# Ontology and DSL co-evolution using graph transformations methods

*Ulitin Boris*

Scientific director:

prof., PhD Babkin Eduard

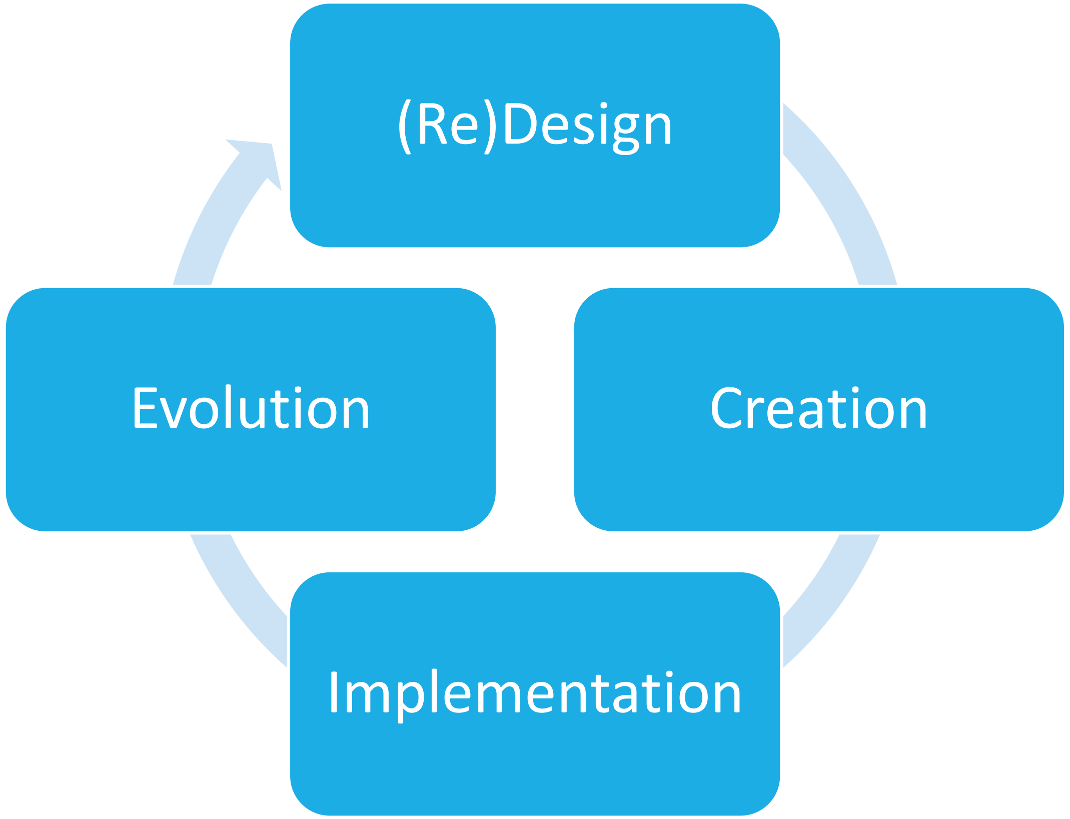
MOSCOW, 2017

## Structure of presentation

# Definition and lifecycle of the DSL

- ***Domain-specific language (DSL)*** – a computer language specialized to a particular application domain (M. Fowler "Domain Specific Languages").

It means, that DSL contains only terms, which are needed for working with a specific domain and for solving tasks of this domain.



# Classification of DSL evolution

## Natural

*Bell, Fowler, Sprinkle, Karsai, Gómez-Abajo, Mengerink,…*

- Caused by changes in the target domain

## Behavioral

*Sprinkle, Karsai*

- Caused by changes in users needs, skills and experience

## Definition of the Ontology

- ***Ontology**** **–** 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.

- ***Application ontology**** describes concepts that depend both on a particular domain and a task (and often combine specializations of both the corresponding ***domain*** and ***task ontologies***).

*G. Guizzardi “Ontological foundations for structural conceptual models”

## Main questions

1. Why and how can ontology be used as a model of the subject area for DSL?

2. How to organize the bridge between the ontology and DSL?

3. How to provide the co-evolution (=corresponding changes without recreation) of the domain (=ontology) and DSL?

Why and how can ontology be used as a model of the subject area for DSL?

How to organize the bridge between the ontology and DSL?

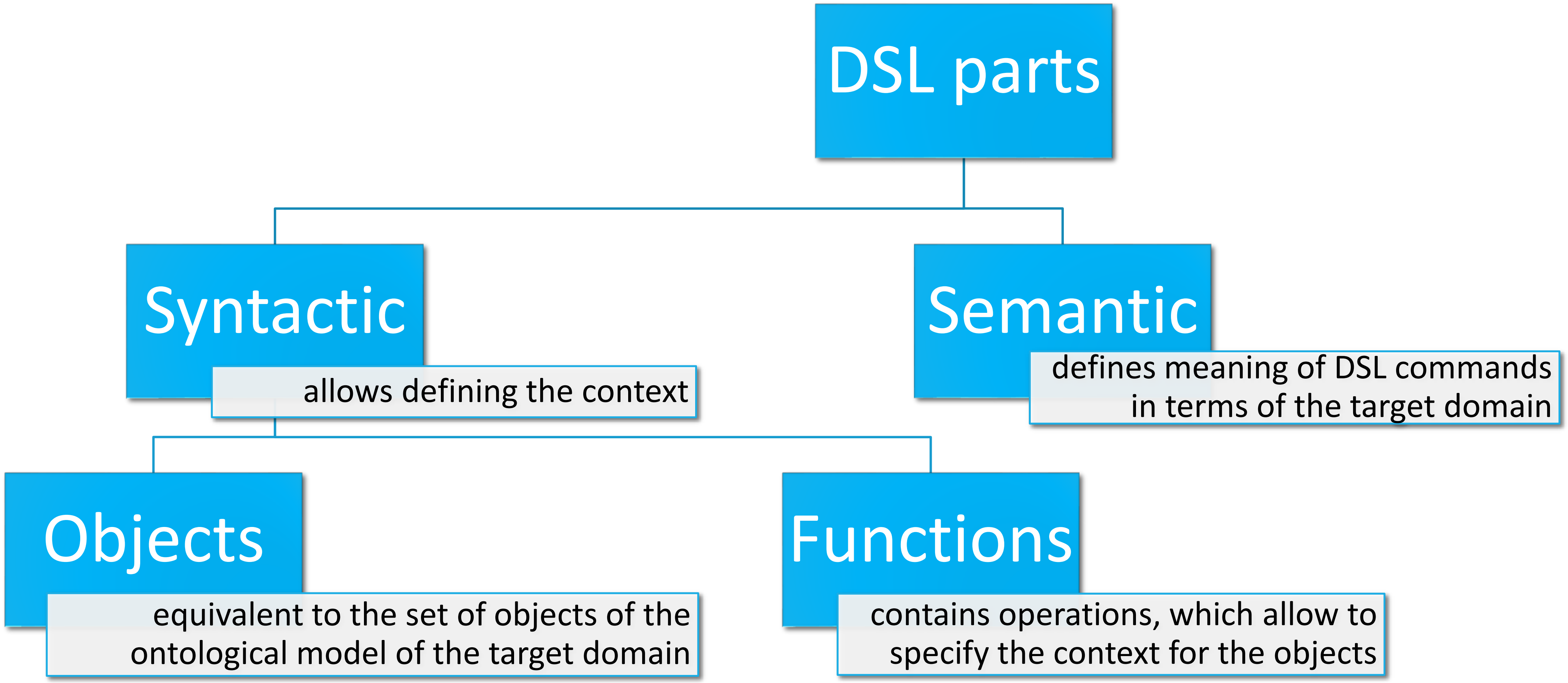
## 1.1 Graph-representation of the Ontology

Ontology can be represented as a triple $(O, R, F)$:

- $O = U \cup C$ is a set of objects;
- $R$ is a set of relations between elements of $O$;
- $F$ is an interpretation function.

**There is a graph-representation of the Ontology**

## 1.2 Structure of the DSL


DSL parts
Syntactic
allows defining the context
Semantic
defines meaning of DSL commands in terms of the target domain
Objects
equivalent to the set of objects of the ontological model of the target domain
Functions
contains operations, which allow to specify the context for the objects

## 1.2 Graph-representation of the DSL

DSL can be represented as follows :

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

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

**There is a similar to the Ontology graph-model scheme:**

$(Obj, Rel, \{Rest, Opp\})$

## Results

- Both Ontology and DSL have similar triples in graph-representation;
- We can organize the transferring elements from one of them into another and vise versa;
- To achieve this goal graph-transformation rules can be used.

## How to provide the co-evolution of the domain (=ontology) and DSL?

## 1.3 Definition of the model evolution

From the formal point of view every model is $(E, R)$ where $E$ is a set of entities:

$$e_i = \{attr_{i_1}, attr_{i_2}, \dots, attr_{i_M}\}, M \in \mathbb{N},, i = 1, N$$

and $R$ is a set of relations between them.

***Evolution of the model*** is a process of changes in the structure of this model

## 1.3 Classification of the model evolution

- ***Vertical:***

  Means the change in level of conceptualization (perspective) of the model:

$$(E^1, R^1) \textit{ is a result of vertical evolution of the model } (E, R) \textit{ if}$$
$$|E| \neq |E^1| \textit{ and } |R| \neq |R^1|$$

- ***Horizontal:***

  Means preserving the level of conceptualization, but changing the sets of attributes for some entities, or changing the set of relations:

$$(E^1, R^1) \textit{ is a result of horizontal evolution of the model } (E, R) \textit{ if}$$
$$|E| = |E^1| \textit{ and } \exists\, r_i \in R, r_j \in R^1 \colon r_i \notin R^1 \vee r_j \notin R \textit{ and}$$
$$\exists\, e_i \in E, e_i^1 \in E^1 \colon Attr_i \cap Attr_i^1 = Attr_i \wedge |Attr_i| \neq |Attr_i^1|$$

## Results[1]

- ***In case of Vertical evolution:***

  We have to **align** the **entities** and **relations** between them.

  It means, that 4 general graph-transformation rules have to be defined: *to transfer entities* between two graph-models (added/deleted) and *to transfer relationships* (added/deleted).

- ***In case of Horizontal evolution:***

  We have to **align** lists of entities' **attributes** and/or set of **relation** between entities.

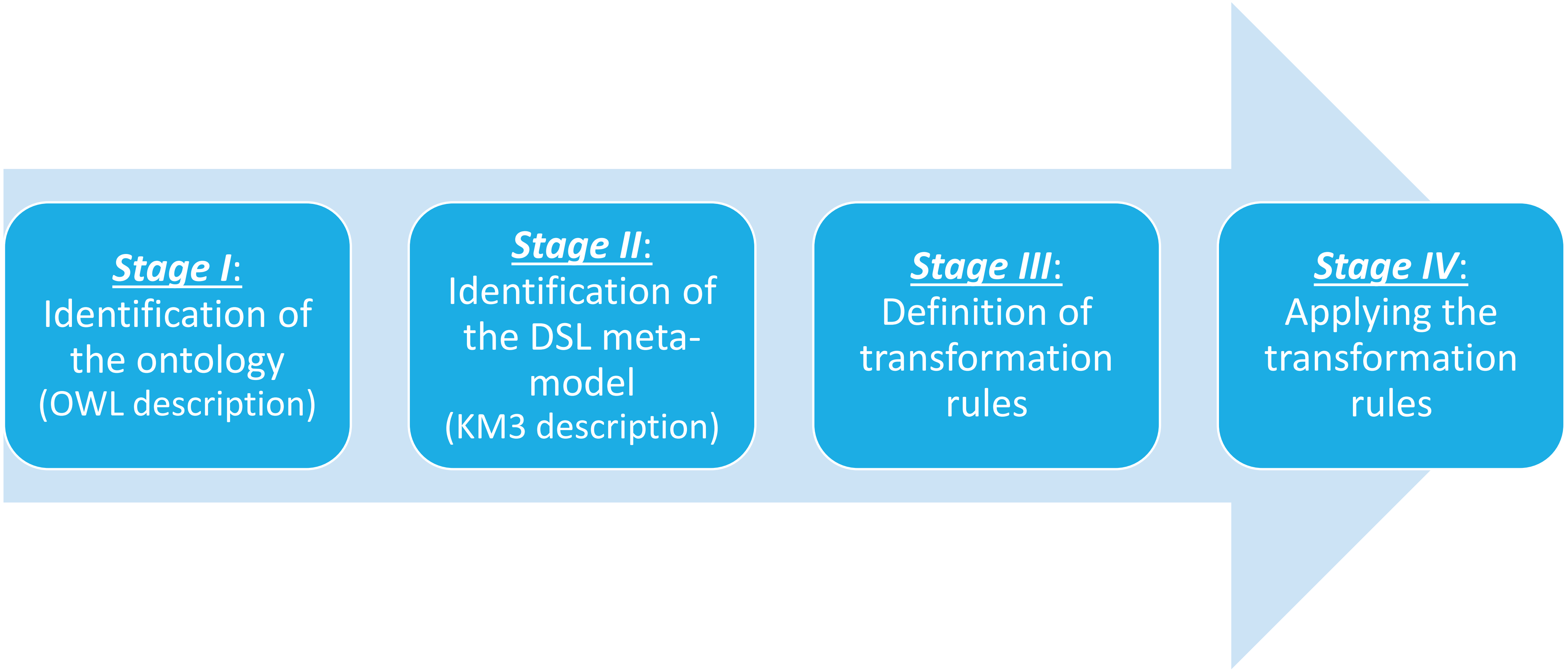
  It means, that 4 general graph-transformation rules have to be defined: *to transfer new attributes* between two graph-models and *to drop deleted*; *to transfer new relationships* and *to drop deleted*.

## Results[2]

- All transformation rules can be separated into 3 categories:
  - ***synchronizing entities:***
    - for added;
    - for deleted;
  - ***synchronizing attributes of entities:***
    - for added;
    - for deleted;
  - ***synchronizing relationships:***
    - for added;
    - for deleted.

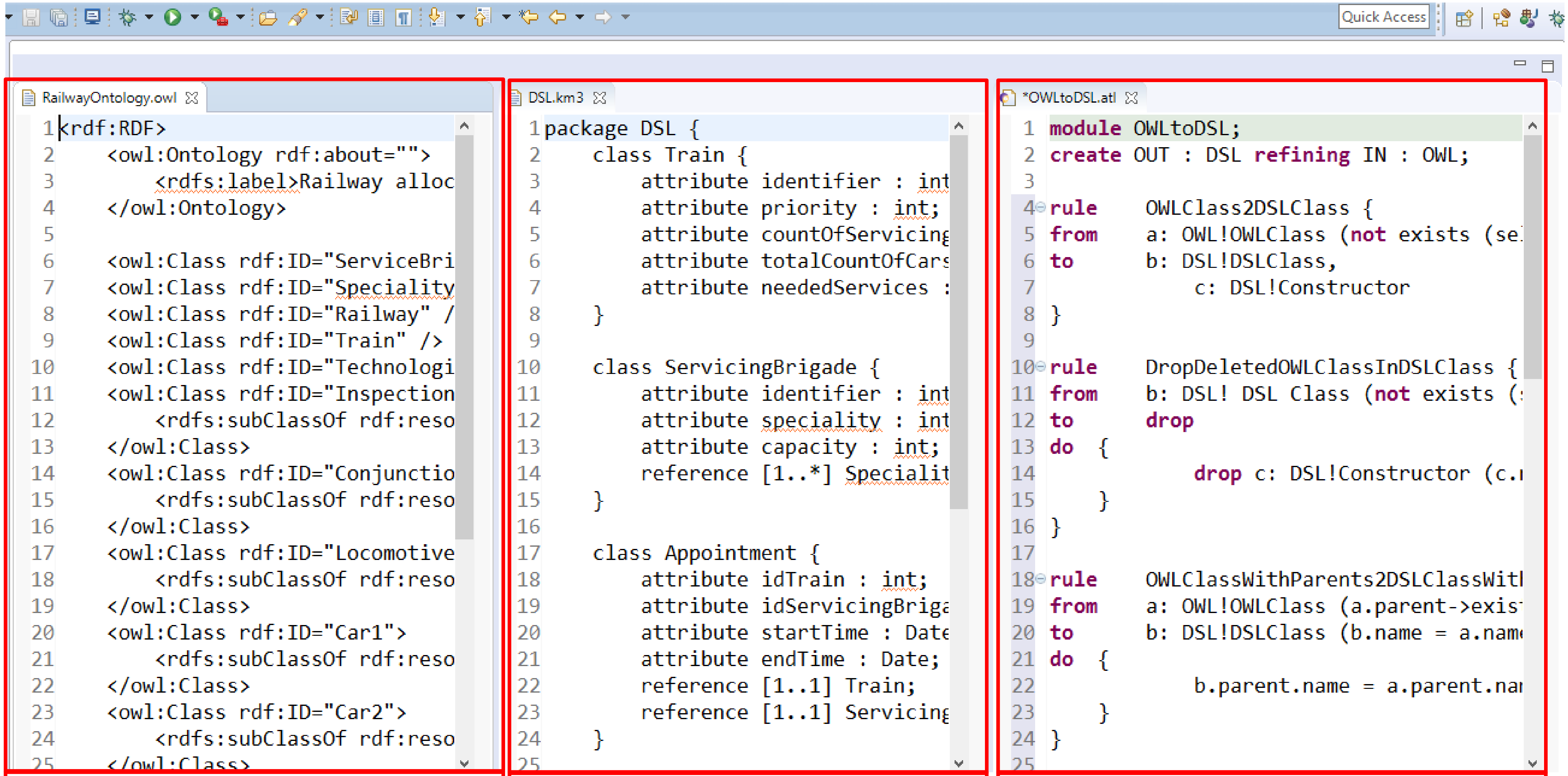
## Solution design

## 2.1 Scheme of created approach


***Stage I***:
Identification of the ontology (OWL description)
***Stage II***:
Identification of the DSL meta-model (KM3 description)
***Stage III***:
Definition of transformation rules
***Stage IV***:
Applying the transformation rules

## 2.2 Using technologies

- ***Eclipse IDE platform***;
- ***Eclipse plugins***:
  - ***TCS (Textual Concrete Syntax)*** – for definition of concrete DSL syntax undo the metamodel ontological terms;
  - ***ATL Transformations*** – for automated connections between the OWL and KM3.



```
<rdf:RDF>
    <owl:Ontology rdf:about="">
        <rdfs:label>Railway alloc
    </owl:Ontology>

    <owl:Class rdf:ID="ServiceBri
    <owl:Class rdf:ID="Speciality
    <owl:Class rdf:ID="Railway" /
    <owl:Class rdf:ID="Train" />
    <owl:Class rdf:ID="Technologi
    <owl:Class rdf:ID="Inspection
        <rdfs:subClassOf rdf:reso
    </owl:Class>
    <owl:Class rdf:ID="Conjunctio
        <rdfs:subClassOf rdf:reso
    </owl:Class>
    <owl:Class rdf:ID="Locomotive
        <rdfs:subClassOf rdf:reso
    </owl:Class>
    <owl:Class rdf:ID="Car1">
        <rdfs:subClassOf rdf:reso
    </owl:Class>
    <owl:Class rdf:ID="Car2">
        <rdfs:subClassOf rdf:reso
    </owl:Class>
```

```
package DSL {
    class Train {
        attribute identifier : int
        attribute priority : int;
        attribute countOfServicing
        attribute totalCountOfCars
        attribute neededServices :
    }

    class ServicingBrigade {
        attribute identifier : int
        attribute speciality : int
        attribute capacity : int;
        reference [1..*] Specialit
    }

    class Appointment {
        attribute idTrain : int;
        attribute idServicingBriga
        attribute startTime : Date
        attribute endTime : Date;
        reference [1..1] Train;
        reference [1..1] Servicing
    }
```

```
module OWLtoDSL;
create OUT : DSL refining IN : OWL;

rule    OWLClass2DSLClass {
from    a: OWL!OWLClass (not exists (se
to      b: DSL!DSLClass,
            c: DSL!Constructor
}

rule    DropDeletedOWLClassInDSLClass {
from    b: DSL! DSL Class (not exists (
to      drop
do  {
            drop c: DSL!Constructor (c.
    }
}

rule    OWLClassWithParents2DSLClassWit
from    a: OWL!OWLClass (a.parent->exis
to      b: DSL!DSLClass (b.name = a.nam
do  {
            b.parent.name = a.parent.na
    }
}
```

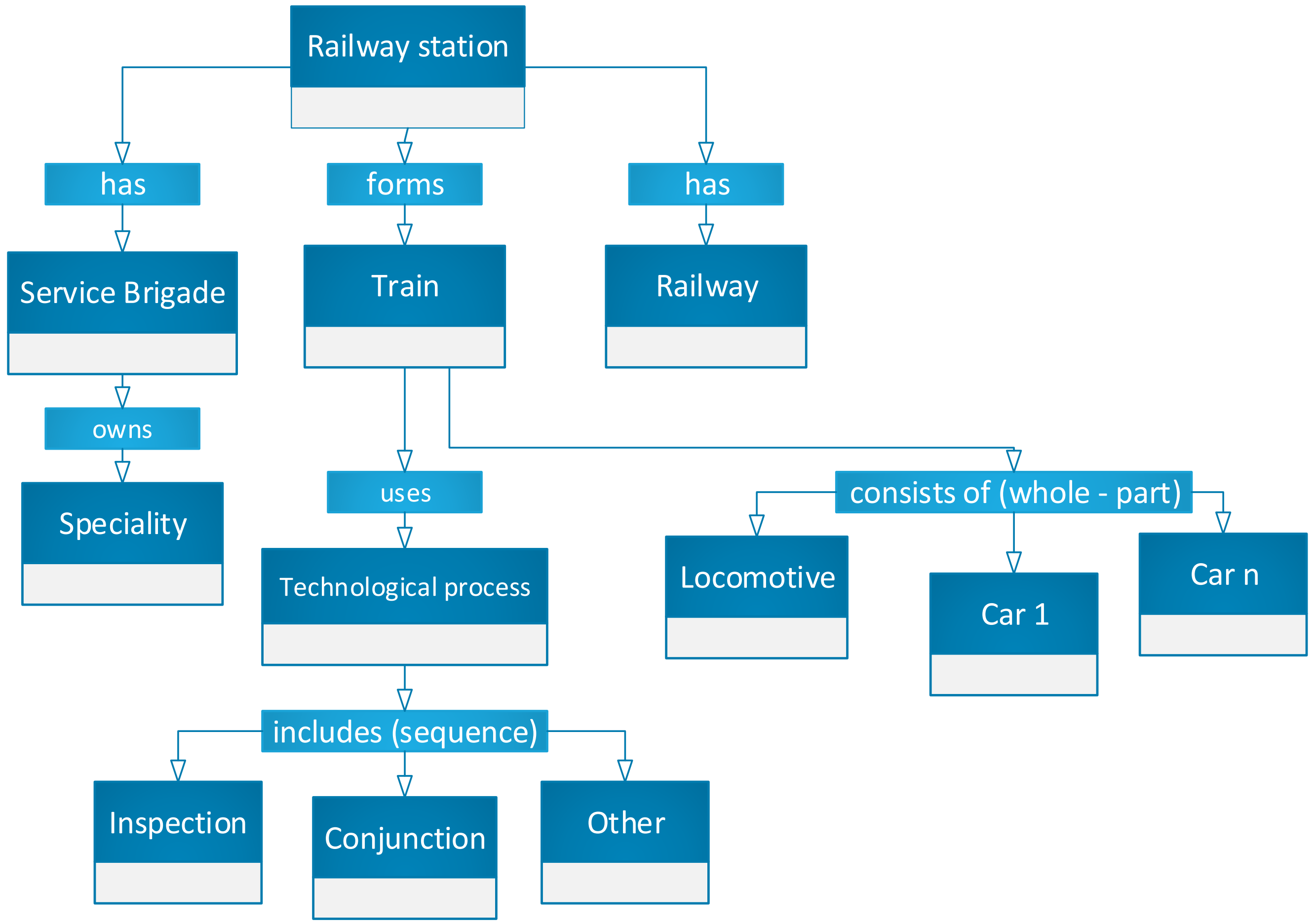
***1st stage***: OWL description of the ontology

***2nd stage***: KM3 description of DSL

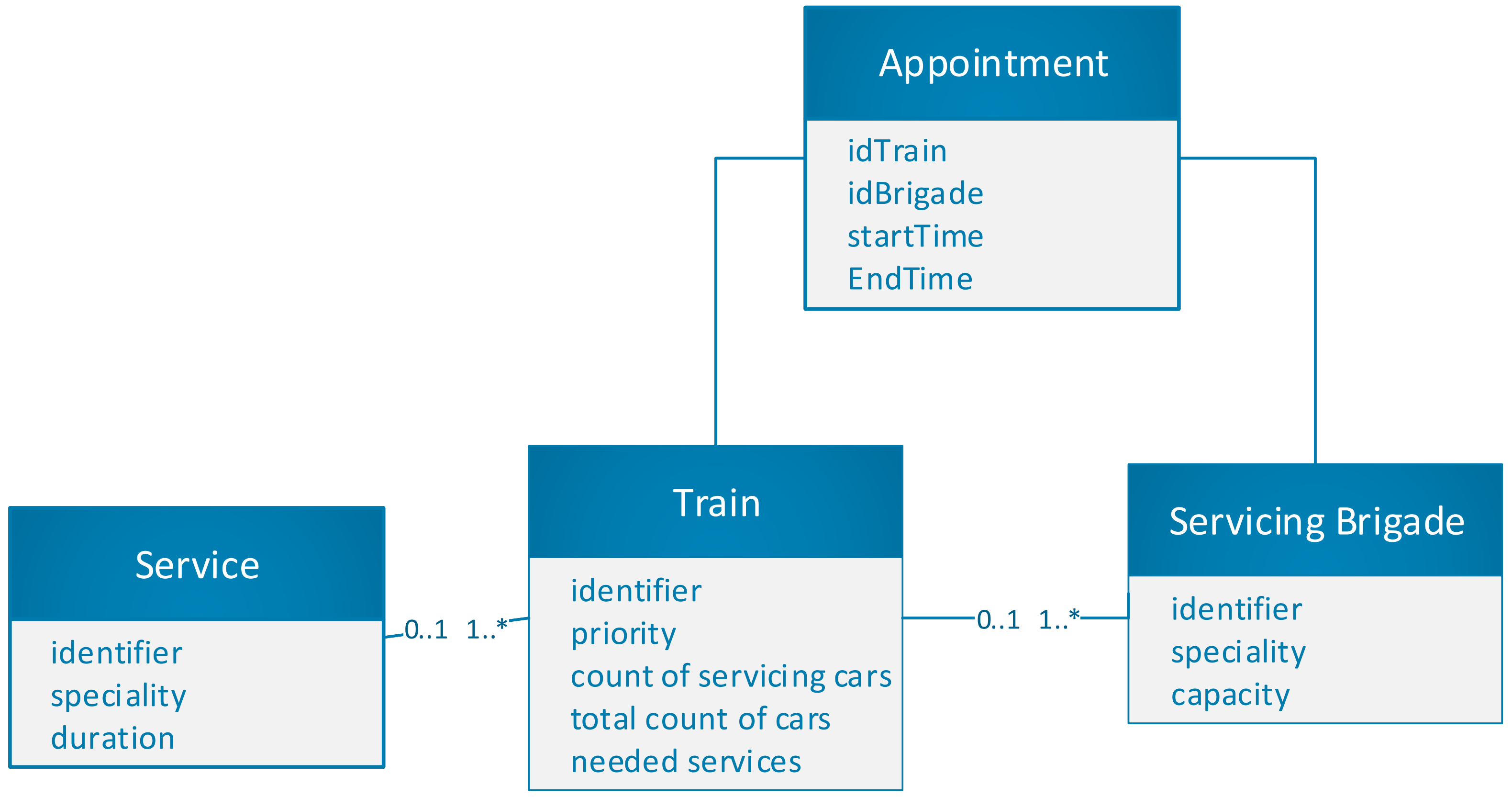
***3rd stage***: Definition of ATL transformation rules

## ***3. Use case:*** co-evolution of the ontology and DSL in the railway allocation domain

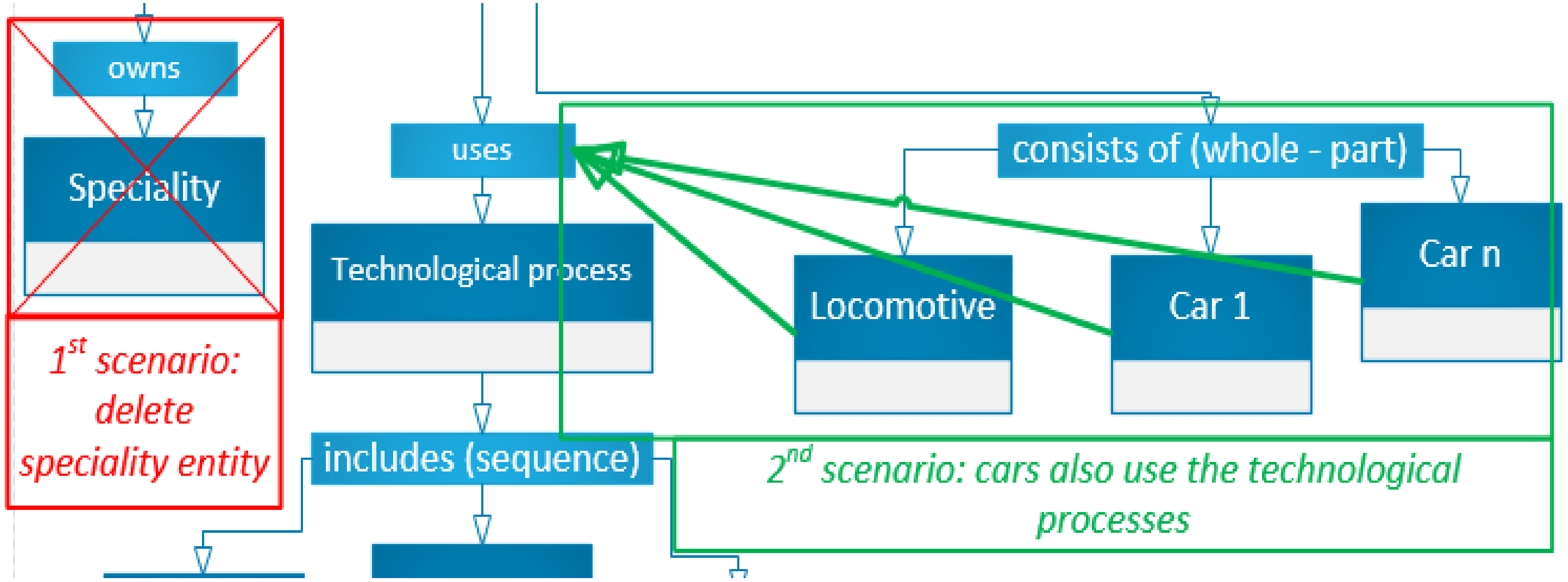
# A part of the railway transportation ontology



# A part of DSL objects scheme



# Sceneries of evolution


owns
Speciality
uses
Technological process
includes (sequence)
consists of (whole - part)
Locomotive
Car 1
Car n
1st scenario: delete speciality entity
2nd scenario: cars also use the technological processes

## Results

- ***In 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;

  this system of rules extends every time, when the ontology changes.

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, using the universal system of graph-transformation rules.

## 4. Conclusion

- DSL is an effective tool for working with the subject area; DSL contains inside the model of the target domain; the ontology can be used as a model for DSL;
- In order to solve the problem of coherence between the subject domain (=ontology) and DSL the graph-transformation approach can be used;
- Proposed approach allows to simplify the needed procedures for achieving the correspondence between the ontology and DSL by creation of system of universal graph=transformation rules;
- In comparison to existing solutions (ex.: Cleenewerck et al. “Component-Based DSL Development”), created approach is not based on some specific DSL, but can be applied for any of them and allows to edit existing ontologies and DSL without recreating them.

***Further steps:***

- Realization of the opportunity to change the functional level of DSL model in accordance with the ontology;
- Implementation of additional ontological concepts as Roles (according to the results of Laird and Barrett from “Towards Dynamic Evolution of Domain Specific Languages”).

## Thank you for attention!

Questions?