RESEARCH ARTICLE

ScienceAsia 42S (2016): **19–25**

doi: 10.2306/scienceasia1513-1874.2016.42S.019

Method for domain-specific mathematical modelling: theory and applications

Vitaliy Mezhuyev

Faculty of Computer Systems and Software Engineering, University Malaysia Pahang, Gambang, 26300, Pahang Darul Makmur, Malaysia

e-mail: mejuev@ukr.net

Received 31 Aug 2014
Accepted 20 Jul 2016

ABSTRACT: A method of domain-specific mathematical modelling (DSMM) whose implementation aims to overcome the shortcomings of existing domain-specific modelling methods is proposed. The levels of the metamodelling architecture and the formal semantics of the DSMM metamodels are defined. Examples of the DSMM application for the development of the metamodels for modelling various domains are considered.

KEYWORDS: domain-specific modelling, metamodel, modelling language

INTRODUCTION

In computer engineering, there are several methods for modelling domains to develop software systems. In any case, a model of domain is created using a universal language, or by development of unique for domain (domain-specific) language.

Along with the possibility of covering multiple domains, universal modelling languages, the most prominent of which is unified modelling language (UML, www.uml.org), have many drawbacks. The need to develop models for a wide range of domains results in a situation, that modelling concrete aspects of specific domains remains outside of the possibilities of universal languages. In this case, to support the modelling-specific properties of domains the dialects (special profiles) of universal languages are developed. For example, the big list of UML profiles fully negates the idea of its 'universality.'

The essence of domain-specific modelling (DSM) is to develop software tools that allow us to define domain-specific languages (DSLs) and to apply them for modelling-specific domains. Due to implementation of the modelling in terms of those domains, but not universal concepts, using DSLs simplifies the process of modelling and increases the adequateness of domain models.

The number of operations needed to develop a software system using universal modelling languages corresponds to the number of operations for direct code writing (www.dsmforum.org/why.html). Thus using 'universal' approaches does not reduce the time needed for software development in comparison with traditional way of programming. Increasing the effectiveness of development a software systems using DSM can be compared with the transition from assembler to high-level programming languages. As each concept of a high-level programming language corresponds to a set of commands in assembler, each concept of a DSL corresponds to a set of commands in a high-level programming language.

Large companies (such as Microsoft) and consortia (like Eclipse) are currently involved in the development of the DSM method. Software tools that implement DSM include METAEDIT+[1], ECLIPSE DSL TOOLKIT[2], MS DSL TOOLS[3], JETBRAINS MPS (www.jetbrains.com/mps). However, despite the power of DSM, its theoretical basis and practical implementation have several limitations: the use of DSM is narrow and mostly limited by the generation of software data and code, while this approach can be extended by over techniques (e.g., mathematical modelling and simulation); the developed with DSM modelling languages are descriptive and not suitable to definite methods for solving domain-specific problems; the syntax of DSLs is limited and often defined as a graph-based notation (e.g., entity-relationship diagram); the existing metamodels do not reflect the mathematical structure of considered domains, which prevents domain adaptation of the metamodels; finally, there are no formal definitions of the object of modelling domain and of the metamodels for DSLs, which prevents the mathematical elaboration of the metamodelling method.

In this paper, we will define the syntax of the metamodels for DSLs on the basis of set theory, graph theory, vector algebra, and other mathematical formalisms. Having all the benefits of modelling in terms of a domain, the result of such the approach is a mathematical model, to which appropriate formal methods are applied. That is why we name the proposed method domain-specific mathematical modelling (DSMM). DSMM allows us apply DSM to other domains such as cyber-physical domains[4,5]. Generation of software code and data remains the important partial cases of the application of proposed method.

This article is organized as follows. First, we discuss the basic ideas of the proposed approach. Then, we define the formal model of a domain, a metamodel, and the levels of the DSMM metamodelling architecture. Finally, we consider proving the concept results.

MATERIALS AND METHODS

The proposed method is essentially based on the definition of the object of modelling domain. In this paper, we consider the domain as allocated by the context of consideration the set of entities D, linked by the structural (S) and the domain-specific (P) relationships:

$$D = \{d_1, d_2, \dots, d_N\}, \quad S, P \subseteq D \times D, \tag{1}$$

where N is a power of D. Each element of D can have attributes, which are considered as unary relationships on D. Attributes define domain-specific properties of D. 0-ary relationships identify the elements of D. Binary and other types of relationships fix the mathematical structure of D.

In the logical aspect, the set D is the domain of interpretation of the model *M1*, built within the metamodel *M2*. Each predicate symbol of *M1* is associated with one n-ary relation on D, and each functional symbol with some n-ary operation on D. These relations and operations are the significant elements of the metamodel *M2* being considered as a formal system.

Mathematically, we define the metamodel *M2* as a triple, which contains the alphabet A (the carrier of the formal system), grammar G, and operations O,

$$M2 = \langle A, G, O \rangle. \tag{2}$$

In particular, the metamodel can be an algebraic system, in which the carrier is the set D, corresponding to the definition (1). The feature of development of metamodels as formal systems is that the result of applying the rules G of the grammar and operations O to the carrier A at this level is used as a carrier of the formal system at the next level of the DSMM architecture. The set D is a carrier for the *M4* (meta-meta-metamodel), by applications of operations on D and structuring D within the rules of grammar G, we obtain metatypes *MT* to be used as a carrier of the formal system for the development of the metamodel at the level *M3* (the meta-metamodel).

Let us consider the levels of the DSMM architecture and their implementation in the corresponding software tools. From a linguistic point of view, the levels of the DSMM architecture: meta-meta-metamodel (*M4*), meta-metamodel (*M3*), and metamodel (*M2*) define the specific modelling languages (DSLs). Each level of the DSMM architecture is used for the definition of the modelling language of the next level.

The levels of DSMM architecture are arranged in a way to ensure the direction of the domain analyses from abstract to concrete. The first stage of the metamodelling and, accordingly, the highest level of abstraction, is a consideration of a domain as a set of heterogeneous entities D (1). Consideration of mathematical structure and domain-specific properties of D is performed at the levels *M3* and *M2*, respectively. At the level *M3* the elements of D are structured as the mathematical metatypes *MT*, and next are used to build domain specific types T at the level *M2*. The essence of the metamodel adaptation is linking defined at the level *M3* mathematical structures of a domain with domain attributes at the level *M2*.

The following relationships are between carriers of the previous (prototype) and the next (image) levels of the DSMM architecture:

$$M_{43}: \quad D \to MT, \qquad M_{43} \subseteq D \times MT, \tag{3}$$

$$M_{32}: \quad MT \to T, \qquad M_{32} \subseteq MT \times T, \tag{4}$$

$$M_{21}: \quad T \to I, \qquad M_{21} \subseteq T \times I, \tag{5}$$

where D is the set of domain entities (1), *MT* metatypes, T domain specific types, and I instances of the types. M_{43}, M_{32}, M_{21} are the mappings, used, respectively, for domain structuring, metamodel adaptation, and types instantiation (Fig. 1).

Mappings M_{43}, M_{32}, M_{21} are homomorphic, i.e., defined on D operations are applicable to the equivalent structures in the sets *MT*, T, and I; e.g., in accordance with the M_{43}, we allocate in the set D the subsets $\{d_1, d_2, \dots, d_n\}$ corresponding to the

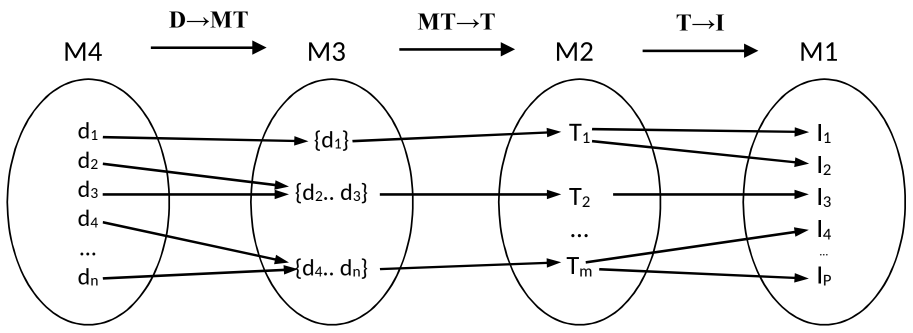


Fig. 1 Mappings of domain structuring, metamodel adaptation, and types instantiation.

elements of *MT*:

$$M_{43}(\{d_1, d_2, \ldots, d_n\}) = MT$$
$$\forall mt \in MT : \exists \{d_1, d_2, \ldots, d_n\} \in D,\ n = \overline{1 \ldots |D|}. \tag{6}$$

Such a mathematical types *MT* can be the model objects of any mathematical apparatus, e.g., nodes and edges of a graph in the case of graph-based structuring of a domain (the examples will be considered in the next section).

The essence of the DSMM method is to identify and incrementally transform the special information objects—metamodels. The meta-meta-metamodel at the level *M4* contains the corresponding to D carrier (alphabet) *A4*; the grammar *G4*, defining the rules for structuring elements of D and the operations *O4* on the elements of the set D. In the software implementation of DSMM[6], each symbol of the alphabet *A4* is typed as the meta-meta-type (*MMT*) the 'element of the D' (Fig. 2).

The grammar rules *G4* allows users to fix the memberships of an element d in the set $D (d \in D)$, and of any subset $\{d_1, d_2, \ldots, d_N\}$ in the set $D (\{d\} \subseteq D)$.

O4 includes operations for creation and deletion of the elements of the set $d \in D$; and for creation, deletion, union, intersection, and subtraction of an arbitrary subset $\{d\} \subseteq D$. At the level *M4* the *API4* (Application Programming Interface) of the DSMM tool has software functions *FO4* and *FG4*, implementing the operations *O4* and the rules *G4*, respectively.

The meta-metamodel *M3* contains as carrier the alphabet of the metatypes *MT* used to produce domain specific types T, the grammar *G3*, defining rules for structuring elements of *MT* and the set of applicable for *MT* mathematical operations *O3*.

Defined at the level *M3* metatypes *MT* show the mathematical structure of D. For example, in the case of using graph theory for structuring hierarchical domains, elements of *MT* are the node and the edge of a graph, *O3* are the operations of addition and deletion of a node and an edge of a graph, *G3* contains the rules for connection of nodes using edges of a graph. The grammar of the level *M3* can be any law, that define the structural relationships S of the elements of D. This is why the proposed metamodelling architecture allow users to model domains having different mathematical structure[7].

The metamodel *M2* is the set of types T inherited from the metatypes *MT* and attributed by domain properties; the rules of the grammar *G2* reflect domain-specific relationships P; operations *O2* are used for the definition of software-based solutions F for domain-specific problems (e.g., code generation).

A user of a DSMM tool develops a domain model *M1* by instantiation of the types T, linking their instances within the rules of grammar *G1*, and application of the set of methods F. Thus the model of a domain is a set of structured within *G1* instances of the types T and processes F in a computer memory. In other words, we consider a model of a domain in DSMM as a specific software system.

Let us generalize the basic principles of the DSMM:

Definition and hierarchical structuring the levels of the architecture of domain-specific modelling to ensure the direction of analyses from abstract to concrete during the development of metamodels and models of domains.

Mathematical definition of the metamodels and formulation of the properties of all levels of the metamodelling architecture inside set theory.

Development of the specific for each level of DSMM architecture metamodel. Metamodel at each level of the architecture defines the modelling language for the next level.

Linking linguistic aspects of metamodelling

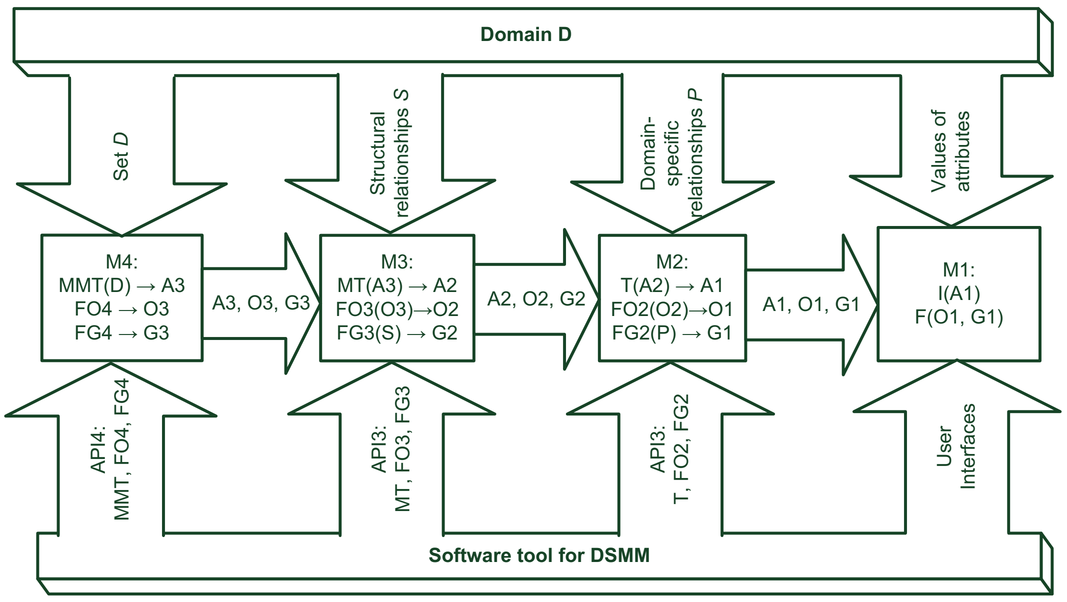


Fig. 2 The scheme of domain-specific mathematical modelling.

with mathematical theory of modelling. This is done by including in the metamodels mathematical operations, which allows users not only to describe properties of domains, but also define methods for solving domain-specific problems.

Predominant role of semantics over syntax. Meta-metamodel defines a syntax of the metamodel, based on a formal (e.g., algebraic) system. Metamodel defines the semantics of a domain (types of attributes, possible structures of systems, laws of grammar and methods for problem solving).

Development of the metamodels for modelling different domains (software, hardware, physical, chemical, etc.), as well as domains, elements of which are heterogeneous.

Support for the users at different levels of analyses and qualifications: a mathematician creates meta-metamodel; a domain expert develops metamodel, and the methods to solve a class of problems; a domain specialist uses a metamodel for the modelling and solving problems that arise in the domain.

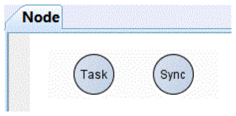
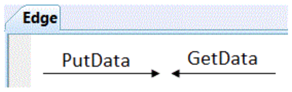
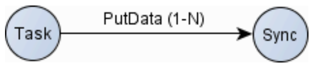
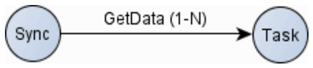
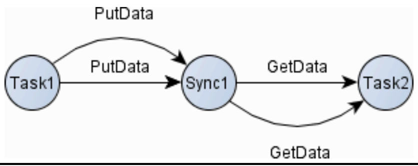
RESULTS

The considered method was implemented in the set of software tools for domain-specific mathematical modelling[6]. These tools were used to specify complex heterogeneous hardware and software systems[8], to develop the metamodels and modelling distributed real time software[9], to desig metamaterials (composite materials with properties, exceeding the properties of natural physical materials)[10], and for some other domains.

Let us consider an example of DSMM application for the development of metamodels and modelling distributed real-time parallel software systems. Since complex software systems are hierarchical, we developed a graph based meta-metamodel, which allows users to define the structure of distributed on computing devices software tasks.

Analysis of interaction of software tasks in a real-time environment allows us to identify the needed properties of the model, in particular, the effect of synchronization when tasks have different temporal semantics (wait, do not wait, wait a time interval)[9]. To develop models of software systems that have the needed properties, the types of tasks and synchronization objects were defined as the alphabet of the metamodel, and the rules of interaction of their instances were formulated as a grammar. For example, the metamodel should guarantee that all tasks in a parallel software system can only communicate through intermediate nodes-synchronization objects. This why we introduce a special type of the metamodel-'synchronization object', which generalizes all the typical for parallel programming synchronization entities like mutex, semaphore, resource, queues, etc. Attributes of this type are synchronization condition and action functions. A user of the DSMM tool defines the semantics of the model of a concurrent system by specifying a synchronization condition and an action that occurs in the case of its truth. This approach allows users to

Table 1 Example of the definition of metamodel $M2_{\text{App}}$ and the model of software in the frame of $M2_{\text{App}}$.

Level	Alphabet	Grammar	Operations and methods
M4	Elements d of the set D	The rules of grammar, based on the relations $d \in D$, $\{d\} \subseteq D$	Create / delete elements and subsets of D
M3	Node $n \in Node$ and edge $e \in Edge$ of graph $G = (Node, Edge)$, $Node, Edge \subset D$	Connection of nodes by edges $e_k(n_i, n_j)$, $n_i, n_j \in Node$, $e_k \in Edge$, $i, j = 1 \dots \lvert Node \rvert$, $i \neq j$, $k = 1 \dots \lvert Edge \rvert$	Add edge $G' = G + e$ Delete edge $G' = G - e$ Add node $G' = G + n$ Delete node $G' = G - n$
M2	**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**
M1		**Task** Task1, Task2; **Sync** Sync1; **PutData** (Task1, Sync1), (Task1, Sync1); **GetData** (Sync1, Task1), (Sync1, Task1);	

develop models of tasks interaction in a distributed parallel software systems that ensure the property of safety synchronization[11].

Table 1 shows an example of the definition of the metamodel $M2_{\text{App}}$ for the modelling concurrent software with the meta-metamodel, based on graph theory. Table 1 also compares visual and textual approaches for the definition of the metamodel $M2_{\text{App}}$ and the corresponding models of the software.

In this sample, a **Node** and an **Edge** are the mathematical metatypes of the graph based meta-metamodel *M3*. The nodes **Task**, **Sync** and the edges **PutData**, **GetData** are the domain specific types, which compose the alphabet of the metamodel and are used to create instances at the level *M1* (for the development of the models of a software system). *M2* also defines grammar rules for linking 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 the development of code generation methods (implemented by traversal of graph of the model *M1*). The model *M1* includes instances of $\text{Task}_1, \text{Task}_2, \dots, \text{Task}_T$ and synchronization objects $\text{Sync}_1, \text{Sync}_2, \dots, \text{Sync}_S$, linked by the channels of interaction PutData, GetData (where T and S are number of tasks and of synchronization objects in the model *M1*, respectively).

DISCUSSION

The DSMM method and corresponding software tools were used to solve several problems:

Web-based DSMM tools were used to organize work of distributed teams to specify the requirements for complex heterogeneous hardware and software systems[8]. Each new project for a system definition was designed as a web portal with the types of content, reflecting proposed by domain expert conceptual metamodel. An alphabet of the metamodel for a system specification contains concepts such as 'requirement', 'specification', 'development task', and 'architecture'. These abstract concepts serve as types for making instances—the statements about properties of a system being developed. The concepts of the metamodel were structured inside the graph based meta-metamodel, which allows users to apply mathematical operations of graph theory to develop the needed methods. For example, a system specification is a graph, whose nodes contain requirements for the architectural elements; the methods of the graph traversal allow users to generate a document of requirements and specifications, select trees of requirements, specifications, and architectural elements for version control, etc.

Another interesting application of the DSMM is the development of logical and algebraic metamodel on the based on vector algebra and logic of syllogisms[12]. The symbols of the alphabet of the meta-metamodel (vectors) are mapped to the categorical statements, which led to the definition of the metamodel 'vector logic'[13]. In this case, the model of a domain consists of the syllogisms, which are instances of the metamodel types 'logical vectors'. Vector algebra was used to define the rules of grammar and methods; in particular, the logical inference (here, the inference is the sum of the vectors, that represent the given assumptions).

For the physical domains, DSMM was used for the development of the metamodel for metamaterials modelling 13. In the frame of set theory, the geometrical meta-metamodel Θ was defined, alphabet of which contains corresponding to the dimensions of a physical space geometrical metatypes (point, line, surface and three-dimensional region). The structure of Θ is caused by the fact, that its symbols are the predominant result of abstraction from geometrical structure of objects in any physical domain. Composition of geometric metatypes allows us to develop domain specific types T. Setting distributions of physical values on geometrical objects links structural (geometrical) and domain specific (physical) properties of the physical model.

CONCLUSIONS

Analysis of the 'universal' approach shows impossibility of developing a language applicable for modelling all existing domains. Thus each domain should have own modelling language to solve arising domain specific problems most effectively. Existing method of DSM has known limitations, to overcome which the method of Domain Specific Mathematical Modelling is proposed.

This article gives the definition of the metamodel as a formal system, which includes carrier (alphabet of types), grammar rules, and mathematical operations. The syntax of DSLs can be defined in terms of set theory, graph theory, vector algebra, or other mathematical system. The approach allows users to consider the result of metamodelling as a mathematical model of domain of discourse, to which the relevant mathematical methods are applied.

The levels of the metamodelling architecture and the stages of domain-specific mathematical modelling are defined. The first stage is a consideration of a domain as a set of heterogeneous entities, linked by structural and domain-specific relationships. The following steps devote to the definition of the structural and the domain-specific types, the grammar and the methods of the metamodels.

Applicability of DSMM has been proven by the development of the metamodels and their use for conceptual and architectural modelling of software systems, for the design of metamaterials and some other domains. Our plan for future research is to expand DSMM method to the different types of mathematical structures (e.g., metric, differential, and topological).

REFERENCES

1. Kelly S, Tolvanen JP (2008) *Domain-Specific Modeling: Enabling Full Code Generation*, Wiley-IEEE Computer Society Pr.
2. Gronback RC (2009) *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, Addison-Wesley Professional, Upper Saddle River, NY.
3. Cook S, Jones G, Kent S, Wills AC (2007) *Domain-Specific Development with Visual Studio DSL Tools*, Addison-Wesley Professional, Upper Saddle River, NY.
4. Mezhuyev V, Samet R (2013) Geometrical meta-metamodel for cyber-physical modelling. In: *Proceedings of 2013 International Conference on Cyberworlds*, pp 89–93.
5. Mezhuyev V, Samet R (2016) Metamodeling methodology for modeling cyber-physical systems. *Cybern Syst* **47**, 277–89.
6. Mezhuyev V (2014) Architecture of software tools for domain-specific mathematical modelling. In: *Proceedings of 2014 International Conference on Computer, Communication, and Control Technology* (Langkawi, Malaysia, Sept 2–4, 2014), pp 166–70.
7. Mezhuyev V (2015) Metamodelling architecture for modelling domains with different mathematical structure. In: *Advanced Computer and Communication Engineering Technology: Proceedings of the 1st International Conference on Communication and Computer Engineering*, Lecture Notes in Electrical Engineering vol 315, Springer, pp 1049–56.
8. Mezhuyev V, Sputh B, Verhulst E (2010) Interacting entities modelling methodology for robust systems design. In: *Proceedings of 2010 Second International Conference on Advances in System Testing and Validation Lifecycle*, pp 75–80.
9. Mezhuyev V (2010) Domain-specific modelling distributed parallel real time applications. *Sist Obrobki Ìnform* **86**, 98–103, [in Ukrainian].
10. Mezhuyev V, Pérez-Rodríguez F (2010) Visual environment for physics modeling and its application for metamaterials design. In: Mora-Ramos ME, Pérez-Álvarez R, Gaggero-Sager LM (eds) *Some Current Topics in Condensed Matter Physics*, Universidad Autónoma del Estado de Morelos, Cuernavaca, Mexico, pp 1–13.

11. Mezhuyev V, Zamli K, Ameedeen MA, Pérez-Sánchez H, Ravi S (2014) Modeling tasks synchronization in the fault-tolerant cyber-physical systems. In: *Proceedings of 2014 IEEE Symposium on Computers & Informatics* (Kota Kinabalu, Malaysia, Sept 28–29 2014).
12. Mezhuyev V (2014) Development of metamodels as logical and algebraic systems. In: *Proceedings of 2014 International Conference on Information Science, Electronics and Electrical Engineering ISEEE* (Hokkaido, Japan, April 26–28, 2014). pp 1850–4.
13. Mezhuyev V (2006) Vector logic: theoretical principles and practical implementations. In: *Visnyk of Zaporizhzhya National Univ: Physical and Mathematical Sciences*, No. 1, 2006, pp 91–7.