UNIVERSITY OF LATVIA
Institute of Mathematics and Computer Science

EDGARS CELMS

TRANSFORMATION LANGUAGE MOLA AND ITS APPLICATION

Summary of Doctoral Thesis

Advisor:
Professor, Dr. habil. sc. comp.
AUDRIS KALNIŅŠ

Riga - 2007

The thesis was elaborated with support of ESF
(European Social Fund)

Advisor:

Professor, Dr. habil. sc. comp. Audris Kalniņš
University of Latvia

Referees:

Professor, Dr. sc. comp. Jānis Bičevskis
University of Latvia

Assoc. Professor, Dr. sc. ing. Mārīte Kirikova
Riga Technical University

Dr. sc. comp. Uģis Sarkans
European Bioinformatics Institute (Cambridge, United Kingdom)

The defence of the thesis will take place in an open session of the Council for Promotion in Computer Science, the University of Latvia, in the Institute of Mathematics and Computer Science, the University of Latvia *(Room 413, Raiņa bulv. 29, Rīga), on 11. September 2007.*

The thesis (collection of works) and its summary are available at the Library of the University *(Kalpaka bulv. 4, Rīga)* .

Head of the Council Jānis Bārzdiņš

Contents

Relevance of the thesis and the achieved results

Relevance of the thesis:
In recent years, it is typical in specific problem areas, to use specialized modeling languages – domain specific languages [16] (DSL – *Domain Specific Language*). DSLs are created specifically to solve problems in a particular domain and is not intended to be able to solve problems outside of it. Development of support tools for such languages is a labor-consuming process. One of the methods to improve the developing process of DSL tools is to create generic metamodel based tool development platforms, which would ensure convenient and fast development of the corresponding tools. A part of the thesis research has been devoted to the issues of development of such generic metamodel based tools.

Another area, to which the research of the thesis has been devoted, is related to model transformation languages. Model transformation languages have become to be an important part of the non-trivial computer software development process. Nowadays in the developing of computer systems ever more often various metamodels and correspondent models are used. They are taking an increasingly important position in software development. Such development method is referred to as model-driven software development (MDSD) [17, 18]. It is a rapidly developing technology, which is used ever more often for designing different types of software. It must be noted that frequently MDSD approach in software development is based on various DSL languages, with assistance of which the corresponding models are formed. The basic idea in application of MDSD approach in software development is use of models, which can be transformed from one design phase to another by means of corresponding model transformations. Therefore very crucial are the instruments with which such model transformations can be executed – model transformation languages. In practice, it has been proven that neither the traditional programming or modeling languages are sufficiently appropriate for those requirements. In order to implement the ideas of MDSD in practice, a special type of language is necessary. The language MOLA, developed within the framework of the thesis, is exactly that kind of language, which can be conveniently applied to MDSD tasks. It must be noted that regardless of research performed during several years on model transformation languages, the world leading organization in establishment of standards in modeling sphere (OMG[19]) has not yet developed a standard for model transformation language.

The most recent research shows that transformation languages are conveniently applicable in DSL tool building. It must be noted that the transformation language MOLA developed within the framework of the thesis is a typical example of a DSL language.

The main results of the work:

- **A generic metamodel based modeling tool** has been developed – actually, it is a fully metamodel based modeling tool building platform. The main advantage and feature of the tool is that there are no predefined (integrated) modeling methodologies. In order to initiate modeling, the tool must at first be "filled up" with metamodel and additional information describing the desired modeling language and the corresponding technology for work with models. Using this modeling tool building platform a modeling tool of industrial quality supporting specific methodology can be created, using a relatively small effort. It must be stressed that this tool has been successfully used also in further research and by using this tool, an editor of the model transformation language MOLA was developed.

- **Model transformation language MOLA** has been developed – a graphical model transformation language, which has already been tested in practice and is conveniently applicable and very suitable language for MDSD tasks.
- **MOLA Tool** has been developed – it is a combination of various components ensuring all necessary services in order to apply the transformations written in MOLA language for MDSD tasks. The tool ensures such services as transformation writing, compilation, execution, data import and export from/to commercial modeling tools.

It must be noted that the achieved outcome is the result of team work. The author of the thesis has contributed significantly to the work and has actively been participating in all research areas, which are related to the achieved results. The personal contribution of the author is specified in the following sections of the summary and in the annex.

General description of the thesis

The Ph.D. thesis "**Transformation language MOLA and its application**" has been elaborated during the time period from year 2001 until year 2007 at the Department of Physics and Mathematics of the University of Latvia and in the Institute of Mathematics and Computer Science (IMCS) under supervision of professor Audris Kalniņš. This work carries on the IMCS traditions of tool building and language development, which had been initiated already in year 1986. The main contents of the thesis is reflected in 13 publications [1-13] and presented in 12 international conferences. The thesis is formed as **a set of thematically related papers** about various aspects related to metamodel based modeling tool building and model transformation languages.

The subject of the research – development of generic metamodel based modeling tools and model transformation languages.

The goals of the research – to develop and implement in practice a generic metamodel based modeling tool, as well as to develop and experimentally test a model transformation language, which could be used in MDSD tasks.

Research stages – the results of the thesis form a closed research cycle, beginning with the research yielding one of the result of the thesis – generic modeling tool (the tool was later applied in development of model transformation language MOLA) and ending with research on transformation language MOLA and its implementation and application to MDSD tasks. The research is summarized in a thematic order (it conforms also to the chronological sequence of research):

- **The first research stage** – from year 2000 until year 2004. The research and results of this stage are described in Chapter 2 of the summary and in corresponding publications [1-5]. The main result of this stage is the developed **generic metamodel based modeling tool**, which in many aspects during that time was unique and superior over other tools, which were built for similar goals. The second significant result was the acquired comprehension and knowledge about model transformations. The research results achieved during this time period were the main idea resource for further development of model transformation language MOLA and for related research. The author of the thesis has actively been participating in all research during this time period, beginning with year 2001. The author's contribution in general constitutes over 40% of all performed research during this research stage.

- **The second research stage** – from year 2003 until year 2006. The research and results of this research stage are described in Chapter 3 of the summary and in the corresponding publications [5-10]. The most important research result is the developed **model transformation language MOLA.** It must be noted that the research in this area are being carried on and currently the next version of the language is being developed. An important research result was also the performed efficiency evaluation of MOLA language pattern matching. The author of the thesis has been actively participating in all research that was performed during this time period and the personal contribution of the author constitutes over 50% from all of the performed research during this research period. It must be noted that the solutions developed during these research later had a very significant practical importance for efficient implementation of MOLA language.

- **The third research stage** – from year 2005 until year 2007. The research and results of this stage are described in Chapter 4 of the summary and in corresponding publications [11-13]. The main research result is the developed **MOLA Tool**. The author has been actively participating in all research that was performed during this time period and the personal contribution of the author in overall constitutes over 30% from all of the performed research during this period. It must be noted that the research in this area is being carried on intensively and currently the next version of MOLA Tool is under development.

The theoretical and practical significance of the research
The main results of the research of the thesis are reflected in 13 publications [1-13]. The most significant results have been reported in important international scientific conferences in the research area. The author of the thesis has presented the results in seven of them. The author has also given demonstration of the MOLA Tool during the tool demonstration section of the international conference ("*European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA)*") [20].

An evident proof of the **practical importance** of the research results is the application of the developed model transformation language MOLA in various projects. Following are listed the most interesting projects:

- EU's Sixth Framework Program (FP6) project – ReDSeeDS (*Requirements-Driven Software Development System*) [21]. In the ReDSeeDS project, the MOLA language is used as an instrument for definition and implementation of transformations.
- Currently at IMCS, a new generation metamodel and model transformation-based tool framework [22] is being developed, within which the model transformations are built in MOLA language.
- A master course "MDA and model transformations" has been elaborated and introduced in the master studies of Computer Science at the University of Latvia. MOLA language has been widely used both in students' practical works and in description of the basic principles of model-driven software development.

The following chapters of the summary contain short description of the most significant results and basic problem settings, which in the corresponding publications are described in more detail:

- In the first chapter, a general description is provided about the basic development principles of model-driven software and about the key terms, thus supplying the reader with the basic knowledge that is required for better understanding of the research performed by the author and about the significance of the achieved results.
- From the second to the fourth chapter, a summarized description is provided about the research performed and results achieved.

- In the conclusion, a synoptic summary of the research results and shortly described further research directions are given.
- In the annex, listed are those international scientific conferences, in which the author has presented the achieved research results, and the personal contribution of the author is specified for each of the publications included in the thesis.

1 Metamodeling and model-driven software development

Nowadays when developing non-trivial computer software, during the development process, ever more often various metamodels and corresponding models are applied. They take an ever more important role in software development. During the system development process these models are subjected to various transformations (modifications). However in order to describe the transformations, the transformation languages are used. Such development method is called model driven software development (MDSD) [17, 18]. In the chapter, a general overview is provided about this area of computer science and application thereof in system development, as well as about the main terms of the sector, thus ensuring the reader with the basic knowledge required for better understanding of further chapters.

1.1 Metamodeling, metamodels, and models

The concept of "model" in computer software development is considered a theoretical construction displaying a specific fragment of the world. A model is a set of interconnected objects, which displays a certain idea. Physically the model usually is a graphical image, in which **according to established rules** various graphical elements are formed and placed. It must be noted that the graphical representation of a model is not a mandatory requirement in model construction. The model can also be in textual format.

As it has been mentioned above, the models are formed in strict accordance to established modeling rules. What establishes these rules? Generally speaking, it can be said that each model is created in compliance with regulations that are established by a certain higher level model - **a metamodel.** A metamodel can be perceived as a general schema or language syntax, according to which models are formed. For instance, one metamodel can be used for illustrating the organizational structure of an enterprise, and another – for description of an information system. In each of the above mentioned cases, the metamodel establishes the rules, according to which the models can be built, or to express it with other words – models would be the instances of some metamodel. For instance, by using a metamodel displaying the organizational structure of an enterprise, each enterprise can create an individual model, following this one common metamodel. For various enterprises these models will turn out different, but they will be unified by a common metamodel and by certain common rules. In a broader understanding, **a model** can be any type of **formalized** (understandable for the computer) artifact, which appears during the development process and the structure of the model can be described with a metamodel.

Metamodel is also called the model of the models. Respectively, the metamodel is the model itself, which has been formed in compliance with a certain higher level metamodel (meta-metamodel). Each specific metamodel and its corresponding models define two metamodeling abstraction levels. Theoretically, it can be talked about arbitrary number of metamodeling levels, where each of the upcoming levels is the meta-level of the previous level. In practice, usually only four abstraction levels are used: M0 – model instance level, M1 – model level, M2 – metamodel level, and M3 – meta-metamodel level.

Currently the most frequently used metamodeling standard is by OMG[19] elaborated MOF[23] (*Meta-Object Facility*) standard, which establishes four abstraction levels (see Figure 1.1). The current MOF version right now is 2.0 [24], and soon it is planned to complete the work with version 2.1, however it is expected that it will not particularly change MOF applicability in practice.

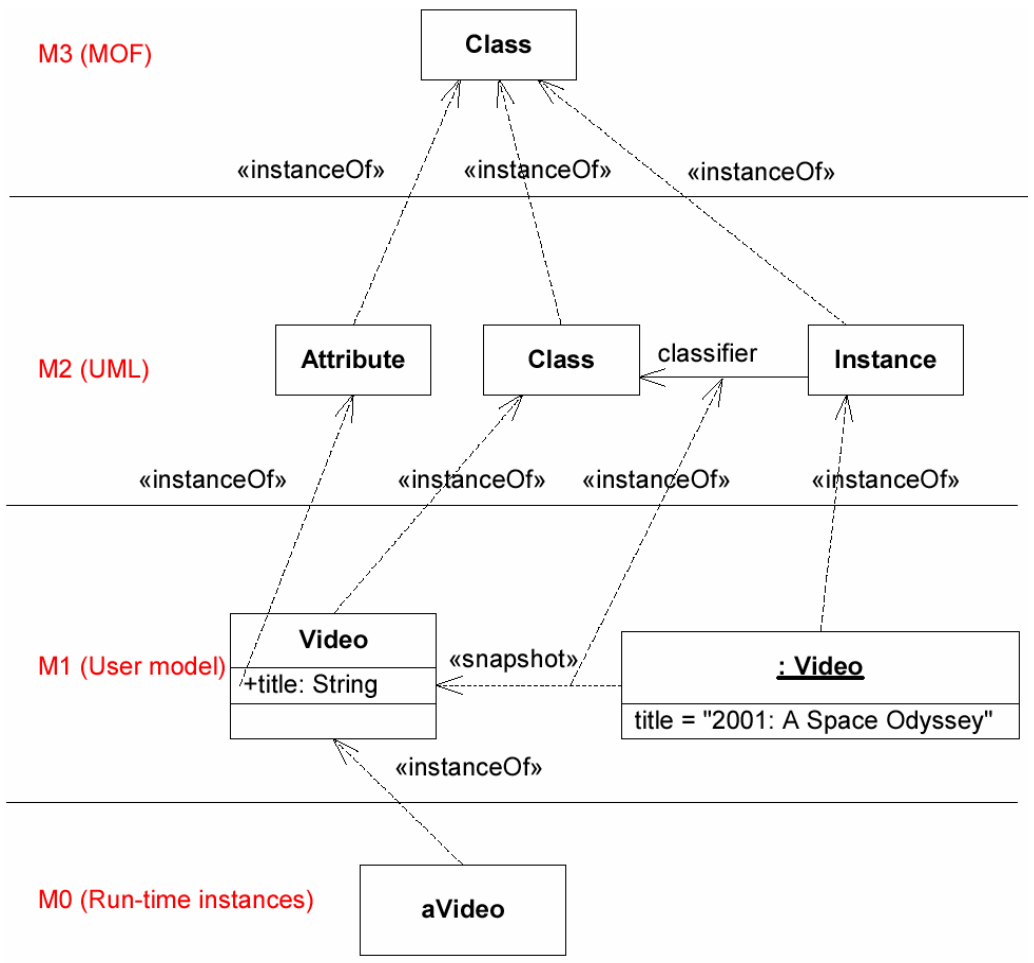

Figure 1.1 OMG meta-level hierarchy example (Figure from [**27**])

In practice, numerous models are described in either of the versions of the modeling language UML[25, 26, 27, 28, 29] (i.e., conforming to the UML metamodel, which in turn is defined in MOF). It must be marked that the models can be described also in other languages, including any of object-oriented (OO) programming languages (the abstract syntax of a specific OO programming also corresponds to the meta-model).

Several typical model examples:

- UML activity diagram – business process model.
- Set of UML use case and activity diagrams – specification of system requirements.

- Set of UML class diagrams – system analysis model.
- UML class diagrams, where J2EE stereotypes are used – design model.
- Java program (written in abstract syntax).
- J2EE program (components + descriptors), also in abstract syntax.
- Workflow definition in either of the defining languages (for instance, BPMN language [30]).
- Formal schema of Web pages (without “decorations”).

1.2 Model-driven software development

At the end of the nineties and during the first part of this decade, a situation had formed that the necessary experience had been accumulated, in order to implement new paradigm and new principles in software development. Such technologies and knowledge was acquired as metamodeling, operations with models, application of UML diagrams in software development, various component development environments (EJB, .NET, CORBA), and object-oriented software languages. However it all together had not rendered the expected effect in software development and a significant efficiency leap had not been achieved.

In year 2000, OMG initiated a new project – Model Driven Architecture (MDA [31, 32]). Specific results appeared only during the second half of year 2001, when OMG published the first version of MDA Guide [33], where the basic ideas and applications of MDA were described. The main idea was that in development of non-trivial systems various meta-modeling primitives should be systematically used. Important was the finding that in practice the models are not appropriately used, their potential is not exploited, and that the models possess various roles during the system development process. Since that moment, MDA and issues related to it have been rapidly developing and even today the development pace has not decreased.

It is interesting that intuitively similar ideas had been used by many in daily practice, but there were no successful attempts to formalize, arrange, and elaborate a methodology of how to use it consequently in software system development.

MDA implemented the model role concept in the system development process. Three types of models were offered:

- Platform Independent model – **PIM**, it is a model where platform-specific elements do not appear. It describes the system, but does not show details of its use of its platform.
- Platform Specific Model – **PSM**, is a model where PIM described matters are combined with specific concepts of a specific platform (for instance, EJB, .NET, *WebServices*, CORBA).
- Computation Independent Model – **CIM**, it is the conceptual or business model of the system. It must be noted that if a CIM model is used in software development process, then it is more in a role of a documentation component and not as a formal document “understandable to the computer”.

Another important formalism, which was implemented in MDA approach, is model transformations. Model transformation is a process that has a very significant role in model driven software development. During the development process, one model needs to be modified into another according to a certain given method, by transferring and incorporating all of the necessary information to the other model. The transformation process itself involves modification of a certain model (source model) or metamodel (source metamodel) into a different model (target model) or metamodel (target metamodel). The most common case that the developers most frequently operate with is the model transformations. It means – to take a certain model of a fixed metamodel (source metamodel) and to transform it into a model of a different (or the same) fixed metamodel (target metamodel) (see Figure 1.2). It must be noted that also transformations of the metamodels are not unusual and are a frequently exploited option in practice.

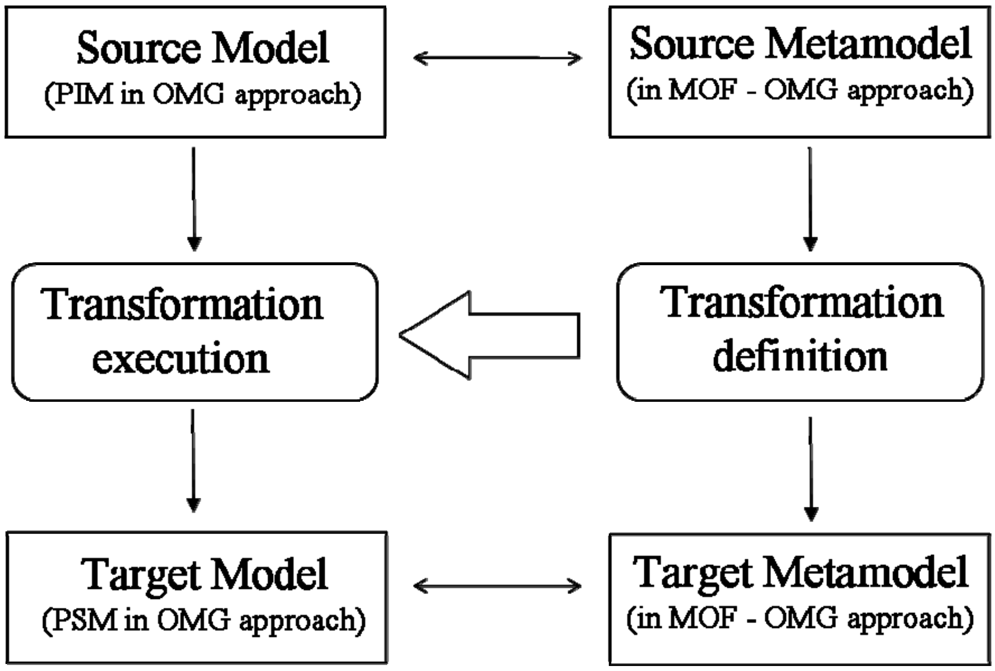

Figure 1.2 Transformation application schema

In the course of time, various approaches have been developed on how the models and corresponding transformations are used in software development. In the classic OMG approach, MDA application is designed in such manner that it consists of one platform independent model and of one or more platform specific models, and complete realization thereof in each of the platforms that the application developer has chosen to support. In addition, during the application development process transformations are applied which are part to the development process. It must be noted that in MDA approach it is considered that the only OMG offered metamodeling instruments are used (MOF, UML).

In the model driven software development (MSDS) it is looked at from a broader perspective. In the MDSD approach, there is no “firm” involvement with MOF and UML, and numerous (of course, formalized) metamodeling instruments can be used. It must be noted that in specific problem sectors, a widespread practice is to use specialized graphical modeling languages – domain specific languages [16] (DSL), which usually are very different from UML.

Another significant difference in MDSD approach, in comparison to MDA, is connected to the model application. In the MDSD approach, the source and target models are any two consecutive models that are used during the development process. The meaning of PIM and PSM in fact is a relative and it is dependent only on the specific level of abstraction, from which it is perceived. For instance, even such a specific model as Java program, if written in its abstract syntax, can be both as a PSM model and as a PIM model. Such Java program could be in a role of PSM model for design class diagrams and in a role of PIM model for a specific development environment platform, which in turn is a PSM model for the above mentioned Java program. The subject matter is only the logical model chain during the development process. It must be added that MDA is only one of the variants of MDSD application.

At the end of the chapter, I would like to emphasize the most characteristic features of the model driven software development approach:

- The metamodels and the corresponding models are the main artifacts of software development. It is a new paradigm in software development – a new method of developing systems. By such development of computer systems, "elevation" to another abstraction level occurs (operations with metamodels and models).
- Use of transformations. By applying the MDSD approach, it is necessary to formally define the transformations; in addition, the transformations are not a part of the model, but instead they are a part of the development process.

1.3 Model transformation languages

As mentioned in the previous chapter, by using a model driven development approach, one model must be transformed (modified) into another according to a specific given method, by transferring and incorporating all of the necessary information to the other model. It must be added that such model transformations can be both manual (used before MDSD) and automatic - transformations performed with formal transformation assistance (in case of MDSD approach). However in order to define the transformation processes, a model transformation language is required. The model transformation language is a programming language, which is provided for describing model transforming processes. A program that is written in either of the model transformation languages in the most typical case would receive a model of a certain metamodel as the input data, and would return a model of another (or the same) metamodel as the data output (see Figure 1.2).

Thus, in order to implement the MDSD ideas in practice, a special kind of language provided specifically for these purposes – the model transformation language – was necessary. In practice, it proved that neither of the existing programming or modeling languages was sufficiently suitable to those requirements that were necessary in model transformations, and in April of 2002, OMG announced a request for proposal (RFP) to the standard of such language – QVT (*Queries/Views/Transformations* [34]).

The model transformation languages are a fundamentally new language class. The main requirements, to which the model transformation languages must conform, are as follows:

- They must service the arbitrary models of metamodels (within understanding of OMG only those conforming to MOF metamodels).
- They must process the source models and produce the target models (must be able to describe and execute the transformations between the models, which conform to specific metamodels).
- They must be metamodel based, respectively, like the transformation program, which is written in either of the transformation languages, is to be perceived as a model that conforms to the metamodel of the given transformation language.
- They can be either graphical or in textual format.
- They must be "easily" understandable both for people and for computers (must be declarative as much as possible).
- There must be a corresponding tool support in order to conveniently create, alter, and execute these transformations.

Various projects on language standard were submitted, which during the course of time did not develop any further or were combined and only one standard project remained – MOF QVT[35], in development of which 16 institutions were participating, including IBM, Sun, and four universities. This project unites multiple initial projects. In March of 2005, the standard language was supposed to be complete according to the plan, however at this moment (year 2007) it is still only in the planning stage and is delivered to the OMG platform task committee (PTC) for development of the final version [35].

Simultaneously a range of other model transformation languages not related directly to the OMG request for proposal were developed – MOLA[6-13], Lx[36], GReAT[37, 38], UMLX[39], ATL[40, 41], TRL[42], Tefkat[43], AGG[44], MTF[45], ATOM[46], VMTS[47], BOTL[48], YATL[49], Fujaba[50], VIATRA2[51], RubyTL[52], ALAN[53], MT[54], and other. It is interesting that still now new model transformation languages are developed and the existing ones – improved. Amongst the above mentioned languages, there are both graphical and textual languages. Particularly must be mentioned the graphical model transformation language MOLA, development of which is a part of the research results performed by the author within the framework of the thesis. A short description of the research performed in this area and description of the MOLA language is provided in Chapter 3 of the summary.

2 Editor definition language and generic modeling tool

In this chapter, the results of the first research stage of the thesis (year 2000 until year 2004) are reviewed, which are summarized in five publications [1,2,3,4,5] and presented in four international conferences. The author has actively been participating in all research during this research period, beginning with year 2001.

2.1 Editor definition language and the prototype of the generic modeling tool

During this time period, research was started, which lead to the development of the model transformation language – MOLA. When initiating the research, the problem sector was diagram processing tools. In practice, there was a necessity for flexible diagrammatic editor, which would be based on a specific metamodel [57]. As a result of the research, a methodology and tools were developed for defining such diagrammatic editors and a generic metamodel based modeling **tool prototype** for supporting this methodology was built. Below listed are the main elements of the developed tool prototype:

- **Logical metamodel** – definition of the logical structure of the diagram objects, by using UML class diagrams. The logical metamodel in our current terminology would be referred to as the domain metamodel. It describes concepts of a certain problem region, as well as relations between them.
- **Editor Definition Language** (EdDL) – a language, with assistance of which the graphical representation of objects and the editor behavior (dynamics) is defined. With the assistance of this language, the following components and features of the editor can be described: graphical representation of the logical elements, use of subdiagrams, various promptings, navigations facilities between the diagrams, symbol palette, and other less important features.
- **Annotation compiler** (*EdDL parser*).
- **Editor engine** – software, which provides editor runtime operations.
- **Graphical diagramming engine** (GDE) – software that implements diagram drawing, automatic placement, and other functionality related to graphical object manipulation.

The greatest contribution of the author in this research stage is development of EdDL basic constructions and the **annotation compiler** and **editor engine** designed and implemented by the author himself.

In order to build an editor, it was necessary to perform several interrelated tool definition stages (see Figure 2.2):

- At first, a metamodel was formed (or more precisely – its view for one diagram) as a UML class diagram.

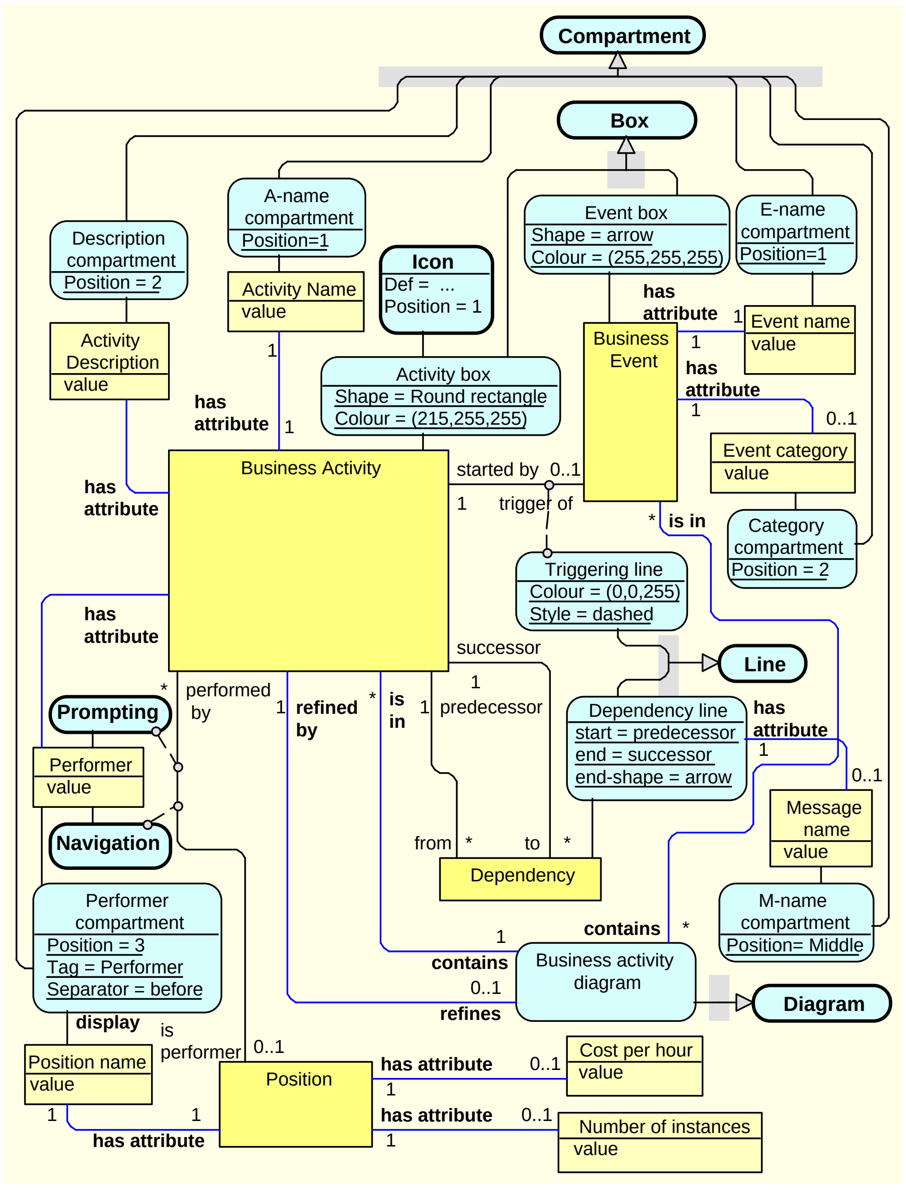

Figure 2.1 With EdDL annotated logical metamodel

- Next, the metamodel was supplemented with the necessary specification of the graphical representation, using EdDL constructions. Respectively, the metamodel was annotated (see Figure 2.1). In the figure, some of the offered EdDL language elements can be viewed. The fundamental idea was to add the presentation level elements into the metamodel, for instance, such as `Box`, `Line`, `Compartment`, `Diagram`, and more, in whose corresponding sub-classes all of the information of the graphical element presentation of the specific diagrammatic tool was stored. However, the presentation elements thereafter were "connected" with the corresponding domain classes, by applying the mapping associations. For instance, for the class of `Business Activity` it is determined that the instances are to be displayed in a form of boxes (the attached subclass `Activity Box` of the class `Box`) with the corresponding default dimensions and color. In addition, it can be seen that this class contains a range of attributes (in the presentation level terms – compartments*)*, which are also correspondingly annotated. The consequences of such compartment annotation were that it was beneficial to create a slightly “bizarre” class diagram, in which the domain class compartments are not displayed in the same class. It must be remarked that this and a range of other inconveniences were eliminated in result of the following research, when presentations and domain metamodels were separated from each other.

- The following step for construction a diagrammatic editor is a compilation of the annotated metamodel. It is implemented by annotation compiler (*EdDL parser*), which transforms the annotated metamodel into its internal format, which is “understood” by the editor engine. The runtime component – editor engine interprets this internal format and the end user “receives” a completely finished diagrammatic editor (see Figure 2.3).

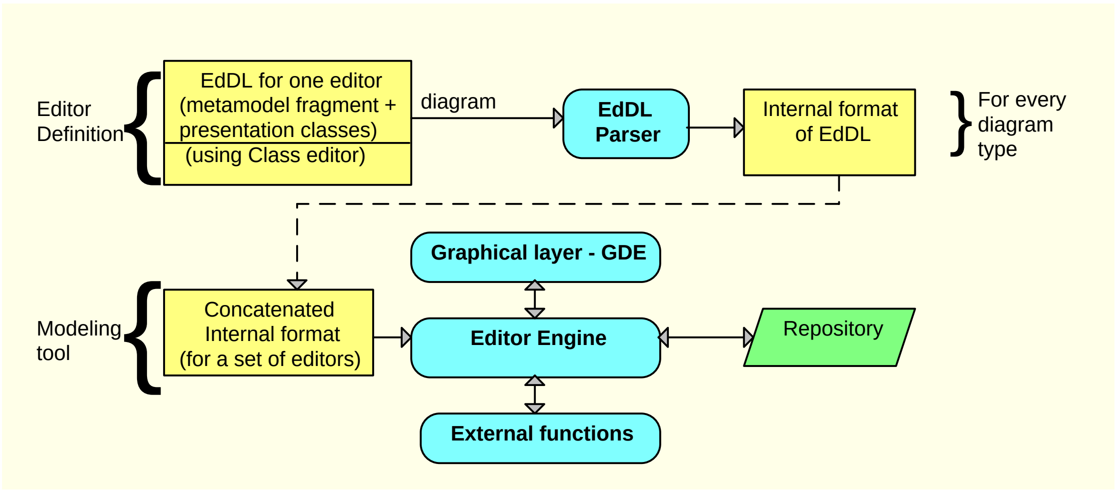

Figure 2.2 The key elements of the generic metamodel based modeling tool prototype

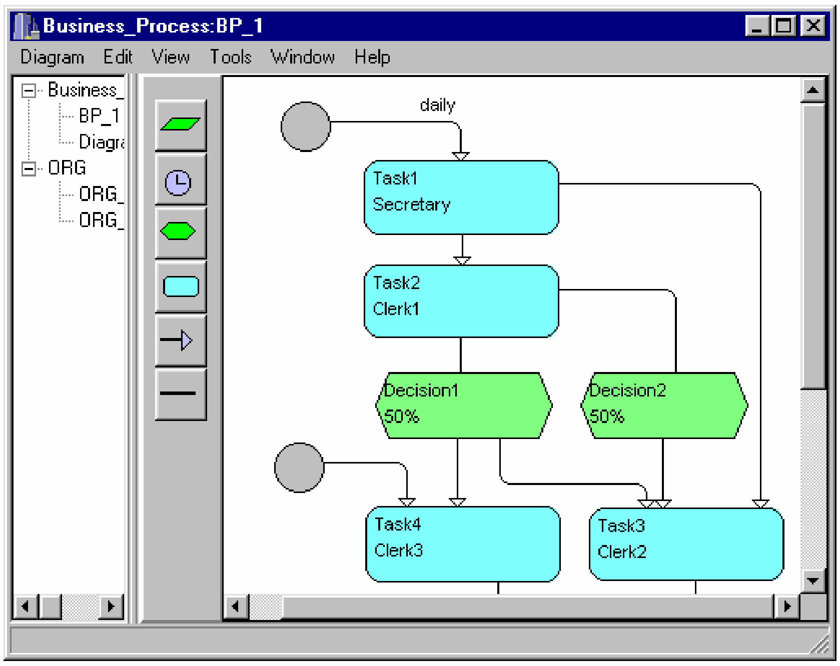

Figure 2.3 Generated editor in action

It must be added that a range of tools, which solved similar problems, had already been developed. The main difference from the IMCS developed generic modeling tool prototype was that they typically were constructed as completed editor sets with vast possibilities of configuration. They could be applied only to a previously defined problem domain. Whereas the IMCS developed tool was more universal. It was based on the logical metamodel and on the relatively independent definition language EdDL, which provided with an opportunity to use the tool for a wide range of the graphical editors. Some of the most interesting tools of the time, which claimed a similar problem space, were:

- **Metaedit**, developer MetaCase Consulting [58]
- **KOGGE**, developer University of Koblenz [59]
- **Toolbuilder**, developer Lincoln Software [60]
- **DOME**, developer Honeywell [61]
- **Moses**, developer ETH Zurich [62]

2.2 Generic modeling tool

Upon initially acquired research results about EdDL and its application in development of diagrammatic editors, a conviction was growing about opportunities and ideas appeared about how this language could be expanded and improved in order to use it in a wider context. The problem research area was development of generic modeling tools. A more detailed attention was paid specifically to the area of business process modeling tools, about which IMCS had already accumulated vast experience.

The situation that had established in the area of modeling tools at that moment was such that none of the available tools was satisfying all of the requirements in the specific sector, for which it was provided. Each of the tools had advantages and disadvantages. Situation was especially difficult in use of domain specific languages [16] (*DSL*) in enterprises. Each of such languages was unique in a way and differed from other. Even in such a relatively stable sector as UML there were problems with tool application. Typically – in large enterprises, the UML tools required various specific additions, which could not be implemented, by using the expansion services offered by the existing tools.

There are various options of how to solve the above mentioned problems in the modeling tool area:

- A tool can be developed individually for each of the modeling methods.
- An attempt to develop utmost generic modeling tool could be performed, in order to satisfy as large of a spectrum of modeling methods as possible.
- It can be attempted to develop a completely metamodel based generic tool development platform, where there would not be any integrated previously defined modeling methods. Moreover, by using such platform, it would be possible to create quickly and at low costs the necessary sets of tools.

By researching these problems and according to the previous experience, which was acquired during development of a generic modeling tool prototype, it was concluded that the most reasonable and also the most interesting problem solution would be development of a generic tool development platform. The requirements were established, which should be satisfied by such tool development platform. Below listed are the main requirements:

- The tool must be completely metamodel based. It would mean that the tool does not have any predefined (integrated) modeling methodologies. In order to begin modeling, the tool must be at first "filled up" with the metamodel and the additional information, which would describe the desired modeling methodology.
- The tool must support various modeling notations, by using a common domain metamodel. It means that the user can create a model in one notation and afterwards operate with it in a different notation (the idea never got completely implemented – the significance of transformations and use thereof had not yet been conceived).
- The tool must ensure universal metamodel based editing options. It means that following only the information existent in the metamodel, the tool should offer the users coherent domain data editing options.

As a result of the research, a methodology and various components (a platform) was developed for development of such generic modeling tools – a generic metamodel based modeling tool (*Exigen Business Modeler*). Below listed are the main components of the developed platform:

- **Domain metamodel.** It defines the modeling ideas (concepts), their attributes (data), and relations amongst them (see bottom part of Figure 2.3).

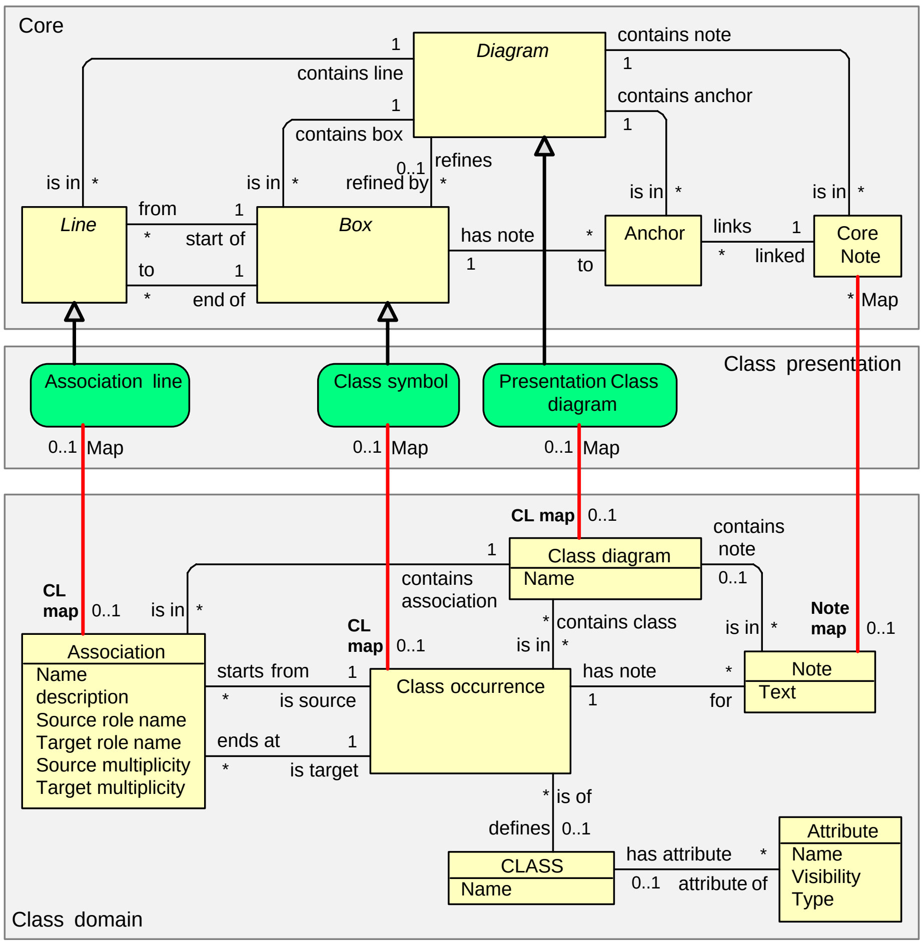

Figure 2.3 A fragment of an extended metamodel example (for a simplified class diagram)

- **Extended metamodel.** It consists of two important parts – metamodel presentation part and metamodel core part (see Figure 2.3). The extended metamodel classes are connected with the corresponding domain metamodel classes, using mapping associations.
- **Diagram annotation definition** (mapping definition) and its runtime interpretation service. The service ensures an option to describe the manner of logical elements (modeling concepts) being displayed as diagram elements (mapping), as well as defines the tool's behavior for operations with diagrams (see Figure 2.4). This component of the platform played a special role within the context of the upcoming research, which were connected with model transformation language MOLA. It is the most important part of the diagram definition in a generic tool. With the mappings it is defined how the logical (domain) elements are displayed as diagram elements. The mapping syntax (see Figure 2.4) is defined with mapping types (patterns). The mapping type contains references to the metamodel classes and associations, which is a "closely related" to the pattern concept, which was later utilized in MOLA language. However the mapping semantics is dependent on the corresponding mapping type and it establishes the semantic operations to be performed with diagram alterations. In modern understanding, these activities were programmatically hard-coded transformations. However within the context of MOLA language it would be the actions to be performed according to the rule, which contains the given MOLA language pattern. In publication [4], it is described in detail, how to define the formal semantics of the mapping by using OCL [63] language. Respectively, to each mapping type a corresponding OCL expression can be defined. Up to limited extent, the tool also ensured the possibility to supplement the existing mapping types with additional OCL expressions.

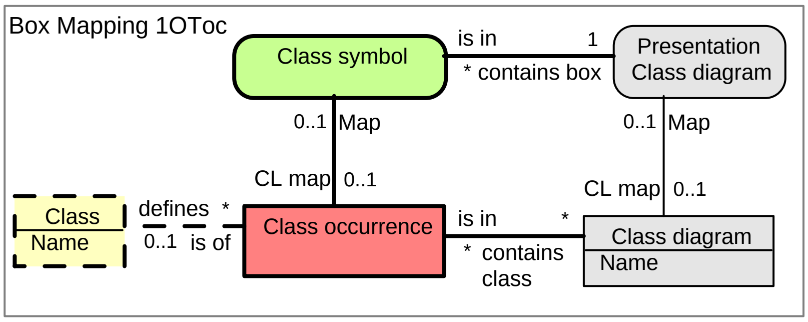

Figure 2.4 Mapping Example (1OToc mapping)

- The definition and interpreting service of the **graphical style of the diagram elements**.
- **Tree model engine** – flexible definition and interpretation of a metamodel based tree model. In definition of the model tree, a range of options are used, with combination of which it is possible to define practically any coherent model tree structure. The most frequently used parts in defining a model tree are:
  - Fixed tree elements – used to form a fixed basic structure of the model tree.
  - Object elements – during tree runtime interpretation (implementation), they are "filled up" with the corresponding metamodel class instances.
  - Reference elements – used for definition of recursive tree elements.
  - The selection criteria for the object elements – one of the "most powerful" tree definition characteristics. It allows setting specific, similar to OCL

selection criteria that are based on the metamodel associations and class attribute values.
    - Definition facilities of pop-up menus – ensures the option to create user defined menus for the tree model elements.
- **Generic property editor engine** – ensures definition and interpretation of metamodel based domain data. By using the domain's metamodel class information (attributes and their types, associations with other classes, cardinalities, etc.), the generic editor generates the dialogue windows (editors), with assistance of which it is possible to perform domain data editing. Various heuristics are applied in editor production, which had proven their usefulness during the upcoming years. For instance, one of them is – with composition type associations attached class instance set is displayed in the editor as included tabular editors. Of course, this was not sufficient to implement all of the necessary non-standard cases. According to the current understanding, they should be implemented by using model transformations, however there were no such facilities available at that time. Nevertheless, there was an option to configure the generated generic editors and in most cases thus the editor quality equivalent to the property editors of commercial modeling tools could be acquired. The MOLA language editor is constructed exactly in such way – by using the configured and partially also the non-configured generic metamodel based property editors (see Figure 2.5).
- **Generic table editor engine** – ensures tabular editor definition and interpretation of metamodel based domain data. It was implemented by applying similar principles as with generic property editors. Its key value is the offered option to review all instances of a certain metamodel class along with the corresponding attributes. During the course of time, this option proved to be a very useful as the default instance browser.

The most significant contribution of the author in this research stage is the development of the basic principles of the generic metamodel based modeling tool architecture, as well as the investment of the author in developing the basic principles of the **diagram annotation definition** (mapping) and the **generic modeling tool manager, model tree engine, generic property editor engine,** and **generic table editor engine** all of them are designed and implemented by the author himself.

The developed generic metamodel based tool was not the only available tool of that kind. According to the functionality, the closest counterparts were Metaedit+[64], GME[65] and ATOM3[66]. The tool developed by IMCS was more universal, more flexible, and with a broader range of various definition options of modeling technology. In the developed tool, a mapping type library is included, which ensures mapping of domain elements in corresponding diagrams. By using the tool, it has been proved that the selected set of mapping types is practically sufficient in order to create a very vast family of modeling tools. However in functionally similar tools, the mapping definition options are very poor. Only the very basic mapping facilities are included in them, for instance, one specific domain element displayed as one presentation (diagram) element. It must be added that in some tools, addition of mapping types is available, by programming in one of the traditional programming languages, for instance, C++ language in GME tool. However it can be performed using inconvenient and labor-consuming low level programming. The fact must be stressed that in the tool developed within framework of the thesis, in comparison with similar tools, a practically applicable graphical editor can be created in a very short time period. For instance, creation of a practically usable UML class editor would not require more than three days of work.

In conclusion, a short summary about this research time period:

- Experience and knowledge was accumulated on how should and, of course, how should not the universal tools be constructed.
- A new generic modeling tool (*Exigen Business Modeler*) was constructed, which is still actively being used for various experiments in IMCS research projects. It must be noted that the editor of model transformation language MOLA was also implemented, by using this tool (see Figure 2.5).
- Mapping, pattern, and transformation concepts were implemented:
    - Mapping definition – a pattern concept, something very "closely related" to the pattern concept, which was later used in MOLA language.
    - Mapping implementation – hard-coded transformations. In MOLA language context, these would be the actions to be performed according to the rule, in which the given pattern is included.
- **Very important** – the achieved results stimulated to think within the context of MDA and MDSD ideas. Knowledge and skills were acquired on how to use the model transformations in tool development, as well as understanding was discovered on what the transformation language should be in order to use it conveniently; also, the transformation significance and its place in software development was conceived. It must be marked that the research results acquired during this stage were a significant source of ideas and inspiration for further research on model transformation language MOLA.

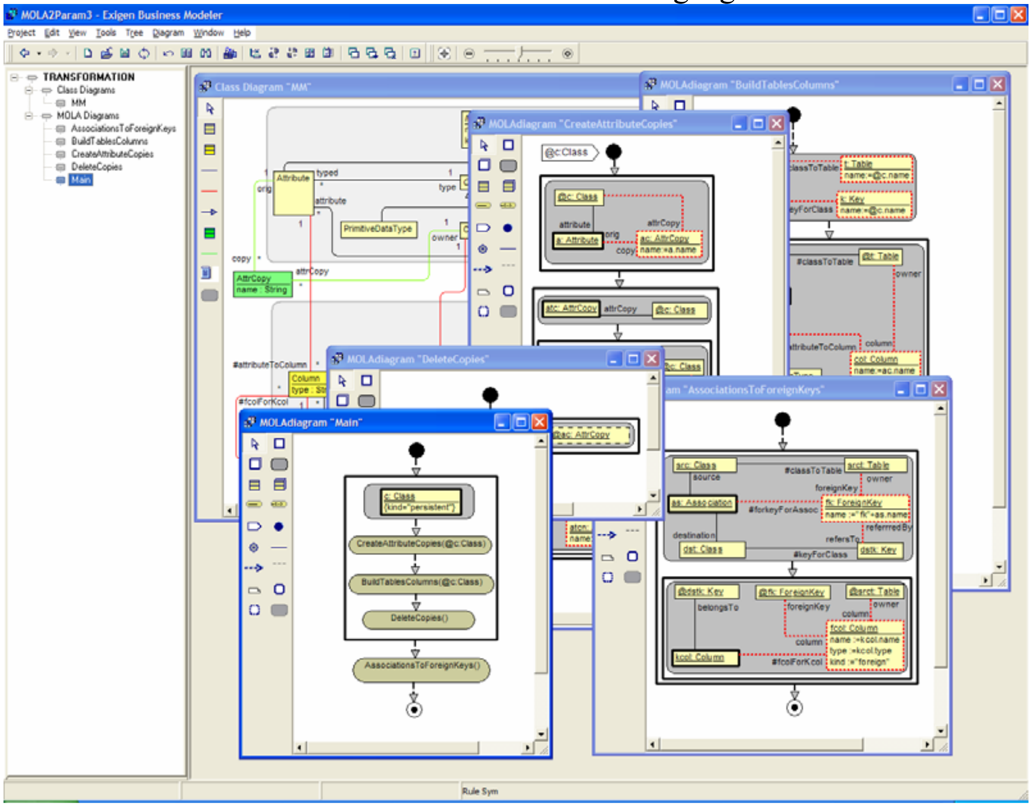
MOLA2Param3 - Exigen Business Modeler
Project Edit View Tools Tree Diagram Window Help
TRANSFORMATION
Class Diagrams
MM
MOLA Diagrams
AssociationsToForeignKeys
BuildTablesColumns
CreateAttributeCopies
DeleteCopies
Main
Class Diagram "MM"
MOLAdiagram "BuildTablesColumns"
MOLAdiagram "CreateAttributeCopies"
MOLAdiagram "DeleteCopies"
MOLAdiagram "Main"
am "AssociationsToForeignKeys"
CreateAttributeCopies(@c:Class)
BuildTablesColumns(@c:Class)
DeleteCopies()
AssociationsToForeignKeys()
Rule Sym

Figure 2.5 Example of operations of the generic modeling tool (MOLA language editor).

3 Model transformation language MOLA and its application methodology

In this chapter, the results of the second research stage (year 2003 – year 2006) of the thesis are reviewed, which are summarized in five publications [6,7,8,9,10] and presented in five international conferences. The author has actively been participating in all activities related to this research stage.

Research about the model transformation language MOLA was a logical continuation of the activities, which were related to the development of universal modeling tools. As it has already been mentioned, the MOLA language editor was built, by using the generic metamodel based modeling tool (see Figure 2.5) developed during the previous research stage.

3.1 Model transformation language MOLA

The experience accumulated during the research of the previous years, the acquired results, and the ideas expressed in OMG's MDA [33] initiative had provided with comprehension and knowledge on how to use model transformations. The significance and role of transformations in software development was realized. Understanding was conceived on what the model transformation language should look like in order to be conveniently usable in model driven software development (MDSD).

When designing the MOLA language developed within framework of the thesis, the main goals were:

- To create a language in which the transformations could be defined as conveniently as possible.
- The transformations should be "easily" understandable (readable) both for the user and for the computer.
- The language should be easily implementable. In addition, later it turned out to be a very important aspect of the language.

The developed MOLA language is a unique graphical model transformation language. Those language elements that also graphically are easily conceivable are displayed graphically. Widely used in the language is pattern application, which is combined with simple iterative control structures, which are adopted from the traditional structural programming. OO elements are also introduced in the language having certain sense in the given context. For instance, such classic OO programming ideas as inheritance and polymorphism are also supported.

Each MOLA program defines one transformation and it consists of a metamodel and MOLA transformation. Metamodel in MOLA language is understood as a class diagram, where included are the parts of source metamodel and target metamodel, and mapping association (see Figure 3.1). In general situation, the source and target metamodel sections can also coincide. MOLA transformation is one or more MOLA procedures (diagrams), of which one is the main.

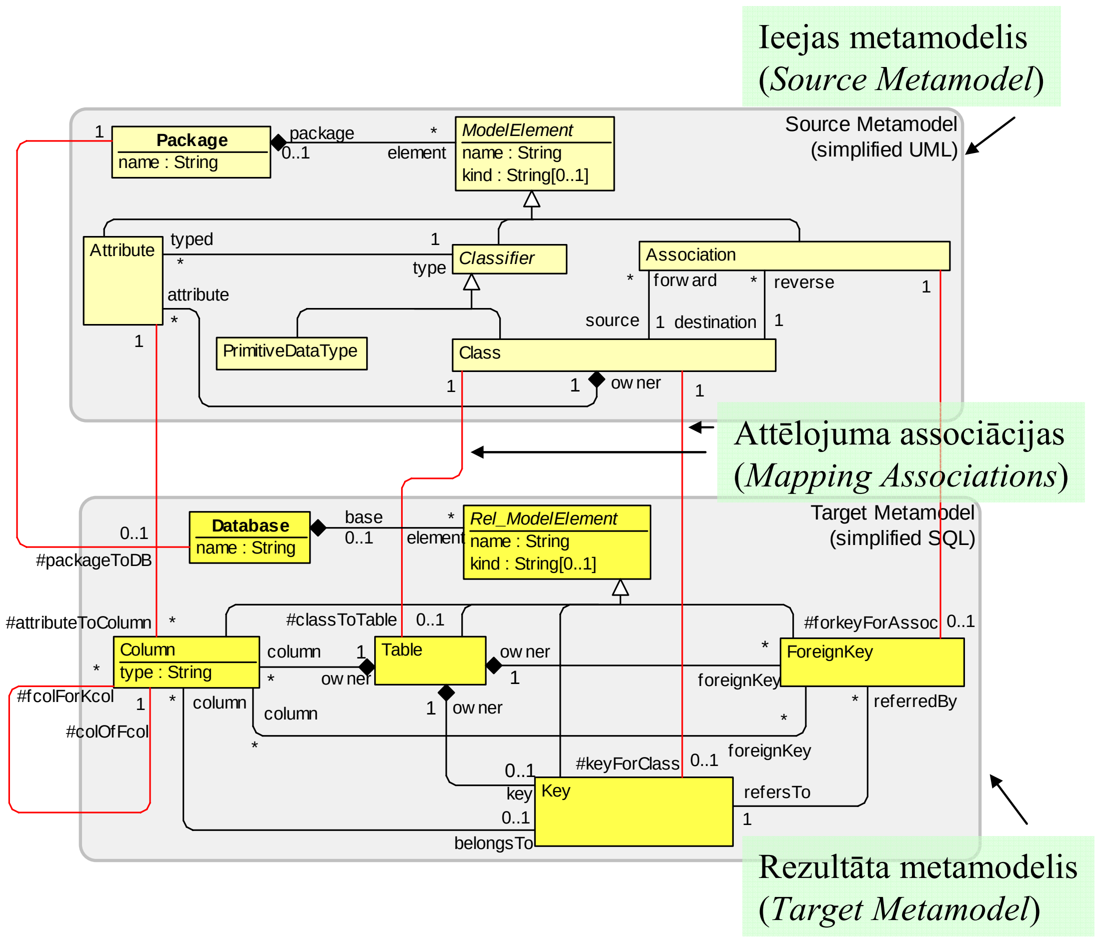

Figure 3.1 Source and target metamodel example

Further, a short description about some of the most frequently used MOLA language basic constructions is given (see also Figure 3.2). MOLA transformation program is a sequence of graphical instructions, which are connected with arrows. One of the most frequently used instruction types is `foreach` loop. The `foreach` loop contains the loop head statement and the loop variable. The loops can also be included in each other. Another very important language construction is rules. It must be added that the loop head is a specific case of the rule. The rule is the pattern, which is combined with actions. Patterns hold a special significance in MOLA language – they establish the instance group, with which certain activities are going to be performed. In MOLA language, a pattern is a fragment of the metamodel, in which each of the class (pattern) elements has a name and in which each class may appear several times with a certain specific role (similar as in the UML communication diagrams the role name is added to the class). The elements are defined in instance notation and they reference to the corresponding metamodel class. The elements can be interconnected with links, which conform to the corresponding metamodel associations. The class elements may have also constraints, which are defined in a very limited OCL subset. It must be stressed that loops, rules, patterns, and corresponding actions to be performed with the instances, are displayed graphically.

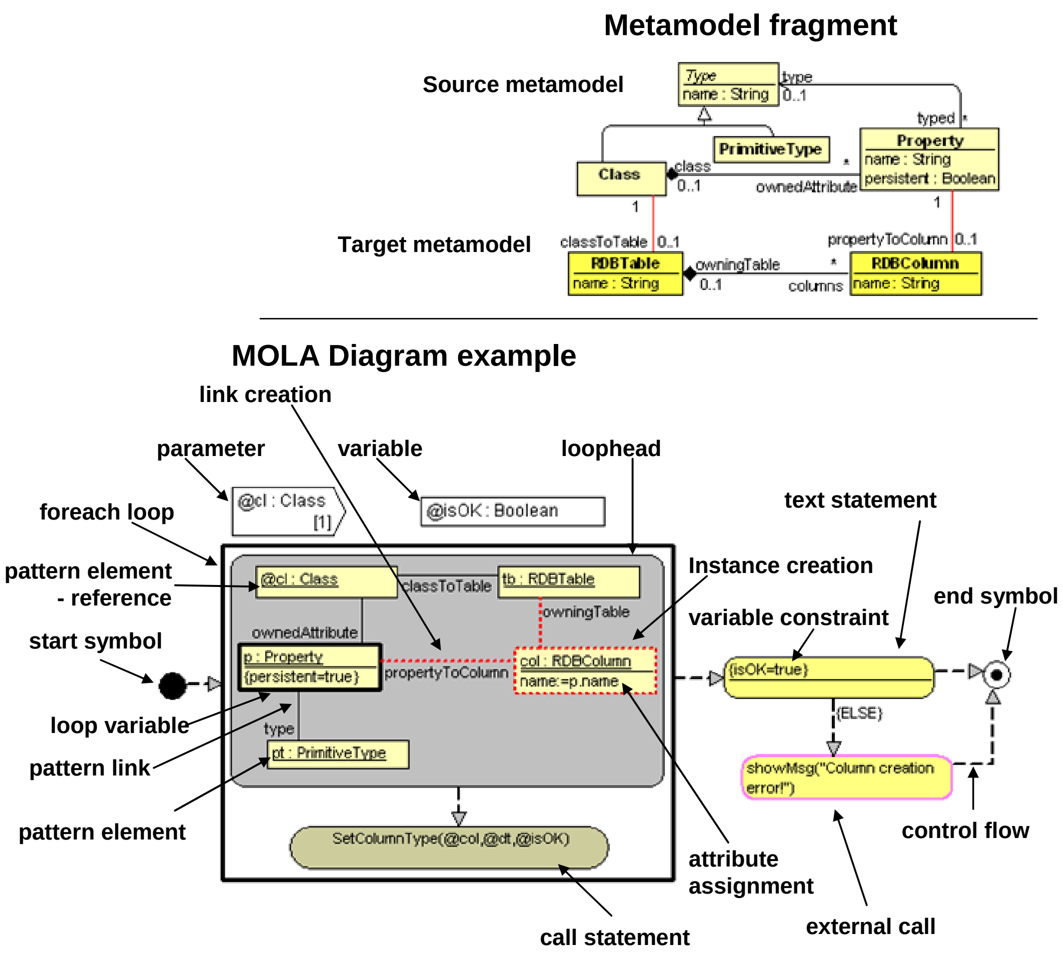

Figure 3.2 MOLA language elements

In order to obtain better understanding of the basic principles of MOLA language construction, it is essential to lay out the general semantics of the language execution:

- MOLA program receives the source model – a group of instances/links, which conforms to the source metamodel.
- The MOLA program creates the target model as a result of the transformation – a group of instances/links, which conforms to the target metamodel.
- Execution of the transformation is initiated with the starting symbol of the main program.
- Call statements call for the corresponding MOLA procedures (subprograms). It must be added that the calls can also be recursive.
- Statement implementation is successive – the loop is executed consecutively for each of the valid loop variable instances.

Further, provided is a more detailed description about the MOLA transformation fragment displayed in Figure 3.2. The image shows a metamodel fragment and a some MOLA procedure. The most interesting section of the procedure is the `foreach` loop. Construction of the given loop would be executed with all of those `Property` instances that have a link (the link conforms to the association with the role name `type` (see the upper right corner of the metamodel fragment in Figure 3.2)) with a `PrimitiveType` instance and a link with such `Class` instance `@cl`, which has been received as a parameter when calling this procedure. However this `Class` instance (`@cl`) must be connected to a certain `RDBTable` instance. In addition, it must be considered that the corresponding `Property` instance must satisfy a constraint that its attribute's `persistent` value is `true`. If all of the above mentioned constraints are satisfied, the action part of the rule is executed. In this case, an instance of `RDBColumn` class is created (displayed as a box with a red interrupted frame) and this newly-created instance is connected with the corresponding instances of `Property` and `RDBTable` classes (dotted red lines). Upon that another MOLA procedure is called – `SetColumnType` with the corresponding parameters. After execution of `SetColumnType`, the next valid `Property` class instance is sought.

As mentioned in Chapter 1.3, MOLA language is not the only model transformation language. Most of the in Chapter 1.3 mentioned languages (for instance, MOF QVT[35], MOLA, and many other) sooner or later became practically complete for traditional model transformation tasks (practically complete here means – tasks can adequately be implemented with programs in the given transformation language). The biggest difference between the languages is that of how easy it is to write and read information contained therein. It has been proven in practice, that MOLA language, in comparison with other languages, possesses high level of expression and readability. Reasonably created transformations in MOLA language are practically self-documenting and it is easy to understand what activities are performed. Work with students has proved that MOLA language can be quickly learned and that it is an easily perceivable transformation language. It is practically enough to have two academic lectures for the students to be able to write wholesome transformations in MOLA language. Amongst the rest of the transformation languages, the most similar with the MOLA language according to the language style are UMLX[39], AGG[44], and the graphical part of the MOF QVT[35] language. However each of these languages has some significant disadvantages. For instance, neither of them have a well-solved language control structure, besides there are relatively weak and inconvenient pattern support solutions. It must be added that a significant disadvantage of many model transformation languages is also an insufficient support of the corresponding tools, resulting in encumbered practical use of such languages. However for the MOLA language it is developed strong tool support (see Chapter 4).

3.2 Usage methodology of MOLA language and efficiency of the pattern matching

This research was performed simultaneously with the development of MOLA language itself. Already from the very beginning, when designing the MOLA language, it was important to consider also efficient implementation of the language, in order to avoid unforeseen problems during later implementation of the language and in application of the language in real MDSD tasks.

As a result of the research, the pattern matching efficiency in MOLA language was examined. Those properties of MOLA language syntax and semantics were separated, which could significantly affect the implementation efficiency of the MOLA transformations.

In order to evaluate the pattern matching efficiency, a hypothetical MOLA language virtual machine was developed. The virtual machine was developed as a group of simple functions, which would ensure simple activities with the repository. Below are the functions of the virtual machine and a detailed description of the meaning is given for some of them:

- `getPatternRoot()` – returns the root element (loop variable) of the pattern.
- `getPatternElement(int i)` – returns the "i" pattern element.
- `getNext(metaClass mcl)` – the function returns he next class `mcl` instance. In case if there are no more instances to returns then the `null` constant.
- `getNextByLink(association assoc, instance sourceInst, metaclass mcl)` – most frequently used pattern matching function. The function consecutively returns he class `mcl` instances, which can be reached from the fixed instance `sourceInst`, by "walking around" the links that conform to the association `assoc`. In case if there are no more instances to returns then the `null` constant. This function also has the initialization function `initializeGetNextByLink` with identical parameters.
- Other less important functions of the virtual machine:
  - `eval(instance inst, oclExpression expr)`,
  - `checkLink(instance sourceInst, instance targInst, association assoc)`,
  - `getNextFromSet(metaClass mcl, set instSet)`,
  - `getNextByLink(association assoc, instance sourceInst, metaclass mcl)`,
  - `getNextByLinkFromSet(association assoc, instance sourceInst, metaclass mcl, set instSet)`,
  - and several other initialization functions.

An algorithm was developed, which implemented pattern matching. Algorithm was developed, by using the facilities of the hypothetical virtual machine and by observing the language programming features. The "good style" was elaborated and offered on how programming in MOLA language should be performed in order to the pattern matching algorithm to operate efficiently. Some of the basic principles of the "good style" were: using metamodel associations with cardinality `0..1` or `1` in the corresponding direction (away from the loop variable) and using reference elements reasonably both in the included loops, as well as where the association cardinalities are not `0..1` or `1` in the corresponding direction. The estimation of the developed pattern matching algorithm was evaluated as O(n), where n is the size of the source model (number of class instances). The estimation was acquired by conditions that the offered style is used in the MOLA language and that the task in fact is possible to be solved algorithmically in linear time.

Certainly, in a general situation the pattern matching can require a search through a potentially large number of elements. It could occur if MOLA program is created unreasonably or some task must be objectively implemented, wherein the programming principles of the offered style cannot be observed. In such understanding, the MOLA language is similar to traditional generic programming languages, wherein a task could be algorithmically implemented with various methods.

The most significant contribution of the author in this research stage is the development of solutions, which are related to such MOLA language aspects as **development of basic constructions, pattern matching efficiency,** and **implementation of OO programming elements** into MOLA language (for instance, such classical OO programming ideas as inheritance and polymorphism are supported in the language). Extensive contribution by the author has been made in development of the hypothetical **virtual machine of the MOLA language**.

4 Implementation of MOLA language and application of the language

In this chapter, the results of the third research stage (year 2004 – year 2007) of the thesis are reviewed, which are summarized in three publications [11, 12, 13] and presented in three international conferences. The author has been actively participating in all of the activities related to this research stage. The MOLA Tool (developed as a result of the research performed during this stage) was demonstrated also in the tool demonstration session [20] of an important international conference ("*European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA)*").

In order to use the developed language conveniently in model transformation tasks, efficient implementation of language and corresponding tool support is required. The main goal of the research of this stage was to develop a set of tools, with which it would be possible to create MOLA language transformations and implement them. The main practical result of the research is the developed MOLA Tool. The MOLA Tool consists of several important components (see Figure 4.1):

- **MOLA TDE** – MOLA transformation definition environment.
- **MOLA TEE** – MOLA transformation execution environment.
- **MOLA Eclipse Plug-in.**

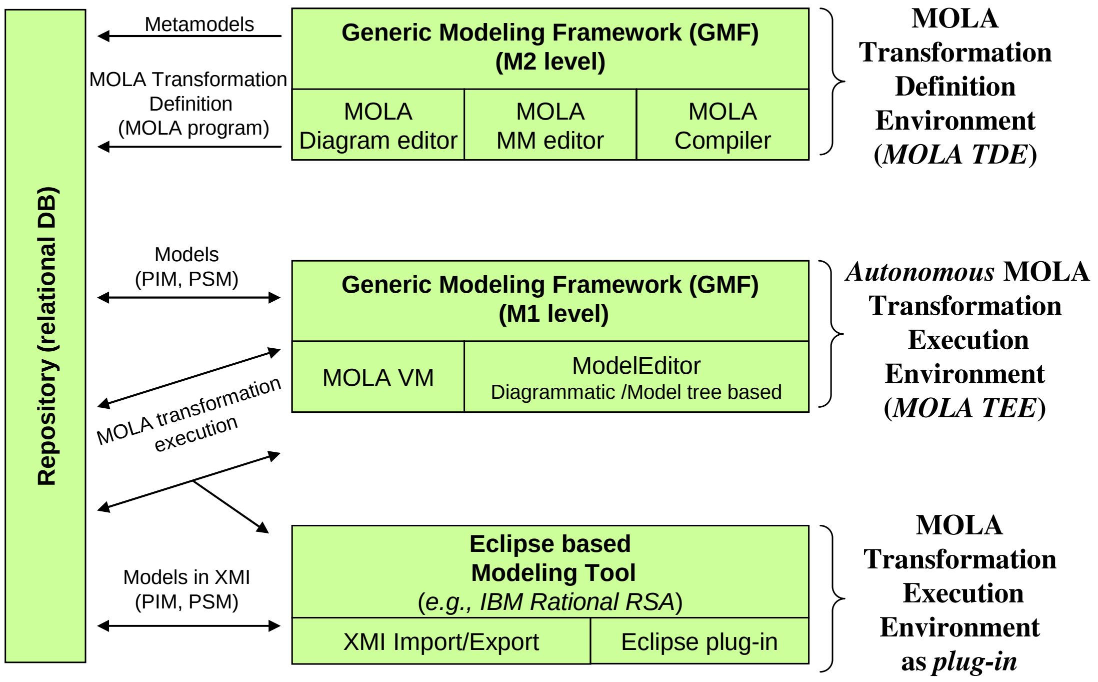
Repository (relational DB)
Metamodels
MOLA Transformation Definition (MOLA program)
Generic Modeling Framework (GMF) (M2 level)
MOLA Diagram editor
MOLA MM editor
MOLA Compiler
MOLA Transformation Definition Environment (MOLA TDE)
Models (PIM, PSM)
Generic Modeling Framework (GMF) (M1 level)
MOLA VM
ModelEditor Diagrammatic /Model tree based
Autonomous MOLA Transformation Execution Environment (MOLA TEE)
MOLA transformation execution
Models in XMI (PIM, PSM)
Eclipse based Modeling Tool (e.g., IBM Rational RSA)
XMI Import/Export
Eclipse plug-in
MOLA Transformation Execution Environment as plug-in

Figure 4.1 Implementation schema of the MOLA Tool

MOLA TDE ensures editor set for operations with metamodels and MOLA language programs and MOLA language compiler. As the base of MOLA TDE, the generic modeling tool was used. Various other additional services were ensured in the environment, for instance, import/export in xml format of separate procedures of MOLA language, which ensures simultaneous work with one MOLA transformation project.

The main component of **MOLA TEE** is MOLA language virtual machine. Its function is to ensure execution of transformations. Due to the fact that in a role of runtime repository is the relation data base, it is very natural and convenient to implement the MOLA language pattern matching as SQL language queries.

MOLA Eclipse plug-in.
In order to apply MOLA language in practice for real MDSD tasks, facilities allowing to use MOLA from the different development tools (wherein model-driven software development was performed) had to be implemented. Therefore intensive research was performed during this time period on how to use MOLA language as a plug-in in other modeling tools. As a result of the research, several components were developed, which ensure execution of transformations written in MOLA language in the modeling tools, which are based on the *Eclipse* platform [67], for instance, a tool of IBM company *Rational Software Architect* [68]. The most interesting of the developed components was the component, which ensures generic UML 2.0 XMI [69] import/export. It ensures data import/export between the MOLA language execution environment and such modeling tool environment, which supports the above mentioned XMI standard.

During this time period, research was performed also on how MOLA language could be applied in typical MDSD tasks. The corresponding transformations developed in MOLA language were built and demonstrated. For instance, in demonstration of MOLA Tool [20] it was demonstrated, how it is automatically feasible with assistance of MOLA language transformations to quickly and conveniently modify (transform) models developed in IBM RSA tool into each other. Specific demonstration was on how from the UML project class diagram it is feasible to automatically obtain a model usable for *Hibernate* platform [70]. It is a classical PIM and PSM model application in the model-driven software development.

The most significant contribution by the author in this research period is:
- The **MOLA language plug-in** designed and developed by the author to be used with IBM modeling tool *Rational Software Architect*. It ensures execution of transformations written in MOLA language within the given tool.
- The generic **UML 2.0 XMI import/export component** designed and developed by the author.
- The **import/export service** in MOLA Tool designed and developed by the author. It ensures import/export of MOLA procedure export, which is an important service in development of large transformation projects, when several persons participate in project development.

5 Conclusions

The thesis involves research on development of generic metamodel based modeling tools and on model transformation languages. **The goals set forth in the thesis have been completed in full.** Below shortly listed are the results of the research:

- **Generic modeling tool prototype** has been designed and implemented – it must be noted that the prototype is practically not being used anymore. The prototype and the editor definition language (EdDL) used in it provided confidence and ideas about the role and significance of model transformations in tool development.
- **Generic metamodel based modeling tool** has been designed and implemented – the tool has particularly well proven itself in practice and it is still actively being used in various experiments in IMCS research projects. By using this tool, the editor of the model transformation language MOLA was also developed.
- **Model transformation language MOLA** has been developed – a unique graphical model transformation language. It has already been tested and successfully applied in international scientific projects, as well as in student training at the master studies in Computer Science at the University of Latvia.
- **MOLA Tool** has been designed and implemented – it ensures all of the necessary services in order to apply transformations written in MOLA language to model transformation tasks.
- Model transformation **MOLA language home page** has been developed [71].

It must be emphasized that the **research** related to transformation languages **is being intensively carried on.** The most interesting project, in which the author of the thesis is actively involved, is development of the new versions of MOLA language and MOLA Tool:

- Currently at the IMCS, a new generation metamodel and model transformation based tool framework [22] is being developed, wherein the model transformations are built in the existing version of MOLA language. In this framework, also the next version of MOLA Tool is being built. Respectively, the next MOLA language version is implemented using the existing version of MOLA language, by applying so-called „*bootstrapping*" [72] method.
- The most interesting research topic related to the next MOLA language version is the research on how in MOLA language such OO programming concepts as templates (*templates* in C++ language or *generics* in Java language) could be introduced.

6 References

6.1 Author's publications in reviewed international conference materials

1. A. Kalnins, K. Podnieks, A. Zarins, E. Celms , J. Barzdins. ***Editor definition language and its implementation.*** - Lecture Notes in Computer Science, Springer, v. 2244, 2001, pp.530-537.
2. A. Kalnins, J. Barzdins, E. Celms et al. ***The first step towards generic modeling tool.*** - Proceedings of the Fifth International Baltic Conference on Databases and Information Systems, Tallin, 2002, v.2, pp.167-180.
3. L. Lace, E. Celms, A. Kalnins. ***Diagram definition facilities in a generic modeling tool.*** - Proceedings of International Conference on Modelling and Simulation of Business systems, Vilnius, 2003, pp.220-224.
4. E. Celms, A. Kalnins, L. Lace. ***Diagram definition facilities based on metamodel mappings.*** - Proceedings of the 3rd OOPSLA (Workshop on Domain-Specific Modeling) , University of Jyvaskyla, 2003, pp.23-32.
5. Celms E. ***Generic Data Representation by Table in Metamodel Based Modelling Tool.*** Scientific proceedings of University of Latvia, Computer Science and Information Technologies, Automation of Information Processing, vol. 669, Riga, Latvia, April 2004, pp. 53-61.
6. A. Kalnins, J. Barzdins, E. Celms. ***Model Transformation Language MOLA.*** - Lecture Notes in Computer Science, Springer, v. 3599, 2005. Model Driven Architecture: European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, June 26-27, 2003 and Linkoping, Sweden, June 10-11, 2004. Revised Selected Papers, pp. 62-76.
7. A. Kalnins, J. Barzdins, E. Celms. ***Basics of Model Transformation Language MOLA.*** - ECOOP 2004 (Workshop on Model Transformation and execution in the context of MDA) , Oslo, Norway, June 14-18, 2004, p. 6.
8. A. Kalnins, J. Barzdins, E. Celms. ***Model Transformation Language MOLA: Extended Patterns.*** - Databases and Information Systems, Selected papers from the 6th International Baltic Conference DB&IS'2004, IOS Press, FAIA (Frontiers in Artificial Intelligence and Applications), vol. 118, 2005, pp. 169-184.
9. A. Kalnins, J. Barzdins, E. Celms. ***MOLA Language: Methodology Sketch.*** - Proceedings of EWMDA-2, Canterbury, England, September 7-8, 2004, pp.194-203.
10. A. Kalnins, J. Barzdins, E. Celms. ***Efficiency Problems in MOLA Implementation.*** 19th International Conference, OOPSLA'2004 (Workshop "Best Practices for Model-Driven Software Development") , Vancouver, Canada, October 2004, p. 14.
11. A. Kalnins, E. Celms, A. Sostaks. ***Tool support for MOLA.*** Fourth International Conference on Generative Programming and Component Engineering (GPCE'05). Proceedings of the Workshop on Graph and Model Transformation (GraMoT) , Tallinn, Estonia, September 2005, pp. 162-173.
12. A. Kalnins, E. Celms, A. Sostaks. ***Model Transformation Approach Based on MOLA.*** ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML '2005). (MoDELS/UML'05 Workshop: Model Transformations in Practice (MTIP)) , Montego Bay, Jamaica, October 2-7, 2005, p. 25.
13. A. Kalnins, E. Celms, A. Sostaks. ***Simple and Efficient Implementation of Pattern Matching in MOLA Tool***. Proceedings of the 7th International Baltic Conference on Databases and Information Systems (Baltic DB&IS'2006). , Vilnius, Lithuania, July 3-6, 2006, pp. 159-167.

6.2 Other publications of the author in reviewed international conference materials (directly non-related to the topic of the thesis)

14. A. Kalnins, J. Barzdins, E. Celms. ***UML Business Modeling Profile***. - Proceedings of ISD'2004, Vilnius, Lithuania, September 9-11, 2004, pp.182-194.
15. J.Viksna, E.Celms, M.Opmanis, K.Podnieks, P.Rucevskis, A.Zarins, A.Barrett, S.Guha Neogi, M.Krestyaninova, M.McCarthy, A.Brazma, U.Sarkans. ***PASSIM - an open source software system for managing information in biomedical studies***. BMC Bioinformatics, vol. 8:52, 2007.

6.3 Other sources used in thesis

16. ***Domain-specific language, DSL.*** Internet – http://en.wikipedia.org/wiki/ Domain-specific_programming_language
17. Stahl Thomas, Volter Markus. ***Model-Driven Software Development***. John Wiley & Sons, Ltd., 2006.
18. Jorn Bettin. ***Model-Driven Software Development: An emerging paradigm for industrialized software asset development.*** 2004. Internet – http://www.softmetaware.com/mdsd-and-isad.pdf.
19. ***Object Management Group (OMG).*** Internet – http://www.omg.org
20. Tool session of the "***European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA)***", November 7-10th, 2005, Nuernberg, Germany.
21. ***ReDSeeDS (Requirements-Driven Software Development System).*** Internet – http://www.redseeds.eu
22. A. Kalnins, J. Barzdins. ***MDA Support by Transformation Based Tool.*** Proceedings of First International Workshop MoRSe 2006, Warsaw, Poland, October 2006, pp. 21-24.
23. Object Management Group. ***Meta Object Facility (MOF).*** Internet – http://www.omg.org/mof
24. Object Management Group. ***Meta Object Facility Core Specification***, version 2.0, 2006. Internet – http://www.omg.org/docs/formal/06-01-01.pdf
25. Booch G., Jackobson I., Rumbaugh J. ***The Unified Modeling Language. Reference Manual***, Addison-Wesley, 1999..
26. Object Management Group. ***Unified Modeling Language: Superstructure.*** Version 2.0 (Final Adopted Specification), 2005. Internet – http://www.omg.org/docs/formal/05-07-04.pdf
27. Object Management Group. ***Unified Modeling Language: Infrastructure***. Version 2.0 (Final Adopted Specification), 2005. Internet – http://www.omg.org/docs/formal/05-07-05.pdf
28. Object Management Group. ***Unified Modeling Language: Superstructure.*** Version 2.1.1, 2007. Internet – http://www.omg.org/docs/formal/07-02-05.pdf
29. Object Management Group. ***Unified Modeling Language: Infrastructure.*** Version 2.1.1, 2007. Internet – http://www.omg.org/docs/formal/07-02-06.pdf
30. Object Management Group. ***Business Process Modeling Notation (BPMN)***, Final Adopted Specification, OMG, 2006. Internet – http://www.omg.org/docs/dtch/06-02-01.pdf
31. Object Management Group. ***Model Driven Architecture (MDA).*** Internet – http://www.omg.org/mda
32. Kleppe A., Warmer J., Bast W. ***MDA Explained. The model driven architecture: practice and promise.*** Addison-Wesley, 2003.
33. Object Management Group. ***MDA Guide Version 1.0.1.*** Internet– http://www.omg.org/docs/omg/03-06-01.pdf
34. Object Management Group. ***Request for Proposal: MOF 2.0 Query / Views / Transformations***, 2002. Internet – http://www.omg.org/docs/ad/02-04-10.pdf
35. Object Management Group. ***QVT – Query/View/Transformation Specification.*** (Final Adopted Specification), 2005. Internet – www.omg.org/docs/ptc/05-11-01.pdf
36. IMCS (LUMII). ***The Base Transformation Language L0***, 2007. Internet – http://l0.mii.lu.lv/L0_plus_CurrVers_2_4.pdf
37. Agrawal A., Karsai G, Shi F. ***Graph Transformations on Domain-Specific Models***. Technical report, Institute for Software Integrated Systems, Vanderbilt University, ISIS-03-403, 2003
38. Vanderbilt University. **GReAT**. Internet – http://repo.isis.vanderbilt.edu/tools/get_tool?GReAT
39. Willink E.D. ***A concrete UML-based graphical transformation syntax - The UML to RDBMS example in UMLX***. Workshop on Metamodelling for MDA, University of York, England, 24-25 November, 2003.
40. Bezivin J., Dupe G., Jouault F., et al. ***First experiments with the ATL model transformation language: Transforming XSLT into XQuery***. 2nd OOPSLA Workshop on Generative Techniques in Context of MDA, Anaheim, California, 2003.
41. Institut national de recherche en informatique et en automatique (INRIA). ***ATL : Atlas Transformation Language***. Internet – http://www.sciences.univ-nantes.fr/lina/atl
42. ***Simple Transformation Rule Language (TRL).*** Internet – http://modfact.lip6.fr/qvtP.html
43. DSTC. ***Tefkat: The EMF Transformation Engine.***Online documentation. Internet – http://www.dstc.edu.au/tefkat
44. TU Berlin, TFS. ***The Attributed Graph Grammar System (AGG).*** Internet – http://tfs.cs.tu-berlin.de/agg

45. IBM. ***Model Transformation Framework (MTF)***.
Internet – http://www.alphaworks.ibm.com/tech/mtf
46. McGill University, Modelling, Simulation and Design Lab. ***ATOM***.
Internet – http://atom3.cs.mcgill.ca
47. Budapest University of Technology and Economics, Department of Automation and Applied Informatics. ***Visual Modeling and Transformation System (VMTS).***
Internet – http://avalon.aut.bme.hu/~tihamer/research/vmts
48. Institut fur Informatik der Technischen Universitat Munchen, Peter Braun, Frank Marschall. ***The Bidirectional Object Oriented Transformation Language (BOTL).***
Internet – http://wwwbib.informatik.tu-muenchen.de/infberichte/2003/TUM-I0307.pdf
49. Octavian Patrascoiu. ***Yet Another Transformation Language (YATL).*** Proceedings of the 1st European MDA Workshop, MDA-IA, pages 83-90. University of Twente, the Nederlands, January 2004. Internet – http://www.cs.kent.ac.uk/pubs/2004/1829
50. Universitat Paderborn, Institut fur Informatik. ***Fujaba.***
Internet – http://wwwcs.uni-paderborn.de/cs/fujaba/documents/user/manuals/FujabaDoc.pdf
51. Budapest University of Technology and Economics, GMT subproject. ***Visual Automated Model Transformations (VIATRA2).***
Internet – http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/index.html
52. Jesus Sanchez Cuadrado, Jesus Garcia Molina, Marcos Menarguez Tortosa. ***RubyTL: A Practical, Extensible Transformation Language.*** Model Driven Architecture - Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006. LNCS 4066, Springer 2006.
53. Kleppe, A. ***Towards general purpose high level software languages***. Model Driven Architecture - Foundations and Applications (A. Hartman and D. Kreische, eds.), vol. 3748 of LNCS, Springer-Verlag, Nov. 2005.
54. Laurence Tratt. ***The MT model transformation language***. Technical report TR-05-02, Department of Computer Science, King's College London, May 2005.
55. Joint revised submission by Compuware Corporation, SUN Microsystems, … (OMG Document ad/2003-08-07). ***XMOF Queries, Views and Transformations on Models using MOF, OCL and Patterns***, 2003. Internet – http://www.omg.org/docs/ad/03-08-07.pdf
56. Compuware. ***OptimalJ (Model-driven development for java).***
Internet – http://www.compuware.com/products/optimalj/
57. EU ESPRIT project, ***Application Development for the Distributed Enterprise (ADDE)***.
Internet – http://www.fast.de/ADDE
58. Smolander, K., Martiin, P., Lyytinen, K., Tahvanainen, V-P. ***Metaedit – a flexible graphical environment for methodology modelling.*** Springer-Verlag, 1991.
59. Ebert, J., Suttenbach, R., Uhe, I. ***Meta-CASE in Practice: a Case for KOGGE***. Proceedings of the 9th International Conference, CAiSE'97, Barcelona, Catalonia, Spain , 1997, pp.203-216.
60. Lincoln Software Ltd. ***IPSYS Toolbuilder Manual***, Version. 2.1, 1996.
61. Honeywell Inc. ***The Domain Modeling Environment (DOME),*** Users Guide.
Internet – http://www.htc.honeywell.com/dome
62. ETH Zurich, TIK. ***The Moses project.*** Internet – http://www.tik.ee.ethz.ch/~moses
63. Object Management Group. ***OCL 2.0 Specification***, Version 2.0, 2006.
Internet – http://www.omg.org/docs/formal/06-05-01.pdf
64. MetaCase. ***MetaEdit+ resources.*** Internet – http://www.metacase.com/papers/index.html
65. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason IV, C., Nordstrom, G., Sprinkle, J., Volgyesi, P. ***The Generic Modeling Environment (GME)***. Workshop on Intelligent Signal Processing, Budapest, Hungary, May 17, 2001.
66. de Lara, J,, Vangheluwe, H., Alfonseca, M. ***Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in AToM3***. Software and Systems Modeling (SoSyM), Volume 3, Number 3, August 2004, pp. 194-209.
67. ***Eclipse***. Internet – http://www.eclipse.org
68. IBM. ***Rational Software Architect (RSA).***
Internet – http://www-306.ibm.com/software/awdtools/architect/swarchitect/index.html
69. Object Management Group. ***MOF 2.0/XMI Mapping Specification***, v2.1, 2005.
Internet – http://www.omg.org/docs/formal/05-09-01.pdf
70. ***Hibernate***. Internet – http://www.hibernate.org
71. IMCS (LUMII). ***MOLA home page***. Internet – http://mola.mii.lu.lv
72. ***Bootstrapping.*** Internet – http://en.wikipedia.org/wiki/Bootstrapping_%28computing%29

7 Annex

7.1 Author's reports on thesis results in international scientific conferences or seminars

1. L. Lace, E. Celms, A. Kalnins. ***Diagram definition facilities in a generic modeling tool.*** - International Conference on Modelling and Simulation of Business systems, Vilnius, 2003, pp.220-224.
2. A. Kalnins, J. Barzdins, E. Celms. ***Model Transformation Language MOLA.*** - Model Driven Architecture: European MDA Workshop: Foundations and Applications, MDAFA 2004, Linkoping, Sweden, June 10-11, 2004.
3. A. Kalnins, J. Barzdins, E. Celms. ***Basics of Model Transformation Language MOLA.*** - ECOOP 2004 (Workshop on Model Transformation and execution in the context of MDA) , Oslo, Norway, June 14-18, 2004.
4. A. Kalnins, J. Barzdins, E. Celms. ***Model Transformation Language MOLA: Extended Patterns.*** - Databases and Information Systems, 6th International Baltic Conference DB&IS'2004, Riga, Latvia, 2004.
5. A. Kalnins, J. Barzdins, E. Celms. ***Efficiency Problems in MOLA Implementation.*** 19th International Conference, OOPSLA'2004 (Workshop "Best Practices for Model-Driven Software Development") , Vancouver, Canada, October 2004.
6. A. Kalnins, E. Celms, A. Sostaks. ***Tool support for MOLA.*** Fourth International Conference on Generative Programming and Component Engineering (GPCE'05). Workshop on Graph and Model Transformation (GraMoT) , Tallinn, Estonia, September 2005.
7. A. Kalnins, E. Celms, A. Sostaks. ***Model Transformation Approach Based on MOLA.*** ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML '2005). (MoDELS/UML'05 Workshop: Model Transformations in Practice (MTIP)) , Montego Bay, Jamaica, October 2-7, 2005.

7.2 Publications included in thesis and personal contribution by the author of thesis

Authors	Publication	Contribution by the thesis author from the total research volume	Description of contribution by the thesis author
A. Kalniņš, K. Podnieks, A. Zariņš, E. Celms, J. Bārzdiņš.	***Editor definition language and its implementation.*** - Lecture Notes in Computer Science, Springer, v. 2244, 2001, pp.530-537.	30%	• Participation in idea elaboration • Development of basic constructions of EdDL language • Designing and development of annotation compiler and editor engine
A. Kalniņš, J. Bārzdiņš, E. Celms,	***The first step towards generic modeling tool.*** - Proceedings of the Fifth	30%	• Participation in idea elaboration • Development of basic

L. Lāce, M. Opmanis, K. Podnieks, A. Zariņš.	International Baltic Conference on Databases and Information Systems, Tallin, 2002, v.2, pp.167-180.		architectural principles of the generic metamodel based modeling tool • Designing and implementation of the generic modeling tool dispatcher • Designing and implementation of the model tree engine
L. Lāce, E. Celms, A. Kalniņš.	***Diagram definition facilities in a generic modeling tool.*** - Proceedings of International Conference on Modelling and Simulation of Business systems, Vilnius, 2003, pp.220-224.	40%	• Participation in idea elaboration • Development of the basic principles of diagram annotation definition (mapping) • Diagram annotation support in generic metamodel based modeling tool
E. Celms, A. Kalniņš, L. Lāce.	***Diagram definition facilities based on metamodel mappings.*** - Proceedings of the 3rd OOPSLA (Workshop on Domain-Specific Modeling) , University of Jyvaskyla, 2003, pp.23-32.	50%	• Participation in idea elaboration • Elaboration of basic principles of mapping types • Development of mapping type library
E. Celms	***Generic Data Representation by Table in Metamodel Based Modelling Tool.*** Scientific proceedings of University of Latvia, Computer Science and Information Technologies, Automation of Information Processing, vol. 669, Riga, Latvia, April 2004, pp. 53-61.	100%	• Participation in idea elaboration • Designing and implementation of the generic table editor engine • Designing and implementation of the generic property editor engine
A. Kalniņš, J. Bārzdiņš, E. Celms.	***Model Transformation Language MOLA.*** - Lecture Notes in Computer Science, Springer, v. 3599, 2005. Model Driven Architecture: European MDA Workshops:	40%	• Participation in idea elaboration • Development of the basic constructions of the model transformation language MOLA

	Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, June 26-27, 2003 and Linkoping, Sweden, June 10-11, 2004. Revised Selected Papers, pp. 62-76.		
A. Kalniņš, J. Bārzdiņš, E. Celms.	***Basics of Model Transformation Language MOLA.*** - ECOOP 2004 (Workshop on Model Transformation and execution in the context of MDA) , Oslo, Norway, June 14-18, 2004, p. 6.	60%	• Participation in idea elaboration • Introduction of OO programming elements into the MOLA language
A. Kalniņš, J. Bārzdiņš, E. Celms.	***Model Transformation Language MOLA: Extended Patterns.*** - Databases and Information Systems, Selected papers from the 6th International Baltic Conference DB&IS'2004, IOS Press, FAIA (Frontiers in Artificial Intelligence and Applications), vol. 118, 2005, pp. 169-184.	60%	• Participation in idea elaboration • Development of the basic principles for extended patterns
A. Kalniņš, J. Bārzdiņš, E. Celms.	***MOLA Language: Methodology Sketch.*** - Proceedings of EWMDA-2, Canterbury, England, September 7-8, 2004, pp.194-203.	60%	• Participation in idea elaboration • Elaboration of methodology for transformation programming for MOLA language
A. Kalniņš, J. Bārzdiņš, E. Celms.	***Efficiency Problems in MOLA Implementation.*** 19th International Conference, OOPSLA'2004 (Workshop "Best Practices for Model-Driven Software Development") , Vancouver, Canada,	80%	• Participation in idea elaboration • Development of the hypothetical virtual machine of MOLA language • Evaluations of pattern matching efficiency for MOLA language • Elaboration of basic

	October 2004, p. 14.		principles of language application, which ensure convenient implementation of the language
A. Kalniņš, E. Celms, A. Šostaks.	***Tool support for MOLA.*** Fourth International Conference on Generative Programming and Component Engineering (GPCE'05). Proceedings of the Workshop on Graph and Model Transformation (GraMoT) , Tallinn, Estonia, September 2005, pp. 162-173.	50%	• Participation in idea elaboration • Elaboration of basic principles of MOLA Tool architecture • Development and implementation of basic principles of generic UML 2.0 XMI import/export • Designing and development of the MOLA Tool Eclipse plug-in
A. Kalniņš, E. Celms, A. Šostaks.	***Model Transformation Approach Based on MOLA.*** ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML '2005). (MoDELS/UML'05 Workshop: Model Transformations in Practice (MTIP)) , Montego Bay, Jamaica, October 2-7, 2005, p. 25.	40%	• Participation in idea elaboration • Development of basic principles of recursion application for MOLA language
A. Kalniņš, E. Celms, A. Šostaks.	***Simple and Efficient Implementation of Pattern Matching in MOLA Tool***. Proceedings of the 7th International Baltic Conference on Databases and Information Systems (Baltic DB&IS'2006). , Vilnius, Lithuania, July 3-6, 2006, pp. 159-167.	20%	• Participation in idea elaboration • Designing and elaboration of import/export services in MOLA Tool