Providing Models of DSL Evolution Using Model-to-Model Transformations and Invariants Mechanisms

Boris Ulitin and Eduard Babkin

Abstract The research is related to the problem of coherent evolution of a domain-specific language (DSL) in response to evolution of the application domain and users' capabilities. We offer a solution of that problem based on a particular model-driven approach. We give the whole definition of DSL in terms of model-oriented approach. Such definition allows us to define the DSL development using the mechanism of consecutive, consistent transformations between DSM, DSL metamodel and DSL concrete syntax model. In our approach we call such transformations as projections.

Keywords Domain-specific language · Domain-semantic model · Projection · Model-driven development · Model-to-model transformations · Evolution · Invariants

1 Introduction

Currently, domain-specific languages (DSL) become more and more widespread. Such popularity can be explained by the fact, that DSL is a fairly simple and convenient way of organizing work in a certain subject area. DSLs contain only the required set of terms of the domain, representing some kind of its reflection, because every DSL uses as its basis some model of the current subject area [1]. As a result, the effectiveness of DSL actually depends on the degree of correspondence between the subject area and its model: a greater level of consistency results in greater flexibility of the language.

First of all, when analyzing the issue of DSL development and use, researchers take into account the fact that DSL should strongly correspond to the subject area for which it is created. All researchers [1–3] note, that the core element of any DSL is a

B. Ulitin (✉) · E. Babkin
National Research University Higher School of Economics, Nizhny Novgorod, Russia
e-mail: bulitin@hse.ru; eababkin@hse.ru

© Springer Nature Switzerland AG 2020

E. Zaramenskikh, A. Fedorova (eds.), *Digital Transformation and New Challenges*,
Lecture Notes in Information Systems and Organisation 40,
https://doi.org/10.1007/978-3-030-43993-4_4

certain model, which is some kind of the reflection of the subject area for which DSL is created. Actually, such a model determines not only the DSL structure, but also its semantic, behavior and mechanisms of working with DSL. For example, Cleenewerck notes that the effectiveness of DSL completely depends on the completeness of its internal model [4].

Researchers also agree, that any domain demonstrates a tendency to changes over time (evolution, in other words). In accordance with the evolution of the subject area, the evolution of its conceptual model also occurs [5, 6]. However, DSL frequently remains unchanged, since it is built on a snapshot of the domain (and its conceptual model) and reflects only the fixed state, without reacting to subsequent changes. This specificity in the process of DSL design results in the emerging the problem of maintaining co-evolution of DSL and its domain and, accordingly, co-evolution of DSL and the conceptual model. At worst, uncoordinated changes can lead to the situation, when DSL, being fixed in its original state, loses its relevance for the significantly changed domain. As a result, the language becomes inapplicable for solving practically important tasks in the subject area. In [7] describe a case of DSL, which could not create entities, unspecified in the original ontology model.

We also need to note that the sceneries of working with DSL may vary due to considerable differences in experience of information needs of different DSL users. As a result, in parallel to the development of the skills and knowledge of the user, the set of DSL terms, that he/she operates with, can also change. It means, that every user defines his own model of DSL and, because every DSL is connected with the domain, creates own domain representation, which may differ from the one originally used in the DSL. In these circumstances, we also face the evolution of the DSL and the subject area, which leads to the inconsistencies between the domain and the DSL model and has to be resolved through the use of interconnected transformations.

Thus, we can argue that DSL is a dynamic system, which can evolve under the influence of various factors. Evolution can occur both under the influence of changes in the subject area itself [3, 4], and under the influence of internal factors, such as changes in the behavior of users of the system [8] or/and their heterogeneity [9].

It is important to note that researchers agree that evolution is an important step in the life cycle of the DSL. However, most of these works doesn't cover the mechanism for tracking changes in the subject area with the consequent translation of them into DSL model. Furthermore, it is believed that by the time of DSL development, the domain model is already created and somehow transferred to DSL model. It is fair to say that some researchers, including Bell [10], Mengerink, Serebrenik, Brand [11], R.R.H. Schiffelers [12] and in particular, Sprinkle [13] are exploring the evolution of graphical domain models. Unfortunately, they do not consider the subsequent transfer of the changes provided (and corresponding rules) into DSL models. On the contrary, they try to keep the structure of DSL unchanged, that does not correspond to the assumption that DSL model is strongly corresponding to the domain model, that means that any change in the domain model should result in an equivalent change in DSL model.

Summarizing, it can be argued that there is a problem of developing mechanisms that provide different models for the DSL evolution. Both under the influence of changes in the domain model, and in the level of user competences.

The most common approach defines a two-level structure of any DSL [14]: the level of meta-model, responsible for the semantic of the DSL, and the level of the concrete DSL syntax. In order to create the meta-model of DSL different grammar tools are used [15], in particular ANTLR, etc. Unfortunately, such grammar-oriented definition of DSL structure is very strong and doesn't allow further modification of the DSL according to the changes in the domain or in users' needs. The only way in this case is to define new DSL, which is not consistent with the previously created. Obviously, such an approach is ineffective in the case when we are dealing with DSLs for rapidly changing domains, where the context is modified not only by changes in the concepts of the domain, but also by the requirements of users [9].

There are some attempts to resolve this problem of DSLs inconsistency. For example, [7] propose to use the formal model of the domain during the DSL development in order to guarantee the maximum coherence between them. As formal domain models, object-oriented UML models and ontologies can be used (in more details both types are described by Guizzardi [16]. Both approaches allow to achieve sufficient flexibility in terms of creating a DSL on the basis of a formal description of the domain. However, in both cases we do not solve the problem with the further evolution of DSL in case of the domain modifications.

There are some attempts to resolve these contradictions. For example, Cleenewerck [3] proposes to segment the domain model and develop the DSL as a set of independent components, connected with each separated fragment of the domain model. The extension of this approach is presented by [10], who not only used the ontology for the domain representation but coordinate changes in it with a change in the DSL. However, these changes were implemented manually, that requires from the users to have skills in the domain conceptualization technics and model transformations.

Another alternative can be using the ontologies as a formal representation of the domain [9]. The ontology allows as to represent the domain as a set of its concepts and relationships between them, as to formalize the constraints of the domain, with special focus on the heterogeneity/taxonomy. Several examples of using the ontologies for the conceptualization of the domain and its application for the DSL quality evaluation are described in articles by Guizzardi [16, 17].

At the same time, all these approaches focus on the alignment of the DSL with the domain model, leaving behind the boundaries the compliance of DSL and the needs of users. This task is interpreted like more applied, implementing at the level of general programming language. On the other hand, Agrawal, Karsai and Shi [18] sufficiently fully justifies that the correspondence between user requirements and changes in DSL can be described in a model-oriented manner. Thus, evolution in the DSL syntax can be organized according to principles similar to the organization of its correspondence with the formal description of the domain.

In our research we support the idea, that ontological model can be the most strong and effective way for the domain representation, because contains not only the

concepts of the domain and relations between them but also restrictions and logical rules, important during the DSL syntax definition. That ontology-based approach seems to be more effective in comparison with similar ones, for example, described by Cleenewerck or by Challenger. Cleenewerck tries to use the mechanism of graph transformation without prior establishing a correspondence between the ontology and DSL. It leads to the need to define a whole complex of disparate transformation rules for each component of the language. Furthermore, this system of transformations has to be changed every time, when DSL modifications are required. In contrast to Cleenewerck, Challenger [7] proposes to abandon the dynamic matching of the subject area and DSL, but to redefine the DSL model whenever the ontology is changed. This approach is also not optimal, since, in fact, it offers not to adapt the existing DSL to changes in the domain, and each time to create a new language.

Much more effective can be using the idea, that DSL can be described in the model-oriented manner. Such definition of DSL allows us to describe the process of its development as a sequence of interconnected, consecutive model-to-model (M2M) transformations. As a result, we can describe definition of DSL evolution in unified manner. Furthermore, we can also guarantee, that DSL dialects (different DSL syntaxes defined under the single meta-model) are consistent and can be transformed between themselves, using the ideas of invariants (resilient metamodel entities). In what follows, we describe our approach in more details, starting with the idea of domain semantic model (DSM) and its further transformation into DSL structure components with the subsequent definition of different DSL dialects with M2M transformations.

2 Background

2.1 Domain Semantic Model (DSM)

Since any DSL contains some domain model, and, as any language, contains semantical and syntactic parts, we argue, that domain model should also contains all needed concepts of the domain semantics. As a result, the domain semantic model should be used as a core element of DSL development.

DSM offers a flexible and agile representation of domain knowledge. DSM can be constituted by either just small pieces of a domain knowledge (e.g. small taxonomies equipped with few rules) or rich and complex ontologies [16] (obtained, for example, by translating existing ontologies). That gives respectively weak or rich and detailed representation of a domain [19]. More formally DSM is a seven-tuple of the form:

$$DSM = (\mathcal{H}_C, \mathcal{H}_R, O, R, A, M, D)$$

where

- $\mathcal{H}_C$ and $\mathcal{H}_R$ are sets of classes and relations schemas. Each schema is constituted by a set of attributes, the type of each attribute is a class. In both $\mathcal{H}_C$ and $\mathcal{H}_R$ are defined partial orders allowing the representation of concepts and relation taxonomies;
- O and R are sets of class and relation instances also called objects and tuples;
- A is a set of axioms represented by special rules expressing constraints about the represented knowledge;
- M is a set of reasoning modules that are logic programs constituted by a set of (disjunctive) rules that allows to reason about the represented and stored knowledge, so new knowledge not explicitly declared can be inferred;
- D is a set of descriptors (i.e. production rules in a two-dimensional object-oriented attribute grammar) enabling the recognition of class (concept) instances contained in O, so their annotation, extraction and storing is possible.

It is also important to note, that DSM usually has a dynamic structure. Any domain demonstrates a tendency to changes over time (evolution, in other words). In accordance with the evolution of the domain, the evolution of its DSM also occurs. As a result, any DSL, based on the corresponding DSM, should be adopted according to the changes. Consequently, the structure of the DSL metamodel should be as close as possible to the structure of DSM in order to guarantee the coherence between DSL and the target domain. So, that is reasonable to select a common meta-meta model which will be used both for definition of a DSL metamodel and DSM. We believe that a widely accepted object-oriented meta-meta model can be suitable for our purposes. The following manifestation of DSL as a special kind of the object-oriented model proves that believe.

2.2 *Manifestation of DSL in Terms of Object-Oriented Models*

Domain-Specific Languages (DSLs) formalize the structure, behavior, and requirements within particular domains problem. Such languages tend to support higher level abstractions than general-purpose modeling languages, and are closer to the problem domain than to the implementation domain. Thus, a DSL follows the domain abstractions and semantics (DSM), allowing modelers to perceive themselves as working directly with domain concepts. Furthermore, the rules of the domain can be included into the language as constraints, disallowing the specification of illegal or incorrect statements.

The definition of a DSL involves at least two aspects: the domain concepts and rules (abstract syntax or metamodel); the notation used to represent these concepts, textual or graphical, (concrete syntax). Such model-oriented definition of DSL allows to use the DSM as a basis for development of DSL metamodel, which is then converted to a concrete syntax. This approach enables the rapid development of languages and some of their associated tools, such as editors or browsers.

From the formal point of view, a metamodel of DSL is derived from the set of entities of the target domain and operations on them. From this point of view, since the structure of every model is a combination (E, R) of some entities and relations between them, the DSL metamodel can be formalized in a model-oriented manner as follows:

- *A set of entities* of the meta-model $Set = \{set_i\}$, $i \in \mathbb{N}$, $i < \infty$, where every entity $set_i = \{SName_i, SICount_i, Attr_i, Opp_i, SRest_i\}$ is characterized by its name ($SName_i$, which is unique within the current model), available amount of exemplars of this entity ($SICount_i \in \mathbb{N}$, $SICount_i \geq 0$), a set of attributes ($Attr_i = \{attr_{j_i}\}$, $j_i \in \mathbb{N}$, $j_i < \infty$), a set of operation on exemplars of this entity ($Opp_i = \{opp_{j_i}\}$, $j_i \in \mathbb{N}$, $j_i < \infty$) and a set of restrictions ($SRest_i = \{srest_{j_i}\}$, $j_i \in \mathbb{N}$, $j_i < \infty$).
- *A set of relations* between the entities $Rel = \{rel_i\}$, $i \in \mathbb{N}$, $i < \infty$, where every relation $rel_i = \{RName_i, RType_i, RMult_i, RRest_i\}$ is identified by its name ($RName_i$, which is unique within the current model), type ($RType_i \in \mathbb{N}$, $RType_i \geq 0$), defining the nature of the relation, the multiplicity ($RMult_i \in \mathbb{N}$, $RMult_i \geq 0$), which defines, how many exemplars of entities, participating in current relation, can be used, and a set of restrictions ($RRest_i = \{rrest_{j_i}\}$, $j_i \in \mathbb{N}$, $j_i < \infty$).

Accordingly we propose to consider the structure of the DSL metamodel as $(E, R, Rest, Opp)$, where the first two E and Rel are responsible for the object-level, and other $Rest = \bigcup_{i=1}^{|E|} SRest_i \bigcup_{i=1}^{|E|} RRest_i$, Opp represent the functional aspects. Interpreting the set E as the set of entities in some domain, R as a set of relations between them and $Rest$, Opp as a set of operations on entities and restrictions to them, we argue that the structure of the DSM and the structure of DSL metamodel can be related by some correspondence. That means, that there is a way for organizing automated development of DSL based on the DSM and vice versa.

The concrete syntax of a DSL provides a realization of its abstract syntax as a mapping between the metamodel concepts and their textual or graphical representation. From this point of view, we can state, that the concrete syntax of DSL can be represented as a reflection of the metamodel, needed for representation of a certain problem situation.

In addition, a syntactic part of DSL can also be separated into two levels: the level of objects and the level of functions. The object-level is equivalent to the set of objects of the metamodel. The functional level contains operations, which allow to specify the operational context for the objects.

As follows, the structure of the syntactic level can be formalized as a triple $(O_{syntax}, R_{syntax}, Rule_{syntax})$, where $O_{syntax} \subseteq E$ and $R_{syntax} \subseteq R$ are the subsets of objects and relations between them of the DSL metamodel respectively, and $Rule_{syntax}$ is a set of rules, describing reflection between metamodel and concrete syntax of DSL.

The most important thing here is, that such definition of DSL concrete syntax based on its metamodel doesn't depend on the way of type of DSL concrete syntax (e.g. textual or visual). For visual languages, it is necessary to establish links between these concepts and the visual symbols that represent them—as done, e.g., with GMF [20]. Similarly, with textual languages links are required between metamodel elements and the syntactic structures of the textual DSL. An example of this approach is TCS.

Under these circumstances, we can tell about the complete model-oriented representation of the DSL syntax structure. The structure allows not only to describe both levels of DSL syntax in structured and unified manner but optimize the process of DSL development and further development by introducing several syntactic DSL dialects on one fixed metamodel. Furthermore, the versification of DSL can be provided in a similar way on the meta-level as well as on the concrete-syntactic level, without need to re-create the whole DSL structure every time, when the changes are required. It's important, since a DSL can have several concrete syntaxes.

3 Proposed Approach

3.1 A Semantic Hierarchy of Model-Oriented DSL Development

Combining the object-oriented model of the DSL structure with the formal definition of DSM on the basis of a single meta-meta model, we can specialize a well-known semantic hierarchy of meta-models for our approach to model-oriented development and evolution of DSL (Fig. 1).

In our case this hierarchy is separated into four layers, according to the stages of the DSL development. Each lower level is based on the model artefacts of the upper level.

A single M3 meta-meta-model determines common grounds for all meta- and models of the lower levels. This meta-level defines also notations in which concrete models will be defined and what rules for their transformations will be used.

The structure of the semantic hierarchy determines the corresponding process of DSL creation. It starts with the definition of DSM, containing all important entities of the target domain and relationships between them. The process of DSM creation is beyond the scope of our current research, we propose its consistency for DSL development. For more details on DSM definition and checking its correctness, see [16].

When DSM is created, we can build the DSL semantic model by the operation of semantic projection. Any semantic projection performs a certain M2M transformation of DSM to some its fragment. Thus, semantic projection fully determines the semantic model of a particular dialect of DSL. In this case the semantic model

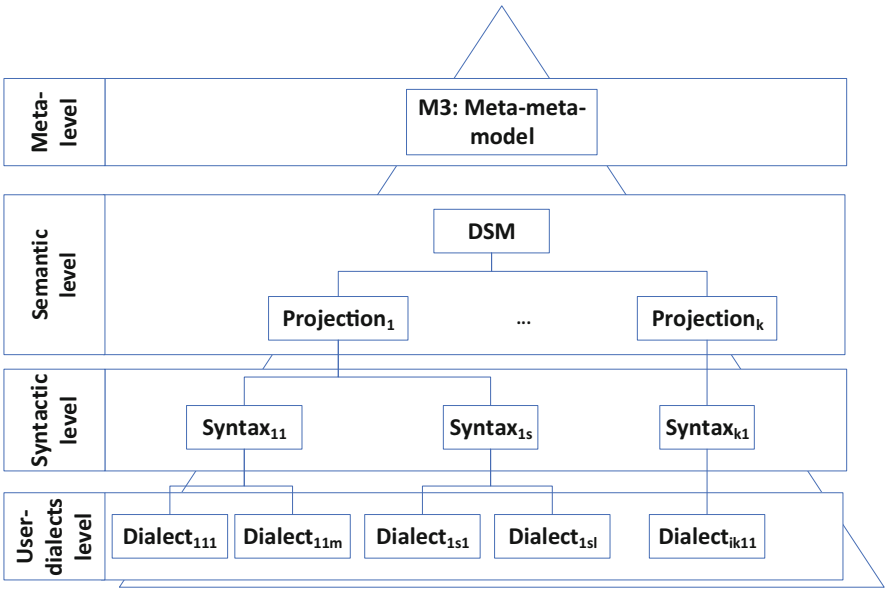

Fig. 1 The semantic hierarchy of projection-based DSL development

becomes an object-temporal structure, because it should be adopted according changes in DSM over the time, thereby defining a new object filling of the DSM.

After the semantic projection was performed, the syntactic level of DSL can be developed by a M2M transformation of the result of the corresponding projection. What is important, these DSL syntactic models are independent of each other and are determined by end-users in accordance with the adaptation of the semantic projection to their own tasks.

Finally, created syntaxes are used by the end-users of DSL, who determine the set of DSL dialects within the single specific syntactic model.

For comparison, traditional approaches start with the manual definition of the DSL concrete syntax which is followed by the translation of the syntax in terms of grammars. Consequently, every change in the target domain leads to the need to redefine the DSL concrete syntax and re-create the corresponding grammar. A similar process repeats in a case, when changes in DSL are caused by the end-users. As a result, outcomes of traditional approaches contain inconsistent dialects of DSL, which cannot be mapped among themselves due to differences in all levels of the DSL structure.

Using the idea, that any model can be represented in graph-oriented form, where an entity is a vertex and the relations between entities are edges between vertexes, in what follows we define the projections mechanism (applied with M2M transformations) in terms of graph transformations rules.

3.2 Defining M2M Transformations with Graph Transformation System

For our purpose, we proposed to use a combination of metamodeling and graph transformation techniques: the *static structure* of a language is described by a corresponding *metamodel* clearly separating static and dynamic concepts of the language, while the *dynamic operational semantics* is specified by *graph transformation*.

Graph transformation provides a rule-based manipulation of graphs, which is conceptually similar to the well-known Chomsky grammar rules but using graph patterns instead of textual ones. Formally, a ***graph transformation rule*** is a triple $Rule = (Lhs, Neg, Rhs)$, where *Lhs* is the left-hand side graph, *Rhs* is the right-hand side graph, while *Neg* is (an optional) negative application condition. Informally, *Lhs* and *Neg* of a rule define the *precondition* while *Rhs* defines the *postcondition* for a rule application.

The ***application*** of a rule to a ***model (graph)*** *M* alters the model by replacing the pattern defined by *Lhs* with the pattern of the *Rhs*. This is performed by (1) *finding a match* of the *Lhs* pattern in model *M*; (2) *checking the negative application conditions Neg* which prohibits the presence of certain model elements; (3) *removing* a part of the model *M* that can be mapped to the *Lhs* pattern but not the *Rhs* pattern yielding an intermediate model *IM*; (4) *adding* new elements to the intermediate model *IM* which exist in the *Rhs* but cannot be mapped to the *Lhs* yielding the derived model *M*.

In this case, graph transformation rules serve as elementary operations while the entire operational semantics of a language or a model transformation is defined by a model transformation system.

A directed unattributed graph $G = (G_V, G_E, src, tar)$ consists of a set of vertices G_V, a set of edges G_E, a mapping $src : G_E \rightarrow G_V$ assigning to each edge a start vertex, and a mapping $tar : G_V \rightarrow G_E$ assigning to each edge a target vertex. A signature $\Sigma = \langle S, OP \rangle$ consists of a set of sort symbols S and a set of operation symbols OP. A Σ-algebra A is an S-indexed family $(A_s)_{s \in S}$ of carrier sets together with an OP-indexed family of mappings $(op^A)_{op \in OP}$ that contains for each $op : s_1 \ldots s_n \mapsto s$ a mapping $op^A : A_{s_1} \ldots A_{s_n} \mapsto A_s$. We denote by $|A|$ the disjoint union of the carrier sets A_s of A, for all $s \in S$, which is usually infinite.

An attributed graph AG where only graph vertices can be attributed is a pair consisting of a directed unlabeled graph G and a Σ-algebra A such that $|A| \subseteq G_V$. The elements of $|A|$ represent potential attribute values which are regarded as special data vertices of the graph (besides the object vertices that model structural entities). An object vertex $v \in G_V$ has an attribute value $a \in |A|$ if there is an edge from v to a in AG.

An attributed type graph ATG is an attributed graph where A is the final Σ-algebra having $A_s = \{s\}$ for all $s \in S$. An attributed instance graph is an attributed graph with an additional typing morphism which specifies the type of all vertices of the graph.

A *graph morphism* $f: G \to H$ is a pair of functions $(f_V: G_V \to H_V, f_E: G_E \to H_E)$ compatible with the graph structure, preserving sources and targets: $f_V \circ s_G = s_H \circ f_E$ and $f_V \circ t_G = t_H \circ f_E$.

An *attributed graph morphism* $f: \langle G_1, A_1 \rangle \to \langle G_2, A_2 \rangle$ is a pair of a Σ-homomorphism $f_A = (f_s)_{s \in S}: A_1 \to A_2$ and a graph homomorphism $f_G = \langle f_V, f_E \rangle: G_1 \to G_2$ such that $|f_A| \subseteq f_V$, where $|f_A| = \bigcup_{s \in S} f_s$ and $A_{1s} = f_V^{-1}(A_{2s})$ for all $s \in S$. Informally, an attributed graph morphism preserves the graph structure of the attributed graphs.

Generalizing these concepts, we can define the typed attributed graph transformation system $GTS = (\Sigma, ATG, X, \mathcal{R})$, which consists of a data type signature Σ, an attributed type graph ATG, a family of variables X over Σ, and a set of attributed graph transformation rules $\mathcal{R}$ over ATG and X. The rules induce a relation $\Rightarrow$ on the set of graphs. One writes $G \overset{r(o)}{\Rightarrow} H$ to denote that graph H is derived from graph G by applying the rule $r \in R$ at occurrence o. A transformation sequence $G_0 \overset{*}{\Rightarrow} G_n = G_0 \overset{r_1(o_1)}{\Rightarrow} \ldots \overset{r_n(o_n)}{\Rightarrow} G_n$ in GTS is a sequence of consecutive transformation steps such that all rules r_i are from $\mathcal{R}$.

From this point of view, we can tell, that in our case we can define the projections between DSM and DSL metamodel and DSL metamodel and DSL concrete syntax as morphisms, that results in the opportunity to define similar transformation rules for all these projections. Such unification of transformations during DSL development leads to the idea of existence of invariants in DSL dialects structure.

In this case we understand the invariant not only as a stable set of entities (graph vertices) that does not change over time. We also can argue, that we have operational invariants—a set of graph transformation rules, which are characterized by the same set of constraints and can be applied equally on structurally identical components. Such invariants allow us not only to unify and simplify the process of DSL development, but also to organize the verification of the consistency of different DSL dialects with the mechanisms of invariants.

4 Conclusion and Future Development

In our research we explored a model-oriented and projection-based approach for DSL development. Proposed approach is based on the idea, that every DSL contains two parts: semantic and the syntactic. Both can be represented as a set of interconnected objects, characterized by specific attributes. Such object-oriented description of DSL levels results in the opportunity to define the DSL as a model-oriented structure, where a certain entity is assigned to each element.

What is the most important, the DSL structure is created automatically from the specific DSM using the so-called semantic projection mechanism. The semantic projection is an operation, which is conducted over the DSM and the result of which

is also a semantic model that is obtained by transforming the original DSM. The result of projection describes the DSL semantic model.

In comparison with the existing approaches to DSL development, which use a traditional cycle of DSL development, starting from the definition of the DSL concrete syntax, our approach starts with the generation of DSM, which is a dynamic, time-varying structure. Under these circumstances, the DSL semantic model can be obtained as a projection of such DSM through M2M transformations. Furthermore, the DSL syntactic model is also the result of the projection of the DSL semantic model onto users' requirements and needs.

As a result, the proposed DSL development process is conducted in full accordance with the conceptual scheme of the target domain, thereby ensuring the participation of end-users in the process of its creation. In addition, the projection-based principle of the DSL development allows the users to achieve the resilience of the DSL created both with the target domain (represented by DSM) and users' requirements. Created DSL can be transformed on the semantic and syntactic levels separately, using M2M transformations for projections realizations. At the same time, the consistency of the created DSL dialects is preserved.

Among advantages of the approach proposed its reusability and end-user orientation should be mentioned. The approach can be transferred to any domain for which the DSM is defined. The model-oriented structure of DSL is also an understandable and convenient for end-user. This format of DSL representation results in the opportunity to make changes to DSL without special programming skills.

Planning further research, the definition of projections using invariants can be provided. Such invariants mechanism will allow to automate the comparison and matching different DSL dialects, using once defined invariant transformation rule to different DSL syntaxes.

References

1. Martin, F. (2010). *Domain specific languages*. Upper Saddle River, NJ: Addison Wesley.
2. Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain specific languages. *ACM Computing Surveys (CSUR), 37*(4), 316–344.
3. Cleenewerck, T., Czarnecki, K., Striegnitz, J., & Volter, M. (2004). Report from the ECOOP 2004 workshop on evolution and reuse of language specifications for DSLs (ERLS). In *Object-oriented technology. ECOOP 2004 workshop reader* (pp. 187–201). Berlin: Springer.
4. Cleenewerck, T. (2003). Component-based DSL development. In *Software language engineering* (pp. 245–264). Heidelberg: Springer.
5. Gómez-Abajo, P., Guerra, E., & De Lara, E. (2016). A domain-specific language for model mutation and its application to the automated generation of exercises. *Computer Languages, Systems and Structures, 49*, 152–173.
6. Popovic, A., Lukovic, I., Dimitrieski, V., & Djuki, V. (2015). A DSL for modeling application-specific functionalities of business applications. *Computer Languages, Systems and Structures, 43*, 69–95.

7. Challenger, M., Demirkol, S., Getir, S., Mernik, M., Kardas, G., & Kosar, T. (2014). On the use of a domain-specific modeling language in the development of multiagent systems. *Engineering Applications of Artificial Intelligence, 28*, 111–141.
8. Laird, P., & Barrett, S. (2010). Towards dynamic evolution of domain specific languages. *Software Language Engineering, LNCS 5969*, 144–153.
9. Pereira, M., Fonseca, J., & Henriques, P. (2016). Ontological approach for DSL development. *Computer Languages, Systems and Structures, 45*, 35–52.
10. Bell, P. (2007). Automated transformation of statements within evolving domain specific languages. Computer science and information system reports. In T. Cleenewerck (Ed.), *Component-based DSL development* (pp. 172–177).
11. Mengerink, J. G. M., Serebrenik, A., Schiffelers, R. R. H., & van den Brand, M. G. J. (2016). A complete operator library for DSL evolution specification. In *MDSE 32nd International Conference on Software Maintenance and Evolution Proceedings* (pp. 144–154).
12. Mengerink, J. G. M., Serebrenik, A., van den Brand, M. G. J., & Schiffelers, R. R. H. (2016). Udapt edapt extensions for industrial application. In *ITSLE 2016 Industry Track for Software Language Engineering Proceedings* (pp. 21–22).
13. Sprinkle, J. (2004). A domain-specific visual language for domain model evolution. *Journal of Visual Languages and Computing, 15*, 291–307.
14. Kosar, T., Bohra, B., & Mernik, M. (2016). Domain-specific languages: A systematic mapping study. *Information and Software Technology, 71*, 77–90.
15. Terence, P. (2012). *Language implementation patterns: Create your own domain-specific and general programming languages*. Pragmatic Bookshelf.
16. Guizzardi, G. (2005). *Ontological foundations for structural conceptual models* (Telematica Instituut Fundamental Research Series: Vol 15). Enschede: Centre for Telematics and Information Technology.
17. Guizzardi, G. (2013). Ontology-based evaluation and design of visual conceptual modeling languages. In *Domain engineering* (pp. 317–347). Berlin: Springer.
18. Agrawal, A., Karsai, G., & Shi, F. (2003). Graph transformations on domain-specific models. *International Journal on Software and Systems Modeling, 37*, 1–43.
19. Ruffolo, M., Sidhu, I., & Guadagno, L. (2007). Semantic enterprise technologies. In *Proceedings of the First International Conference on Industrial Results of Semantic Technologies: Vol. 293* (pp. 70–84).
20. Elipse Graphical Modeling Project (GMP). Accessed December 26, 2019, form http://www.eclipse.org/modeling/gmp/