A Practical Approach for Constraint Solving in Model Transformations

Youness Laghouaouta, Pierre Laforcade

A practical approach for constraint solving in model transformations

Youness Laghouaouta[1] and Pierre Laforcade[1]

Computer Laboratory of Le Mans University, France
{youness.lahouaouta, pierre.laforcade}@univ-lemans.fr

Abstract. As MDE (Model Driven Engineering) principles are increasingly applied in the development of complex software, the use of constraint solving in the specification of model transformations is often required. However, transformation techniques do not provide fully integrated supports for solving constraints, and external solvers are not well adapted. To deal with this issue, this paper proposes a pattern-matching based approach as a promising solution for enforcing constraints on target models. A transformation infrastructure is semi-automatically generated and it provides support for specifying patterns, searching for match models, and producing valid target models. Finally, a use case is presented in order to illustrate our contribution.

Keywords: Model Driven Engineering, Model Transformation, Pattern Matching, Constraint Satisfaction Problem.

1 Introduction

In Model Driven Engineering (MDE), the primary focus is on models rather than computing concepts. Models represent all artifacts handled by a software development process and can be used as first class entities in dedicated model management operations (e.g. model transformation, model composition, model validation...).

The model transformation operation is one the pillar of MDE. It underpins the automatic generation of target models from source ones (i.e. generally higher level models). The managed models conform to metamodels that define the structure and well-formedness rules. Besides, a transformation specification/definition includes descriptions of how constructs of source metamodels can be transformed into constructs of target metamodels [15]. Several related techniques have been proposed to provide developers with supports to implement transformation scenarios (e.g. [6][11][16]).

The obtained target models have to conform to the structuring defined by the implied metamodel and satisfy all the related constraints. In practice, constraints cannot be expressed by means of metamodel constructs and require the use of additional formalisms (e.g. OCL [17]). Likewise, model transformation techniques are not well supported for enforcing all constraints. Indeed, developers are constrained to use external constraint solvers or libraries.

However, expressing a Constraint Satisfaction Problem (CSP) is a complex and error prone activity. This is due to the fact that constraint solvers support numeric values while model transformations are expressed against model elements. Hence, developers

are faced with the two domains divergence and are obliged to establish and manage non evident correspondences between these domains.

Our contribution is then a practical constraint solving approach for model transformations. The objective is to allow enforcing constraint on target models while simplifying the expression of the constraint based problem. To this aim, a CSP is considered as a pattern matching problem specified by means of model elements. Besides, the pattern matching is included in a global process that allows producing the expected target models of a given transformation scenario. In [13], we have presented the base principles of the proposed approach and give the primary results. The current chapter extends this work and focuses on parametric patterns.

The remainder of this chapter is structured as follows. In Section 2, we present the context of this research work and motivate the need for a practical constraint solving approach for model transformations. Section 3 gives a global overview of the proposed approach, while Section 4 focuses on implementation details. In section 5, we demonstrate the soundness of our approach using an illustrative transformation scenario. Section 5 lists related work. Finally, Section 6 summarizes this paper and presents future work.

2 Motivation

This research work is conducted in the context of the *Escape it!* project. The objective is to develop a serious game to train visual skills of children with ASD (Autistic Syndrome Disorder). Given the specific needs of autistic children, it was crucial to involve ASD experts in the first development stage. The aim is to guarantee that the proposed game fits to ASD characteristics while to be individually adaptive to each child.

MDE provides principles and techniques that allow domain experts to take part of the design activity and guide the development of the game. Indeed, the domain elements (i.e. children profiles, game components and game scenarios) can be expressed in a high level of abstraction so that the implication of domain experts does not require a technical background. Also, model transformations make it possible to deal with the scenarization process (i.e. the other alternative would be to design and implement all possible configurations of scenarios). Indeed, the profile model and game component model can be transformed to automatically produce adapted scenarios. These latter are used to validate the domain elements and rules that are relevant for the generation of scenarios and will subsequently form a basis for real exploitation within the game.

In [12], we have proposed a metamodel for structuring all the dimensions related to the game. As for the generation of scenarios adapted to children profiles, it is implemented as a model transformation written in Java/EMF [20]. Certainly, the proposed implementation allows optimizing the validation task in the sense that game scenarios are generated in demand and without additional effort. However, the problem arises when domain experts suggest alterations of the domain rules that drive the generation.

The identification of transformation fragments impacted by an expressed change is a complex task. Indeed, the way the transformation is specified does not reveal matches between each experts direction/requirement (i.e. considered here as a constraint) and the transformation fragments that allow building conformed models. In addition, the

experts directions/requirements are not easy to implement even when the transformation is specified from scratch. Several constraints are expressed in order of priority. Moreover, they are global constraints as they are attached to a set of model elements and not to separated ones. In fact, the proposed model transformation uses an external constraints solving library to tackle some very specific generation steps.

As a feedback from the co-design sessions conducted with ASD experts, we realize that MDE provides support for adaptive generation of scenarios and allows varying situations proposed to domain experts without significant effort. However, the way to implement the production of learning scenarios is problematic (i.e. especially when changes are expressed). Hence, we have exploited other model transformations languages/supports (i.e. ETL language [11] and the meta-language *Melange* [3]) to express the generation of scenarios.

Although ETL allows specifying the transformation in a much more structured way compared to Java/EMF (e.g. rules, operations, pre and post blocks), the lack of a CSP support raises a significant issue. As for *Melange* (i.e. a language workbench that allows expressing operational semantics by augmenting meta-classes with behaviors), the generation concern is specified in a modular manner which helps to identify the components (e.g. metaclasses, operations) related to the expressed change. Also, it is possible to reuse existing Java libraries for CSP solving. However, like the Java/EMF transformation, it is not easy to express the direction/requirement of the expert by means of CSP. This is due to the divergence between domains of values supported by the CSP solver (essentially integer and real values) and the concerned model elements.

The next section details our proposal for a model transformation approach that facilitates the expression of problems addressed by constraints satisfaction. Our goal is to deal with the objectives below:

1. the generation of target models by transforming source ones;
2. the specification of constraints applied to target models in a simple manner and constraint solving;
3. the modification or reconfiguration of the transformation in case of constraint changes.

We have to notice that details concerning the last objective are out of the paper scope.

3 Global overview

This section explains how a model transformation implying constraint solving could be considered as a pattern matching problem. We give the base principles of our approach and present an illustrative example. Thereafter, we detail the structuring the pattern matching problem.

3.1 Base principles

Basically, a CSP is defined as a set of variables, variable domains (i.e. possible values for each variable) and a set of constraints. A solution is an assignment of values to each

variable that satisfies every constraint. As for graph pattern matching, it is based on *(sub)graph isomorphism* and requires finding an image (i.e. match) of a given graph (i.e. pattern graph) in another graph (i.e. source graph) [14].

In the literature, several work address the joint use of CSP and pattern matching [19][14][21]. Essentially, the graph pattern matching is expressed and resolved as a CSP. The aim is to improve matching performance by exploiting the rich and advanced research work done in the CSP field. In order to obtain the CSP equivalent of a pattern matching problem, some matches have been established between concepts of the two domains [19]:

- CSP variables correspond to the objects of the pattern graph;
- variable domains correspond to the source graph objects to be matched into;
- constraints correspond to the restrictions that apply to a graph morphism.

Our approach is based on a reverse use of these matches. The core principle is to consider a pattern matching problem as a high level specification of a CSP. Hence, a CSP problem over a model can be directly expressed by means of model elements rather than establishing non evident matches to basic variable domains supported by CSP solvers (e.g. integers, reals).

Figure 1 gives a global overview of our approach. A model transformation that implies constraint solving is decomposed into two steps: a pattern matching step and a transformation step. The idea is to express all constraints to enforce on target models through a relevant pattern. The found match is then transformed into valid target models. Therefore, the expression of the pattern has to consider the following requirements:

- although the pattern is expressed by means of source model elements, it has to ensure the satisfaction of all constraints related to target models;
- a match model has to be sufficient enough to ensure a complete generation of target models.

Furthermore, the expression of a pattern is decomposed into two parts. The structure part refers to the elements to be matched into source models, while the constraint part refers to the different constraints that force the identification of a match model. This decoupling makes it possible to associate multiple constraints (i.e. classed by order of priority) to the same pattern. Also, variation of a constraint does not affect the transformation because this latter is specified using the pattern structure.

3.2 Illustrative example

The transformation scenario we have chosen to illustrate the base principles of our proposal consists in producing a piling up of triangles. These triangles are matched from those belonging to the source model (see Figure 2). This latter contains a set of squares with a colored background with numbered triangles of different areas. Each produced triangle must preserve the same area of its source equivalent and have the background color of the container square. Besides, the produced model must include as many triangles as source squares.

As for constraints to to be satisfied by the target model, they are listed by order of priority:

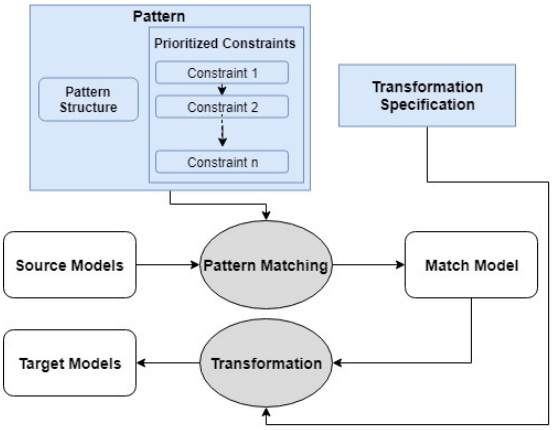

Fig. 1. Global overview of the transformation process

1. the piling up must be coherent (i.e. the area of the contained triangle must be less than the container one) and the target triangles must have different color.
2. the piling up must be coherent.

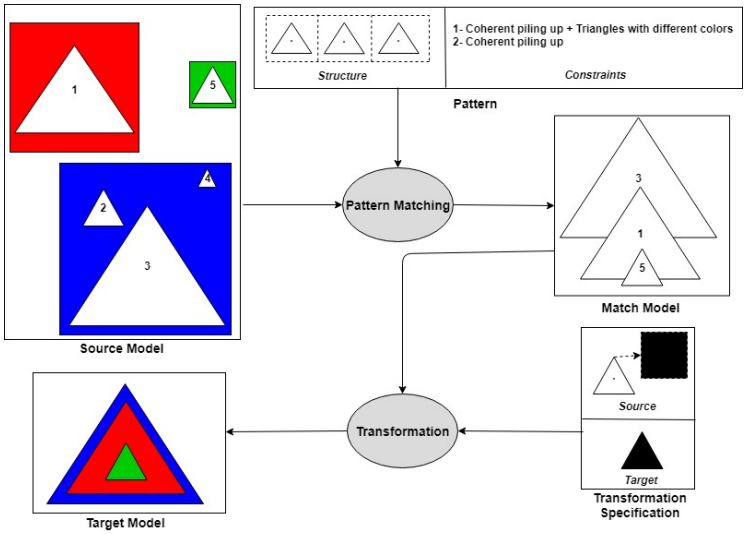

Fig. 2. Simple application example

For this transformation scenario, the pattern structure consists of an ordered set of three triangles (because the source model contains three squares). Each one can match one of the source triangles. As for the constraints part, we specify that the piling up must be coherent and triangles must have different colors. The less prioritized constraint

allows producing a coherent piling up of triangles regardless their colors. We have to notice that the two constraints express what has to be satisfied by target models, but they cannot be directly applied on the pattern structure elements. In fact, other constraint are derived from expressed ones to specify the validation logic of a model matched using the pattern structure. The relevant correspondences are given below:

- The piling up must be coherent: the matched triangles are in descending order of area.
- target triangles must have different color: the matched triangles must belong to different squares.

The matching process uses the high level constraint to search for a valid model. In case of no match is found (e.g. considering that triangles 1 and 5 have the same area), models matched using the pattern structure are validated against the less prioritized constraint. Once a valid match occurs, the transformation is applied on each matched triangle for copying it and assigning the background color of its container. We have to notice that squares are not matched by the pattern. They are derived from matched triangles.

3.3 Configuration metamodel

As discussed before, the pattern specification (i.e. structure and constraints parts) underpins the proposed transformation approach. The relevant information is considered as a configuration for generating the transformation infrastructure. It is stored in a model which conforms to the metamodel depicted in Figure 3.

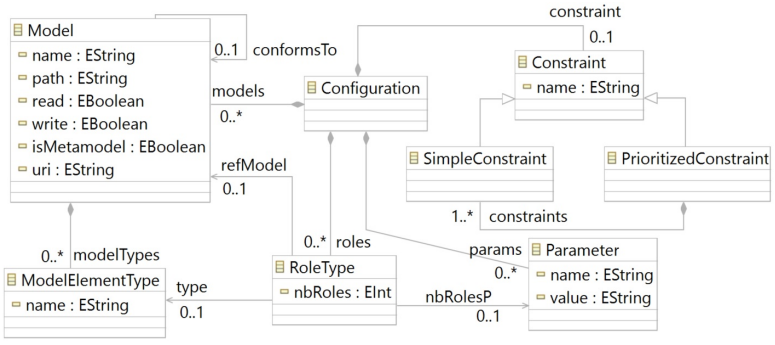

Fig. 3. Configuration metamodel

A configuration references different models (i.e. source models, target models and the implied metamodels). As for the pattern structure, a configuration is defined by multiple role types. Each one can be considered as an abstraction of a set of concrete roles (i.e. pattern elements used to match source model elements) sharing same characteristics or managed as a set. Indeed, a *RoleType* is characterized by the number of

concrete roles and it references a specific source model element type. The number of concrete roles can be fixed (expressed using the *nbRoles* attribute) or parametric (expressed using the *nbRolesP* reference). In the latter case, the roles number corresponds to the resolution of the expression value which depends on the managed models.

For example, the pattern depicted in Figure 2 can be expressed by one *RoleType* instance. This latter references the model element type corresponding to triangles, while the number of concrete roles is parametric and corresponds to the number of source squares. Therefore, the declared role type is an abstraction of three concrete roles and each of them is used to match a unique triangle of the source model.

As for to the pattern constraint part, a configuration expresses if a match model is validated against one constraint level (*SimpleConstraint*) or multi-levels and prioritized constraints (*PrioritizedConstraint*). A constraint is characterized by a name that gives an idea of the validation logic. One can note that the complete constraints specification (i.e. by means of conditions for example) is not covered by the proposed metamodel. Indeed, the configuration model does not ensure the generation of the entire transformation infrastructure. This latter includes resources that have to be manually completed by developers. The next section details these aspects by presenting the infrastructure generation process.

4 Transformation infrastructure

In this section, we detail the generation process of the infrastructure supporting our approach for CSP solving in model transformations (see Figure 4).

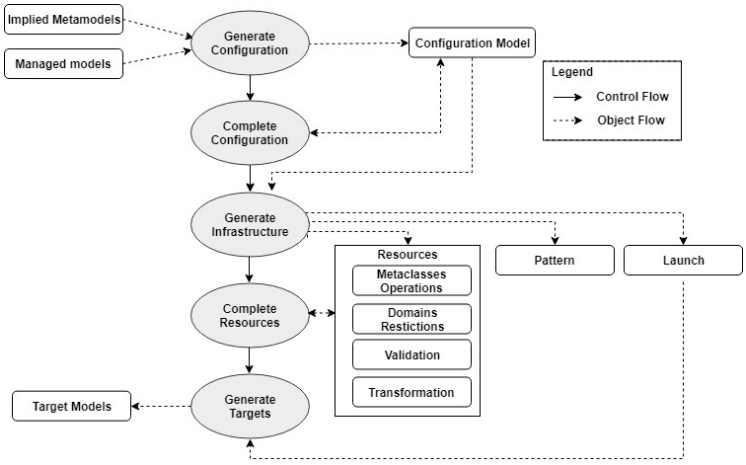

Fig. 4. Process for generating the transformation infrastructure

4.1 Generate configuration

The first step to produce the transformation infrastructure is the generation of the configuration. For that, we provide developers with a GUI allowing them to specify all paths of the managed models and metamodels to which they conform. A configuration model can then be automatically generated. It includes the input information as well as other automatically derived data (e.g. metamodels URIs, model elements types).

4.2 Complete configuration

Recaling from Section 3.3, the configuration model has to be completed with the pattern structure as well as with the constraints type (i.e. simple or hierarchical). To make this task easier for developers, we have associated a concrete textual syntax to the configuration metamodel and implemented a dedicated XText editor [1].

4.3 Generate infrastructure

Once the configuration is completed, developers can ask for the automatic generation of the transformation infrastructure. This is concretely done by associating a specific EPL pattern [9] to each constraint level (i.e. in case of hierarchical constraints).

EPL is a language that provides support for the specification and detection of structural patterns in models that conform to diverse metamodels [9]. Essentially, an EPL pattern consists of a set of typed roles used to capture adequate combinations from source models and a match condition to evaluate the validity of a combination. In our case, typed roles are derived from *RoleType* instances (i.e. with respect to *nbRoles*, *nbRolesP*, *type* and *refModel* values) while the match condition is viewed as a Boolean operation that references a considered constraint.

Besides, the sequencing of the patterns execution is described as an ANT-based Epsilon workflow [8]. For each EPL pattern, a dedicated target and task pair is generated. Besides, *depends* properties of each generated target are specified in order to prohibits the execution of a successor pattern (i.e. with respect to constraint priorities which is derived from the order of declaration) if a match has already been found for the current pattern.

These details are hidden to developers by separating the patterns (i.e. generated automatically) from some required resources to be completed (i.e. transformation, validation, domain restrictions and metaclasses operations). Figure 5 depicts details of the patterns execution activities.

4.4 Complete resources

Since all the patterns have the same structure (i.e. derived from the configuration model), a developer needs to provide only one specification for transforming match models. The transformation is specified by means of EOL operations [10] that are applied the match model elements to produce a valid target model.

Regarding constraints, they are expressed in the validation resource. Indeed, for each constraint, a specific EOL operation is generated and it allows accessing the match

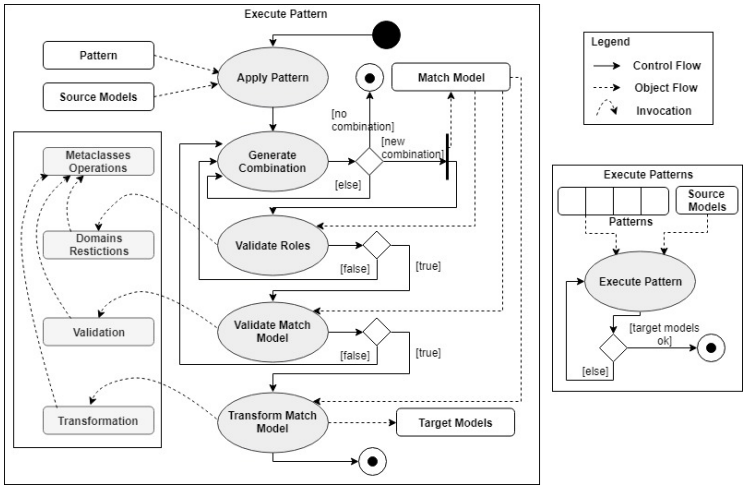

Fig. 5. Patterns execution

model elements. The generated operations must be implemented by the developer in order to express the validation logic (i.e. when a model captured by pattern roles is considered to be a valid match).

Domain restriction resources make it possible to refine the constraints specification. Unlike the validation which applies on an entire match model, the restriction concerns only one single role (e.g. do not capture a triangle if its area exceeds a threshold). Finally, the remaining resource allows the developer to assign operations to metaclasses. These operations can be called by other resources.

4.5 Generate targets

In order to encapsulate the patterns execution details, the transformation infrastructure includes a launch configuration that allows automatically calling the ANT workflow and therefore producing the target models. Nevertheless, changing the source models implies to synchronize the transformation infrastructure. Indeed, the ANT workflow and the launch file has to be regenerated in order to reference the new models paths. In addition, the patterns have to be adapted with respect to the resolved values of parametric concrete roles numbers. The different resources remain unchanged because they are independent from the managed models and the number of concrete roles.

The way in which the transformation infrastructure is structured brings further benefits. When a constraint changes, the transformation resource is not impacted. Besides, the operations associated to the implied metaclasses can be reused when changing the pattern structure or the transformation scenario as long as the same metamodels are involved.

5 Application

This section is dedicated to the application of the proposed approach. First, we briefly present the serious game that motivates the overall proposal and we describe the selected use case. Then, we illustrate each step of the process of generating the transformation infrastructure.

5.1 Use case

The application context is the *Escape it!* project which aims to develop a mobile learning game (i.e. a serious game with learning purposes) dedicated to children with ASD (Autistic Syndrome Disorder). The game intends to support the learning of visual skills and it will be used both to reinforce and generalize the learning skills. These skills will be initiated by ”classic” working sessions with tangible objects. The proposed serious game is based on a minimalist ”escape-room” gameplay. The child (player) has to drag objects, sometimes hidden, to their correct locations in order to unlock the room's door and get to the next level.

The global domain elements required for the generation of game sessions are structured into three related parts: game description elements, profile-related elements, and scenario elements. The required constructs have been defined by a dedicated metamodel [12].

The game description model describes all the real game elements (skills, resources or exercisers, in-game objects...). As for the profile model, it represents a player's (child's) profile. These models are transformed into game scenario. This later is built after three steps.

- objective scenario: it is related to the selection of the visual performance skills in accordance with the current child profile.
- structural scenario: it refers to the selection of the various scenes where game levels will take place. This scenario extends the previous one. It is generated from knowledge domain rules stating the relations between scenes and the targeted skills they can deal with.
- features scenario: it expresses the additional inner-resources/fine-grained elements to be associated to each selected scene (e.g. objects appearing in a scene, their positions...). The features scenario includes components of previous scenarios. It specifies the overall information required by a game engine to drive the set-up of a learning game session.

In [13], we have selected the generation of the objective scenario as an illustrative use case (top part of Figure 6). This paper extends the application scope by presenting the way structural scenarios can be generated using the proposed constraint solving approach. The selected transformation scenario takes as input the objective scenario presented in [13] as well as an extended version of the game description model that includes the structural dimension (i.e exercises) (butom part of Figure 6). The managed models conform to the metamodel depicted in Figure 7. It is worth noting that the presented metamodel is an excerpt of the global one that defines all the game constructs [12].

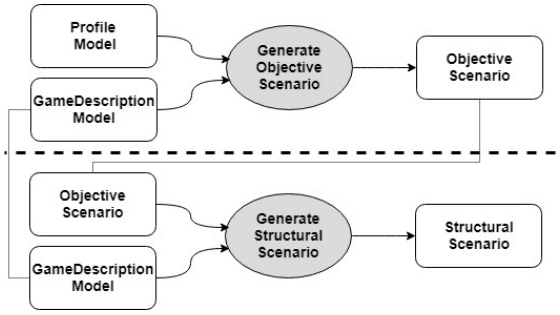

Fig. 6. selected transformation scenario

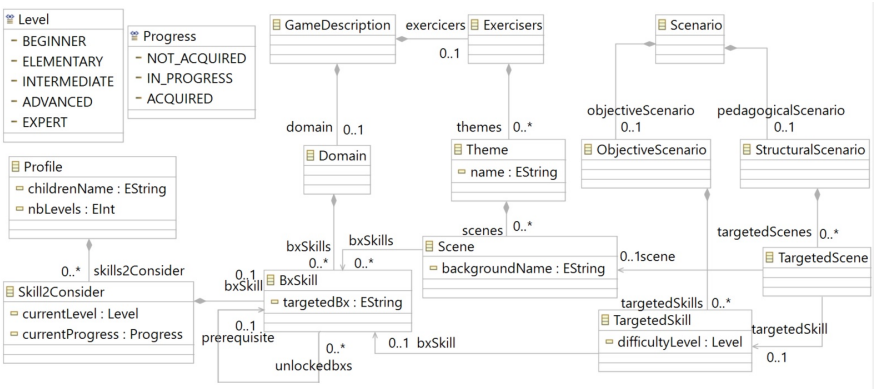

Fig. 7. Excerpt of the ”Escape It!” metamodel

The game description model has been specified on the base of experts' requirements. It expresses four visual performance skills B3-B4-B8-B25 (respectively matching object to object, matching object to image, sorting categories of objects, making a seriation) and their dependency relations. The game description model expresses also the different supported scenes organized by themes. Figure 8 shows an excerpt of this model that focuses on exercisers. Relations between scenes and the targeted skills are depicted with dashed lines.

The structural scenario is generated from the objective scenario depicted in Figure 9. It includes a possible combination of skills to be trained by the child. We recall that the corresponding fictive child profile as well as details about the generation process are given in [13]. The number of targeted scenes to be added to the structural scenario has to be equals to the number of the targeted skills. Besides, each selected scene corresponds to one of the selected targeted skills (i.e. based on the *targets* reference). In addition, the domain experts have expressed some constraints to enforce on the generated structural scenario. They are listed by priority order:

1. all scenes must be different and belong to the same theme;
2. all scenes must belong to the same theme. In addition, two successive scenes must be different;

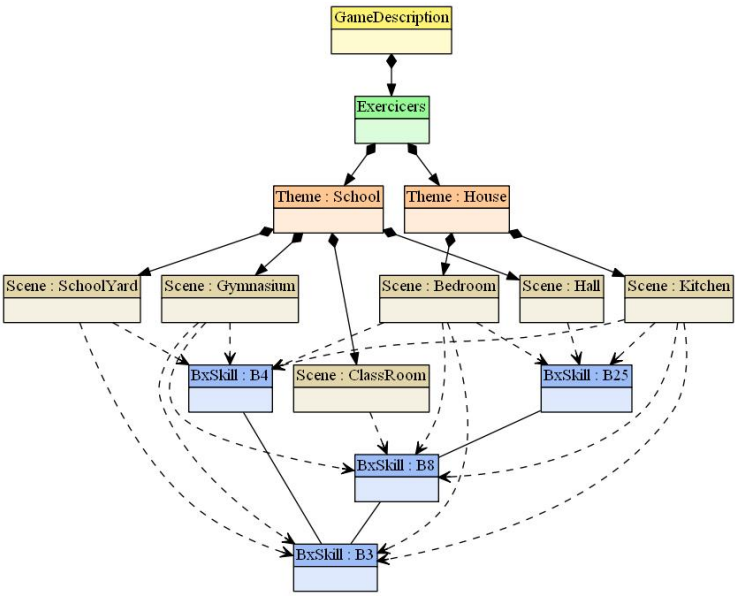

Fig. 8. Game description model

3. all scenes must be different (no constraints on themes);
4. two successive scenes must be different (no constraints on themes).

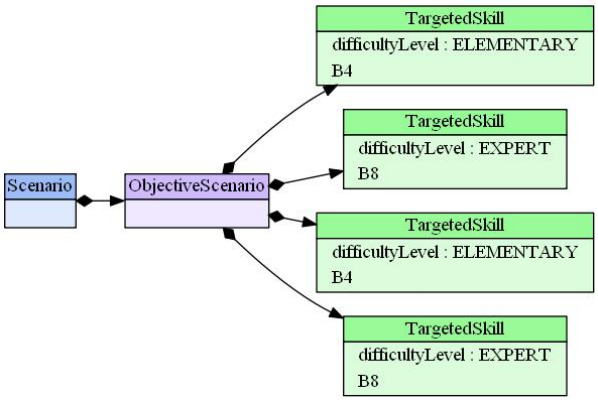

Fig. 9. objective scenario

5.2 Infrastructure generation

In order to perform the described transformation scenario, we start by generating the relevant configuration. This latter can be completed by defining the pattern structure and identifying constraints through a dedicated Xtext editor (Figure 10).

For the presented use case, the pattern comprises three role types. The first one allows matching an *ObjectiveScenario*, the second role type corresponds to the *targetedSkill* instances to be matched, and the last one corresponds to the selected scenes. The two last roles are related to a parameter (i.e. parametric number of roles) to express the need to match all targeted skills belonging to the source objective scenario and the same number of scenes. Indeed, each pair of *TargetedSkill* and *Scene* elements is viewed as the source equivalent of a targeted scene (*TargetedScene*) to be generated. As for the constraints presented above, they are expressed as prioritized constraints while greatest priority is given to the first declared one. Aside from the pattern structure and constraints parts, all the other elements are automatically generated.

```
Configuration{
    Models:
        metamodel "mmcs" "C:/Eclipse/workspace/IcsoftExt/models/GameDescription.xmi"
            uri: "http://mmcs/1.0/"
            Types:
                "Domain"
                "BxSkill"
                "GameDescription"
                "Exercisers"
                "Scene"
                "Profile"
                "Skill2Consider"
                "Scenario"
                "ObjectiveScenario"
                "StructuralScenario"
                "TargetedSkill"
                "TargetedScene"
                "Theme"
        model "m1" "C:/Eclipse/workspace/IcsoftExt/models/Scenario.xmi" read write mmcs
        model "m2" "C:/Eclipse/workspace/IcsoftExt/models/GameDescription.xmi" read mmcs
    Pattern:
        roleType m1!mmcs.ObjectiveScenario[1]
        roleType m1!mmcs.TargetedSkill [$p]
        roleType m2!mmcs.Scene [$p]
    Parameters:
        "p": "m1.Scenario.conceptualscenario.targetedskill"
    Constraint:
    ComplexConstraint "scenesConstraint":
        SimpleConstraint "dSuT"
        SimpleConstraint "uTdSS"
        SimpleConstraint "dS"
        SimpleConstraint "dSS"
}
```

Fig. 10. Configuration model

Once the configuration model is completed, the transformation infrastructure can be generated (see Figure 11). Recalling from Section 4.3, an EPL pattern is automatically generated for each constraint level and the related details are hidden to developers by separating patterns and the required resources. For the application example, four EPL patterns are generated respectively for the aforementioned constraints (cf. Section 5.1). Listing 1 illustrates an excerpt of the pattern generated for constraint 3 (i.e. all scenes must be different). For convenience, the excerpt focuses on the pattern structure and the

resources invocation. Some code fragments (e.g variables declaration, stop searching, randomness...) have been removed to simplify the pattern's interpretation.

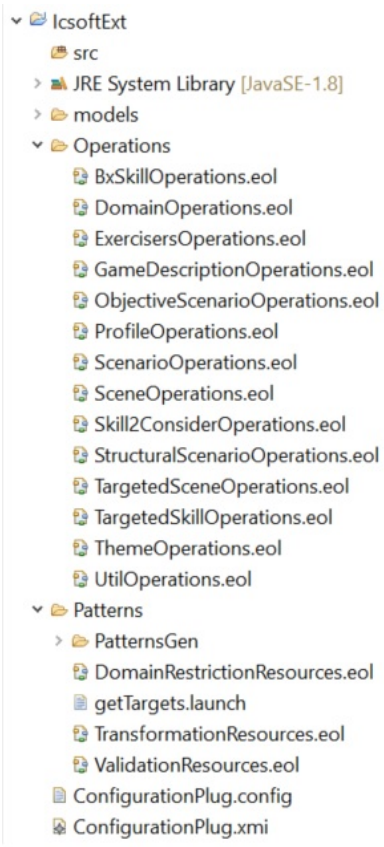

Fig. 11. Transformation infrastructure

As for the domain restriction resource, it is possible to specify guard conditions in order to restrict the possible elements to be caught by a role. Given that the restriction mechanism is not applicable for the selected use case, the generated operations remains unchanged and they allows capturing all possible elements (cf. Listing 2).

Listing 3 depicts an excerpt of the validation resource. Rour operations are automatically generated with respect to the constraints type and names. We complete these operations with action blocks that express the specific validation logic for models matched by the pattern (the added code is highlighted). For example, the third operation implements the validation logic the constraint 3 (i.e. all scenes must be different). The first statement verify if the matched targeted skills are all different (i.e. because EPL allows matching the same element multiple times) while the second one is applied on the matched scenes. Also, the *For* statement verify if each matched scene is compatible

with one of the targeted skills. the *targets*() operation implements this behavior and it is expressed in the context of the *Scene* metaclass (i.e. metaclass operations resources).

As for *allDifferent*(), it is a predefined operation. Indeed, we defined a list of operations (e.g. *followingNotMatch*(), *sort*(), *randSequence*()...) which are automatically added to the operations resource.

```
pattern Pattern
        r0 : ml!ObjectiveScenario
                guard : r0.ObjectiveScenarioDomainRestriction(),
        r1 : ml!TargetedSkill
                guard : r1.TargetedSkillDomainRestriction(),
        r2 : ml!TargetedSkill
                guard : r2.TargetedSkillDomainRestriction(),
        r3 : ml!TargetedSkill
                guard : r3.TargetedSkillDomainRestriction(),
        r4 : m2!Scene
                guard : r4.SceneDomainRestriction(),
        r5 : m2!Scene
                guard : r5.SceneDomainRestriction(),
        r6 : m2!Scene
                guard : r6.SceneDomainRestriction(),
        r7 : m2!Scene
                guard : r7.SceneDomainRestriction(),
        r8 : m2!Scene
                guard : r8.SceneDomainRestriction(),
{
match: continue and validatePatterndS(r0,Sequence{r1,r2,r3,r4},Sequence{r5,r6,r7,r8})
onmatch
        {
        //code fragment depends on the matching mechanism (all possible match, first match, random match)
        //use the continue boolean to stop searching for possible combinations.
        }
        do{
        //code fragment depends on the matching mechanism
        transformPattern(r0,Sequence{r1,r2,r3,r4},Sequence{r5,r6,r7,r8});
        }
}
```

Listing 1.1. Excerpt of the EPL pattern generated for constraint 3

```
operation ml!ObjectiveScenario ConceptualScenarioDomainRestriction(): Boolean{
return true;
}
operation ml!TargetedSkill TargetedSkillDomainRestriction(): Boolean{
return true;
}
operation m2!Scene SceneDomainRestriction(): Boolean{
return true;
}
```

Listing 1.2. Domain restriction resource

```
operation validatePatterndSuT(r0 : ml!ObjectiveScenario,r1 : Sequence ,r5 : Sequence ) :Boolean{
        //code fragment implementing the constraint 1
}
operation validatePatternuTdSS(r0 : ml!ObjectiveScenario,r1 : Sequence ,r5 : Sequence ) :Boolean{
        //code fragment implementing the constraint 2
}
operation validatePatterndS(r0 : ml!ObjectiveScenario,r1 : Sequence ,r5 : Sequence ) :Boolean{
        if(not allDifferent(r1)) return false;
        if(not allDifferent(r5)) return false;
        for(i in Sequence{0..r1.size()-1}){
                if(not r5.get(i).targets(r1.get(i))){
                        return false;
                }
        }
        return true;
}
operation validatePatterndSS(r0 : ml!ObjectiveScenario,r1 : Sequence ,r5 : Sequence ) :Boolean{
        //code fragment implementing the constraint 4
}
```

Listing 1.3. Validation resource

As for the transformation resource (see Listing 4), we completed it with actions to be applied on the match model in order to produce a valid structural scenario. Basically, a new *StructuralScenario* element is created as a target equivalent of the matched

objective scenario. In addition, for each targeted skill, the compatible scene is selected (with respect to the sequencing order), and both elements are used to create a new *TargetedScene* element.

```
operation transformPattern(r0 : ml!ObjectiveScenario,r1 : Sequence ,r5 : Sequence ){
        var ss=createStructuralScenario(r0);
        for(i in Sequence{0..r1.size()-1}){
                ss.targetedscene.add(createTargetedScene(r1.get(i),r5.get(i)));
        }
}
operation createStructuralScenario(p0 : ml!ObjectiveScenario) : ml!StructuralScenario {
        var ss= new ml!StructuralScenario;
        p0.eContainer().pedagogicalscenario=ss;
        return ss;
}
operation createTargetedScene(p0 : ml!TargetedSkill,p1 : m3!Scene) : ml!TargetedScene {
        var ts= new ml!TargetedScene;
        ts.targetedskill=p0;
        ts.scene=p1;
        return ts;
}
```

Listing 1.4. Transformation resource

Figure 12 presents the generated structural scenario. We can verify that the selected scenes are compatible with the targeted skills. However, the proposed skills do not belong to the same theme. In fact, no combination of the possible skills allows enforcing the first constraint. For that, the transformation generates a scenario with respect to a less prioritized constraint (i.e. constraint 2). Indeed, the selected scenes belongs to same theme (i.e School), and you can see that each one is different from its successor scene.

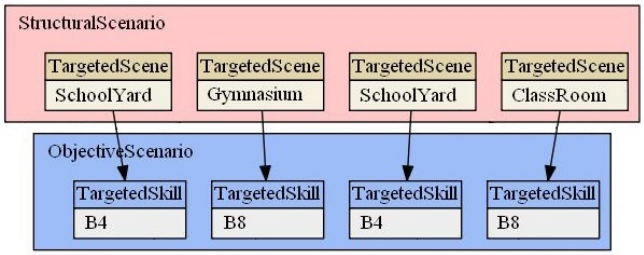

Fig. 12. The generated structural scenario

6 Related work

Our approach for constraint solving is based on expressing constraints to enforce on target models by means of source model elements. This proposal is inspired from graph transformations techniques where constraints on the involved graphs can be expressed through application conditions [4]. Besides, the proposed transformation process (i.e. including the match and transformation steps) is similar to the application of graph transformations. For these latter, the source graph fragments concerned with the application of a transformation are first determined with respect to the LHS (Left Hand Side) graph. Then, the matched fragments are replaced with the structure of the RHS (Right Hand Side) graph.

The main difference is the way the pattern is defined. In fact, the pattern structure is separated from the constraints which allows expressing different and prioritized constraints for the same pattern. Besides, it is much easier to express complex constraints within our approach (e.g. textual syntax, feature navigation, predefined operations...). In contrast, for graph transformations the pattern is defined as one block (i.e. the LHS graph) and constraints are expressed like sub-graphs.

Several proposals have addressed the problem of directly enforcing constraints on target models. Petter et al. [18] have proposed an implementation to extend the QVT-Relations language [16] with constraint solving capabilities. However, the proposal focuses on constraints related to attribute values and disregards global constraints.

Other related work address the automatic generation of models. In this case, models are not considered as targets of applying model transformations but are viewed as valid instances of constrained metamodels [7]. Cabot et al. [2] have proposed an approach where metamodels and OCL constraints are translated into a CSP and a dedicated solver allows producing a valid instance. Based on similar principles, Ferdjoukh et al. [5] have proposed an approach for model generation while dealing with performance.

In the limited scope of the presented use case, we have experimented the use of model generation techniques to perform the transformation scenario. The idea was to express the source models information, the way to construct the target model and the expert requirements, as OCL constraints. Hence, a model generation support (we chose Grimm [5]) can be used to deal with the generation of the expected objective scenario. However, the tool failed because it does not support some essential OCL operations.

7 Conclusion

This paper presents a practical approach for constraint solving in model transformations. The base principle is to consider a pattern matching problem as a high level specification of a CSP. Besides, a transformation infrastructure that underpins the conceptual proposal can be generated in a semi-automatic manner. Indeed, this infrastructure provides support for pattern specification, match model search, and transformation into valid target models. A use case extracted from the *Escape It!* serious game has been selected to illustrate these tasks.

The way the pattern definition is carried out allows some benefits. By decoupling the pattern structure from the validation constraints, it is possible to associate multiple constraints to a same pattern and therefore allows specifying shared transformation rules for all validation logic. In addition, the proposal supports parametric patterns. Hence, the same pattern definition can be used in various transformation scenarios even if involving slightly different match models.

The integration of our proposal in the co-design framework for the presented serious game opens up many perspectives. The future work deal with two main issues: (i) the cognitive effort to be implicated by the domain expert in order to specify/interpret the pattern and (ii) the change impact analysis of domain rules. To address the first issue, we are exploring a new approach to express the constraint satisfaction problem by means of target model elements. The corresponding source pattern can be automatically generated by exploiting some relevant information (i.e. source-figtarget correspondences,

one time or multiple match...). As for the change of domain rules, the regeneration of the transformation infrastructure must consider the extent of the variation expressed by the expert (e.g. adding a constraint must imply changing the validation resource without impacting the transformation and operations resources).

References

1. Bettini, L.: Implementing domain-specific languages with Xtext and Xtend. Packt Publishing Ltd (2016)
2. Cabot, J., Claris, R., Riera, D., et al.: Verification of uml/ocl class diagrams using constraint programming. In: First International Conference on Software Testing Verification and Validation, ICST 2008. pp. 73–80. IEEE (2008)
3. Degueule, T., Combemale, B., Blouin, A., Barais, O., Jézéquel, J.M.: Melange: A meta-language for modular and reusable development of dsls. In: Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering. pp. 25–36. ACM (2015)
4. Ehrig, H., Ehrig, K., Habel, A., Pennemann, K.H.: Constraints and application conditions: From graphs to high-level structures. In: International Conference on Graph Transformation. pp. 287–303. Springer (2004)
5. Ferdjoukh, A., Baert, A.E., Chateau, A., Coletta, R., Nebut, C.: A csp approach for metamodel instantiation. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence. pp. 1044–1051. IEEE (2013)
6. Jouault, F., Kurtev, I.: Transforming models with atl. In: International Conference on Model Driven Engineering Languages and Systems. pp. 128–138. Springer (2005)
7. Kleiner, M., Del Fabro, M.D., Albert, P.: Model search: Formalizing and automating constraint solving in mde platforms. In: European Conference on Modelling Foundations and Applications. pp. 173–188. Springer (2010)
8. Kolovos, D., Rose, L., Garcia-Dominguez, A., Paige, R.: The epsilon book (2017) (2017)
9. Kolovos, D.S., Paige, R.F.: The epsilon pattern language. In: 9th IEEE/ACM International Workshop on Modelling in Software Engineering, MiSE@ICSE 2017. pp. 54–60. IEEE (2017)
10. Kolovos, D.S., Paige, R.F., Polack, F.A.: The epsilon object language (eol). In: European Conference on Model Driven Architecture-Foundations and Applications. pp. 128–142. Springer (2006)
11. Kolovos, D.S., Paige, R.F., Polack, F.A.: The epsilon transformation language. In: International Conference on Theory and Practice of Model Transformations. pp. 46–60. Springer (2008)
12. Laforcade, P., Laghouaouta, Y.: Supporting the adaptive generation of learning game scenarios with a model-driven engineering framework. In: Lifelong Technology-Enhanced Learning - 13th European Conference on Technology Enhanced Learning, EC-TEL 2018, UK. pp. 151–165. Lecture Notes in Computer Science, Springer (2018). https://doi.org/10.1007/978-3-319-98572-5_12
13. Laghouaouta, Y., Laforcade, P., Loiseau, E.: A pattern-matching based approach for problem solving in model transformations. In: Proceedings of the 13th International Conference on Software Technologies, ICSOFT 2018, Portugal. pp. 113–123. SciTePress (2018). https://doi.org/10.5220/0006847901130123
14. Larrosa, J., Valiente, G.: Constraint satisfaction algorithms for graph pattern matching. Mathematical Structures in Computer Science **12**(4), 403–422 (2002)

15. Mens, T., Gorp, P.V.: A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science **152**, 125–142 (2006)
16. OMG: Meta object facility (mof) 2.0 query/view/transformation specification (2008)
17. OMG: Object constraint language 2.4 specification (2014)
18. Petter, A., Behring, A., Mühlhäuser, M.: Solving constraints in model transformations. In: International Conference on Theory and Practice of Model Transformations. pp. 132–147. Springer (2009)
19. Rudolf, M.: Utilizing constraint satisfaction techniques for efficient graph pattern matching. In: International Workshop on Theory and Application of Graph Transformations. pp. 238–251. Springer (1998)
20. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, 2nd edn. (2009)
21. Taentzer, G., Ermel, C., Rudolf, M.: The agg approach: Language and tool environment. Handbook of graph grammars and computing by graph transformation **2**, 551–603 (1999)