unidirectional association “SuperClass_SubClass” binds entity “Entity” with itself, it allows any instance of “Entity” to have parent (another instance of “Entity”) in ERD-models. In ERD metamodel between entities “Attribute” and “Abstract” the aggregation “Belongs” is set (in figure this relationship is presented by an arc with a diamond end), therefore in ERD-models instances of entities “Relationship” and “Entity” can be connected by aggregation with the instances of entity “Attribute”.

For correct transformation execution the additional attributes in the source metamodel should be added. To determine what entity is a parent, and what entity is a child it is necessary to add the mandatory attributes of a reference type (“Child” and “Parent”) to relationship “SuperClass_SubClass”.

The entity “Relationship” should be transformed to the reference between relational tables, therefore we will add to “Relationship” additional mandatory attributes-references of “LeftEntity” and “RightEntity” and attribute of logical type “Has_Attribute”, which will facilitate the replacement of the left-hand side of the production rule by the right-hand side.

For transformation definition we will use the traditional rules of conversion of the ERD notation to a relational model, for this purpose we will define the following rules.

The rule “Entity” which transforms the instance of entity “Entity” to the single table looks like:

```
CREATE TABLE <<E.Entity.Name>>
(id INTEGER primary key)
```

Here <<E.Entity.Name>> is a dynamic part of the template which allows to get a name of corresponding entity.

As there is not inheritance relationship in a relational model, it is necessary to specify the rule “Inheritance”, which for each instance of the relationship “SuperClass_SubClass” in the “SubClass” table creates foreign key for connection with the “SuperClass” table.
This rule looks like:

```
ALTER TABLE <<R.SuperClass_SubClass.Child>>
ADD <<R.SuperClass_SubClass.Parent>>ID INTEGER
```

The rule “Relationship_1M” allows to transform instance of entity “Relation”, which does not have attributes and its multiplicity is “1:M”, to the reference between tables.

The rule has the following appearance:

```
ALTER TABLE <<E.Relationship.LeftEntity>>
ADD FOREIGN KEY (<<E.Relationship.RightEntity>>ID)
REFERENCES <<E.Relationship.RightEntity>> (id)
```

In this rule at first in the table corresponding to the left entity the additional column with the name <<E.Relationship.RightEntity>>ID is added, and then the foreign key (correspondence between this additional column and a column containing the identifiers of right table rows) is created. This rule contains the constraint on the pattern occurrence:

```
E.RelationshipMultiplicity = 1:M AND
E.Relationship.HasAttribute = False
```

The rule “Relationship_M1” allows to transform instance of entity “Relation”, which does not have attributes and its multiplicity is “M:1”, to the reference between tables.

The rule looks like:

```
ALTER TABLE <<E.Relationship.RightEntity>>
ADD FOREIGN KEY (<<E.Relationship.LeftEntity>>ID)
REFERENCES <<E.Relationship.LeftEntity>> (id)
```

The content of this rule is similar to the content of the rule “Relationship_1M”. This rule contains the constraint on the pattern occurrence:

```
E.RelationshipMultiplicity = M:1 AND
E.Relationship.HasAttribute = False
```

For each instance of entity “Relationship”, which has the attributes, or has the multiplicity “1:1” or “M:M”, it is necessary to create the single table that contains the key columns of each entity involved in relationship. We call this rule “Relationship_MM”, it has the following appearance:

```
CREATE TABLE <<E.Relationship.Name>>
(id INTEGER primary key,
<<E.Relationship.LeftEntity>>ID INTEGER,
<<E.Relationship.RightEntity>>ID INTEGER)
ALTER TABLE <<E.Relationship.Name>>
ADD FOREIGN KEY (<<E.Relationship.LeftEntity>>ID)
REFERENCES <<E.Relationship.LeftEntity>> (id)
ALTER TABLE <<E.Relationship.Name>>
ADD FOREIGN KEY (<<E.Relationship.RightEntity>>ID)
REFERENCES <<E.Relationship.RightEntity>> (id)
```

This rule contains the constraint on the pattern occurrence:

```
E.RelationshipMultiplicity = M:M OR
E.RelationshipMultiplicity = 1:1 OR
E.Relationship.HasAttribute = True
```

The rule “Attribute” adds the columns corresponding to attributes of instances of entities and relationships to the created tables:

```
ALTER TABLE <<E.Abstract.Name>>
ADD <<E.Attribute.Name>>
<<E.Attribute.Type>>
```

Let’s consider an example, apply the described transformation to the model “University” presented in Fig. 3.

![Fig. 3. Simplified model “University” on the ERD notation](image-url)

As the result the following text has been generated by the MetaLanguage system:

```
CREATE TABLE Man (id INTEGER primary key)
CREATE TABLE Student (id INTEGER primary key)
CREATE TABLE Lecturer (id INTEGER primary key)
CREATE TABLE ExamCards (id INTEGER primary key)
CREATE TABLE PassExam (id INTEGER primary key)
CREATE TABLE ExamCards (id INTEGER primary key)
ALTER TABLE ExamCards ADD StudentID INTEGER
ALTER TABLE Lecturer ADD FOREIGN KEY (StudentID)
REFERENCES Student (id)
ALTER TABLE PassExam ADD FOREIGN KEY (LecturerID)
REFERENCES Lecturer (id)
ALTER TABLE Student ADD ManID INTEGER
ALTER TABLE Lecturer ADD FOREIGN KEY (ManID)
```

...
It should be noted that this transformation does not take into account complex conversions the ERD notation to the database schema, for example, those which would allow to create single dictionary table on the base of attribute, because it requires a special description language of templates and it is one of the areas for further research. Although such conversion could be done by adding to the entity “Attribute” the attribute “Is_a_Dictionary” and setting the constraints on pattern occurrence.

Transformation “model-model”. Transformation of this type allows to produce conversion of model from one notation to another or to perform any operations over model (creation of new elements, reduction, etc.). Such transformation will allow to export model to external systems, and to provide the ability to convert the domain-specific language that was created by the user in one of most common modeling language, for example, UML, ERD, IDEF0, etc.

The left-hand side of a production rule of this type transformation is some fragment of the source metamodel, and the right-hand side of the rule is some fragment of the target metamodel. At the production rule definition also it is necessary to describe the rules for converting the attributes of entities and relationships. The created model should not contain dangling pointers, therefore the process of the transformation executions begins with the creation of entity instances and only then instances of relationships are created. If in the process of model building the dangling pointers are still found the system will delete them.

At transformation execution it is necessary to consider the following elementary conversions:
- conversion “entity → entity”;
- conversion “relationship → relationship”;
- conversion “entity → relationship”;
- conversion “relationship → entity”.

Let’s suppose that in the source model the instances of entities and relationships of pattern are already found.

For fulfillment of the conversion ee: Ent_L → Ent_R it is necessary to create in the new model the instance Ent_R of the appropriate entity from a rule right-hand side and to perform transformation of attributes. The created instance of entity will have the same name, as the name of source entity instance.

For execution the conversion rr: Rel_L → Rel_R at first it is necessary to found in the source model the instances of entities Rel_L,SEI and Rel_L,TEI, which are connected by the relationship instance Rel_L, then the images of these instances should be found in the new model, and an instance of the relationship from a rule right-hand side should be lead between them. After that it is necessary to fulfill transformation of attributes.

For fulfillment of the conversion er: Ent_L → Rel_R it is necessary to find in source model the nodes Ent_L, Ent_L, which are adjacent to entity instance Ent_L. Let’s denote their images in the target model as Source and Target. In the target model the relationship instance Rel_R between nodes Source and Target should be lead. Further it is necessary to execute defined transformation of attributes.

Conversion re: Rel_L → Ent_R transforms the instance of relationship Rel_L found in the source model to the entity instance Ent_R of target model. For conversion execution it is necessary to create the entity instance Ent_R, to perform the specified transformation rules of attributes. The name of Ent_R will be the same as the name of the relationship instance Rel_L. At the next step it is necessary to find entities instances Rel_L,SEI, Rel_L,TEI, which are connected by relationship instance Rel_L.

Further the instances of relationships that connect an entity instance Ent_R with nodes Source and Target, which are images of the nodes Rel_L,SEI and Rel_L,TEI, accordingly, are created with keeping of orientation of relationship instance.

It is possible to present the rest conversions of “model-model” type by a combination of these elementary operations.

Let’s consider an example, perform the transformation of the model on ERD notation to UML Class Diagrams.

Since the transformation is done at the metamodel level, then at the first step it is necessary to create/open source and target metamodels. The ERD metamodel was presented in the Fig. 2. Metamodel of UML Class Diagrams is shown in the Fig. 4. It contains the following elements: the entity “Class” and three relationships “Inheritance”, “Association”, “Aggregation”. Let’s define the production rules that determine the transformation.

![Fig. 4. Metamodel of UML Class Diagrams](image)

The rule “Abstract-Class” allows to convert the instances of entities “Entity” and “Relationship”, which are connected at least with one instance of entity “Attribute”, to the instance of entity “Class”.

This rule has the following appearance:
The rule “Entity-Class” allows to convert the instance of entity “Entity”, which is not associated with any instance of the entity “Attribute”, to the instance of an entity “Class”.

The rule has the following form:

```
Entity  →  Class
```

The rule “Relationship-Association” converts instances of the entity “Relationship” of the source model to instances of the relationship “Association” of the target model.

This rule looks like:

```
Relationship  →  Association
```

The rule “Inheritance” puts in correspondence to each instance of the relationship “SuperClass_SubClass” of source model a particular instance of the relationship “Inheritance” of target model. This rule has the following form:

```
SuperClass_SubClass  →  Inheritance
```

After definition of all rules, which are included in the transformation, it is possible to execute conversion on a specific model. Let’s perform this transformation on the considered earlier model “University” (see Fig. 3). The result of the transformation execution is presented in Fig. 5.

```
Student

<table>
<thead>
<tr>
<th>is a</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETS</td>
<td>ExamCards</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>is a</th>
<th>Lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKES</td>
<td>ExamCards</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>is a</th>
<th>Man</th>
</tr>
</thead>
<tbody>
<tr>
<td>PassExam</td>
<td>ExamCards</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>is a</th>
<th>Professor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks</td>
<td>ExamCards</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>is a</th>
<th>Senior lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepares</td>
<td>ExamCards</td>
</tr>
</tbody>
</table>
```

Thus, at creation of metamodel “ERD” the mapping of metalanguage constructions in metamodel entities and relationships is fulfilled. So the metalanguage construction “Entity” is mapped in the entities “Abstract”, “Attribute”, “Entity”, “Relationship”.

Then at construction of the domain-specific metamodel “University” the elements of metamodel “ERD” are mapped in instances of entities and relationships of the metamodel “University”. For example, on the basis of entity “Attribute” its instances “Direction”, “Duration”, “Name”, “Post”, “Task” are built.

At the creation of model “Exam” the entities and relationships of the domain-specific metamodel “University” are mapped in elements of the model “Exam”. So on the basis of entity “Lecturer” the elements “Professor”, “Senior lecturer” are created.
At the stage of models validation and transformation the MetaLanguage system fulfills interpretation of models elements at various hierarchy levels. So at transformation of the domain-specific metamodel “University” in the SQL-query the language workbench should determine with what entities and relationships the elements of metamodel “University” are created, since transformation rules are described at level of metamodels. For example, at fulfillment of the previously described transformation “model-text” the MetaLanguage system will determine that elements “Man”, “Student”, “Lecturer”, “ExamCards” are instances of the entity “Entity” and will apply to them the transformation rule “Entity”.

V. CONCLUSION

Models transformations are a central part of the model-based approach to system development, because an existence in one system of models, which are fulfilled from the different points of view, with a different level of details and with use of different modeling languages, requires of existence of model transformation tools, which allow to convert models both between various levels of hierarchy, and within one level (at transition from one modeling language to another).

The presented approach has been implemented in a transformer of MetaLanguage system. This component allows to convert models described on visual domain-specific languages to text or other graphical models. The component has a convenient and simple user interface, therefore not only professional developers, but also domain specialists, for example business analysts, can work with it.

With the usage of this approach some languages and models have been developed. As example, the domain specific languages for the queuing system simulation have been designed and rules for transformation of visual simulation models into code in GPSS language have been described [26]. Generated model has been used for simulation running.

REFERENCES