VOL. 9, NO. 7, JULY 2014 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

METAMODELLING ARCHITECTURE FOR MODELLING DOMAINS WITH DIFFERENT MATHEMATICAL STRUCTURE

Mezhuyev
Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, Gambang, Malaysia
E-Mail: mejuev@ukr.net

ABSTRACT

The new metamodelling approach for domain specific modelling is proposed in the paper. The additional level of the metamodelling architecture is introduced, which gives the possibility of metamodels development in the different mathematical semantics. This allows to take into account the mathematical structure of modelled domains, and to use the mathematical operations for development of new effective methods for solving domain specific tasks. The applicability of the approach for development of metamodels for modelling different domains is shown.

Keywords: domain specific modelling, metamodel, metamodelling architecture, mathematical structure, formal system.

INTRODUCTION

The methodology of Domain Specific Modelling (DSM) becomes more and more popular today, allowing overcoming the known issues of the "universal" modelling approach [1]. The sense of DSM is development of Domain Specific Languages (DSLs), applicable for modelling properties of particular domains. A DSL is built inside a so called metamodel, defining the concrete syntax of the language. The ab-stract syntax of a DSL is defined in the frame of the meta-metamodel as e.g. MOF [2], GOPPRR [3], MGA [4] etc.

Emphasizing the power of the existing DSM approaches, they have a number of issues, caused by the lack of generalization and formalization:

- the metamodel based DSLs are mostly descriptive, i.e. not expressive for the definition of methods for solving domain specific tasks;
- the applicability of a DSL by the generation of software data and code is limited;
- while the DSM approach is intended for using by domain experts, the obligatory involvement of IT specialists for development of code generators is needed;
- for code generation an additional external language should be used, which is not linked with specifics of a modelled domain;
- the meta-metamodel, used for metamodels development, does not reflect the mathematical structure of a considered domain and is hardcoded inside a DSM tool.

Let's consider the principles of the proposed approach to the metamodels development, allowing overcoming the specified above issues:

- the formal definition of the object of modelling - the domain, as the set of entities, linked by the forming mathematical structure and the domain specific relationships;
- the definition of the meta-metamodel and the metamodel as the formal systems, allowing to fix correspondingly the structural and domain specific properties;
- the mathematical structure of a domain is defined at the meta-metamodel level and next is used as the carrier of domain specific properties;
- the additional level of the metamodelling architecture is introduced, which allows to develop the meta-metamodels, having different mathematical semantics.

While the existing metamodelling approaches use the predefined mathematical for-malisms (mostly, graphs) for structuring domain properties, here the development of meta-metamodels in the different mathematical semantics is possible. Additional level of the metamodelling architecture allows to express properties of domains in terms of set theory and to reflect different mathematical structures (algebraic, topological, differential, geometrical etc.). Corresponding mathematical operations are integrated in the metamodel and used for solving domain specific tasks. Generation of software data and code becomes the partial case of the proposed metamodelling approach.

The paper is organized as follows. First the new metamodelling architecture is dis-cussed in comparison with existing approaches. Section 3 of the paper shows applicability of the proposed approach for producing the graph based metamodels for modelling software systems. Section 4 expands the practical applications for requirements engineering, business process modelling and solving tasks of multidimensional physical domains. The conclusion, plan of future research and references list finalize the paper.

Metamodelling architecture

The methodology MOF (Meta Object Facility) [2] was used by the OMG (Object Management Group) consortium for development of the Unified Modelling Language (UML). MOF has the four levels of the metamodelling architecture. The top level is the meta-metamodel (M3), defining the language for development of the metamodels (having the level M2). The level M2 (here, UML) used for development of the domain models of the level M1 (the UML-models). The last is the level of

data (M0), describing the concrete instances of M1. The MOF architecture is based on the object-oriented methodology of software systems design.

The meta-metamodel GOPPRR (Graph-Object-Property-Port-Role-Relationship) allows producing metamodels inside the graph based notations, by means of connection of objects by relationships, definition of domain properties (attributes) and roles [3]. Each of the GOPPRR concepts a metatype is called. As MOF, the metamodelling architecture of the GOPPRR in four levels can be shown (see the Figure-1).

The proposed approach also has the multiple-level metamodelling architecture, but it semantics differs from the existing methodologies. All of the metamodels are considered to be formal systems; they contain an alphabet of types, a grammar and operations. We introduce the additional level of the metamodeling architecture - the meta-meta-metamodel (M4), as a formal system, that is built on the basis of set theory. M4 includes the meta-metatype "element of a set", set operations and grammar rules, which (taken together) allow us to specify a set structure. This approach allows us to consider a domain as a set of heterogeneous entities, having domain specific properties and linked by different kinds of mathematical structures.

Formally, we define a domain as a set of entities *D*, linked by structural *S* and domain specific *P* relationships:

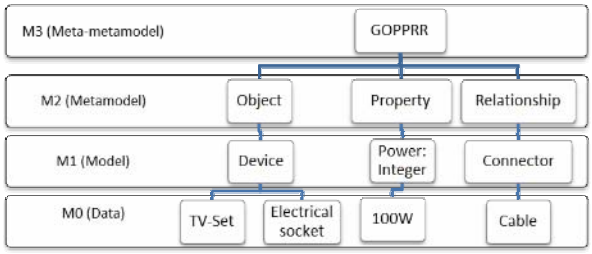

Figure-1. The GOPPRR metamodelling architecture.

$$D = \{d_1, d_2 ..... d_N\}, S, P \subseteq D \times D \quad (1)$$

where *N* is a power of *D*. Each element of *D* can have attributes, which we consider as unary relationships on *D*. 0-ary relationships are used to identify elements of *D*. Binary and other relationships are used to fix mathematical structure of D.

All of the levels of the proposed metamodelling architecture contain not only descriptive elements, such as in MOF or GOPPRR, but also procedural part, implemented with software functions.

Following our proposal, the architecture for development of the graph based metamodel on the Figure-2 is shown. Here a node and an edge of a graph serve as the mathematical metatypes for development of domain specific metamodels types (an attribute is the inherent part of a node and of an edge). The node and the edge are produced from the meta-meta-metamodel as the having algebraic structure subsets of the composing domain entities. Note, while GOPPRR [3] and MGA [4] also use the graphs for structuring domain specific properties, this is a partial case of the proposed approach, where development and using the different mathematical structures is possible.

The implementation of mathematical operations of the metamodels at all levels of the proposed architecture, forms the Application Program Interface (API) of the corresponding software tool. The API of M4 contains the methods for manipulation with the elements of a set of composing domain entities. The API of M3 is the operations with subsets (e.g., with a node and an edge of a graph, and in the general case with any model objects of the considered domain). For M2, the API contains the metamodel processing routines (here, the metatypes of the level M3 become domain-specific types, i.e., to the mathematical subsets the semantics of the domain is assigned). M1 contains instances of the types and definitions of domain-specific methods, implemented with the APIs of all the previous levels. M0 is data values and processes in the computer memory (instances of the methods, defined at the level M1).

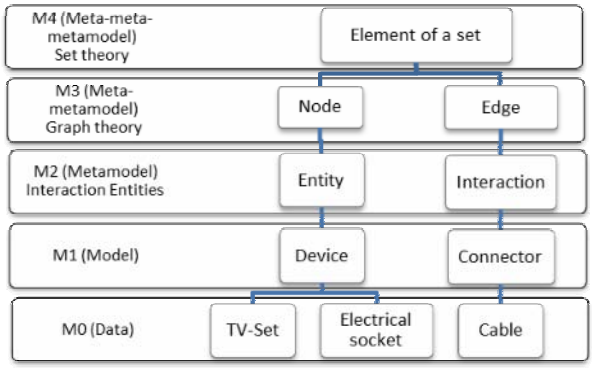

Figure-2. The levels of the proposed metamodelling architecture.

Development of graph-based metamodels

Let us consider the mathematical method for producing the graph-based meta-metamodel in the context of proposed approach. Its alphabet includes the metatypes node *N* and edge *E* of the graph *Gr = (N, E);* the grammar G_{Gr} is the set of rules, defining the possibility *{true, false}* of the connection of nodes n_i, n_j by the edge $e_k = (n_i, n_j)$, $n \in N$, $e \in E$

$$G_{GR} = \left\{ \begin{matrix} \left(n_i n_j\right) \middle| g_k \in \{true, false\}, n_i n_j \in N, i, \\ j = 1..M, k = 1..K \end{matrix} \right\} \quad (2)$$

where *M* is a power of *N*. The number of rules *K* depends on the properties of the graph *Gr* (is it directed, are loops possible, etc.).

At the level of metamodel development, to the nodes and the edges of the meta-metamodel the semantics

of domain is assigned. For example, the node *N* can be the metatype for definition of the types of software tasks and synchronization objects, and the edge *E* can be the metatype for definition of the types of channels (communication protocols) between tasks and synchronization objects. This metamodel will include the alphabet, containing typical for parallel programming synchronisation objects (critical section, mutex, semaphore, resource, FIFO etc.) and software tasks (driver, application etc.); the grammar rules, specifying the valid interactions of software tasks via synchronisation objects, and operations, used for definition of code generation functions.

Table-1 shows an example of the definition of the metamodel for modelling the parallel concurrent software system inside the graph based meta-metamodel.

In this example, a Node and an Edge are the mathematical metatypes of graph based meta-metamodel M3. Domain specific types are the nodes Task, Sync and the edges PutData, GetData, which compose the alphabet of M2 metamodel and are used to create instances at the M1 level.

M2 also defines the grammar rules for combining instances of the types by using predicates PutData (Task, Sync) and GetData (Sync, Task). These grammar rules correspond to the edges of the graph-based meta-metamodel and are used for development of code generation methods (implemented by walking the graph based model M1). The M1 model of software system includes instances of $Task_1$, $Task_2$ ... $Task_T$ and synchronization objects $Sync_1$, $Sync_2$ ... $Sync_S$, linked by the channels of interaction PutData, GetData (where *T, S* - are the number of tasks and the number of synchronization objects in the model respectively).

For the interesting reader, to show the applicability of described graph based metamodel, we can refer to the metamodel of interacting entities [5], which was used for development of a real-time operation system [6] and for modelling distributed parallel real-time software [7].

The definition of the metamodel alphabet as the set of attributed types and the domain model as the instances of the types, having the concrete values of attributes, make possible the formal checking a model in its state space. Due to including mathematical methods in the metamodel the checking properties of behaviour of a real-time system (e.g. absence of deadlocks) was applied. The graph based methods (e.g. Dijkstra's algorithm) for development of the code generation functions (e.g. routing table of a real time operation system) were used.

Table-1. Levels of metamodelling architecture for a software system modelling.

Level		Alphabet	Grammar	Operations / Methods
M 4	Math structures	Elements *d* of the set *D*	The rules of grammar, based on the relations $d \in D$, $\{d\} \subseteq D$	Create / delete element *d*, subset *{d}*
M 3	Math structures	Node $n \in Node$ and edge $e \in Edge$ of graph $G = (Node, Edge), Node, Edge \subseteq D$	Connection of nodes by edges $e_k(n_i, n_j)$, $n_i, n_j \in Node$, $e_k \in Edge$, $i, j = 1..\|Node\|$, $i \neq j$, $k = 1..\|Edge\|$	Add edge $G' = G + e$ Delete edge $G' = G - e$ Add node $G' = G + n$ Delete node $G' = G - n$
M 2	Domain specific properties	Node Task, Sync; Edge PutData, GetData;	PutData(Task, Sync); GetData(Sync, Task)	Add / delete a type of task Task / sync object Sync, create communication channel PutData, GetData
M 1	Domain specific properties	Task Task1, Task2; Sync Sync1; PutData (Task1, Sync1); GetData (Sync1, Task1);		

OTHER APPLICATIONS OF THE METAMODELLING APPROACH

Except development of graph-based metamodel for software systems design, the applicability of the proposed approach was proven for the next domains:

- Requirements Engineering (RE), where conceptual metamodelling for systems specification was used. The set of the typical for the RE concepts formed the alphabet of the metamodel, which symbols were the types for instantiation - definition of the concrete statements describing a system properties and behaviour. The methods of the graph based meta-metamodel were used to check correspondence of the graph of architectural decomposition to the graph of initial requirements, generate the document of systems specifications, made the control of versions etc. The conceptual metamodel was further expanded by the Finite State Machine formalism [5]. This allows us to build the domain specific models of processes on the base of the ontology of a domain. To each concept of ontology the state transition attribute was added. The process grammar was the set of rules, defining the state transitions of conceptual model of a system description. E.g. only after capturing requirements user can move to the specification stage, next to the phase of architectural modelling etc. Such the approach allows us to manage users activity to achieve the goal of a process in a given time (up to deadline);

- Development of the metamodel, based on the vector algebra and the logic of syllogisms. Here vectors were used as the metatypes for producing the logical types of the metamodel alphabet. In the practical implementation [8], the alphabet of the metamodel on the base of the types of categorical syllogisms was developed. Due to using vector algebra for the definition of the metamodel, the operations on syllogisms as operations on vectors in linear vector space were implemented. This allows us to develop the algorithm for automatic geometrical theorem proving. The approach was used for development of the logic for optical computers, where at physical level vectors were implemented as laser beams;
- Development of the metamodel for multidimensional physical domains [9]. The alphabet of the meta-metamodel was defined as the set of the basic (corresponding to the dimensions of the physical space) geometrical objects, i.e. point, line, surface and 3D region. For metamodels development we set distributions of physical properties among the defined with the meta-metamodel geometrical structures. Due to considering objects as the sets of geometrical points in the physical space, the grammar of the metamodel in the terms of Boolean operations on geometrical subsets was defined. This grammar limits the possible compositions of the geometrical objects in the 3D space. The mathematical methods of the metamodel correspond to the solutions of multidimensional tasks of the integral and deferential calculus. As the interesting application of the metamodelling approach for physical domains the design of metamaterials (artificial composites with specific optical properties) can be mentioned [10].

PLAN OF FUTURE RESEARCH

The plan of research is further exploring the properties of the metamodels, allowing fixing different mathematical structures:

- the formal definition of the metamodels, the mathematical structure of its types, grammars and operations at all levels of the metamodelling architecture;
- learning the linguistic properties of the metamodels, incl. the possibility of reduction of the grammars into the normal Chomsky form;
- definition of the method for metamodels composition, allowing to combine the declarative and imperative constructs (alphabet, grammar and operations);
- exploring the textual and the visual forms of expression of metamodels and development of the method for its combination;
- expansion of the approach on the other types of mathematical structures (metric, geometrical, differential, topological, etc.).

CONCLUSIONS

The new approach for metamodels development is proposed. The metamodelling architecture is decomposed into the layers, allowing fixing the structural and the domain specific properties. This allows taking into account the mathematical structure of considered domains. The additional set-based level of the metamodelling architecture is introduced, which allows defining the meta-metamodels in the different mathematical semantics.

REFERENCES

[1] Robert B. France, Sudipto Ghosh, Trung Dinh-Trong, Arnor Solberg. “Model-Driven Development Using UML 2.0: Promises and Pitfalls,” IEEE Computer, February 2006.

[2] ISO/IEC 19502:2005, Information technology. Meta Object Facility. - ANSI. 2007. p. 292.

[3] Steven Kelly and Juha-Pekka Tolvanen. “Domain-Specific Modeling: Enabling Full Code Generation,” Wiley-IEEE Computer Society Pr. 2008. p. 427.

[4] Gregory G. Nordstrom. Metamodeling - rapid design and evolution of domain-specific modeling environments // Dissertation for the degree of Doctor of Philosophy in Electrical Engineering. Nashville, Tennessee, 1999. p. 170.

[5] Vitaliy Mezhuyev, Bernhard Sputh and Eric Verhulst. Interacting Entities Modelling Methodology for Robust Systems Design // 2nd International Conference on Advances in System Testing and Validation Lifecycle. CPS Publishing. 2010. pp.75-80.

[6] Formal Development of a Network-Centric RTOS / Eric Verhulst, Raymond T.Boute, José Miguel Sampaio Faria, B.H.C. Sputh and Vitaliy Mezhuyev. Springer, 2011. p. 227.

[7] Vitaliy Mezhuyev. Domain specific modelling distributed parallel real time applications // The systems of information processing. 2010. 5(86). pp. 98-103.

[8] Vitaliy Mezhuyev. Vector logic: theoretical principles and practical implementations // The papers of Zaporizzia National University. - Zaporizzia: ZNU. 2006. pp. 91-97.

[9] Vitaliy Mezhuyev, Oleg Lytvyn. Metamodel for visual modelling multidimensional domains and its practical applications // Control systems and machines. 2010. 4: 31-43.

[10] Vitaliy Mezhuyev, Felipe Pérez-Rodríguez. Visual Environment for Metamaterials Model-ling // Some current topics in condensed matter physics. Universidad Autónoma del Estado de Morelos. 2010. pp. 1-13.