Visualizing OCL Constraint Patterns with VOCL

Ali Hamie
Computing Division, Brighton University, Brighton and Hove, U.K.

Keywords: Constraint Pattern, Template Package, OCL, Visualization.

Abstract: A specification pattern is a generic constraint expression that can be instantiated to solve a class of specification problems. It captures and generalizes frequently used logical expressions in models. Specification patterns have been defined and represented in the Object Constraint Language OCL for UML/OCL modelling. The notation of visual OCL (VOCL) is a visualization of OCL and can be considered as an alternative solution to the textual OCL. This paper provides the visualization of some OCL constraint patterns in VOCL. For this purpose, we introduce *constraint pattern templates* in VOCL to represent constraint patterns in a diagrammatic form. The benefits of the visualization is that some patterns will be available in a intuitive diagrammatic notation that follows the UML notation.

1 INTRODUCTION

Constraint patterns (Ackermann and Turowski, 2006; Costal et al., 2006; Wahler et al., 2006; Wahler et al., 2010) are essential to developing constraint specifications for UML class models (OMG, 2006). Using constraint patterns accelerates the development and maintenance of constraint specifications. This is because a pattern abstracts from concrete textual syntax and thus reduces typical syntactic and semantic errors by providing predefined constraint templates. Patterns also overcome the difficulty of writing constraints because class models can express complicated relations between classes, including subtyping, reflexive relations, or potentially dealing with infinitely large instances, and specifying such facts requires addressing this complexity. A pattern captures and generalizes frequently used logical expressions. It is a parameterizable constraint expression that can be instantiated to solve a class of specification problems.

In the context of UML modelling, the Object Constraint Language OCL (Warmer and Kleppe, 2003) has been used to define and represent constraint patterns. OCL is a formal specification language that was developed as an extension to the Unified Modelling Language UML . The main purpose is to describe additional constraints on UML models that are difficult to describe using the diagrammatic notation of UML. OCL is based on textual syntax and provides basic data types and a library of collection types such as sets, bags and sequences. The type of constraints that can be described in OCL include invariants on classes and types, preconditions and postconditions of operations or methods. However, modeling with UML and OCL may lead to some difficulty in that the user has to learn two different languages to represent common modelling elements such as objects and links.

The Visual OCL (VOCL) (Winkelmann, 2005; Bottoni et al., 2001) is a graphical representation of OCL capable of visualizing the textual constraints of UML models. It was developed to overcome the problem stated above. Based on the OCL meta model, VOCL follows the UML notation and its graphical representation as far as possible. This makes a direct integration of OCL in UML diagrams easier. Like OCL, VOCL is a formal, typed and object-oriented language. Since the user does not need to learn another textual language, it is claimed to be an advantage over the textual OCL. The language uses simple diagrams to represent new data types such as collections, and operations such as *forall*, *select*, *union*, etc. Logical expressions are represented as Peircian graphs using different kinds of box to express conjunctions and disjunctions.

In this paper we provide a visualization of OCL specification patterns using VOCL. The basic tool for representing and combining constraint patterns is a generic form of package, called a pattern or template package. In the package, a generic constraint framework can be defined which acts as a macro-like template that can be applied in many places. A template can contain modeling constructs in the form of both diagrams and text. In addition, any name can be written as a <*placeholder*>, which will be substituted when it is applied to a model. The process of building and composing constraint patterns using template packages will be explained. The advantage of the visualization is that some constraint patterns will be available to users in a intuitive diagrammatic notation that follows the UML notation. With the help of appropriate tools, constraint frameworks simplify the process of developing diagrammatic constraints in VOCL. The approach will be illustrated using an example.

2 VIDEO STORE MODEL

In this section, we present a partial class model for the *video rental store* system that we will use as an example throughout the remainder of this paper. Figure 1 illustrates the model of the video rental store using a UML class diagram. The main classes are VideoRentalStore, Title, Member and Rental. The video rental store has a collection of members which is represented by the association with role name members between classes VideoRentalStore and Member. The association between VideoRentalStore and Title indicates that a store has many titles and a title belongs to exactly one store. The association between Title and Rental indicates that a title has many rentals and a rental belongs to one title. The association between Member and Rental indicates that a member has many rentals and a rental is only for one member. In addition the diagram shows a number of attributes for each class.

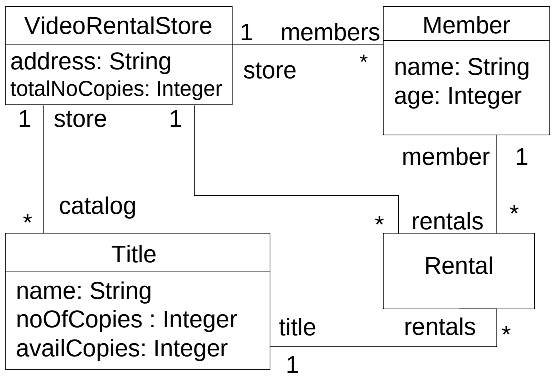

Figure 1: A partial class model for video rental store.

3 CONSTRAINTS AND PATTERNS IN VOCL

An OCL constraint is visualized as a rounded rectangle with two sections, the section of the context and the section of the body which can contain a condition. The context section includes the keyword *context* followed by the model element which can either be a class, type or operation, of the constraint followed by the kind of the constraint e.g. *inv*, *pre*, *post* or *def*. In the body section, the body of the constraint is visualized. The condition section contains the conditions of the constraint declared using variables defined in the body. If there is a condition section it is separated from the rest of the body by a dashed line. In order to represent constraint patterns we use a generic, or *template*, form of package. Inside the template, some types and their features can be defined using placeholder names. These names can be substituted for actual values. the package is designed to be imported with substitutions. Unfolding the package provides a version of its contents that is specialized based on the substitution made.

3.1 Attribute Value Restriction

Consider the constraint that each member in the video store should be over 18 years old. This can be specified textually in OCL as follows.

```
context Member inv :
self.age > 18
```

The first line declares the context of the constraint Member followed by the type of the constraint inv, indicating that the constraint is an invariant. The second line is the actual invariant represented as an OCL boolean expression using the variable self that refers to an object of class Member.

The visualization of this simple constraint in VOCL is shown in Figure 2. The main difference here is that the object self of class member has been visualized using UML as a rectangular box. The context is specified like in OCL.

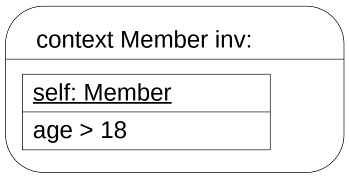

Figure 2: Properties of an object attribute.

Restricting the value of an attribute is a common kind of constraint. Abstractly, the above constraint restricts the value of an attribute of the context class (age) to be greater than a given value (18). We therefore generalize this constraint by introducing the *Attribute Value Restriction* pattern, which can be used to restrict the values of attributes for all instances of the attributes'class. We define it as VOCL template package, as shown in Figure 3. The template has place holders (parameters) that specify classes and objects that can be substituted when the pattern is instantiated in a particular context. For this pattern the parameters are the class C, the attribute att, the comparison operator operator and the value value. Placeholders within the constraint template are written with angle brackets (<>). An alternative way is to include the parameters after the name of the pattern.

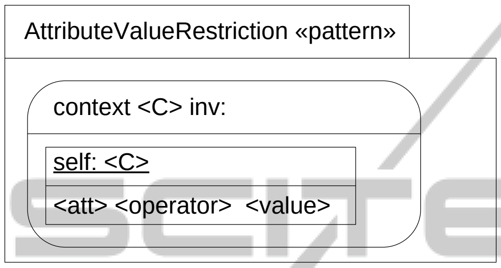

Figure 3: Properties of an object attribute pattern.

Now the constraint that members must be over 18 can easily be generated from a *pattern application* (see Figure 4). The place holders inside the pattern definition are identified with actual type names when the pattern is applied. This is the effect of the labeled arrows when the pattern is applied. The example also shows name substitution in the form [pattern-name\applied-name], we have used to substitute values for the attribute att and parameters value and operator. This text form and the arrows are equivalent. It is sometimes convenient to write instead of drawing pictures:

```
AttributeValueRestriction[C\Member, att\age,
                          value\18, operator\ >]
```

This indicates that Member, age, 18 and > are substituted for C, att, value and operator respectively.

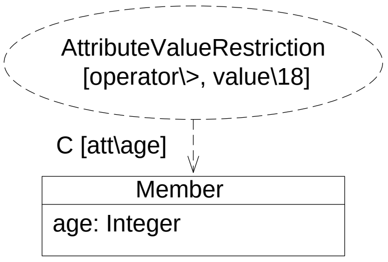

Figure 4: Example of applying a constraint pattern.

In the resulting constraint, the context will be a predefined class or type with features appearing in the body, the condition also includes features defined by the pattern, as name-substituted by the application. Working out the complete constraint is called *unfolding*. A good tool can show an unfolded version on demand.

In the context of OCL a constraint pattern named AttributeValueRestriction is defined in (Wahler et al., 2010) to capture this kind of constraints. The pattern is defined as an OCL template as follows.

```
pattern AttributeValueRestriction
        (property: Property,
         operator, value: OclExpression) =
self.property operator value
```

The pattern has three parameters. property represents the attribute of the object, operator represents the comparison operator, and value represents the actual value.

Using the pattern, we can express the constraint in Figure 2 as follows.

```
context Member inv :
AttributeValueRestriction(age, >, 18)
```

The VOCL pattern can be regarded as a visualization of the OCL constraint pattern. The only difference is that the OCL pattern does not include the context of the constraint whereas the VOCL pattern does. However, it is possible to formulate the OCL pattern by including an additional parameter for the context of the constraint.

3.2 Restricting the Multiplicity of Associations

The multiplicities of properties (associations) can only be roughly constrained in a diagrammatical way in class models. However, there are situation where the multiplicity of an association depends on the value of an attribute. For example, an object of class Title can have an arbitrary number of rentals which cannot exceed the total number of copies for that title. So there is a dependency between the association with role name rentals and the attribute noOfCopies. The following OCL constraint captures this dependency.

```
context Title inv RentalsRestriction :
 self.rentals->size() <= noOfCopies
```

The above constraint can be visualized in VOCL as shown in Figure 5. This constraint specifies that a title has less than noOfCopies rentals. In this constraint the OCL operation size is visualized and applied to a collection (in this example a set) of rentals. The variable n contains the number of elements in this collection.

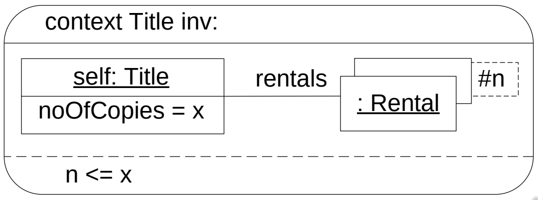

Figure 5: Restricting association multiplicity.

This constraint can be represented as an instance of the *Multiplicity Restriction* pattern. This pattern restricts the multiplicity of associations. The multiplicity of associations can be restricted in UML class models, however this pattern allows model developers to define multiplicity restrictions that depend on properties of the model instance, e.g. an attribute value. The pattern is defined by the template in Figure 6.

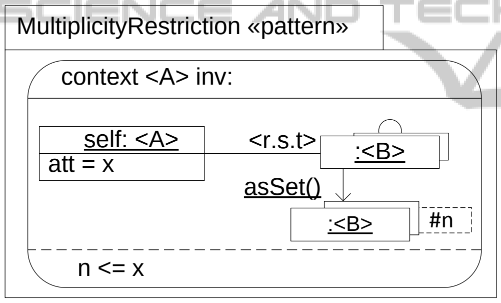

Figure 6: Restricting association multiplicity pattern.

This pattern has four parameters: A, B, representing classes, att, attribute of the context class and <r.s.t>, which represents a sequence of properties, thus allowing the use of expressions such as self.catalog.rentals. Since in OCL the navigation of more than one association may result in a bag, the OCL operator asSet() is used to convert the resulting collection into a set.

In some cases, there may not be a direct link between the object self and an object of class B, however the the link can be regarded as a derived link. This is the case when the navigation uses more than one association.

In general, when you build a pattern one must make certain assumptions about the things that are substituted for the placeholders in order for the application of the pattern to work as intended. For example, to apply the pattern in Figure 6, as a prerequisite, the classes substituted for <A> and <B> should be linked by an association with appropriate role name and multiplicity; if they do not, they are not suited for the MultiplicityRestriction template.

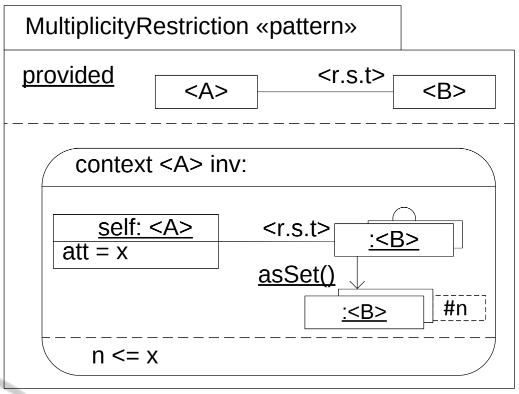

Figure 7: Explicit substitution provisions in MultiplicityRestriction pattern.

In a separate section of the package we provide information for the modeler of a template to state the conditions under which the pattern can be meaningfully applied. Figure 7 shows an improvement of MultiplicityRestriction template. It says that if you have two types reachable via a sequence of associations, then it is OK to say that the navigation results in a collection with restricted size.

In the precondition section, one can put any model to which the substituted types must *already* conform. Thus, one can require that substituted types have some relationship or satisfy some predicate. When the pattern is applied one must check, perhaps with the help of a tool, that all other parts of his or her model imply the properties laid down as preconditions.

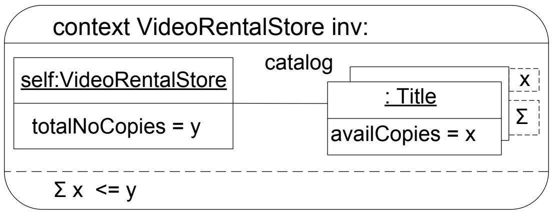

Figure 8: Sum restriction constraint.

3.3 Attribute Sum Restriction

In the video rental store model the sum of the available copies of all titles must not exceed the total number of copies in the store. This dependency between the attributes totalNoCopies and availCopies is captured by the following OCL invariant.

```
context VideoRentalStore inv :
self.catalog.availCopies−>sum()
                <= self.totalNoCopies
```

This can be visualized in VOCL by the diagram in Figure 8. In this case the operation sum has a frame in which the element whose values are summed up, is visualized. This element is depicted at the frame above the $\sum$ symbol. To refer to the result of the sum in the condition section, the $\sum$ symbol is used.

To capture this constraint, we define the *Attribute Sum Restriction* pattern, which has five parameters. Besides the parameters A, B, which represent classes, and r.s.t, which denotes a path expression to a related class, this pattern has two parameters. Parameter summation refers to the property in the context class that denotes the value that must not be exceeded, and summand refers to the property in the related class that is accumulated. The pattern is shown in Figure 9.

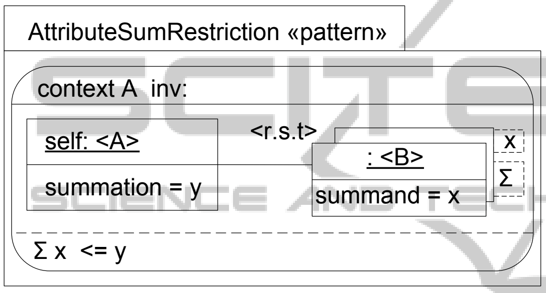

Figure 9: Attribute sum restriction pattern.

3.4 Uniqueness Constraints

The unique identity constraints are very frequent in modelling. For example, it is required that each title in the video rental store has a unique name in order to distinguish one title from another. Such constraint can be expressed in OCL by using the class operation allInstances that returns the set of existing instances of the class. It is given as follows.

```
context Title inv UniqueName :
    Title.allInstances−>forAll(t1,t2: Title |
        t1 <> t2 implies t1.name <> t2.name)
```

This constraint is visualized in VOCL by the diagram given in Figure 10. The feature allInstances results in a set of all instances of the type which exists at the specific time when the expression is evaluated. It is visualized by a set of type of the class name.

In the constraint above the navigation starts at this set, since there is no instance self from which it could be started. After this a forall operation is applied to that set and inside the forall an implies operation. forall operation is defined on collections and has one or two iterators. It has a frame which contains the expression which has to be true for each collection element. By isIn all instances of the set of all titles can be accessed.

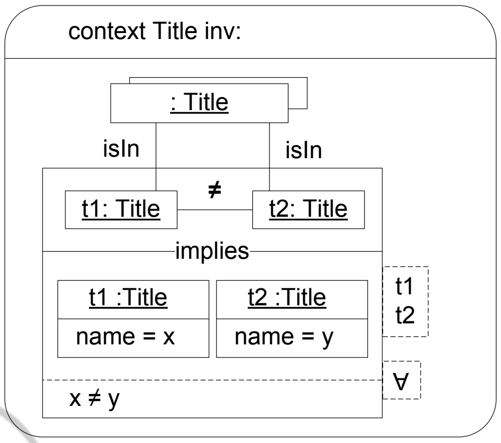

Figure 10: Unique identity constraint.

These instances correspond to the iterator/iterators. The constraint specifies that all instances of type Title have unique names. The link labeled isIn visualizes that single objects are contained in the collection.

By abstracting from the above constraint a visual pattern is obtained and shown in Figure 11. The pattern has a parameter for the context class and another for the property or attribute that has to be unique.

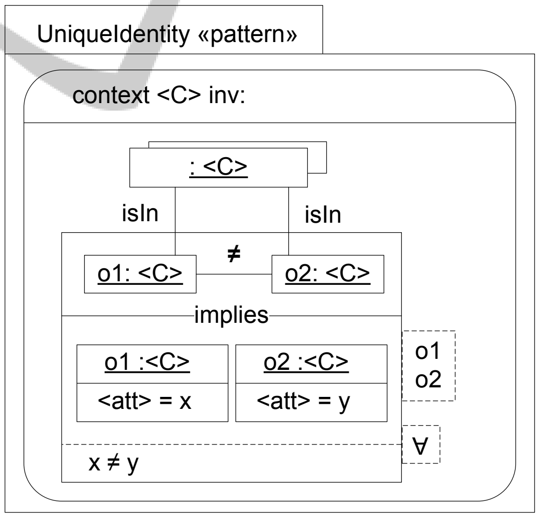

Figure 11: Unique identity constraint pattern.

3.5 Composing Patterns

Associations in class models represent relations between the associated classes, therefore they can have properties such as *injectivity*, *surjectivity* and *bijectivity*. These properties can be specified on class diagrams by appropriate multiplicity constraints. However, there are situation where these properties depend on some model attributes making it necessary to write textual constraints. Patterns for injective, surjective and bijective associations have been introduced in (Wahler et al., 2010) as OCL templates. Figure 12 shows a visualization of a modified formulation of the *Injective Association* pattern.

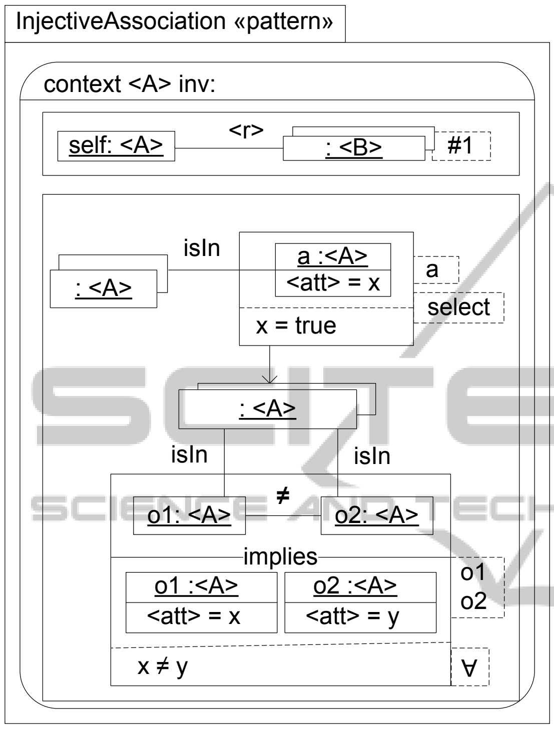

Figure 12: Injective pattern.

The pattern uses two subexpressions which are visualized below each other, which in VOCL are automatically combined by and.

Assuming that a pattern called *Surjective Association* has been defined, then the *Bijective Association* pattern can be defined in terms of the *injectivity* and *surjectivity* patterns as shown in Figure 13. This corresponds to the definition that a bijective association is both injective and surjective.

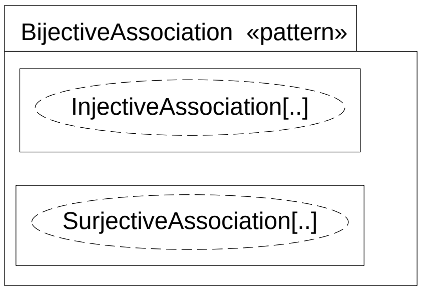

Figure 13: Bijective pattern.

4 CONCLUSIONS

In this paper we provided a visualization of OCL constraint patterns using the VOCL notation. The patterns are represented as generic packages containing placeholder definitions. A placeholder is a name that can be substituted when the pattern is used. Each use or application of the pattern provides its own substitutions of the placeholders. Placeholder names are distinguished with angle brackets (<>). The names of attributes and associations of placeholder classes are themselves placeholders.

The benefit of the visualization is that for some patterns it provides an intuitive way for representing them within the UML. However there will be situations where the textual representation of a pattern is simpler than its diagrammatic representation. The pattern-based approach helps to avoid syntactic and structural errors because the developer can generate VOCL diagrams instead of drawing them by hand. Further research based on case studies is needed to evaluate this approach and compare the visual and textual representations with respect to expressiveness and readability.

REFERENCES

Ackermann, J. and Turowski, K. (2006). A library of OCL specification patterns for behavioral specification of software components. *Lecture Notes in Computer Science*, 4001/2006:255–269.

Bottoni, P., Koch, M., Parisi-Presicce, F., and Taentzer, G. (2001). A visualization of OCL using collaborations. In *<<UML>>'01, 4th International Conference on the Unified Modelling Language*, pages 257–271. Springer-Verlag.

Costal, D., Gómez, C., Queralt, A., Raventós, R., and Teniente, E. (2006). Facilitating the definition of general constraints in UML. In *MoDELS*, volume 4199 of *Lecture Notes in Computer Science*, pages 260–274. Springer.

OMG (2006). *Unified Modeling Language Specification 2.0: Infrastructure*. OMG doc. smsc/06-02-06.

Wahler, M., Basin, D., Brucker, A. D., and Koehler, J. (2010). Efficient Analysis of Pattern-Based Constraint Specifications. *Software and Systems Modeling*, 9(2):225–255.

Wahler, M., Koehler, J., and Brucker, A. D. (2006). Model-driven constraint engineering. *Electronic Communications of the EASST*, 5.

Warmer, J. and Kleppe, A. (2003). *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley, Reading, MA.

Winkelmann, J. (2005). Specifcation of visualOCL: A visualisation of the object constraint language. Master's thesis, TU Berlin. (in German).