Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 52-13 (2019) 1138–1144

Ontology-based reconfigurable DSL for planning technical services

Boris Ulitin* Eduard Babkin**

**National Research University Higher School of Economics, Nizhny Novgorod, Russia (e-mail: bulitin@hse.ru)*

***National Research University Higher School of Economics, Nizhny Novgorod, Russia (e-mail: eababkin@hse.ru)*

Abstract: The article is related to the problem of decision support in dynamic business contexts where multiple facts of domain knowledge (conditions, values and goals) frequently change over time, and users should participate continuously in the problem definition. In our research we explore an opportunity to use reconfigurable domain specific languages (DSL) as the most convenient and user-oriented way to implementation of decision support systems in the context of modern digital enterprises. What is the most important, the DSL proposed can be evolved in real time using the framework according to the changes in the target domain and competences of end-users. Applicability of the framework is demonstrated using a real-life example of resource allocation process in the railway transportation.

Keywords: enterprise knowledge management, domain-specific language, planning technical service, resource allocation problem, ontology, evolution.

1. INTRODUCTION

Modern trends in industry, services and logistics leverage importance of formal methods for representation of the relevant knowledge and contextual constraints within organizational frameworks of continuous improvement of operational processes and business models (Lucassen et al. (2017)).

A number of research results demonstrate suitability of using Domain-Specific Languages (DSL) when defining the context for different knowledge modeling and management tasks of modern companies. For example, Sprinkle (2016) describes the implementation of DSL for modelling logistic interactions within the organisation. Pereira et al. (2016) prove the effectiveness of DSL usage for the definition of the context of the resource allocation problem (in terms of materials and even human resources). However, in both works authors focus on the case, when DSL is considered unchangeable and requires a stable context of the problem being solved. To overcome such a drawback Hodgson (2012), Cleenewerck et al. (2004) offer to separate the model, which is responsible for the solving procedure, and a mechanism, which allows one controlling this model (Wang et al. (2017)). As a result, the original DSL becomes the dynamic structure, which can be adopted in accordance with changes both in the domain and in the needs of users.

Cited publications lead to a more generic insight about DSL, as a bearer of relevant knowledge and the driver of improvements and digitalization of business models. To play such a role DSL should be accompanied by specific methods of its evolution and reconfiguration.

What is more important, every DSL uses as its basis some model of the current subject area (according to Fowler (2010)). 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. This is especially useful, considering that the DSL structure contains semantic and syntactic levels. The syntactic level is responsible for the opportunity to define some context, while the semantic level reflects this context on the concepts of the target domain. As a result, the semantic level absolutely depends on the model of the target domain, used during DSL creation.

At the same time, the syntactic level of DSL is a result of combination of the general domain concepts with the problem-specific tasks of the users. This means, the syntactic part of DSL is a result of adaptation of the DSL semantic model to information and operational needs, experience and skills of every specific user. 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.

All this mean, that the approach, allowing not only to define the tasks in terms of DSL, but also to modify DSL itself should be developed. However known research achievements do not offer complete solutions. For example, Cleenewerck et al. (2004) try to identify the tree of DSL dialects using graph-transformations for transition between different DSL dialects. Another model-oriented approach is described by Laird et al. (2010), but they concentrate only on the opportunity to transform the DSL semantic level into the syntactic level, without further evolution of them. Our new contributions provide a reusable approach to continuous reconfiguration and adaptation of DSL in respond to changing conditions. Although models and algorithms in our approach can be applied to different domains, we demonstrated core features of our approach for the case of the railway real-time

10.1016/j.ifacol.2019.11.349

allocation problem, because this problem has a great impact on knowledge-based digitalization of railway companies.

We present our results as follows. In Section 2 and 3 we give some facts from the theory of design of domain-specific languages, define the DSL structure in a model-oriented manner and analyze various types of its evolution. Section 3 offers main components of the developed framework, its implementation is represented and evaluated in Sections 4 and 5 correspondingly. We conclude the article with analysis of the results and specification of the future research directions.

2. DSL EVOLUTION AS MODEL TRANSFORMATION

Since it was mentioned above, DSLs represent really efficient and convenient opportunity to model and manage different aspects of all spheres of modern companies. Such effectiveness can be explained because of the strong relationship between DSL and knowledge of domain users used in the process of designing DSL.

However, as any dynamic complex structure, any domain demonstrates the tendency to the evolution over time: new concepts may arise, while others unite into more general ones or become obsolete. In accordance with these changes, DSL should also support the possibility of evolution.

The simplest option in this case is to rebuild the DSL whenever the domain model changes. But this process has several disadvantages. First of all, the process of DSL development is really time-consuming, since DSL contains internal and external parts, connected with the domain model and DSL syntax correspondingly. Secondly, DSL development, since DSL is a language, is often associated with the use of grammar tools that require special skills from developers. Finally, while a new version of DSL is being created, the domain can be changed again. Thus, the DSL changes may not be synchronized with the domain changes, making DSL not fully compliant with the needs of the end users.

Taking into account all these arguments, it is much more effective to be able to implement changes in DSL in parallel with changes in the domain, carrying out their co-evolution. This approach is also more convenient from the point of view that both, as DSL and the domain, can be represented in the model-oriented approach. Consequently, for the organization of their co-evolution, methods of model transformations can be used.

For example, Kesentini et al. (2019) describes the opportunity to automate the transformation between the metamodel on the domain and specific models, based on this metamodel. The main idea of automation in this case is that the specific model contains terms described at the metamodel level. Thus, they can be obtained by straightforward transformations. The only limitation in this case is the need to use a single notation to represent the structure of the metamodel and models. Such approach can be used on the stage of designing the DSL meta-model, using the domain model as its basis.

On the other hand, this approach cannot be applied at the stage of development a specific DSL syntax, since it does not support defining restrictions and syntactic constructions associated with them. To eliminate this omission, the approach described by Khelladi et al. (2017) can be used. In this work authors represent the opportunity to transfer grammatically correct terms with OCL constraints between two (meta)models, using Automated Transformation Language (ATL). Unfortunately, such transformations are unidirectional due to the specificity of ATL.

In what follows we combine and extend all these approaches, trying to develop a really reconfigurable framework, allowing designers not only to define concrete scenarios for the specific domain, but also to change DSL in real time, using ideas of DSL model-oriented specification.

3. BACKGROUND

3.1 Definition of Domain Semantic Model (DSM)

According to Parr (2012) 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 (obtained, for example, by translating existing ontologies). That gives respectively weak or rich and detailed representation of a domain (Bell (2007)). More formally DSM is a seven-tuple of the form:

$$DSM = (H_C, H_R, O, R, A, M, D) \quad (1)$$

- H_C and H_R are sets of classes and relations schemas, 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 enabling the recognition of class (concept) instances contained in O, so their annotation, extraction and storing is possible.

The DSM above can be separated into three interconnected layers: the meta-layer (H_C and H_R), which identifies the way to define the DSL itself; the object-layer (O and R), which represent the main concepts in the target domain and relations between them and the logical layer (A, D and M), which postulates the rules, existing in the target domain and different restrictions on them. In practice, all these layers are important and have to be considered during the effective DSL development. However, based on the DSL definition, we are interested mostly in the object-layer of DSL, which consists of the objects and relations between them, which form the basis of the future DSL. In what follows, we will be focused on this object-layer of DSM.

It is also important to note, that DSM usually has a dynamic structure, demonstrates a tendency to changes over the time (evolution, in other words). According to Laird et al. (2010) 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.

3.2 Model-oriented Definition of DSL

A domain-specific language (DSL) is a computer language specialized to a particular application domain. This is in contrast to a general-purpose language, which is broadly applicable across domains, and lacks specialized features for a particular domain (Fowler (2010)). Laird et al. (2010) identified two parts of the DSL: (1) a syntactic part, which defines the constructions of DSL; and (2) a semantic part, which manifests itself in the semantic model. The first part allows users defining the context for working with the second one, which defines meaning of DSL commands in terms of the target domain.

From the formal point of view, a semantic part of DSL is derived from the set of objects of the target domain and operations on them. The structure of every model is a combination *(E,R)* of some entities and relations between them, where each entity is a set of its attributes

$$e_i = \{attr_{i_1}, attr_{i_2}, ..., attr_{i_M}\}, M \in \aleph, i = 1, N \quad (2)$$

By applying the object-oriented approach to this definition, we can formalize the semantic DSL level in a model-oriented manner as a combination *(O,R)* of some objects of the target domain and relations between them, where each object is a set of its attributes and operations

$$\begin{aligned} &o_i = (Attr_i, Opp_i) = \\ &(\{attr_{i_1}, attr_{i_2}, ..., attr_{i_M}\}, \{opp_{i_1}, opp_{i_2}, ..., opp_{i_K}\}), \\ &M, K \in \aleph, i = 1, N \end{aligned} \quad (3)$$

In these terms, the syntactic part of DSL can be represented as a subset of the semantic level, needed for representation of a certain problem situation. The very one difference is, that the syntactic part may not absolutely reflect the semantic constructions but identify its own definitions (pseudonyms) for the semantic constructions, according to the user's needs.

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 semantic level. The functional level contains operations, which allow to specify the operational context for the objects.

As follows, the structure of the syntactic DSL level can be formalized as a triple (O_{syntax}, R_{syntax}, $Alias_{syntax}$), where $O_{syntax} \subseteq O$ and $R_{syntax} \subseteq R$ are the subsets of objects and relations between them of the semantic DSL level respectively, and $Alias_{syntax}$ is a set of pseudonyms for objects' components (attributes and operations), similar to the set of descriptors D in the DSM.

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

3.3 Evolution of DSL

As already mentioned above, DSL, which is created for working with some specific domain, contains inside some model of this domain. However, the subject area can evolve. In this connection, the models created within the given domain should be changed accordingly. On the other hand, DSL structure can be defined in model-oriented manner, that lead us to the idea, that instead of the concept of co-evolution of DSM and DSL, we can consider a single concept of the model evolution.

As it was mentioned before, from the formal point of view, the structure of every model is a combination *(E,R)* of some entities in the certain domain and relations between them correspondingly. In these terms, evolution of the model is a process of changes in the structure of this model. However, the structure includes not only entities, used in some specific model, but and relations between them.

The very problem in these circumstances is to prove, that one model is the result of evolution of another. In simple way, it means, that one model can be derived during the transformation of another, that means, they contain a set of the same entities and/or relations between them, which are identified as invariants for these models. But before the definition of the model evolution in terms of invariants, the classification of it should be stated.

According to Cleenewerck et al. (2004), model evolution can be separated into two classes: vertical evolution and horizontal evolution.

Vertical evolution means the change in level of conceptualization (perspective) of the model. In this case of evolution new entities are added into the model (with(out) changes in the set of relations). It means, that vertical evolution can be formalized by the following manner: (E^1,R^1) is a result of vertical evolution of the model *(E,R)* if $|E| \neq |E^1|$ and $|R| \neq |R^1|$. In terms of models it means, that new entities can be added or some previously created entities can be deleted, with corresponding changes in the set of relations.

In terms of DSLs it means, that semantic layer will be changed: some objects will be introduced, some will be removed from the language – alongside with the corresponding changes on the syntactic level.

Horizontal evolution means preserving the level of conceptualization, but changing the sets of attributes for some entities, or changing the set of relations. It means, that vertical evolution can be formalized as follows: (E^1,R^1) is a result of horizontal evolution of the model (E,R) if $|E| = |E^1|$ and $\exists e_i \in E$, $e^1_i \in E^1$: $Attr_i \cap Attr^1_i = Attr_i \wedge |Attr_i| \neq |Attr^1_i|$ and $\exists r_i \in R$, $r_j \in R^1$: $r_i \notin R^1 \vee r_j \notin R$. In terms of models it means, that the set of objects will be the same, but some of them can be specified by adding new (or deleting old) attributes. In addition, the set of relations can be changed. In terms of DSL it means, that the semantic level will be changed only in terms of objects attributes with corresponding changes on the syntactic level or in terms of connections between them (in this case changes on the syntactic level depend on the nature of changed connections).

When we described two types of the DSL evolution in terms of the structure changes, we can formulate them using the mechanisms of invariants.

There are two types of invariants: object invariants and functional invariants. Object invariants are defined over two sets of objects.

Definition 1. A set of objects O is called object invariant for sets of objects A and reflection $g:A\rightarrow A$ iff $A \cap g(A) = O$.

In other words, object invariant consists of objects, which are represented in both sets of objects.

Definition 2. Allow A, B – sets of objects, G – a set of reflections $A\rightarrow A$. Reflection $f:A\rightarrow B$ is called functional invariant for G iff $\forall a\in A$ and $g\in G$: $f(a) = f(g(a))$.

This definition means, that functional invariant is a reflection, which allows one to check, which objects from sets A and B (elements of which are the result of reflection G) make up the object invariant.

Provided, we tell about vertical model evolution, the set G consists of the *equivalent* reflection. In this case, the invariant reflection *f* is the *rename* operation, which allows one to change the name of some entity without changes in its structure. It's reasonable since in case of vertical evolution entities can't be changes structurally, only new entities can be added, or some existing dropped.

Another situation arises in the case of horizontal evolution, when changes go to the structure level of entities. It means, that we should describe invariants on the level of attributes instead of entities. As a result, the set of reflections G consists of the operation of *getAttribute*, which derives set of attributes from the concrete entity. And at this level, the invariant *f* is the *rename* operation, which enables changes in the name of some attributes.

From these follows, that in order to identify the object invariant in the case of DSL structure evolution, we should find objects, which were changed only in the part of its names, without changes in its structure, and objects, attributes of which were also renamed only. All other objects can be formulated as object invariants, because need more complicated transformations, than equivalent transformation.

4. PROPOSED FRAMEWORK

In our case, we separated the structure of the framework (combination of the programming interfaces with a set of incorporated algorithms and mechanisms of their realization) into two sections: the first one is responsible for processing scripts in terms of DSL. The second one is responsible for DSL transformation. Such separation allows us to design the DSL evolution in parallel to its verification: previously created scenarios can be adopted to new DSL syntax automatically in real time. The first section was described in details in Ulitin et al. (2016), thus in this article we focus mostly on the second one.

In fig. 1 main steps of DSL scenario processing are described. From this scheme we can see, that first of all every DSL scenario is parsed in order to separate grammatically correct scenario terms and concrete data from this scenario. After the correct syntactic terms were identified, we also parse them, comparing with the structure of the objects of the DSL semantic level. This stage allows us to identify, what objects have to be created for successful processing of the scenario and what data have to be fixed in them. Such separation of syntactic and semantic parsing allows us not only to provide them independently and optimize the scenario processing, but and organize effective evolution of DSL on both levels separately.

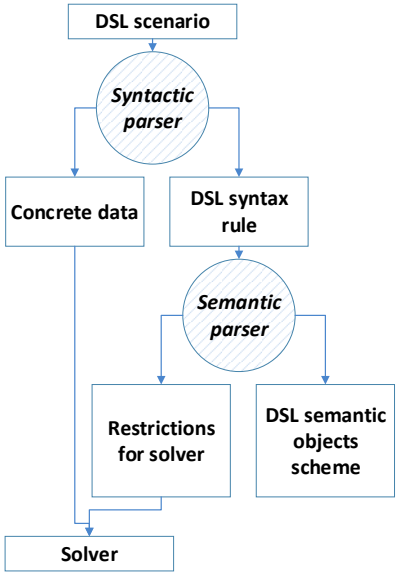

Fig. 1. Scheme of DSL scenario processing.

When we want to evolve DSL, we should follow the stages, described in fig. 2. According to this scheme, we can provide evolution of DSL both from the syntax to semantic and vice versa. For both vectors of evolution model-to-model (M2M) transformations are used. In more details these rules and their

implementation for different types of DSL evolution are described in Ulitin et al. (2017).

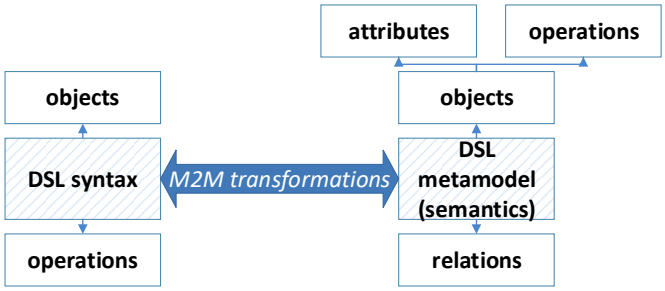

Fig. 2. Scheme of DSL evolution implementation.

However, what is the most important, such organization of DSL evolution allows us to unify this process, since objects and operations are represented on both levels of DSL and can be transformed between themselves. In order to organize such transformations, the mechanisms of invariants are implemented: the evolution of DSL is organized using M2M transformations, where two models assume the preservation of both object and functional invariants. It results in the opportunity to organize the interconnected transformations between these models and guarantees their consistency at the structural level.

Based on these assumptions, we can formulate the following algorithm for DSL evolution:

1. Identify the original state of the DSL model M1.
2. Identify the final state of the DSL model M2.
3. Identify the object invariants between M1 and M2 in order to guarantee their consistency.
4. Apply the equivalent transformations to object invariants.
5. Describe the transformations for objects and functions, not presented in the object invariants.

This algorithm provides the opportunity to organize any type of DSL evolution, since it was shown, that any DSL evolution contains invariants at different levels of the structure of the model that describes the corresponding level of DSL.

5. DEMONSTRATION IN A PRACTICAL CASE

5.1 DSM and DSL for railway allocation domain

The domain of railways services represents an interesting and significant case of dynamic management of knowledge and enterprise digitalization. In particular the context of the railway allocation problem can change frequently because of arrivals of new trains, or changing the priority of existing services. As a result, we have to have a clear and simple way to adapt new changes in terms of the proposed framework, responsible for finding the optimal resource allocation. In the process of DSL design for the railway allocation process, it's vital to identify all the types of resources in this domain.

There are three general resources for any railway station, each with specific attributes: railways, trains and service brigades. All of them are represented in the DSM for the corresponding domain, which is more complete in comparison with that considered in our previous work (Ulitin et al. (2017)), since it contains the specification of the requirements (Skills) both for the Services and for the Brigades providing them.

After the DSM created, we can identify the semantic level of DSL, describing the DSL metamodel. For this purpose, M2M transformation rules can be used, as it was described in Ulitin et al. (2018). This is reasonable since both DSM and DSL meta-model are described in a model-oriented way. In addition, M2M transformations are independent from the notation of model definition, that allows us to describe DSM and DSL meta-model independently, in the most appropriate way. As a result, we will have the complete DSL metamodel, which can be used during the following DSL syntax definition. This definition includes two parts: definition of objects for DSL syntax, which are the equivalents to the objects, described on the semantic level of DSL, and grammar, describing the operations and correct terms for the future DSL syntax. In our case, we used the Backus-Naur form of grammar definition, because this form allows us to identify rules, based on the previously created objects, and automatically convert the resulting rules into an abstract, language-independent form.

As a result, the created DSL semantic and syntactic levels are wholly coherent and can be evolved using transformations in real time. In addition, such changes are provided separately, since the invariants on both levels are identified.

5.2 Subject-oriented GUI

As it was mentioned before, the interface created contains two parts: the first one, responsible for the DSL scenario definition and processing, and another one, needed for evolution of DSL. Both parts are represented in fig. 3, 5 correspondingly.

The first component of the interface (fig. 3) contains only a visual panel, representing all the DSL components needed for definition of DSL scenarios. Lines identify railways; rectangles indicate arriving trains and their cars (different colours reflect current status of the corresponding car: red, yellow and grey correspond to unattended, serviced, free); blue rectangles with "+/-" buttons allow a user to increase or decrease number of brigades for the train. Text boxes contain additional details about railways, trains and other resources of the terminal. In more details this part of the interface is described in Ulitin et al. (2016).

The most important new thing here is, that this part of DSL is automatically adopted when the DSL evolution is provided. Such an automation allows us to support DSL evolution by end-users without the need to re-compile the whole framework and to have special programming skills.

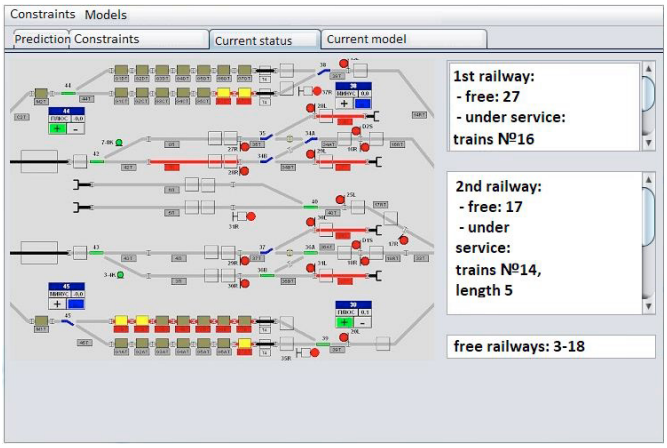

Fig. 3. GUI components for DSL scenario definition.

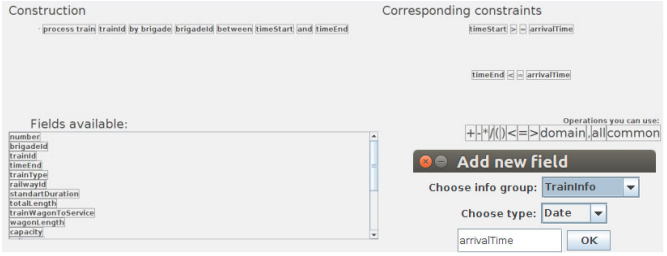

Fig. 4. Evolution of DSL implementation.

In order to design such evolution, the second part of the interface is used (fig. 4). This part contains three main components: the component to define/change a new/existing command of DSL, the component for definition of constraints, connected with the command and the component for definition of syntactic terms, related to the new command. All these components are identified in accordance with the structure of DSL: objects, which contain attributes and operations and relations between them. As a result, the created interface allows us to define the whole DSL structure and change it in real time without need to re-create the DSL manually.

6. EVALUATION

In this article we concentrated our attention to the implementation of DSL evolution using a specially designed subject-oriented interface. According to the idea, that there are two types of DSL evolution, two corresponding scenarios will be described.

The first scenario realizes the horizontal evolution of DSL – we want to extend syntactic part of DSL by adding new command: *process train trainId by brigade brigadeId from timeStart till timeEnd*. This command uses existing objects for the DSL semantic level: a train and a brigade, but implements a new syntactic term. In order to implement this command, the second part of GUI is used. First of all, the user should define a needed command, using the block of available fields of DSL objects. As a result, the following construction is defined (fig. 4).

Fig. 5. Scenario with added command.

After the term definition, constraints, related to this, are defined (fig. 4). Finally, the created term is compiled and added to the DSL, ready to use.

Using this interface, other sceneries of DSL evolution can also be implemented, for example, we can add a new attribute *arrivalTime* to the *TrainInfo* object. For this purpose, the corresponding interface component can be used (fig. 4).

What is the most important, in this case, we only define new commands, without need to re-create the DSL structure and can use them in sceneries in real time. For example, the result of added command is represented in fig. 5. As follows, the approach proposed allows us to implement all types of DSL evolution in real time, correctly transforming new commands into DSL syntactic and semantic objects and terms.

Currently existing approaches, allowing also to allocate resources of railway station, are targeted to one concrete type of resources (for example, to brigades by Wang et al. (2017), or to trains by Chen et al. (2018)). Furthermore, such approaches use static models of resource allocation and cannot be adopted according to new types of resources or solving models in real time. In comparison to existing approaches, the approach proposed is independent from the nature of the resources and can be adopted to any other domain.

The only limitation for our approach is the fact, that the framework represented can provide the opportunity to define only unidirectional transformation of DSL, according to changes in the domain model. This limitation can be explained by the fact, that languages of model transformations do not support bidirectional transformations, because symmetric transformation means using the opposite to the original operation (delete instead of add, etc.). However, such limitation can be resolved using the idea of closure operations necessary for organizing the DSL evolution (Ulitin et al. (2017), Demuth et al. (2016)).

7. CONCLUSION

In our research we explored an opportunity to use DSL as an effective way for definition of the knowledge context of tasks in the specific domain. The interface developed not only allows to define the scenarios using DSL, but also permits modifications of this DSL in accordance with changes in the domain and competences of the end-users. As a result, DSL becomes really user-oriented, because takes into account the relevant knowledge and contextual constraints within organizational frameworks.

We may note, that in our approach the DSL structure is created automatically from the specific DSM. After the DSL structure is created, DSL can be evolved on both levels semantic and syntax in real time without the need to re-create the whole DSL manually. 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.

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. The defining a specific syntax for different versions of DSL is maximally automated and is carried out at the level of models, rather than in the program code. Such automation significantly reduces the time required to complete the DSL development cycle, that was demonstrated by the examples.

In comparison to existing solutions (like Bell (2007) or Laird et al. (2010)), our approach allows users not only to specify the concrete DSL terms without opportunity to redefine them automatically in real time, but also to edit existing DSM and the whole DSL structure without recreating them according to changes both in the domain and competences of end-users. Furthermore, in contrast to using the graph-transformations for conversions between DSM (the ontology) and DSL, our proposed solution is not based on some specific DSL, but can be applied for any of them.

All these advantages were demonstrated by the real example of DSL evolution in railway domain. Planning further research, we consider to implement whole invariants mechanisms for realization of interconnected changes in DSL and DSM instead of current one-directional from DSM to DSL. For achieving this goal an approach, described by Bell (2007), can be used. Bell (2007) tries to implement the coherent changes in several DSL meta-models, using the automated transformations, that can allow to organize the consistent changes in the semantic and syntactic DSL levels models within our approach.

The approach proposed will facilitate adopting various semantic technology applications to the current state of customer requirements and their experience. Furthermore, the mechanism of invariants will allow users to apply contextual constraints with saving sustainable digital transformation of organizational structures and processes of modern companies.

REFERENCES

Bell, P. (2007). Automated Transformation of Statements within Evolving Domain Specific Languages. *Computer Science and Information System Reports,* pp. 172–177.

Chen, W., and Dong, M. (2018). Optimal resource allocation across related channels. *Operations Research Letters,* pp. 397-401.

Cleenewerck, T., Czarnecki, K., Striegnitz, J., and Volter, M. (2004). Report from the ECOOP 2004 Workshop on Evolution and Reuse of Language Specifications for DSLs (ERLS). *Object-Oriented Technology. ECOOP 2004 Workshop Reader,* pp. 187–201.

Demuth, A., Riedl-Ehrenleitner, M., Lopez-Herrejon, R.E., and Egyed, A. (2016). Co-evolution of metamodels and models through consistent change propagation. *Journal of Systems and Software,* pp. 281–297.

Fowler, M. (2017). *Domain Specific Languages*. Addison Wesley.

Hodgson, M. (2012). On the Limits of Rational Choice Theory. *Economic Thought,* pp. 94–108.

Kessentini, W., Sahraoui, H., and Wimmer, M. (2019). Automated metamodel/model co-evolution: A search-based approach. *Information and Software Technology,* pp. 49–67.

Khelladi, D.E., Bendraou, R., Hebig, R., and Gervais, M.-P. (2017). A semi-automatic maintenance and co-evolution of OCL constraints with (meta)model evolution. *Journal of Systems and Software,* pp. 242–260.

Laird, P., and Barrett, S. (2010). Towards Dynamic Evolution of Domain Specific Languages. *Software Language Engineering,* pp. 144–153.

Lucassen, G., Robeer, M. Dalpiaz, F., Werf G.M., and Brinkkemper, S. (2017). Extracting conceptual models from user stories with Visual Narrator. *Requirements Engineering, v. 22(3),* pp. 339–358.

Parr, T. (2012). *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*. Pragmatic Bookshelf.

Pereira, M., Fonseca, J., and Henriques, P. (2016). Ontological approach for DSL development. *Computer Languages, Systems&Structures,* pp. 35–52.

Sprinkle, J. (2016). A safe autonomous vehicle trajectory domain specific modelling language for non-expert development. *Proceedings of the International Workshop on Domain-Specific Modeling,* pp. 42-48.

Ulitin, B., Babkin, E., and Babkina T. (2016). Combination of DSL and DCSP for decision support in dynamic contexts. *Lecture Notes in Business Information Processing Issue 261: Perspectives in Business Informatics Research,* pp. 159–173.

Ulitin, B., and Babkin, E. (2017). Ontology and DSL co-evolution using graph transformations methods. *Lecture Notes in Business Information Processing Issue 295: Perspectives in Business Informatics Research,* pp. 233–247.

Ulitin, B., Babkin, E., and Babkina T. (2018). A Projection-Based Approach for Development of Domain-Specific Languages. *Lecture Notes in Business Information Processing Issue 330: Perspectives in Business Informatics Research,* pp. 219–234.

Wang, H., Wang, X., and Zhang X. (2017). Dynamic resource allocation for intermodal freight transportation with network effects: Approximations and algorithms. *Transportation Research Part B: Methodological,* pp. 83-112.