Ontology and DSL Co-evolution Using Graph Transformations Methods

Boris Ulitin(✉) and Eduard Babkin

National Research University Higher School of Economics,
Nizhny Novgorod, Russia
{bulitin, eababkin}@hse.ru

Abstract. The article is related to the problem of implementing consistent changes in the model of some subject area and in the syntax of domain specific language (DSL), dealing with problems in that domain. In our research, we explore an opportunity to provide the method of co-evolution of the ontology, used as a model of the subject area, and DSL. This method combines graph representation of the ontology and DSL with the set of rules, formulated in terms of an automated graph-transformation language. Applicability of the proposed approach is demonstrated using a real-life example of co-evolution of the ontology and DSL in the railway transportation domain.

Keywords: Domain-specific language · Ontology · Evolution · Graph transformation · Railway transportation

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.

The ontological approach can provide such a degree of accuracy in the description of the subject area, because an ontology reflects a set of concepts of the target domain and the connections between them [2]. Consequently, application of the ontology as a model of DSL guarantees that DSL is identical to the corresponding domain, thereby allows interacting with it more effectively.

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 ontological representation also occurs [3, 4]. However, DSL frequently remains unchanged, since it is built on a snapshot of the domain (and the ontology) 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 ontology. At worst, uncoordinated changes can lead to the situation, when DSL, being fixed in its

B. Johansson et al. (Eds.): BIR 2017, LNBIP 295, pp. 233–247, 2017.
DOI: 10.1007/978-3-319-64930-6_17

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 [15] Challenger, et al. 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 have to resolve them through the use of interconnected transformations.

In order to avoid such problems, automated methods of conflict detection that may arise in the course of the co-evolution of the domain ontology and DSL are needed. An important addition to such methods can be the ability to resolve emerging contradictions. We believe that a new approach is needed which combines the formal ontology mechanism with modern mathematical models of graph transformation [3, 5].

That ontology-based approach seems to be more effective in comparison with similar ones, for example, described by Cleenewerck or by Challenger. In [14] 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 in [15] 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.

In this article, we concentrate on the object level co-evolution and only to a certain extent on co-evolution in a functional sense, primarily because of the complexity of developing a universal description for the set of operations in any domain.

The article describes our proposed approach as follows. In Sect. 2 we give some facts from the theory of design of ontologies and domain-specific languages and some principles of processes of model evolution. Section 3 describes a universal graph-model for DSL and also contains the analysis of using graph-transformation techniques in presence of different types of evolution. Section 4 demonstrates advantages of our approach in the case of the railway transportation domain instead of applying graph transformations without establishing a correspondence between the ontology and DSL. We conclude the article with analysis of the results and specification of the future researches.

2 Background

2.1 Definition and Classification of Ontologies

Ontology is a representational artifact, comprising a taxonomy as proper part, whose representations are intended to designate some combination of universals, defined classes, and certain relations between them [2]. In accordance with formalization framework [6] the ontology can be naturally perceived as a graph (O, R), with a set of functions of constraints F.

On the other hand, the ontology is some kind of representation, created by the designer [7] which has a certain goal. As a result, in [2, 6] the following ontology kinds are detected: ***top-level ontologies***, ***domain (task) ontologies***, and ***application ontologies***, which describe concepts that depend both on a particular domain and a task, and often combine specializations of both the corresponding domain and task ontologies.

In the current research, we pay attention only to the Application ontologies, especially in the domain of railway processes – the railway resource allocation procedure. But as the Domain ontology contains only the necessary concepts of a subject area, applied ontology operates with a subset of these concepts necessary to achieve a certain goal. That allows us to consider the Application ontology as the reduced Domain ontology.

2.2 Definition and Structure 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 [1]. In [8] two parts of the DSL are identified: (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 defining the context for working with the second one, which defines meaning of DSL commands in terms of the target domain.

In addition, a syntactic part of DSL can 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 ontological model of the target domain. The functional level contains operations, which allow to specify the context for the objects. The combination of these two levels determines the meta-model of DSL, which identifies the entities, used in DSL, and relations between them.

This two-level division of the DSL terms into objects and commands allows not only to achieve the maximum correspondence between the ontological model of the target domain and the DSL model, but also to construct the most convenient way of organizing conversions between them. In order to provide such transformations, it is sufficient to adjust the system of matching rules between the components of the ontological model of the target domain and the components of the DSL model. One of the solutions to this problem can be the mechanism of graph transformations between the graph representation of the ontology and the DSL model. In such mechanism, formal specification of transformations is defined by one of specialized graph-transformation languages such as

ATL Transformation Language [9], GReAT (Graph REwriting And Transformation) [10, 11], AGG (Attributed Graph Grammar) (http://www.user.tu-berlin.de/o.runge/agg/), etc.

In our research, we propose to use ATL, because this language allows to describe the transformation rules from any original model into any target model, conducting a transformation at the level of meta-models. ATL has a simple scheme for defining the transformation rules with only 4 components: the name of the rule, the type of the element of the original model, the type of the equivalent element of the target model and the block for definition of the additional actions with transforming and corresponding components of the original and the target model (Fig. 1).

```
rule    rule_name {
from    type_of_the_element_of_original_model (conditions_expression)
to      action or type_of__the_element_of_target_model
do {    additional_actions    }
}
```

Fig. 1. The scheme for definition the transformation rules in ATL

2.3 Definition and Classification of Model Evolution

As already mentioned above, the ontology is some kind of a model of the certain area and 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. These assumptions lead to the idea, that instead of the concept of co-evolution of ontologies and DSL, we can consider a single concept of the model evolution.

From the formal point of view, the structure of every model is a combination (E, R) of some entities (each entity is a set of its attributes $e_i = \{attr_{i_1}, attr_{i_2}, \ldots, attr_{i_M}\}, M \in \mathbb{N}, i = 1, N)$ in the certain domain and relations between them correspondingly. In these terms, evolution of the model is a process of changes in structure of this model. However, structure means not only entities, used in some specific model, but and relations between them. As follows, model evolution can be separated into two classes [4]: 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: $\left(\mathrm{E}^1, \mathrm{R}^1\right)$ is a result of vertical evolution of the model (E, R) if $|\mathrm{E}| \neq \left|\mathrm{E}^1\right|$ and $|\mathrm{R}| \neq \left|\mathrm{R}^1\right|$. In terms of ontologies it means, that new objects (universals and/or classes) can be added or some previously created objects can be deleted, with corresponding changes in the set of relations. In terms of DSLs it means, that the object level will be changed: some objects will be introduced, some will be removed from the language – alongside with the corresponding changes on the functional 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: $\left(\mathrm{E}^1, \mathrm{R}^1\right)$ is a result of horizontal

evolution of the model (E, R) if $|E| = |E^1|$ and $\exists e_i \in E, e_i^1 \in E^1 : Attr_i \cap Attr_i^1 = Attr_i \bigwedge |Attr_i| \neq |Attr_i^1|$ and $\exists r_i \in R, r_j \in R^1 : r_i \notin R^1 \vee r_j \notin R$. In terms of ontologies it means, that the set of objects (universals and classes) 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 object level will be changed only in terms of objects attributes with corresponding changes on the functional level (so-called constructors of objects) or in terms of connections between them (in this case changes on the functional level depend on the nature of changed connections).

In these circumstances, the question is how to synchronize changes in the ontological model of the domain and in the DSL model. To answer this the graph-model of DSL have to be introduced.

3 Problem of Co-evolution of the Ontology and DSL

3.1 The Graph-Based Model of DSL

In order to organize the transformation rules between the graph representation of the ontology and the DSL meta-model, we need to specify rules in terms of the graph transformation formalism. According to principles described in [12, 13], we will use the following concepts:

- ***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 ($RRest_i = \{rrest_{j_i}\}, j_i \in \mathbb{N},\ j_i < \infty$).

In this case, the graph of the meta-model can be characterized as an oriented pseudo-metagraph GMM = (V, E), for which the following constraint holds:

$$V = Set \bigcup_{i=1}^{|Set|} Attr_i \bigcup_{i=1}^{|Set|} Opp_i \bigcup_{i=1}^{|Set|} SRest_i \bigcup Rel \bigcup_{i=1}^{|Set|} RRest_i$$

$$E = ESA \bigcup ESO \bigcup ESR \bigcup ERR \bigcup ESRR$$

where

- $ESA = \{esa_i\}, \overline{\imath = 1, |Set|}$ – a set of arcs connecting each entity with a set of its attributes;
- $ESO = \{eso_i\}, \overline{\imath = 1, |Set|}$ – a set of arcs connecting each entity with a set of operations on it;
- $ESR = \{esr_i\}, \overline{\imath = 1, |Set|}$ – a set of arcs connecting each entity with a set of restrictions imposed on it;
- $ERR = \{err_i\}, \overline{\imath = 1, |Rel|}$ – a set of arcs connecting each relation with a set of restrictions imposed on it;
- $ESRR = \{esrr_i\}, i \in \mathbb{N},\ i < \infty$ – a set of arcs connecting between the entity and relations, in which it is involved.

Simplifying this model, the set of attributes of entities can be considered within a set of entities. Moreover, we can say that the sets of restrictions, both for entities and relations, also can be combined in one set. Accordingly we propose to consider the structure of the DSL model as $(Obj, Rel, Rest, Opp)$, where the first two $Obj = Set \bigcup_{i=1}^{|Set|} Attr_i$, and Rel are responsible for the object-level of DSL, and other $Rest = \bigcup_{i=1}^{|Set|} SRest_i \bigcup_{i=1}^{|Set|} RRest_i$, Opp represent the functional structure of DSL. Interpreting the set Obj as the set of objects in some domain, Rel 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 ontological model of some domain and the structure of DSL can be related by some correspondence. That means, that there is a way for organizing automated co-evolution of them.

3.2 Vertical Evolution of Domain

In order to identify this type of evolution we have to create a set of rules, which transform new entities of the ontology into entities of the DSL model. The first assumption is that the DSL meta-model has the following structure scheme (Table 1).

The second assumption is that an initial correspondence exists between the ontology and the DSL model. It means, when DSL was created, the rules of correspondence between the ontology elements and the elements of the DSL model were determined.

Table 1. Description of DSL scheme components

Element	Meaning
SchemeName	Defines the name of the current DSL model
Class	Defines the class/type of objects (with some attributes inside)
Function	Defines the function (with a set of classes, instances of which are used in the function)
Constructor	Defines the constructor for creation of the current object with the list of attributes and its values

Considering these, we identify two classes of ATL rules for vertical transformation between elements of the OWL ontology representation and the DSL model (Fig. 2).

```
rule   OWLClass2DSLClass {
from   a: OWL!OWLClass (not exists (select b|
                                    b.isTypeOf (DSL!DSLClass)
                                    and a.name = b.name))
to     b: DSL!DSLClass,
       c:DSL!Constructor
}
rule   DropDeletedOWLClassInDSLClass {
from   b: DSL! DSL Class (not exists (select a|
                                      a.isTypeOf (OWL!OWLClass)
                                      and a.name = b.name))
to     drop
do   { drop c: DSL!Constructor (c.name = b.name) }
}
```

Fig. 2. ATL rules for synchronizing added and deleted elements between the ontology and DSL

According to the these ATL rules, OWLClass have to be replaced with specific OWL class, and DSLClass – with a corresponding DSL class. The first rule provides the reflection of all new elements in the ontology, which are absent in the DSL model, and creates corresponding constructor-procedures. The second one drops all elements in the DSL model, removed from the ontology.

Of course, there are situations, when these two rules are not enough. For example, if new entity in the ontology is associated with one of the previously created. In this case, additional rule has to be identified to create all needed relations between the DSL model entities (Fig. 3).

```
rule   OWLClassWithParents2DSLClassWithParents {
from   a: OWL!OWLClass (a.parent->exists())
to     b: DSL!DSLClass (b.name = a.name)
do   { b.parent.name = a.parent.name }
}
```

Fig. 3. ATL rules for reflecting connected elements from the ontology into DSL

Three designed ATL rules (Figs. 2, 3) are sufficient to organize basic vertical co-evolution of the ontology and the DSL model, because they cover all possible changes, which are peculiar to this kind of transformation.

3.3 Horizontal Evolution of the Domain

Following the assumptions established in the previous section and taking into account the fact that horizontal transformation includes changes in terms of entity's attributes and/or relations, the following variants to organize the horizontal co-evolution of the

ontology and the DSL model are possible: (1) changes occurred in the attributes of entities or (2) in the relations between entities.

The first situation (1) means, that some attributes of entities in the ontology were deleted, or new ones were added. In terms of formal rules in ATL language, these cases can be identified by the next rules (Fig. 4).

```
rule   OWLClassAttributes2DSLClassAttributes {
from   a: OWL!OWLClass!OWLObjectProperty (
              not exists (select b|
                          b.isTypeOf (DSL!DSLClass!DSLObjectProperty)
                          and a.name = b.name))
to     b: DSL!DSLClass!DSLObjectProperty (b.owner.name = a.owner.name),
       c: DSL!Constructor!DSLObjectProperty (b.owner.name =
c.owner.name)
}
rule   DropDeletedOWLClassAttributesInDSLClassAttributes {
from   b: DSL!DSLClass!DSLObjectProperty (
              not exists (select a|
                          a.isTypeOf (OWL!OWLClass!OWLObjectProperty)
                          and a.name = b.name))
to     drop
do   { drop c: DSL!Constructor!DSLObjectProperty (c.name = b.name) }
}
```

Fig. 4. ATL rules for synchronizing changes in attributes of elements in the ontology and DSL

The first rule transforms new attributes from entities in the ontology into corresponding entities attributes in the DSL model and adds these attributes as arguments into constructors of corresponding entities. The second one drops attributes, deleted in the ontology, from the corresponding entities and constructors in the DSL model.

The second situation (2) means, that a set of relations in the ontology was changed. To support it the next rules can be used (Fig. 5).

```
rule   OWLClassWithParents2DSLClassWithParents {
from   a: OWL!OWLClass (a.parent->exists())
to     b: DSL!DSLClass (b.name = a.name),
do   { b.parent.name = a.parent.name }
}
rule   DropDeletedOWLClassParentsInDSLClasses {
from   b: DSL!DSLClass (
              not exists (select a|
              a.isTypeOf (OWL!OWLClass)
              and a.name = b.name and a.parent.name = b.parent.name))
to   drop b.parent
do { drop c: DSL!Function!DSLObjectProperty(
                              c->getProperties()->contains(b.parent)) }
}
```

Fig. 5. ATL rules for reflecting changes of relations in the ontology and DSL

The first rule is the same with vertical evolution. The second one provides the opportunity to drop all relations, deleted in the ontology, with corresponding reduction of the list of arguments of the functions, which are used for initialization of corresponding relations.

3.4 Using ATL IDE for Co-evolution of the Ontology and DSL

After description of all possible variants of the model evolution, we have to identify all important stages, used for identification of the process of co-evolution of the ontology and DSL. In other words, we have to define the mechanism, which connects changes between two models: the ontology and DSL meta-model. For this goal, we use ATL Integrated Environment (IDE) [9], which allows to transform the ontology into the target model (DSL model in this case), using the system of defined rules. In order to create the whole consistency between the ontology and DSL meta-model we perform the following steps.

Step 1: *Identification of the ontology*. On this stage, we formulate the original state of the ontology in terms of one of available formats. In our case, we use the OWL format for this propose.

Step 2: *Identification of the DSL meta-model*. We define a set of entities that form the basis of the DSL object-level. For this propose we use the KM3 language (http://wiki.eclipse.org/KM3), which is an implementation-independent language to write meta-models and thus to define abstract syntaxes of DSLs.

Step 3: *Definition of transformation rules*. We introduce the system of the transformation rules, which are responsible for establishing the correspondence between them. Examples of such rules were given in Sects. 3.2 and 3.3.

Step 4: *Applying the transformation rules to the ontology*. We initialize the transformation procedure – apply defined earlier transformation rules to the current ontology. As output, we have the ready DSL model, which corresponds to the structure of the ontology and is a complete description of the object level of the DSL. After the DSL structure is ready, we need only to (re)configure the syntactic analyzer, which transforms the terms of DSL into the terms of some general programming language.

4 A Use Case: Co-evolution of the Ontology and DSL in the Railway Allocation Domain

4.1 Ontology-Part and DSL Description

In the current research, we will consider the ontology of railway transportation. A fragment of the application ontology of this domain is represented in Fig. 6 in the form of a generic semantic network, whose vertices are basic concepts, and arcs express relations between them (part-whole, gender-type, reservation, etc.).

According to this ontology the structure of DSL's object-level was designed (Fig. 7). This structure contains all needed elements of the ontology, which are used in the process of allocation railway station resources. Some ontology entities are transformed into equivalent entities, other (connected between each other) become the attributes for entities of higher level. Also, the relations were transformed into the corresponding fields of classes or into separate classes of connectedness, which contains links to the connected classes. In addition, we need to include into DSL some helper classes to finalize the syntax of DSL.

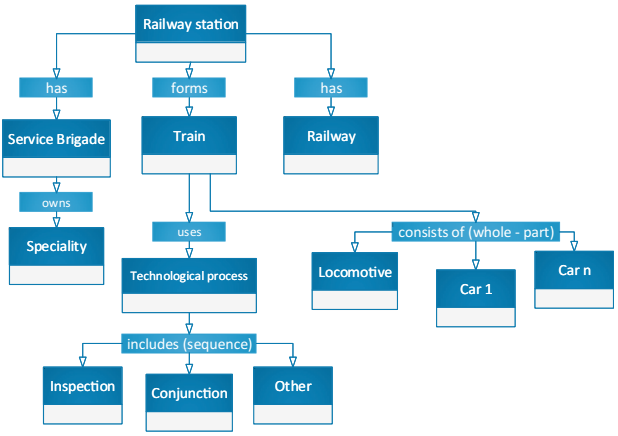

Fig. 6. A part of the railway transportation ontology

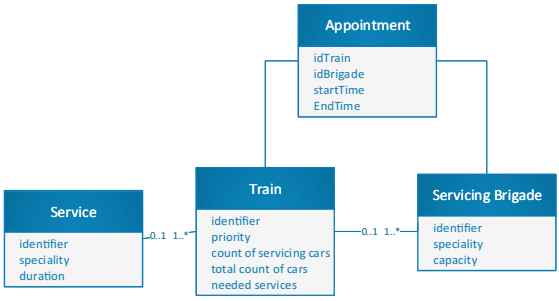

Fig. 7. The structure of DSL: the object level

In order to work with the designed structure of main and helper classes, the next structure of DSL constructors on the functional level is developed (Fig. 8).

On Fig. 8 the meta-symbol '*' means, that a preceding sequence of objects can be written multiple times. The meta-symbols '[' and ']' specify a list of single-typed entities.

```
Trains
({id priority length timeArrival timeDeparture [services]
   wagonsToService})*
EndTrainsBlock;
Services
   ({name mayBeInParallel [speciality] standartDuration [equipment]})*
endServiceBlock;
Servicing Brigades ({id [ speciality ] capacity})*endBrigadeBlock;
Appointments
   ({idBrigade->idTrain timeStart timeEnd})*
endAppointmentBlock;
```

Fig. 8. Constructors for DSL objects

The ontology and DSL above are used in order to appoint servicing brigades between arriving trains. Both contains only terms, which are used throw this process. Other objects and relations ignore, since they have no effect on the procedure under consideration.

4.2 Sceneries of Evolution and Solving Conflicts

In Sect. 4.1 we introduced the ontology for the railway transportation domain and the corresponding DSL. During their design, we assumed that both are universal and can be used for the allocation servicing brigades between trains on any types of stations. However, as can be seen, the original ontology has no whole consistence with the DSL meta-model: some entities of the ontology have no equivalent entities in DSL (or has, but with differences in names) and vice versa. This means that in addition to the previously defined in Sects. 3.2 and 3.3 universal rules, which transforms only equivalent entities of the ontology and DSL, we have to create an additional system of specific rules. The specific rules allow to treat all the differences between the ontology and DSL. Below we consider two situations of evolution scenarios of the ontology and DSL.

The first scenario is that starting in 2018 all servicing brigades will become universal. It means, that all brigades will have no unique skills, providing all types of services. Due to these modifications, the ontology has to be changed. We delete from it such an entity, as Speciality and the corresponding relation as well.

But DSL lefts unchanged and ceases to fully comply with the subject area. Because now such attribute as skills has no equivalent in the ontology. From the pragmatic point of view, it means, that user will continue operate with this term of DSL language and fill the value of this attribute for every brigade. But now he/she will have to input all values of the skills list, otherwise there may be a situation when a brigade cannot be allocated to the maintenance of the train due to an incomplete list of skills. That contradicts the actual state of the subject area. To solve this conflict between the target domain (ontology) and DSL we have to add to the universal rules and apply the next ATL rule (Fig. 9). That rule deletes one particular class ‘Speciality’ and transforms the new state of the ontology into the updated DSL model.

```
Rule   DropSpecialityOWLClassInDSLClassAttributes {
From   b: DSL!DSLClass!DSLObjectProperty("Speciality" = b.name)
to     drop
do { drop c: DSL!Constructor!DSLObjectProperty("Speciality" =c.name)}}
```

Fig. 9. The ATL rule for DSL correction after deleting entity Speciality from the ontology

We have to use this rule because changes in the ontology are vertical: one entity should be deleted. But we need no other rules, since entity Speciality has no other children entities and requires to delete only corresponding attributes in DSL objects and Constructors. After applying this rule, the new scheme of DSL objects will be the following (Fig. 10).

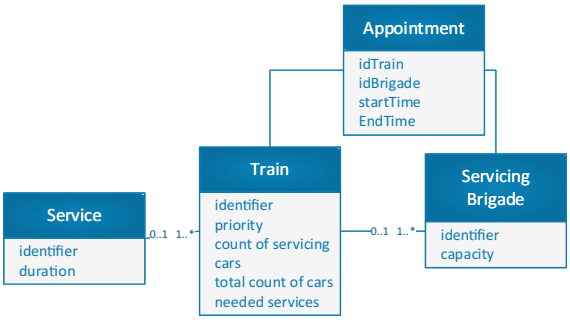

Fig. 10. The structure of DSL after corrections: the object level

As we see, the attribute speciality, which was the equivalent to the entity Speciality in the ontology, is absent. Corresponding changes will be done in the Constructor of the Servicing Brigades and Services on the functional level of DSL (Fig. 11).

```
Services
  ({name mayBeInParallel standartDuration [equipment]})*
endServiceBlock;
ServicingBrigades ({id [ speciality ] capacity})* endBrigadesBlock;
```

Fig. 11. Constructor for DSL objects Servicing Brigades after corrections

The second scenario is caused by the fact, that there are some types of trains, which need organizing services not within the whole train, but for each car separately. It means, that in terms of the ontology, our relation between Services and Trains have to be expanded on Cars. And this change is not so trial, as a first one. Here we have a combination of the vertical and horizontal transformation.

The former transformation is needed because of creation a new type of relation between Service and Car. And the later transformation is the result of DSL earlier structure: because we had no need to operate every Car separately, this entity of the ontology was replaced with the attribute 'total count of cars' in Trains. But now this concept is incorrect. To resolve these contradictions, we extend the previously created rules by the following system (Fig. 12).

The first rule deletes the property 'total count of cars' from DSL. Other rules add skipped entities and relations from the ontology into DSL. But we see, that all rules are specific and are caused by the fact that the structure of the DSL was not originally adapted to the structure of the ontology.

Summarizing two given scenarios of co-evolution, it can be argued that in the case, when the ontology and DSL has no whole coherence, we need to complete the system of universal transformation rules with a set of specific rules, which aims to resolve differences between the ontology and DSL. Furthermore, this system of rules extends every time, when the ontology changes. This significantly complicates the procedure for the organization of co-evolution of the ontology and DSL.

```
rule    DropCarsPropertyInDSLClassAttributes {
from    b: DSL!DSLClass!DSLObjectProperty (b.name = "total count of
cars"))
to      drop
do {
        drop c: DSL!Constructor!DSLObjectProperty (c.name ="total count
of cars")
   }
}
rule    OWLClassCars2DSLClass {
from    a: OWL!OWLClass (a.nam->contains("Car"))
to      b: DSL!DSLClass, c:DSL!Constructor
}
rule    OWLClassServiceCars2DSLClassServiceCars {
from    a: OWL!OWLClass (a.parent->exists() and a.name->contains("Car"))
to      b: DSL!DSLClass (b.name = a.name),
do    { b.parent.name = a.parent.name }
}
```

Fig. 12. The rule for DSL correction after adding the relation between services and cars

In order to avoid the above problems, we have to build a DSL model as a complete reflection of the ontology when the latter is designed for the first time. For this purpose, we proceed through all the stages, described in Sect. 3.4. In this case, we don't create the structure of the DSL model manually, but automatically obtain it from the ontology. As a result, the structure of DSL can be updated every time, when the structure of the ontology is changed – we need only to identify the principle for running the previously created system of universal rules of correspondence between the ontology and DSL.

As a result of this approach, the model of DSL will be described as follows (Fig. 13).

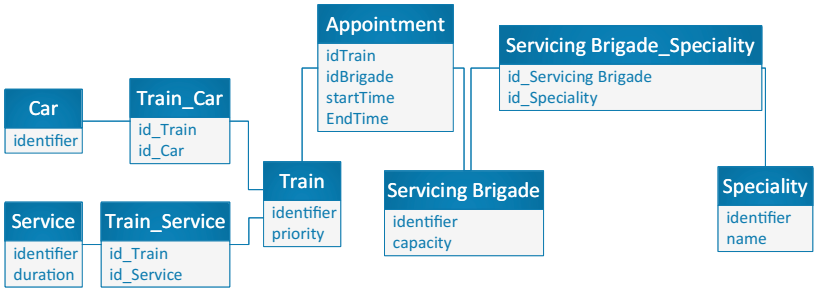

Fig. 13. Structure of DSL in the case of co-evolution with the ontology

Of course, in a manner, such a structure of DSL may seem superfluous. But in fact, from the point of view of designing further changes, this structure of DSL guarantees the simplicity of correspondence with ontology updates, because co-evolution needs only universal rules, described in Sects. 3.2 and 3.3. To optimize this structure of DSL we can use the level of functions, which is not so trivial from the position of establishing equivalent transformations with the ontology, because this level of DSL can differ from the similar one in the ontology because of the limitations of the latter.

5 Conclusion

To summarize, we can say that DSL is an effective tool for working with the subject area. At the same time, it is necessary to control all the changes that occur in the target domain and reflect them in the DSL structure, because otherwise there can be a contradiction between the model of DSL and the subject domain, thus DSL may become completely inapplicable.

To solve this problem, mechanisms for the organizing the co-evolution of the domain ontology and the DSL model can be used. We believe that it is very important to consider what kind of evolution occurs: vertical or horizontal. The first one means changes in the level of the abstraction with affection of the amount of entities in the ontology. The second one happens, when only relations between an object in the ontology or their attributes are modified. Vertical changes can be solved by a simple mechanism of finding differences in the set of entities in DSL and the ontology and organizing corresponding transformations of them. The horizontal evolution is more complicated, because it requires not only mirror changes in the system of relations between entities, but also transformations in terms of the functional level of DSL.

The proposed approach allows to simplify the needed procedures for achieving the correspondence between the ontology and DSL. This approach uses the graph-based definition of the ontology and the DSL model and allows to create the universal system of rules, which provides all needed changes in objects and constructors of the functional level of DSL.

In the paper, it was shown that the proposed approach can be applied also in the case when the designer of DSL only informally relied on the domain ontology in the development of the DSL, and in the case when DSL is constructed as a complete reflection of the ontology. In both cases a system of universal transformation rules has to be used, but in the first case we need to extend it with a set of specific rules, which resolve inconsistencies between the ontology and DSL.

In comparison to existing solutions (like [14, 15]), which use the graph-transformations for conversions between the ontology and DSL, our proposed solution is not based on some specific DSL, but can be applied for any of them. In addition, the proposed method of transformations allows to edit existing ontologies and DSL without recreating them.

The application of the proposed approach is most useful in areas where the structure of the ontology changes frequently: new entities or connections between them appear. For example, in the areas of administrative and economic planning, where the ontology, used in the process of resource allocation, changes constantly because of the emergence of new regulatory documents. Another sphere of potential usage of co-evolution mechanism is the sphere of modelling DSLs, which are used for creation of some models – in this case the changes in components of model have to be reflected in terms of DSL, otherwise DSL is not useful for updated version of the model specification.

In the future, we expect to modify this approach by adding opportunity to change the functional level of DSL model in accordance with the ontology. In addition, such concept of ontologies as Roles have to be examine.

References

1. Fowler, M.: Domain Specific Languages. Addison Wesley, Boston (2010)
2. Arp, R., Smith, B., Spear, A.D.: Building Ontologies with Basic Formal Ontology. The MIT Press, Cambridge (2015)
3. Kosar, T., Bohra, B., Mernik, M.: Domain-specific languages: a systematic mapping study. In: Information and Software Technology, pp. 77–90. Elsevier (2016)
4. Cleenewerck, T., Czarnecki, K., Striegnitz, J., Völter, M.: Evolution and reuse of language specifications for DSLs (ERLS). In: Malenfant, J., Østvold, Bjarte M. (eds.) ECOOP 2004. LNCS, vol. 3344, pp. 187–201. Springer, Heidelberg (2005). doi:10.1007/978-3-540-30554-5_18
5. Pereira, M., Fonseca, J., Henriques, P.: Ontological approach for DSL development. In: Computer Languages, Systems & Structures, pp. 35–52. Elsevier (2016)
6. Guizzardi, G.: Ontological foundations for structural conceptual models, vol. 15. Telematica Instituut Fundamental Research Series, The Netherlands (2005). ISBN 90-75176-81-3
7. Guizzardi, G., Halpin, T.: Ontological foundations for conceptual modeling. Appl. Ontol. **3**, 91–110 (2008)
8. Parr, T.: Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages. Pragmatic Bookshelf, Raleigh (2012)
9. ATL Transformation Language. http://www.eclipse.org/atl/
10. Agrawal, A., Karsai, G., Shi, F.: Graph transformations on domain-specific models. Int. J. Softw. Syst. Model. **37**, 1–43 (2003). Vanderbilt University Press, Nashville
11. GReAT: Graph Rewriting and Transformation. http://www.isis.vanderbilt.edu/tools/great
12. Sprinkle, J.: A domain-specific visual language for domain model evolution. J. Vis. Lang. Comput. **15**, 291–307 (2004)
13. Bell, P.: Automated transformation of statements within evolving domain specific languages. In: Computer Science and Information System Reports, pp. 172–177 (2007)
14. Cleenewerck, T.: Component-based DSL development. In: Software Language Engineering, pp. 245–264 (2003)
15. Challenger, M., Demirkol, S., Getir, S., Mernik, M., Kardas, G., Kosar, T.: On the use of a domain-specific modeling language in the development of multiagent systems. In: Engineering Applications of Artificial Intelligence, pp. 111–141 (2014)