Ontology Based Development of Domain Specific Languages for Systems Engineering

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

Abstract—The new approach to development of Domain Specific Languages (DSL) for solving tasks of Systems Engineering (SE) is proposed. A DSL alphabet is defined on the base of an ontology of the SE domain as the set of its typical concepts. A DSL grammar on the base of Finite State Machine (FSM) formalism is defined, which allows to link the SE ontology with the different SE processes (specification of requirements, work planning, development, validation, standardisation etc.). A DSL is mapped with the set of mathematical methods, which allows to solve arising in the SE domains tasks. The approach on the sample of development of the DSL for specification of requirements is illustrated.

Keywords—Domain Specific Language, Ontology, Metamodel.

I. INTRODUCTION

The domain of Systems Engineering (SE) is a big and diverse one. The existing attempts of using one unified language (as e.g. SysML [1]) for modelling all SE artefacts meet with the reasonable critics of theoretical researchers and practical developers. The way to overcome the issues of the universal modelling approach is the development of a Domain Specific Language (DSL) for each concrete SE domain. This paper describes the new approach to development of DSLs, which using increases the effectiveness of solving tasks and managing processes in different SE domains.

A DSL is developed within a so called metamodel, defining the alphabet and the grammar of the language. In the proposed approach we define a DSL on the base of ontology of a considered domain. The set of the typical for a SE domain concepts, as e.g. requirement, specification, architectural module, implementation task, work package etc. forms the alphabet, and the rules of linking instances of these concepts - the grammar of the DSL. The symbols of the DSL’ alphabet are used for instantiation by concrete statements, describing a system properties and behaviour.

The proposed approach is a quite similar with development of an ontology of a domain, where defined at intentional level the set of the general concepts is expanded by the individual concepts at extensional level [2]. The difference with an ontology development is in clear separation of these two levels and considering the general concepts as an alphabet of a DSL, which symbols are used as the types for making instances, composing the conceptual model of a SE domain.

Other principle of the approach can be formulated as “in order to develop a right system it should be developed in a right way”. While the best practices of using e.g. SysML exist, there are always the features, depending on the specifics of a SE domain. As Chourabi et al [3] mentioned, a valuable design knowledge, situated in the context of a concrete problem, is usually lost. The idea of the proposed approach is the development of DSLs, allowing to fix not only the declarative but the procedural part of domain-specific knowledge.

This why the metamodel of a DSL should define not only the typical for a SE domain concepts, but also the procedure, how these concepts will be used (e.g. instantiated). This forms the grammar of a DSL, allowing to define and manage the different processes of systems engineering - specification of requirements, system validation, work planning, standardisation etc.

The paper is organized as follows. First the basic ideas of the approach is described. The metamodel for the DSL on the base of the model of ontology is defined. Formalisation of the DSL grammar on the base of the Finite State Machine (FSM) is proposed. The necessity of expanding the metamodel by mathematical methods is considered. Proving the concept samples are given throughout the paper. The conclusions, plans of future research and the references list finalize the paper.

II. USING ONTOLOGY FOR METAMODEL DEVELOPMENT

Considerable progress has been made in developing techniques, which allow knowledge reusing and sharing [4]. Together with ontologies for fixing static knowledge, problem solving methods have been created for reasoning across domains and tasks. There are also several works devoted to using semantic web technologies for systems engineering, see e.g. [5; 6; 7]. Ontologies are quite often applied for building the systematic, consistent, reusable and interoperable knowledge models. The SE Conceptual Model, as a joint effort of the INCOSE, the ISO AP-233 project, and the SE DSIG [8] worth
mentioning here. The SE Conceptual Model captures the essential concepts of systems engineering, but use it as the unified information model, expressed in the form of a UML class diagram.

To capture the essential specifics of concrete SE domain, an engineer should have the possibility of development of own DSL. For the moment there are no works, devoted to using ontologies for DSLs development. In this paper we propose to take the model of ontology as a base for definition of the metamodel for the different SE DSLs. This is possible due to the fact, that concepts of ontology and metamodel are closely linked. As stated Pidcock [9] “a valid metamodel is an ontology, but not all ontologies are modelled explicitly as metamodels”.

Following the definition, given by Gruber, an ontology is explicit specification of conceptualization [10], where conceptualization is “a set of objects which an observer thinks exist in the world of interest and relations between them”. So the ontology $O$ of the domain $D$ can be considered as the set of the concepts and the relationships between the concepts:

$$O^D = \langle X, \mathcal{R} \rangle$$  \hspace{1cm} (1)

where

$$X = \{ x_k \mid k = 1, \ldots, K \}$$ is the finite set of the concepts of the domain $D$.

$$\mathcal{R} = \{ r_i \mid i = 1, \ldots, L \}$$ is the finite set of the relationships between the concepts $X$.

In order to the ontology (1) be applicable for development and application of the DSLs, we will define it at two levels:

1. The metamodel level (we will use the M2 signature for designation of this level). Here the sets of the concepts $X$ and their relationships $\mathcal{R}$ are the Types, used for instantiation and so for the definition of the model of domain $D$ (we will use $T^X$ and $T^\mathcal{R}$ signatures correspondingly).

2. The model level (we will use the M1 signature for designation of this level). Here the sets of concepts $X$ and their relationships $\mathcal{R}$ are the Instances of the types, were defined at the M2 level (further, $I^X$ and $I^\mathcal{R}$).

So the ontology $O^D$ is decomposed into the ontology of types $O^T$ and ontology of instances $O^I$. This make a difference with the classical definition of Gruber, where the ontology $O^D$ contains as general concepts (types) and individuals (instances of the types).

Let’s consider the simple $O^T$ ontology, corresponding to the classical V-model of a system development. It will be used for the development of a DSL. See fig. 1 for the graphical representation of $O^T$.

![Fig. 1. The sample of the SE ontology (at the M2 level)](image)

The elements of the $T^X$ (the types of concepts) are the nodes of the graph $T^O$

$$T^X = \{ \text{Requirement}, \text{Specification}, \text{Architectural Module}, \text{Development Task}, \text{Validation Task} \}$$  \hspace{1cm} (2)

The elements of $T^\mathcal{R}$ (the types of relationships) are the edges of the graph $T^O$, linking the elements of $T^X$:

$$T^\mathcal{R} = \{ \text{Concretisation}, \text{Description}, \text{Implementation}, \text{Verification}, \text{Validation} \}$$  \hspace{1cm} (3)

The set $T^\mathcal{R}$ by specification of the pairs of elements of the $T^O$ can be also defined

$$T^\mathcal{R} = \{ \langle \text{Requirement, Specification} \rangle, \langle \text{Specification, Architectural Module} \rangle, \langle \text{Architectural Module, Development Task} \rangle, \langle \text{Development Task, Verification} \rangle, \langle \text{Validation Task, Validation} \rangle \}$$
Thus $T^\mathcal{X}$ defines the rules of connection of instances of elements of $T^\mathcal{X}$ and can be considered as the simplest grammar of the DSL. Mathematically, (4) defines the edges linking the nodes (3). Note, the graph based formalisation of conceptualisation is the most common in ontology engineering. But in the general case, the different types of formalisms can be used for the definition of the metamodel grammar (we will return to this idea in the future work section, proposing metamodels generalisation as integrated logical and algebraic systems).

Let’s define the metamodel M2 by expanding the model of ontology (1) by the grammar rules $G$:

$$M2 = \{T^O, G\}$$  \hspace{1cm} (5)

where $T^O = \{T^X, T^\mathcal{X}\}$,

$G = \{g_m \mid m = 1, \ldots, M\}$ is the finite set of the grammar rules.

The rules of the grammar (4) define the possible ways of using $T^\mathcal{X}$ (e.g. instantiation and connecting the instances). To capture the procedural knowledge, the grammar $G$ should allow to fix the specific for SE processes, e.g. the order of steps of a system development. In the given above sample, the elements of the set (2) as the stages of a system development process can be considered: after capturing requirements they should be transformed into specifications, next architectural modules have to be allocated, further becoming concrete tasks and work packages for implementation etc.

The derived from $T^O$ the ontology $I^O$ is a conceptual model of SE domain, which sample on the fig. 2 is shown. The concepts $I^X$ are the instances of the corresponding elements of $T^\mathcal{X}$ and compose the nodes of the graph $I^O$

$$I^X = \{ \text{“Car should be fast”}, \text{“Acceleration to 100km/h in 6s”}, \text{“Engine (and other entities)”}, \text{“Engine development”}, \text{“Is acceleration to 100km/h in 6s?”}, \text{“Is a car fast enough?”} \}$$  \hspace{1cm} (6)

Fig. 2. The sample of the SE ontology (at the M1 level)

The mapping $T^O \rightarrow I^O$ is the instantiation relationship between the elements of $I^O$ and $T^O$ and shows the process of a system definition, resulting in the conceptual model of a system under development.

III. DEFINITION OF THE METAMODEL GRAMMAR

Let’s consider the method for modelling SE processes on the base of ontologies of SE domains. The typical way to model a process is to allocate its steps and the order of their execution. We consider steps of a process as the nodes of an ontology $T^O$, which common semantics in SE is a user action, a technical operation, a resource changing etc.

The directed graph on the fig. 1 allows to formalise and visualise the process of a system development. The nodes have the semantics of the steps of the process, and the edges define the direction of the process execution. In this model of a process the next step depends on the result of execution of the previous step only. To expand the semantics of the modelled domains, let’s consider the DSL process grammar on the base of Finite State Machine (FSM).

The FSM defines a model of process as the finite number of states and the rules of transitions between those states. The state is the set of values of attributes of all objects, composing the ontological model of a domain $D$. Each concept of the ontology $T^O$ has attributes, defining the properties of a considered domain, as e.g. an author of a requirement, a responsible of a task execution, a version of review, a deadline etc. To transform the ontology of domain into a process ontology we will add to the concepts $T^X$ the attributes $A$, defining the possible state transitions.

Let’s formalize the DSL process grammar as a Mealy machine, defined by the tuple $S$
\[ S = \langle S, \Sigma, \Gamma, S_0, \delta, \omega \rangle \]  

(7)

where \( S \) is a finite, non-empty set of states.

\( \Sigma \) is the input alphabet (a finite, non-empty set of symbols).

\( \Gamma \) is the output alphabet (a finite, non-empty set of symbols).

\( S_0 \) is an initial state, \( S_0 \in S \).

\( \delta \) is the state-transition function, \( \delta : S \times \Sigma \rightarrow S \).

\( \omega \) is the output function, \( \omega : S \times \Sigma \rightarrow \Gamma \).

The definition (7) specifies the process grammar of a (regular) domain specific language. The input alphabet \( \Sigma \) on the base of the concepts of ontology \( T^O \) is defined. These concepts are expanded by the state transition attributes \( A \), defining a system behaviour. The accepted by the FSM language results in the output alphabet \( \Gamma \).

The input alphabet \( \Sigma \) is a Cartesian product of the set of concepts \( T^X \) of a domain ontology and the state attributes \( A \)

\[ \Sigma = T^X \times A \]  

(8)

The simple example of the attribute, defining the state transitions of a model of a system under development

\[ A = \{ 'Work Started', 'Approved' \} \]  

(9)

The example of the definition of the output function \( \omega \):

\[ \omega(\xi, S) = \begin{cases} 
\gamma \leftarrow Yes, \bar{I}^X \leftarrow \xi : \forall a \in A \land a = 'Approved' \\
\gamma \leftarrow No, \text{otherwise} 
\end{cases} \]  

(12)

where \( \xi \in \Sigma \), \( \gamma \in \Gamma \).

Output \( \omega \) is a function of two variables - the symbol \( \xi \) of the input alphabet \( \Sigma \) and the state \( S \). It sets the symbol \( \gamma \) of the output alphabet \( \Gamma \) into 'Yes', if all attributes \( a \in A \) in \( I^X \) have the value 'Approved'. Otherwise it sets the symbol \( \gamma \) of the output alphabet \( \Gamma \) into 'No'. The definition of the output function \( \omega \) allows to manage the execution of SE processes. This is a way to ensure that a user will execute a process properly, e.g. approve specifications if only all corresponding requirements were approved. Introducing the time attributes allows to guarantee that a user completes all tasks, requested to achieve the objective, up to deadline.

The proposed approach allows not only to model the different SE processes (development, testing, validation, standardisation etc.), but also organize the users activity correspondingly to the model of process. This needs linking the rules of grammar (state transition conditions) with methods, implemented as functions in the corresponding software tools [11]. Such the functions allows to manage users by setting permissions on its actions with the conceptual model of a SE domain. For example, it can be permission to change the status attribute from 'In work' to 'Approved', if all previous nodes in the graph \( I^X \) were set to 'Approved'. In other words, a user
cannot make an action which does not correspond to the grammar rules of the metamodel.

4. MODELLING PROCESSES OF SE

Thus, each edge of the \( I^O \) with a method is linked, whose sequential execution composes a process of a SE domain. Formally, the domain specific process \( P^D \) as a rout on the graph \( I^O \) can be defined

\[
P^D = I^X_1 \times I^X_2 \times \ldots \times I^X_N
\]

where \( N \) is a number of the steps in a process.

I.e. the process on \( I^O \) is the finite sequence of edges \( I^X : I^X_1 \times I^X_2 \times \ldots \times I^X_N \). The result of the process is achievable, if the corresponding to (13) sequence of nodes \( I^X_1 \times I^X_2 \times \ldots I^X_N \) exists and its walking satisfies the property, expressed by the rules of metamodel grammar.

Exactly the process grammar defines the possible routs on the graph \( I^O \). Each rule allows to user to make a step \( S \), where the general amount of steps is Cartesian product \( S \subseteq I^X \times I^X \).

The semantics of a step depends on the considered domain, e.g. relation between the nodes (‘software module’, ‘release’) can be called implementation; (‘specification’, ‘test task’) – validation; (‘specification’, ‘standard’) – audit etc. A sequence of the steps composes a SE process. For example, the sequence of steps (3) corresponds to the process of a system definition. Note, to specify the direction of process the oriented graphs are used, so (4) should contain the ordered pair of nodes.

5. EXPANDING THE METAMODEL BY THE METHODS

The directed graph of a SE ontology, represented on the fig. 2, was considered as a state transition diagram, where the nodes correspond to the states of system, and the edges represent the possible state transitions.

A grammar rule of a DSL is a condition that would be met to enable the state transition of a model of a system. To support the user management in the corresponding software tools, a method is called when a condition of a transition is met. Mathematically, this needs expanding the metamodel by the set of methods \( F \)

\[
M 2 = \langle T^O, G, F \rangle
\]

where \( F = \{ f_q | n = 1, \ldots, Q \} \) is the finite set of methods, used in \( M 2 \).

With set of permissions, allowing to users make actions correspondingly to the rules of the grammar, the other class of methods exists. It depends on the mathematical formalisms, used for an ontology definition. Let’s consider the methods for processing the graph based ontologies of SE domains, which include the editing graph structure, the graph walking, the graphs mapping, forming sub-graphs etc.

The methods for editing graph structure include: create node, delete node, create edge, delete edge, add attribute, delete attribute of the graph of an ontology. Nodes of \( I^O \) are created as the instances of the types \( T^X \), edges as the instances of \( T^X \). Thus, the types instantiation as the method for a system definition is used. Adding attributes to \( T^O \) allows to reflect the quantitative characteristics of a system under development (e.g. roles of users, weight of requirements, priorities of tasks etc.) and is used with waited graphs methods.

A user of corresponding software tool [11] can make a step, if only it is allowed by the grammar of the metamodel. This allows to implement the process, as conditional walking the graph \( I^O \). During the process, a user updates the values of the different attributes of a ontology graph (as e.g., the text attribute at reformulation of requirements).

SE processes are usually cycles, because they reflect the property of iterativeness of a system definition. Possibility of giving the formal definitions for the different properties and processes of SE is a big advantage of the proposed approach, which allows to improve correctness and preciseness of SE.

E.g., we can consider realisation of the property “iterativeness of a system definition process” as cyclical routes on the graph \( I^O \). The partial case of a cycle is the loop (then a rout starts and ends on the same node), which is used for the definition of different graph based methods. Existing parallel edges between nodes (so called multi graph), means that several different methods can be applied or the processes can be executed. Here, the task of optimisation of a SE process arose, if there are the different routes, linking the same nodes.

In addition to optimisation, the approach allows mathematically solve the different tasks of systems engineering. The samples of the graphs based methods, applicable to \( I^O \) are

- finding a rout in an oriented graph, to answer the question if a process achievable, e.g. if the result can be achieved up to deadline etc.;
- calculating amount of nodes in the rout to measure complexity of a system description;
- find a minimal and maximal path between nodes to estimate the time needed for result achievement;
choosing the optimal variant of a route on the base of some criteria, e.g. optimization of process the base of significance of requirements, priority of tasks, costs measures etc.;

graphs mapping, incl. forming sub-graph. The mapping \( f : X \rightarrow Y \), if for all \( x \in X \) we have \( f(x) \in Y \). E.g on the fig. 4 the sample of mapping requirements graph on the graph of architectural elements is shown.

Fig. 4. Mapping graphs of a system requirements and architecture

The applicability of the mapping method is quite wide, e.g. it can be used for checking correspondence of a system to the standards (by mapping the graph of specification on the graphs of standards) etc.

CONCLUSION

The new approach to development of DSL for solving tasks of SE is proposed. The definition of the metamodel expands the model of ontology by grammar rules and mathematical methods. The approach allows: a) take into account the specifics of SE domains by development of corresponding DSLs; b) fix in the metamodel the procedural part of domain knowledge and so implement SE process modelling; c) manage users correspondingly to the developed model of a process to guarantee the correctness of a process execution (e.g. correspondence to a standard, technology, method etc.); d) application of mathematical methods to solve arising in the SE domain tasks (e.g. system definition, validation, certification etc.); e) formalisation of the different SE properties and technologies, nowadays having only the informal conceptual definition (as e.g. iterativeness, agile development etc.).

FUTURE RESEARCH AND DEVELOPMENT

Our future work will increase expressiveness of the grammar and the methods of the ontological metamodels for DSLs development. We will learn the possibility of using different formalisms for development of metamodels inside their general definition as integrated logical and algebraic systems [12]. We will also finalize software tools, implementing the proposed approach.

REFERENCES

[9] Pidcock, Woody. What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a meta-model? http://infogrid.org/trac/wiki/Reference/PidcockArticle