Using OCL in Model Validation According to Stereotypes*

Zdenek Rybola[1] and Karel Richta[2,3]

[1] Faculty of Information Technology, Czech Technical University in Prague
rybolzde@fit.cvut.cz
[2] Faculty of Mathematics and Physics, Charles University in Prague
richta@ksi.mff.cuni.cz
[3] Faculty of Electrical Engineering, Czech Technical University in Prague
richta@fel.cvut.cz

Abstract. Model-Driven Development approach became popular in past years. Domain-specific profiles are defined for various domains and tools are used to transform models using these profiles to source code artifacts. However, rules need to be defined for the profile elements usage so the transformation can be effective and reliable.
This paper deals with an approach of expressing these rules using special type of metamodel with UML class diagrams with the stereotypes defined in the profile – we call them constraint diagrams. Each class in this metamodel represent all classes in the model with the same stereotype. Using stereotyped associations, we can link classes with different stereotypes and restrict the usage of such stereotype only to relations between specific stereotyped classes in the model. OCL constraints can be generated from the constraint diagram to enable validation of the model according to the rules in the metamodel. This paper deals with the description of the constraint diagram creation and OCL constraints generation.

Keywords: UML, OCL, constraint, stereotype, validation

1 Introduction

Model Driven Development [10] is a modern and popular software development approach. It is based on creation of model of different abstraction levels and transformations between those models. It also includes forward and reverse engineering methods. Forward engineering becomes especially popular for generation of source code from models.

Models of software systems are usually created using UML and transformed to source code using a tool that generates all required artifacts from the model. For instance, many various artifacts for J2EE application such as Entities, Session beans, Message-driven beans and many others can be generated from an

* This research was partially supported by Grant Agency of CTU No. SGS12/093/OHK3/1T/18 and Czech Grant Agency (GAČR) No. GA201/09/0990

J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2012, pp. 93–102, ISBN 978-80-7378-171-2.

UML class model. Because of various domains of software systems, domain-specific UML profiles are usually defined. UML profile [9] is a meta-model that allows the definition of stereotypes and tagged values for model elements. If a profile is defined and used, model transformation can be adapted according to the specified stereotypes and tagged values. However, special rules usually come with the profile to restrict usage of various profile artifacts that needs to be satisfied by any model based on the profile. To make those adapted transformations effective, model validation against the defined rules is required.

In our current research, we deal with an approach to generate OCL constraints for domain-specific profiles using an instance of the meta-model defined in the profile. Creating a constraint diagrams using artifacts of the profile allows us to graphically express domain-specific rules. These diagrams can be easily transformed to OCL constraints that can be used for model validation using various CASE and OCL tools. Whole process is shown in Fig. 1.

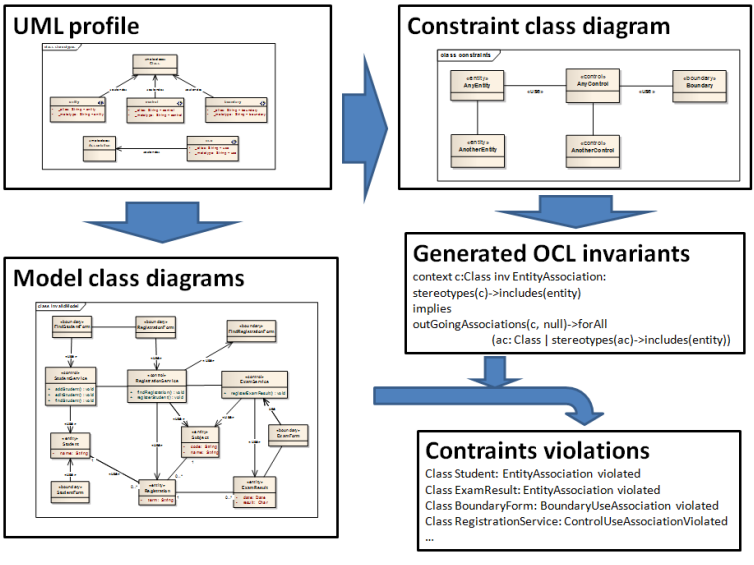

Fig. 1. Graphical description of the model validation process - model diagrams and a constraint diagram are created using the profile; the constraint diagram is transformed to OCL constraints; model diagrams are validated using OCL constraints resulting in a set of constraints violation messages

In this particular paper, we deal with generation of OCL constraints for rules restricting connections between various stereotypes. An example is based on fundamental analysis class model specification [2]. In this specification, only classes with Entity, Control and Boundary stereotypes can be used with some

restrictions for connections between various stereotypes. However, our approach can be used for any profile defined for any domain.

The paper is structured as follows: Section 2 presents related work and their difference from our approach. In section 3, we show an example of model with a profile and definition of rules for stereotype usage. In section 4, we describe the constraint diagram for expressing the rules. Generation of OCL invariants is explained in section 5. Conclusions and further work plans are given in section 6.

2 Related work

There have been done some research on model checking and model validation using OCL. Richters and Gogolla [11] presented an approach of animating model snapshots and validating it against OCL constraints. The authors use USE tool [12] for model animation and validation. The authors also introduced a method of automatic model snapshot generation in USE tool [5]. However, the authors only generate snapshot of the model and the constraints must be defined as required directly in OCL.

Some research was made on model validation against fundamental properties of models defined in UML. Chae et al. [2] focuses on analysis class model used in many standard object-oriented processes where three basic stereotypes are defined for analysis classes - boundary, control and entity. The authors define a set of constraints in OCL that any analysis class model should satisfy including constraints for associations between classes of particular stereotypes. The authors also demonstrate a case study of validation of analysis models against defined OCL constraints using OCLE tool [3]. However, the authors only define particular constraints for these three basic stereotypes of analysis classes.

In [6], Guizzardi defines ontological extension of UML class diagram with ontology stereotypes and constraints called OntoUML. He presents a fine-grained approach for domain modelling using stereotypes to distinguish between various types of domain artefacts and relationships. In [1], Benevides and Guizzardi present a graphical editor for OntoUML and validation of OntoUML model using OCL constraints. These constraints are based on stereotypes of model elements and defined according to the specification of OntoUML.

In comparison to the approaches mentioned above, in our approach we do not define any particular constraint for any particular domain. Instead, we propose a general approach to create a constraint diagram using domain-specific and user-defined profile to model domain-specific business rules. This diagram can be used to generate OCL constraints for validation of any model based on the user-defined profile. We deal with generating OCL constraints for UML class diagrams in this particular paper.

There is also a lot of tools that support model transformation and validation against OCL constraints. Dirigent [7] is an open-source MDA tool for generation of source code from model. The tool reads model stored in XMI format and generate source code files according to specified patterns in Apache Velocity.

These patterns can be based on model element stereotype and generate a set of source files. If extended appropriately, Dirigent would be able to parse the proposed constraint diagram and generate OCL constraints for validation of a model based on the same profile.

Dirigent could also be extended to validate the model using the generated OCL constraints itself. However, there are also many other tools that support model validation using OCL constraints. OCLE tool [3] can be used as mentioned above. DresdenOCL [4] is a toolkit for creating and validating OCL constraints for specified model. It also supports model validation and model transformation together with the constraints to SQL and Java with AspectJ.

3 Defining rules

To make transformations of domain-specific model in UML to source code files effective, domain-specific UML profile should be defined. This profile includes mostly stereotypes of classes and associations for class diagrams and can define tagged values of such stereotypes as well. In the model, various defined stereotypes are used to model various types of artifacts using classes and their associations. During transformation to source files, various stereotyped classes can be transformed to various source code artifacts such as J2EE entities, session beans or servlets.

Lets imagine an example shown in Fig. 2 where a class model of a part of a university information system is shown. The model is created using analysis model profile with stereotypes entity, control and boundary to distinguish between three kinds of analysis classes. We use only a part of the analysis model profile required for our model. The original analysis model profile with full set of rules is defined in [2].

For each of these stereotypes, various source code artifacts can be generated during transformation. For instance of a J2EE application:

- an entity class, a data access object class and a SQL creation script for each entity-stereotyped class,
- a session bean local interface and implementation for each control-stereotyped class,
- and a servlet for each boundary-stereotyped class can be generated.

With generation of source code artifacts for each model element, references between those generated artifacts can be also generated. For instance:

- relationships between entities for each association between entity-stereotyped classes,
- reference between session beans,
- and reference to session bean from servlets can be generated.

However, to generate such artifacts and connections effectively and correctly, the model must satisfy specific rules. These rules must be defined and obeyed during

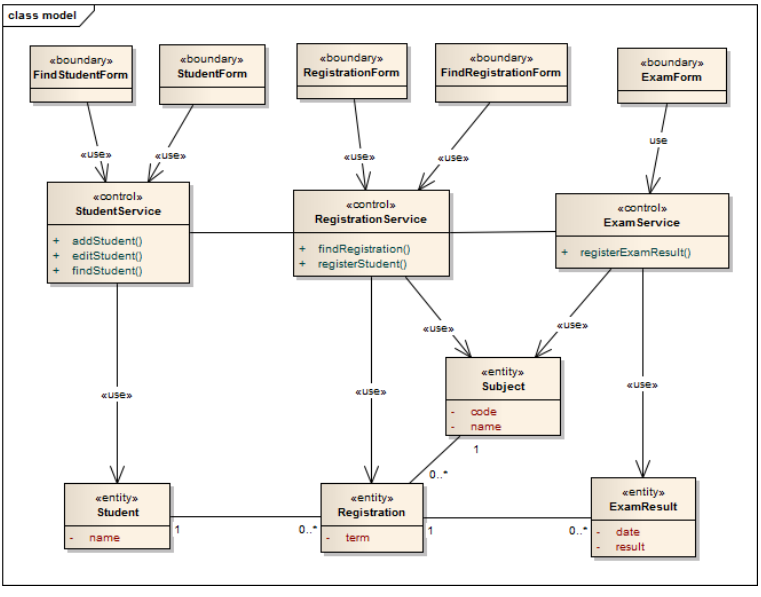

Fig. 2. Model of a part of a university information system with analysis model stereotypes

modeling. The rules for analysis class model are defined in [2]. However, for our example with simplified profile, only some of these rules are important and they can be expressed as follows:

1. Each entity class can be related by stereotype-free association only to another entity class.
2. Each control class can be related by use-stereotyped association to entity class.
3. Each control class can be related by stereotype-free association only to another control class.
4. Each boundary class can be related by use-stereotyped association to control class.

Notice, that the definition is always defined in the direction from the source class in the relation. This is to eliminate redundancy and to make each relation checked only once. If these rules are obeyed in the model, the transformation to the source code artifacts will be correct and no error shall appear.

4 Modeling constraints

Unfortunately, developers and analysts make mistakes so we need to validate and check the model before transformation and source code artifacts generation.

Therefore, the rules must be defined in some formal way so a tool such as OCLE [3] or DresdenOCL [4] can be used to validate the model against the defined rules for stereotype usage and relations restrictions.

Object Constraint Language (OCL) [8] is the part of UML used to define model constraints in a form of invariants that each instance of the element in the context must satisfy. However, definition of the required OCL invariants can be a bit tricky for unfamiliar developers. Not many developers or even analysts posses the knowledge of OCL. Therefore, we propose an approach of creating a special class diagram – we call it *constraint diagram* – to model the rules using stereotypes used in the analysis model and to generate the OCL constraints for model validation.

Definition 1. *A* constraint diagram *is a UML class diagram with classes and relations using domain-specific stereotypes used to define domain-specific rules for relations allowed between various used stereotypes and to generate OCL invariants to validate domain-specific UML class diagrams.*

In the constraint diagram, we have to model all relations allowed in the model between classes with particular stereotypes. Therefore, we add classes with defined stereotypes as required and add relations with defined stereotypes and direction. Names of the classes and multiplicity values are not important in this diagram, they just represent any instance of a particular stereotype and any relation of a particular type and stereotype in the model. Each relation in the constraint diagram stands for a rule that restricts relations between particular stereotypes. Each of them can be transformed to OCL constraint as described later in section 5.

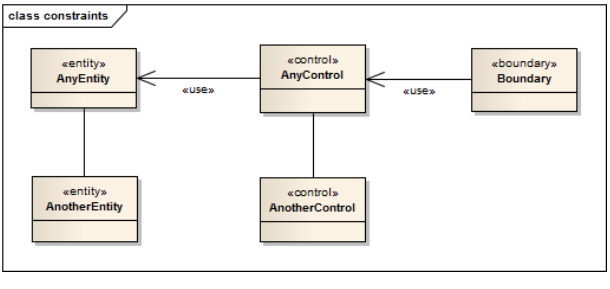

Fig. 3. Constraint diagram for the used part of analysis model profile

In Fig. 3, constraint diagram is shown for the rules defined in section 3. We have created two classes with stereotype *entity*. These represent any class with that stereotype used in the model. We add an association between *AnyEntity* and *AnotherEntity* – in this direction – with no stereotype to express that any class with stereotype *entity* can be related by stereotype-free association to any other class with stereotype *entity*. This relation stands for the rule 1.

To define rule 2 in the constraint diagram, we add a class with stereotype *control*. Then we connect it with a class with stereotype *entity* by a use-stereotyped association to express that each control class can be related to any entity by a use-stereotyped association. In our case on Fig. 3 we connected *AnyControl* to *AnyEntity* class.

Similar to the rule 1, to define rule 3 in the constraint diagram, we add another class with stereotype *control* and connect it from the class *AnyControl* to express the allowed stereotype-free association between control-stereotyped classes. Finally, similar to rule *rule:control-entity*, a class with stereotype *boundary* is added to the diagram with a connection to class *AnyControl* by a use-stereotyped association to define the rule *rule:boundary-control*.

Notice that for correct rules definition and constraint generation, some constraints for the constraint diagram must be obeyed as well. For instance, only one relation of each partial type and stereotype can be connected from each particular-stereotyped classes, otherwise there would be redundant rules defined in the diagram for the same relation and the same source artifact. Also, no directed relations can be used in the diagram. As mentioned in section 3, only outgoing relations stand for a rule to eliminate redundancies. If a relation of both directions between two various-stereotyped classes can be used in the model, two distinct relations must be created in the constraint diagram, each with the different direction – source and target classes – of the relation.

Although we have to check the constraint diagram for these rules satisfaction, we do this only for this single constraint diagram, while all the model diagrams based on the same profile with the rules defined in the constraint diagram can be validated and checked automatically by the generated constraints.

5 Generating OCL constraints

A set of OCL constraints can be generated from a constraint diagram using a tool such as Dirigent [7] to parse the model diagrams. The tool traverses the model and for each relation in the diagram, an OCL invariant is generated. To explain the constraint structure, let us define several terms used in the constraint first.

Definition 2. Source Class Stereotype *is the stereotype of the source class of the relation in the constraint diagram just being processed by the generation tool.* Target Class Stereotype *is the stereotype of the target class of that relation.* Relation Stereotype *is the stereotype of that relation.*

The structure of such constraint is shown in Fig. 4. The invariant is defined in the context of Class with its name generated from the *Source Class Stereotype*, the *RelationStereotype* and the type of the relation. Then, two functions are defined – *sts()* return a set of names of stereotypes of the classifier parameter – i.e. a class or an association –; *associationSet()* return a set of associations – that have no stereotype or include the given stereotype according to the stereotype given as parameter – from the given classifier. Then, the main constraint body is

defined – if the class in context includes the *Source Class Stereotype* then target classes of all associations with the same stereotype as the *RelationStereotype* must include the *Target Class Stereotype*.

```
context c:Class inv <SourceClassStereotype><RelationStereotype>Association:
let sts(c:Classifier) : Set(String) =
  c.stereotype->collect(name)->asSet()
let associationSet(c:Classifier, s:String) : Set(Association) =
  c.association.association.connection->
  select(isNavigable = true)->
  select(a:Association | (s = "" and sts(a)->empty())
    or sts(a)->includes(s))->asSet()
sts(self)->includes(<SourceClassStereotype>) implies
  associationSet(self, <RelationStereotype | "">)->collect(participant)->
  select(ac:Classifier | ac <> self)->
  forAll(ac:Classifier | sts(ac)->includes(<TargetClassStereotype>))
```

Fig. 4. General form of OCL invariant generated from a constraint class diagram with wildcards for *Source Class Stereotype*, *Target Class Stereotype* and *RelationStereotype*

When parsing the constraint diagram shown in Fig. 3, the tool will find four associations to generate OCL invariants – stereotype-free associations between two entities and two control classes, respectively, a use-stereotyped association from *AnyControl* class to *AnyEntity* class and a use-stereotype association from *AnyBoundary* class to *AnyControl* class. Therefore, four OCL invariants are generated as shown in Fig. 5. All generated invariants use the same functions *sts()* and *associationSet* as defined in Fig. 4, however, they are not displayed in the figure because of limited space in the paper.

Now, having these OCL invariants describing rules for the use of the profile stereotypes, we can use any tool supporting model validation using OCL constraints such as OCLE or DresdenOCL Toolkit. Using such a tool, we can validate any part of a model of a system using the same profile against the defined rules. The tool will find any classes that does not satisfy any of our invariants pointing the class is connected to some other class using wrong-stereotyped association or the target class have wrong stereotype attached.

6 Conclusions

Model-Driven Development approaches for software development became popular in last years. Many software development processes use a tool to transform models and to generate source code artifacts. Many domain-specific UML profiles are also created for various domains or software projects and generation tools are adapted to utilize these profiles during transformation and generation. However, this approach brings the need of domain rules definition of how the profile should be used.

```
-- rule 1
context c:Class inv EntityAssociation:
sts(self)->includes("entity") implies
  associationSet(self, "")->collect(participant)->
  select(ac:Classifier | ac <> self)->
  forAll(ac:Classifier | sts(ac)->includes("entity"))

-- rule 2
context c:Class inv ControlUseAssociation:
sts(self)->includes("control") implies
  associationSet(self, "use")->collect(participant)->
  select(ac:Classifier | ac <> self)->
  forAll(ac:Classifier | sts(ac)->includes("entity"))

-- rule 3
context c:Class inv ControlAssociation:
sts(self)->includes("control") implies
  associationSet(self, "")->collect(participant)->
  select(ac:Classifier | ac <> self)->
  forAll(ac:Classifier | sts(ac)->includes("control"))

-- rule 4
context c:Class inv EntityAssociation:
sts(self)->includes("boundary") implies
  associationSet(self, "use")->collect(participant)->
  select(ac:Classifier | ac <> self)->
  forAll(ac:Classifier | sts(ac)->includes("control"))
```

Fig. 5. OCL invariants for the rules to check and validate the model

In this paper, we presented an approach of modeling these domain rules for the use of user-defined stereotypes and relations between each other using UML class diagram. We presented a method how such diagram can be created for a set of example rules. We also presented a technique how to generate OCL invariants from the diagram to be used for model validation. Whole process was illustrated on an example using analysis model profile with standard class stereotypes *entity*, *control* and *boundary*.

In our further research, we would like to extend our approach to enable to model directed association constraints to validate usage of directed associations in the model, other types of relations such as generalization or dependency. Some research can also be done to define multiplicity constraints for the number of related classes using each particular stereotyped relation. Finally, extending the approach by other types of model artifacts such as actors, components or use cases can be researched.

References

1. Benevides, A.B.: A Model-based Graphical Editor for Supporting the Creation, Verification and Validation of OntoUML Conceptual Models. Ph.D. thesis, Federal University of Espírito Santo (UFES), Vitória, E.S., Brazil (Feb 2010)
2. Chae, H.S., Yeom, K., Kim, T.Y.: Specifying and validating structural constraints of analysis class models using OCL. Information and Software Technology 50(5), 436–448 (Apr 2008), http://www.sciencedirect.com/science/article/B6V0B-4NVH7T5-1/2/02217cd36c68c34c93fc63253c28bf62
3. Chiorean: OCLE 2.0 - object constraint language environment. http://lci.cs.ubbcluj.ro/ocle/index.htm (Feb 2012), http://lci.cs.ubbcluj.ro/ocle/index.htm
4. Demuth, B.: DresdenOCL. http://www.reuseware.org/index.php/DresdenOCL (Jan 2011)
5. Gogolla, M., Bohling, J., Richters, M.: Validating UML and OCL models in USE by automatic snapshot generation. Software and Systems Modeling 4(4), 386–398 (2005)
6. Guizzardi, G.: Ontological foundations for structural conceptual models. PhD thesis, University of Twente, Enschede, The Netherlands (Oct 2005), http://eprints.eemcs.utwente.nl/7146/
7. Hubl, K.: Dirigent (Jan 2012), code.google.com/p/dirigent/
8. OMG: Object constraint language, version 1.3. http://www.omg.org/spec/OCL/2.2/PDF (Feb 2010)
9. OMG: UML 2.4. http://www.omg.org/spec/UML/2.4/ (Aug 2011), http://www.omg.org/spec/UML/2.4/
10. OMG, Miller, J., Mukerji, J.: MDA guide version 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf (Jun 2003)
11. Richters, M., Gogolla, M.: Validating UML models and OCL constraints. UML 2000 - THE UNIFIED MODELING LANGUAGE, PROCEEDINGS - ADVANCING THE 1939, 265–277 (2000)
12. Richters, M., Buettner, F., Gutsche, F., Kuhlmann, M.: USE - a UML-based specification environment. http://www.db.informatik.uni-bremen.de/projects/USE/ (Jan 2011)