MODEL OF PROBLEM DOMAIN "MODEL-DRIVEN ARCHITECTURE FORMAL METHODS AND APPROACHES"

Elena Chebanyuk, Krassimir Markov

***Abstract**: Model-Driven Architecture MDA can be regarded as a part of Model-Driven Development (MDD), where the modeling operation and transformation languages are standardized by Object Management Group (OMG).*

This article is devoted to designing of domain model that illustrates interconnections between mathematical foundations used to design formal approaches of software models transformation. Then, review of related researches advantages, according to MDA promising, is represented. A summary of requirements to model to model transformation approach according to MDA promising is outlined. The scope of mathematical foundations for designing model to model transformation techniques is defined. During domain model designing, a controlled vocabulary containing description of basic mathematical foundations that are used for model to model transformations is composed. Proposed domain model will serve as a template for choosing proper mathematical approaches and means for model to model transformation performing with given level of accuracy. It should be considered when new transformation methods are designed.

Means of increasing productiveness for transformation methods designing, involving proposed domain model, are formulated in conclusions.

***Keywords**: software model, model transformation, Model-Driven Architecture (MDA), theory of categories, first-order logic, metalogic, formal language, transformational grammar.*

***ACM Keywords**:*

***- Classification of 2012 year**: Software and its engineering, Software system structure, Software system model, Model driven system engineering.*

***- Classification of 1998 year:** D2 software engineering, D 2.0 Tools*

Introduction

Software models, often represented as Unified Modeling Language (UML) diagrams, are key artifacts in Model-Driven Development (MDD) approach. The key idea behind MDA is to separate the specification of the system functionality from its implementation on specific platforms increasing the degree of automation and achieving interoperability with multiple platforms. Thus, model processing, is a key and the most common activity for different software development lifecycles processes.

Software model transformation is a key activity of Model-Driven Architecture (MDA). Target of model transformation activity is to analyze together information from different software development lifecycle processes. In order to archive this goal the next model transformation activities are implemented: Model to Model (M2M), Model to Text (M2T), and Text to Model (T2M) transformations. Text in M2T and T2M transformations means analytical representation of software model or skeleton of program. In the first case, the role of text is to be subsidiary artifact that saves information about model. In the second case the role of text to be target of transformation [Truyen, 2006].

Other transformation' aspects are horizontal and vertical software model transformations. A horizontal transformation is a transformation where the source and target models reside at the same abstraction level. Typical examples are refactoring (an endogenous transformation) and language migration (an exogenous transformation). A vertical transformation is a transformation where the source and target models reside at different abstraction levels. A typical example is refinement, where a specification is gradually refined into a full-fledged implementation, by means of successive refinement steps that add more concrete information. Also code generation operation are considered as vertical software model transformation [Czarnecki and Helsen, 2006].

The key idea behind MDA is to separate the specification of the system functionality from its implementation on specific platforms increasing the degree of automation and achieving interoperability with multiple platforms.

The promise of Model Driven Architecture is to facilitate the creation of machine-readable models with a goal of long-term flexibility in terms of:

• Technology obsolescence: new implementation infrastructure can be more easily integrated and supported by existing designs;

• Portability: existing functionality can be more rapidly migrated into new environments and platforms as dictated by the business needs;

• Productivity and time-to-market: by automating many tedious development tasks; architects and developers are freed up to focus their attention on the core logic of the system;

• Quality: the formal separation of concerns implied by this approach plus the consistency and reliability of the artifacts produced all contribute to the enhanced quality of the overall system;

• Integration: the production of integration bridges with legacy and/or external systems is greatly facilitated;

• Maintenance: the availability of the design in a machine-readable form gives analysts, developers and testers direct access to the specification of the system, simplifying their maintenance chores;

• Testing and simulation: models can be directly validated against requirements as well as tested against various infrastructures. They can also be used to simulate the behavior of the system under design;

• Return on investment: businesses are able to extract greater value out of their investments in tools [Mens and Van Gorp, 2006].

To realize software model transformation tasks many successful researches are done. These researches are aimed to solve actual software engineering tasks of implementing software model transformation.

Consider papers that make a strong contribution in several MDE promising by means of using tools or means operating with some mathematical solutions.

A review of mathematical foundations for providing realization of model transformation techniques is outlined in [Rabbi et al, 2016].

The MDE promising are achieved solving actual software engineering tasks by means of implementing software model transformation [Favre and Duarte, 2016].

Paper [Greiner et al, 2016] represents a case study dealing with incremental round-trip engineering of UML class models and Java source code.

The MoDisco [Bruneliere et al., 2010] framework is used to parse the Java source code into a model representation. QVT-R is used to formalize a bidirectional model-to-model transformation between the UML model and the Java model. This aspect is an interesting feature of QVT-R, as a transformation

developer may provide a single relational specification which may be executed in both directions, rather than writing two unidirectional transformations separately. Moreover, QVT-R is chosen because of its declarative nature where the developer is supposed to focus on relations and dependencies between the metamodels rather than on single execution steps [Greiner et al, 2016].

Described approach tries to prevent information loss during round-trip engineering by using a so called trace model which is used to synchronize the platform independent and the platform specific models. Furthermore, the source code is updated using a fine grained bidirectional incremental merge. Also, information loss is prevented by using Javadoc tags as annotations. In case model and code are changed simultaneously and the changes are contradicting, one transformation direction has to be chosen, which causes that some changes might get lost [Greiner et al, 2016].

Declarative notation of QVT-R language allows involving predicate logic to express transformation rules.

A review of metamodeling tools is represented in paper [Favre and Duarte, 2016] and several metamodeling frameworks are described. Two points of this review are interesting for us from point of view of providing descriptions of mathematical foundations that consist the basic of metamodeling tools and frameworks:

1. Varró and Pataricza presented a visual and formally precise metamodeling framework that is capable of uniformly handling arbitrary models from engineering and mathematical domains. They propose a multilevel metamodeling technique with precise static and dynamic semantics (based on a refinement calculus and graph transformation) where the structure and operational semantics of mathematical models can be defined in a UML notation [Varro and Pataricza, 2003]. In order to verify metamodel some expressions are composed. First-order logics may be used as mathematical foundations for it. Different semantics (for example static and dynamic semantics) are used to verify the content of metamodels.
2. Also, logics are used to design modeling languages. For example modeling language “Allow” is based on first-order relational logic. It can be a base for creating frameworks for metamodels and model processing by means of analyzing their analytical representation. For example, Boronat and Messeguer describe an algebraic, reflexive and executable framework for metamodeling in MDD [Boronat and Meseguer, 2010]. The framework provides a formal semantic of the notions of metamodel, model and conformance relation between a model and a metamodel.

Authors of [Favre and Duarte, 2016] provided a metamodeling framework based on MOF and the algebraic formalism that focus on automatic proofs and tests. The central components of the proposed approach are the definition of the algebraic language NEREUS and the development of tools for formal metamodeling: the NEREUS analyzer and the NEREUS-to-CASL translator.

Let's consider main possibilities of NEREUS syntax: classes may declare types, attributes, operations and axioms which are formulas of first-order logic. They are structured by different kinds of relations: importing, inheritance, sub-typing, and associations [Favre and Duarte, 2016].

Descriptive notation of NEREUS allows adding new construction to describe variety of connections between objects and operations. First-order logic and algebraic formalisms used to describe relationships between NEREUS components and to compose new expressions for interconnecting NEREUS with other metamodeling tools.

Considering that there exist many formal algebraic languages, NEREUS allows connecting any number of source languages such as different Domain Specific Languages (DSLs) and target languages (different formal languages).

The contribution of paper [Rabbi et al, 2016] is a new web-based metamodeling and model transformation tool called WebDPF based on the Diagram Predicate Framework (DPF). WebDPF has been developed using HTML5 and JavaScript. Any HTML5 and JavaScript enabled web browser can be used for metamodeling with WebDPF. Algorithms, related to model transformation and analysis in WebDPF, are written in JavaScript and therefore executes on the client machine. WebDPF supports multilevel diagrammatic metamodeling and specification of model constraints, and it supports diagrammatic development and analysis of model transformation systems. In WebDPF, one can graphically specify constraints and model transformation rules. Transformation rules have been introduced in WebDPF for two purposes:

i) automatic rewriting of partial (incomplete) models so that they can be made to conform to the underlying metamodel;

ii) modelling the behavior of systems.

The support for model transformation systems in WebDPF can be exploited to (i) support auto-completion of partial models thereby enhancing modeling efficiency, and (ii) provide execution semantics for workflow models.

The WebDPF metamodeling environment supports multilevel metamodeling [Rutle, 2010]. In WebDPF, one can graphically specify constraints and model transformation rules, based on graph transformation rules. The rules are linked to predicates and the standard double-pushout (DPO) approach is used. Attached transformation rules for a predicate p, is given by a set of coupled transformation rules ρ(p) where the meta-models remain unchanged. A rule r ∈ ρ(p) of a predicate p has a matching pattern (L), a gluing condition (K), a replacement pattern (R), and an optional negative application condition. The matching pattern and replacement pattern are also known as left-hand side and right-hand side of a rule, respectively. WebDPF performs termination analysis based on principles adapted from layered graph grammars [Ehrig et al., 2006]. In a layered typed graph grammar, transformation rules are distributed across different layers. The transformation rules of a layer are applied as long as possible before going to the next layer. WebDPF generalizes the layer conditions from [Ehrig et al., 2006] allowing deleting and non deleting rules to reside in the same layer as long as the rules are loop-free [Favre and Duarte, 2016].

Authors of [Zaraket and Noureddine, 2014] proposed a method of checking software with first order logic specification using And-Inverter-Graph (AIG) solvers. AIG can be viewed as a restricted C++ program, specifically a concurrent program in which all variables are either integers, whose range is statically bounded, or Boolean-valued, and dynamic allocation is forbidden. Using first order logics it permits developing and modifying semantics for model checking operations. Using conjunctive normal forms it support defining templates to estimate software models quality.

The results of literature review, matching strong contribution of considered transformation methods and techniques to MDA promising are represented in Table 1.

The first column contains features of the MDA promising. The next four columns contain analysis of transformation methods and techniques outlined in the considered papers:

- Formal MOF Metamodeling and Tool Support [Favre and Duarte, 2016];
- Model Checking Software with First Order Logic Specifications using AIG Solvers [Zaraket and Noureddine, 2014];
- Bidirectional transformations approach with QVT-R [Greiner et al, 2016];
- Web-based metamodeling and model transformation tool called WebDPF [Rabbi et al, 2016].

The last column contains our proposal what features we expect from the full automated method model to model transformations which has to be developed.

Table 1. Matching of considered transformation methods and techniques to MDA promising

Methods and tools / MDA promising	Formal MOF Metamodeling and Tool Support	Model Checking Software with First Order Logic Specifications using AIG Solvers	Bidirectional transformations approach with QVT-R	Web-based metamodeling and model transformation tool called WebDPF	Full automated method model to model transformations (to be developed)
Summary	Approach integrates MOF meta-language with formal specification languages based on the algebraic formalism. More concretely, NEREUS, as a formal metamodeling language, supports processes for reasoning about MOF-like metamodels such as ECORE metamodels.	Synthesis and verification frameworks to validate programs. And-Inverter-Graph (AIG) is a Boolean formula with memory elements, logically complete negated conjunction gates, and a hierarchical structure.	Method uses extensible model for model to model transformations by means of adding relations and elements	WebDPF is based on the Diagram Predicate Framework (DPF). WebDPF supports multilevel diagrammatic metamodeling and specification of model constraints, and it supports diagrammatic development and analysis of model transformation systems.	Provide a both top-down and bottom-up software model transformation from any type of UML diagram to another
Technology obsolescence	Framework is extensible because all of its components are described declaratively.			WebDPF specification allows adding new programming languages	Using existing and new formal approaches and tools

Methods and tools / MDA promising	Formal MOF Metamodeling and Tool Support	Model Checking Software with First Order Logic Specifications using AIG Solvers	Bidirectional transformations approach with QVT-R	Web-based metamodeling and model transformation tool called WebDPF	Full automated method model to model transformations (to be developed)
Portability				Such open standards as javaScript and HTML5 allow using WebDPF for different platforms	To be compatible with open standards, open data specifications, and open model processing environments or tools
Productivity and time-to-market		Effective optimization of model checking operations lets to shrink time for processing large amount of software models	Using tools lets to proceed many software models and modules of code raising effectiveness of software development processes		Providing bidirectional transformations for different types of software models. (productivity). (*) Reusing information from different software models (time to market). (*)

Methods and tools / MDA promising	Formal MOF Metamodeling and Tool Support	Model Checking Software with First Order Logic Specifications using AIG Solvers	Bidirectional transformations approach with QVT-R	Web-based metamodeling and model transformation tool called WebDPF	Full automated method model to model transformations (to be developed)
Quality	Comparability with constraint language allows to design high quality metamodels	Setting model quality characteristics and involving different verification techniques to modeling process allows estimating resulting models		By understanding both SysML semantic and programming languages constructions	Using modularity, restriction languages, and semantic tools
Integration	Framework uses formal language and it is compatible with other model processing tools and plugins, namely CASL, HETS and AST			Web environment allows integrating proposed tool with other metamodeling frameworks	Considering operations on meta-level allows designing integration tools for different platforms
Maintenance:		Quality models are sources for effective software lifecycle development processes	Using open standards simplifies the model maintenance procedure		Maintenance procedure based on matching transformation techniques with visualization ones

Methods and tools / MDA promising	Formal MOF Metamodeling and Tool Support	Model Checking Software with First Order Logic Specifications using AIG Solvers	Bidirectional transformations approach with QVT-R	Web-based metamodeling and model transformation tool called WebDPF	Full automated method model to model transformations (to be developed)
Testing and simulation			Using tools following open standards notations allows verifying all intermediate results in model transformation process		Creating environment for model processing allows combining analytical results with existing plugins and tools for model testing and simulation

Task and challenges

Task: to design model of problem domain "Model-driven architecture formal methods and approaches".

A subsequent task is to represent information for designing model to model transformation automated method, covering all MDA promising.

Explanation, how to follow MDA promising by means of mathematical foundations, is represented in the last grey column of the Table 1.

Challenges to this model:

- Reflect interconnection between mathematical foundations used to design new transformation methods or techniques;
- Serve as a template for defining collaboration between mathematical foundations involved to transformation techniques;
- Simplify the procedure defining compatible data formats for transmitting information about models between different stages of transformation methods.

Domain model design

There is no standard for domain model design, but there are articles containing precise description of this process. According to many recommendations, the first step of domain model designing is to compose a controlled vocabulary. Such one is represented in Table 2.

Table 2. Controlled vocabulary of problem domain "MODEL-DRIVEN ARCHITECTURE FORMAL METHODS AND APPROACHES"

Term	**Definition**
Logic	A particular system or codification of the principles of proof and inference: *Aristotelian logic.* Study of correct reasoning, especially as it involves the drawing of inferences. [Britannica, 2016c] The formal mathematical study of the methods, structure, and validity of mathematical deduction and proof. [MathWorld. 2016g]
Formal logic	Abstract study of propositions, statements, or assertively used sentences and of deductive arguments. From the content of these elements, the discipline abstracts the structures or logical forms that they embody. Alternative titles: mathematical logic; symbolic logic [Britannica, 2016a].
First-order logic	The set of *terms* of first-order logic (also known as first-order predicate calculus) is defined by the following rules: 1. A variable is a *term*. 2. If *f* is an *n*-place function symbol (with n>=0) and $t_1, ..., t_n$ are terms, then $f(t_1,...,t_n)$ is a *term*. If *P* is an *n*-place *predicate* symbol (again with n>=0) and $t_1, ..., t_n$ are *terms*, then $P(t_1,...,t_n)$ is an *atomic statement*. Consider the *sentential formulas* $\forall xB$ and $\exists xB$, where *B* is a *sentential formula*, $\forall$ is the *universal quantifier* ("for all"), and $\exists$ is the *existential quantifier* ("there exists"). *B* is called the scope of the respective quantifier, and any occurrence of variable *x* in the scope of a quantifier is bound by the closest $\forall$x or $\exists$x. The variable *x* is free in the

Term	Definition
	formula *B* if at least one of its occurrences in *B* is not bound by any quantifier within *B*. [MathWorld. 2016e]
First-order predicate calculus	The set of *sentential formulas* of first-order predicate calculus is defined by the following rules: 1. Any *atomic statement* is a *sentential formula*. 2. If *B* and *C* are *sentential formulas*, then ***¬B*** (NOT B), ***B Λ C*** (B AND C), ***B V C*** (B OR C), and ***B⇒C*** (B implies C) are *sentential formulas* (cf. *propositional calculus*). 3. If *B* is a *sentential formula* in which *x* is a *free variable*, then *∀xB* and *∃xB* are *sentential formulas*. In formulas of first-order predicate calculus, all variables are object variables serving as arguments of functions and predicates. The set of axiom schemata of first-order predicate calculus is comprised of the axiom schemata of propositional calculus together with the two following axiom schemata: $\forall x\ F(x) \Rightarrow F(r)$ (1) $F(r) \Rightarrow \exists x\ F(x)$ (2) where *F(x)* is any sentential formula in which *x* occurs free, *r* is a term, *F(r)* is the result of substituting *r* for the free occurrences of *x* in sentential formula *F*, and all occurrences of all variables in *r* are free in *F*. Rules of inference in first-order predicate calculus are the *Modus Ponens* and the two following rules: $\frac{G \Rightarrow F(x)}{G \Rightarrow \forall\, xF(x)}$ (3) $\frac{\mathrm{F(x)} \Rightarrow \mathrm{G}}{\exists xF(x) \Rightarrow G}$ (4) where *F(x)* is any sentential formula in which x occurs as a free variable, *x* does not occur as a free variable in formula *G*, and the notation means that if the formula above the line is a theorem formally deducted from axioms by application of inference rules, then the *sentential formula* below the line is also a *formal theorem* [MathWorld. 2016e].

Term	Definition
Metalogic: Second-and Higher-order Logic	Study and analysis of the semantics (relations between expressions and meanings) and syntax (relations among expressions) of formal languages and formal systems [Britannica, 2016a]. Metalogic may be second or higher order logic. Second-order logic is an extension of first-order logic where, in addition to quantifiers such as "*for every object* (in the universe of discourse)," one has quantifiers such as "*for every property of objects* (in the universe of discourse)." This augmentation of the language increases its expressive strength, without adding new non-logical symbols, such as new predicate symbols. For classical extensional logic, properties can be identified with sets, so that second-order logic provides us with the quantifier "*for every set of objects*." There are two approaches to the semantics of second-order logic. They differ on the interpretation of the phrase "*for every set of objects*." Does this have some fixed meaning to which we can refer, or do we need to consider the variety of meanings the phrase might have? In the first case (which will be called *standard semantics*), we are taking for granted certain mathematical concepts. In the second case (which will be called *general semantics*), much less is being taken for granted. In this case, to be considered valid, a sentence will need to be true under *all the allowable meanings* of the phrase "*for every set of objects*." [Stanford, 2015]. In second-order predicate calculus, variables may denote predicates, and quantifiers may apply to variables standing for predicates [MathWorld. 2016e]. There is no need to stop at second-order logic; one can keep going. We can add to the language "*super-predicate*" symbols, which take as arguments both *individual symbols* (either variables or constants) and *predicate symbols*. And then we can allow quantification over *super-predicate symbols*. And then we can keep going further. [Stanford, 2015]
Language	Set (finite or infinite) of sentences, each finite in length, and constructed out of a finite set of elements [Chomsky, 1957].
Formal Language	Formal language is normally defined by an alphabet and formation rules. The alphabet of a formal language is a set of symbols on which this language is built. Some of the

Term	Definition
	symbols in an alphabet may have a special meaning. The formation rules specify which strings of symbols count as well-formed. The well-formed strings of symbols are also called words, expressions, formulas, or terms. The formation rules are usually recursive. Some rules postulate that such and such expressions belong to the language in question. Some other rules establish how to build well-formed expressions from other expressions belonging to the language. It is assumed that nothing else is a well-formed expression. [MathWorld. 2016f]
Notation	A series or system of written symbols used to represent numbers, amounts, or elements in something such as music or mathematics [Oxford, 2016].
Grammar	Grammar is best formulated as a self-contained study independent of semantics. We consequently view grammars as having a tripartite structure. A grammar has a sequence of *rules from which phrase structure can be reconstructed* and a sequence of *morphophonemic rules* that convert strings of morphemes into strings of phonemes. Connecting these sequences, there is a sequence of *transformational rules* that carry strings with phrase structure into new strings to which the morphophonemic rules can apply [Chomsky, 1957].
Transfor-mational Grammar (Transforma-tional-gene-rative Grammar)	System of language analysis that recognizes the relationship among the various elements of a sentence and among the possible sentences of a language and uses processes or rules (some of which are called transformations) to express these relationships. Transformational grammar assigns a *"deep structure"* and a *"surface structure"* to show the relationship of such sentences. A type of grammar that describes a language as a system that has a deep structure which changes in particular ways when real sentences are produced [Britannica, 2016e].
Generative grammar	Precisely formulated set of rules whose output is all (and only) the sentences of a language — i.e., of the language that it generates. There are many different kinds of generative grammars, including transformational grammar as developed by Noam Chomsky from the mid-1950s [Britannica, 2016b].

Term	Definition
Algebra	The part of mathematics in which letters and other general symbols are used to represent numbers and quantities in formulae and equations [Corry, 2005]. Term algebra usually denotes various kinds of mathematical ideas and techniques, more or less directly associated with formal manipulation of abstract symbols and/or with finding the solutions of an equation. Examples of algebras include the algebra of real numbers, vectors and matrices, tensors, complex numbers, and quaternions. (Note that linear algebra, which is the study of linear sets of equations and their transformation properties, is not an algebra in the formal sense of the word.) Other more exotic algebras that have been investigated and found to be of interest are usually named after one or more of their investigators. This practice unfortunately leads to entirely unenlightening names which are commonly used by algebraists without further explanation or elaboration [MathWorld. 2016b].
Modern algebra	Branch of mathematics concerned with the general algebraic structure of various sets [Britannica, 2016d]. Modern algebra, also called abstract algebra, is the set of advanced topics of algebra that deal with abstract algebraic structures rather than the usual number systems. The most important of these structures are groups, rings, and fields. Important branches of abstract algebra are commutative algebra, representation theory, and homological algebra. [MathWorld. 2016a].
Category theory	A general mathematical theory of structures and of systems of structures. It is a language, or conceptual framework, allowing us to see the universal components of a family of structures of a given kind, and how structures of different kinds are interrelated Category theory is an alternative to set theory as a foundation for mathematics. As such, it raises many issues about mathematical ontology and epistemology. Categories are algebraic structures with many complementary natures, e.g., geometric, logical, computational, combinatorial, just as groups are many-faceted algebraic structures [Stanford, 2014]. Category theory is a branch of mathematics which formalizes a number of algebraic

Term	Definition
	properties of collections of transformations between mathematical objects (such as binary relations, groups, sets, topological spaces, etc.) of the same type, subject to the constraint that the collections contain the identity mapping and are closed with respect to compositions of mappings. The objects studied in category theory are called *categories*. [MathWorld. 2016c]. A category consists of three things: a collection of objects, for each pair of objects a collection of morphisms (sometimes call "arrows") from one to another, and a binary operation defined on compatible pairs of morphisms called composition [MathWorld. 2016d].

Analyzing Table 2 and defining interconnections between vocabulary entities, domain model of mathematical approaches to be used for model transformation tasks is designed and represented on Figure 1.

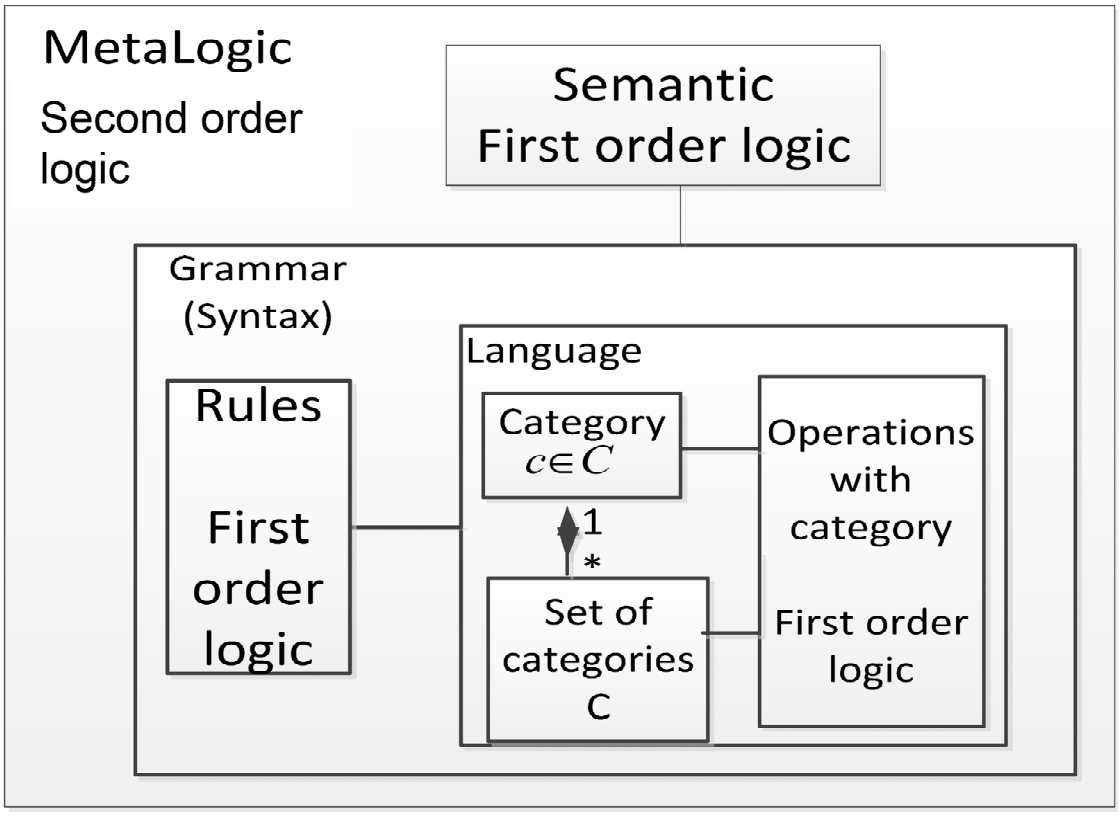

Figure 1. Model of problem domain

Domain model description

Consider mathematical foundations for software model elements description (Table 2).

Languages that are used for designing software models contain a set of elements to be combined for creation of complex software model structures. Considering theory of categories, one may see that each language element corresponds to some category. Combination of language elements allows creating more complex constructions. Also theory of categories is used to describe complex structure from some elements. Thus, it shows complex relations between components of complex structure.

Of course, some rules for designing these constructions should be followed. And first-order logic allows expressing these rules.

From software model elements whole model is designed. Thus, in turn, consider mathematical foundations for whole software model description.

Variety of complex constructions permits design precise software models. Grammar of language, namely its syntax, defines space of rules for checking correctness of software model in a whole. Also, first-order logic allows expressing rules and restrictions for correct model creation.

Semantic rules are used for analysis of software model content. To express semantic rules, first or second order logic is used. First order logic allows composing expressions. Second-order logic is aimed to estimate group of expressions or expressions that works with group of objects. Then chosen logic is used for estimating composed expressions. Also mathematical apparatus for semantic checking should be compatible with tool of software model designing and representation.

Finally, metalogic, or second order logic, as a foundation that studies interconnections between syntax and semantic, should cover all aspects of software model representation and all model processing operations.

Further researches

Further researches are:

1 Analyze model to model transformation techniques in order to define the typical stages of transformation operations.

2 Using controlled vocabulary (Table 2) and domain model (Figure 1), define mathematical foundations for every of these tasks and their interconnections. It has to be done with aim to ground the choice of mathematical foundations for every stage of transformation algorithms.

3 Represent formal description of all model to model transformation operations in terms of chosen mathematical foundations. Then propose techniques, which allow combining different formal solutions of transformational tasks.

Conclusion

Model of problem domain “MDA formal methods and approaches” is proposed in this article. Designed model illustrates relationships between different mathematical foundations that are used in formal software models transformational methods. It is proposed to use domain analysis artifacts, namely controlled vocabulary and domain model, by the following way:

- Decompose transformation techniques into steps;
- Match tasks of every step with possibilities of mathematical foundations, represented in controlled vocabulary (Table 1);
- Define a set of mathematical tools for solving tasks formulated above, analyzing controlled vocabulary and domain model;
- Co-ordinate different mathematical foundations for software model representation and processing.

Performing such operations allows:

- Formulating requirements to data specifications;
- Considering math apparatus for solving transformation tasks more attentively;
- Simplifying software model verification operations;
- Co-ordinate hidden relations between mathematical descriptions.

Bibliography

[Boronat and Meseguer, 2010] Artur Boronat, José Meseguer. An algebraic semantics for MOF. Formal Aspects of Computing. Springer Verlag, 2010, 22 (3), pp.269-296. <10.1007/s00165-009-0140-9>. <hal-00567269> https://hal.archives-ouvertes.fr/hal-00567269/document

[Britannica, 2016a] First-order Logic. Encyclopædia Britannica. 2016. https://www.britannica.com/search?query=first-order%20logic

[Britannica, 2016b] Generative Grammar. Encyclopædia Britannica. 2016. https://www.britannica.com/topic/generative-grammar

[Britannica, 2016c] Logic. Encyclopædia Britannica: 2016. https://www.britannica.com/topic/logic

[Britannica, 2016d] Modern Algebra. Encyclopædia Britannica. 2016.https://www.britannica.com/search?query=Modern%20Algebra

[Britannica, 2016e] Transformational grammar. Encyclopædia Britannica: 2016. https://www.britannica.com/topic/transformational-grammar

[Bruneliere et al., 2010] Hugo Bruneliere, Jordi Cabot, Frederic Jouault, Frederic Madiot. MoDisco: A Generic And Extensible Framework For Model Driven Reverse Engineering. 25th IEEE/ACM International Conference on Automated Software Engineering (ASE 2010), Sep 2010, Anvers, Belgium. pp.173-174, 2010. <hal-00534450> http://hal.univ-nantes.fr/hal-00534450/document

[Chomsky, 1957] Noam Chomsky, Syntactic Structures. Mouton publishers, Eilenberg: Mac Lane The, Hague, 1945 - 1957. ISBN 90 279 3385 5. p.107. http://ewan.website/egg-course-1/readings/syntactic_structures.pdf

[Corry, 2005] Leo Corry. History of algebra. In: Encyclopædia Britannica. 2005. http://www.tau.ac.il/~corry/publications/articles/pdf/algebra%20EB.pdf

[Czarnecki and Helsen, 2006] Krzysztof Czarnecki , Simon Helsen. Feature-based survey of model transformation approaches. IBM Systems Journal Vol. 45 No.:3: 2006. pp. 621 - 645. ISSN :0018-8670, DOI: 10.1147/sj.453.0621. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=5386627

[Ehrig et al., 2006] Ehrig, H., Ehrig, K., Prange, U., and Taentzer, G. (2006). Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. Springer. ISBN 978-3-540-31188-1. http://www.springer.com/us/book/9783540311874

[Favre and Duarte, 2016] Liliana Favre and Daniel Duarte. Formal MOF Metamodeling and Tool Support. In: MODELSWARD 2016, Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development. Edited by S. Hammoudi, L.F. Pires, B. Selic and P. Desfray. SCITEPRESS – Science and Technology Publications, Lda. Portugal, 2016. ISBN: 978-989-758-168-7. pp. 99-110. DOI:10.5220/0005689200990110, http://www.scitepress.org/DigitalLibrary/ProceedingsDetails.aspx?ID=j1i7qrX33Ns=&t=1

[Greiner et al, 2016] Sandra Greiner, Thomas Buchmann, Bernhard Westfechtel. Bidirectional Transformations with QVT-R: A Case Study in Round-trip Engineering UML Class Models and Java Source Code. In: MODELSWARD 2016, Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development. Edited by S. Hammoudi, L.F. Pires, B. Selic and P. Desfray. SCITEPRESS – Science and Technology Publications, Lda. Portugal, 2016. ISBN: 978-989-758-168-7. pp. 15-27. DOI:10.5220/0005644700150027 http://www.scitepress.org/DigitalLibrary/PublicationsDetail.aspx?ID=efZXth7Zbbg=&t=1

[Rabbi et al, 2016] Fazle Rabbi, Yngve Lamo, Ingrid Chieh Yu, Lars Michael Kristensen. WebDPF: A Web-based Metamodelling and Model Transformation Environment. In: MODELSWARD 2016, Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development. Edited by S. Hammoudi, L.F. Pires, B. Selic and P. Desfray. SCITEPRESS – Science and Technology Publications, Lda. Portugal, 2016. ISBN: 978-989-758-168-7. pp. 87-98. DOI:10.5220/0005686900870098, http://www.scitepress.org/DigitalLibrary/PublicationsDetail.aspx?ID=lzjjeczBZuA=&t=1

[MathWorld. 2016a] Abstract Algebra. Wolfram MathWorld. 2016. http://mathworld.wolfram.com/AbstractAlgebra.html

[MathWorld. 2016b] Algebra. Wolfram MathWorld. 2016. http://mathworld.wolfram.com/Algebra.html

[MathWorld. 2016c] Category Theory. Wolfram MathWorld. 2016. http://mathworld.wolfram.com/CategoryTheory.html

[MathWorld. 2016d] Category. Wolfram MathWorld. 2016. http://mathworld.wolfram.com/Category.html

[MathWorld. 2016e] First-Order Logic. Wolfram MathWorld. 2016. http://mathworld.wolfram.com/First-OrderLogic.html

[MathWorld. 2016f] Formal Language. Wolfram MathWorld. 2016. http://mathworld.wolfram.com/FormalLanguage.html

[MathWorld. 2016g] Logic. Wolfram MathWorld. 2016. http://mathworld.wolfram.com/Logic.html

[Mens and Van Gorp, 2006] Tom Mens, Pieter Van Gorp. A Taxonomy of Model Transformation. Electronic Notes in Theoretical Computer Science 152 Elsevier B.V., 2006. pp. 125–142. http://staffwww.dcs.shef.ac.uk/people/A.Simons/remodel/papers/MensVanGorpTaxonomy.pdf

[Oxford, 2016] Notation. Oxford Dictionaries. Oxford University Press, 2016. http://www.oxforddictionaries.com/definition/english/notation

[Rutle, 2010] Rutle, A. Diagram Predicate Framework: A Formal Approach to MDE. PhD thesis, Department of Informatics, University of Bergen, Norway. (2010). http://bora.uib.no/handle/1956/4469

[Stanford, 2014] Category Theory. The Stanford Encyclopedia of Philosophy. The Metaphysics Research Lab, Center for the Study of Language and Information (CSLI), Stanford University. 2015. Library of Congress Catalog Data: ISSN 1095-5054 http://plato.stanford.edu/entries/category-theory/

[Stanford, 2015] Second-order and Higher-order Logic. The Stanford Encyclopedia of Philosophy. The Metaphysics Research Lab, Center for the Study of Language and Information (CSLI), Stanford

University. 2015. Library of Congress Catalog Data: ISSN 1095-5054 http://plato.stanford.edu/entries/logic-higher-order/#4

[Truyen, 2006] Frank Truyen. The Fast Guide to Model Driven Architecture. The Basics of Model Driven Architecture (MDA). Cephas Consulting Corp, 2006. http://www.omg.org/mda/mda_files/Cephas_MDA_Fast_Guide.pdf

[Varro and Pataricza, 2003] Varro, V., Pataricza, A., VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. Journal of Software and System Modeling, Vol.2, Issue 3. Springer-Verlag, 2003. pp. 187-210. ISSN: 1619-1366 (print version), ISSN: 1619-1374 (e-version), doi:10.1007/s10270-003-0028-8 http://link.springer.com/article/10.1007/s10270-003-0028-8

[Zaraket and Noureddine, 2014] Fadi A. Zaraket, Mohamad Noureddine. Model Checking Software Programs with First Order Logic Specifications using AIG Solvers. arXiv, Cornell University, Ithaca, New York 14850. arXiv:1409.6825v1 [cs.SE] 24 Sep 2014. https://arxiv.org/pdf/1409.6825v1.pdf

Authors' Information

Elena Chebanyuk – *Software Engineering Department, National Aviation University, Kyiv, Ukraine,*

Major Fields of Scientific Research: *Model-Driven Architecture, Model-Driven Development, Software architecture, Mobile development, Software development,*

e-mail: chebanyuk.elena@ithea.org

Krassimir Markov – *Information Modeling Department, Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Sofia, Bulgaria*

Major Fields of Scientific Research: *Software Engineering, Cognitive Science, Information Modeling, Multi-dimensional Graph Data Bases, Business informatics, General Information Theory*

e-mail: markov@ithea.org