Defining OCL constraints for the Proxy Design Pattern Profile

N. C. Debnath
Winona State University
Department of Computer Science
Winona, MN 55987 USA
ndebnath@winona.edu

A. Garis, D. Riesco, G. Montejano
Universidad Nacional de San Luis
Ejército de los Andes 950
(5700) San Luis-Argentina
{agaris, driesco, gmonte}@unsl.edu.ar

Abstract

Profiles allow extend UML vocabulary and the design patterns define a common vocabulary for software designers, therefore it is possible to use profile to define a pattern vocabulary in UML. If profiles are used to represent patterns then it is not required to define a special notation neither a particular CASE tool for patterns (UML tool is used).
Three mechanisms are in the profiles: stereotypes, tag values and constraints. Stereotypes extend the UML vocabulary and it is possible to associate to it tag values and constraints. When these elements are introduced in models; patterns can be clearly visualized, software developers improve communication and establish a common vocabulary. Also profiles allow add information to the model to transform it to other models.
OCL (Object Constraint Language) constraints are semantic restrictions added to UML elements. This work shows a way in which OCL constraints are used to define semantic restrictions over stereotypes included in a profile of pattern. The definition of OCL constraints for proxy design pattern is shown as an example of our proposal. An interaction between users and UML tool is proposed for solving the loss generality when OCL constrains are imposed.

Keywords: UML Profiles, OCL, Design Patterns

1. Introduction

Sometimes design patterns are instantiated in UML (Unified Modeling Language) design models.
Because often UML is not enough expressive to model the semantic of a particular design pattern, it is required to extend it.
UML Profiles extend its syntax and semantic in order to model specific elements of particular domains. They provide three elements for this purpose: stereotypes, tag values and constraints. Stereotypes extend the UML vocabulary and it is possible to associate to it constraints. Constraints are semantic restrictions added to UML elements.

Profile features are used in the Design Pattern context: As profiles extend UML vocabulary, it is possible to use profile to define a pattern vocabulary in UML. One of the reasons for defining profiles is that they add information to the model to transform it to other models or to code. Other is that if UML tools are profile aware is not required to define a specific tool for patterns.
When a profile of a pattern is defined, constraints are associated to stereotypes. If constraints are attached, later all model elements associated it must conform its restrictions.
UML allows using natural language for defining constraints; however Object Constraint Language (OCL) is a best option because it is a more precise language.
The verification of OCL constraints with a UML tool can be done combining two check techniques: delay consistency check and on-line consistency check. The first refer about OCL specifications that they may be performed after the model is completed. The seconds verify OCL specifications while the stereotypes are introduced in a model.
The profile of a pattern should be general enough to describe the essential spirit of the pattern. If the definition of a pattern is too restricted, the benefits of using pattern could be loss when it is instantiated in different situations. Levels of abstraction for a pattern can be specified with OCL constraints. An approach for this is presented in this work.
We illustrate a way in which OCL constraints are used in a profile of pattern**.** The definition of OCL constraints for proxy design pattern is shown as an example of our proposal.
This work is structured as following. In the next section, "UML profiles" and "OCL" are introduced. Later, specification of design patterns is explained and specification of design patterns with UML profiles too. Afterwards, the advance for defining OCL constraints in "Proxy Pattern Profile" is described. Finally, the conclusions are presented.

2. UML Profiles

The UML infrastructure is composed of different conceptual levels [1]. In the M0 level there are *instances*

of a particular real system. In the M1 level there are *models* of particular systems and its instances are M0 elements. M2 level is *the model of the model* and its elements are the languages for Modeling (its instances are M1 elements; for example, "Person" is an instance of the "Class" element). M3 level is the *model of M2 model*. Profiles change UML metamodel in M2.

A package "Profile" [1] (see Figure 1) is defined in UML version 2.0. Three mechanisms are included in the package: stereotypes, tag values and constraints. The UML semantic is adapted to particular requirements without changing the UML metamodel.

UML Profiles extend its syntax and semantic in order to model specific elements of particular domains or platform. In UML version 2.0 the package "Profile" [1] (see Figure 1) is defined: Stereotypes, tag values and constraints are included in this package.

Stereotypes allow extending the UML vocabulary. The specification of a stereotype is the association of itself with an element of the metamodel (class, attribute, operation). The way for denoting it is <<stereotype-name>>. It is possible to associate to it tag values and constraints.

Tag values associate attributes to elements extended by the profile. It is denoted by a pair with the attribute name and its value associated, i.e, {name=value}.

Constraints are semantic restrictions added to UML elements. If constraints are attached to a stereotype, later all model elements associated must verify the stereotype's restrictions. Natural language or OCL can be used for defining constraints however the second option is the best option because it have more precision.

2.1. OCL Constraints are often described in natural language but usually this result in ambiguities. OCL is a formal language for the specification of constraints. It allows writing unambiguous constraints. It is not only usable to persons with a strong mathematical background, it is easy to read and write [2]. It can be used to specify constraints in the models and the UML metamodel. Different purposes for which OCL can be used are described in [2]:

"• As a query language

• To specify invariants on classes and types in the class model

• To specify type invariant for Stereotypes

• To describe pre- and post conditions on Operations and Methods

• To describe Guards

• To specify target (sets) for messages and actions

• To specify constraints on operations

• To specify derivation rules for attributes for any expression over a UML model."

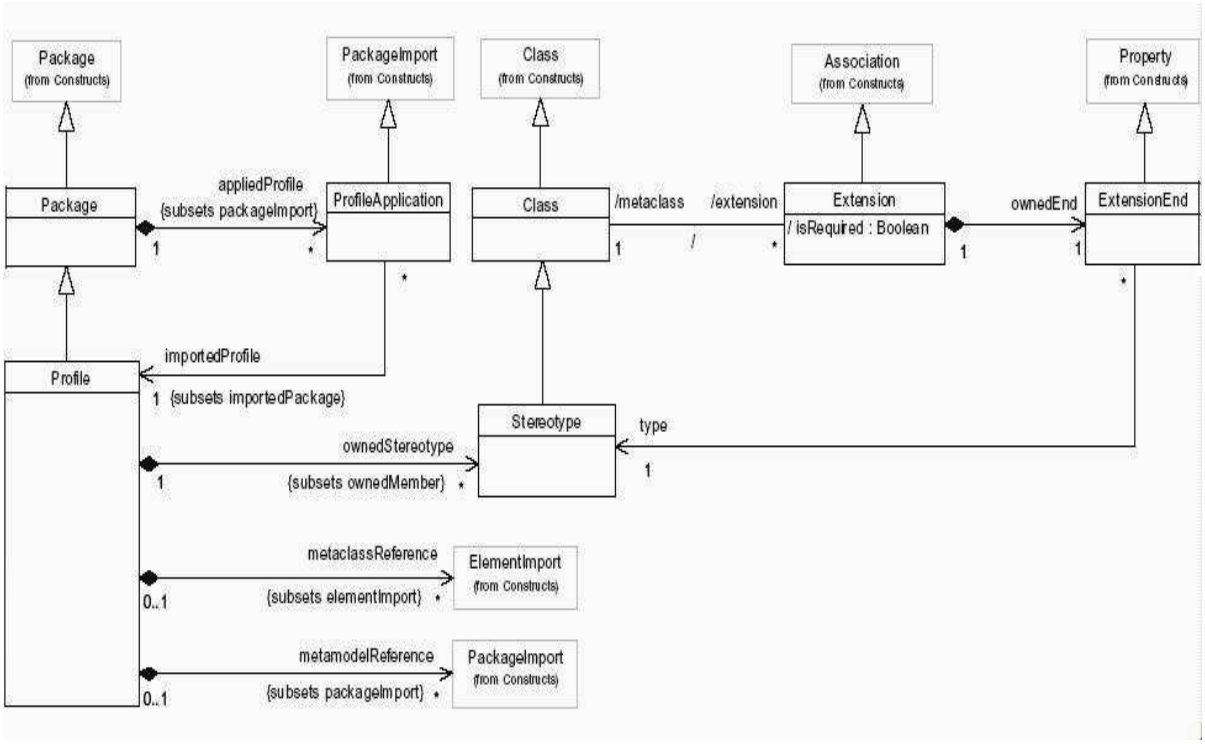

Figure 1. Profile Package

3. Specification of design patterns

There are different approaches to specify design patterns. Formal specification is used in [6], and [3]. Lano and Goldsack in [6] propose the formal prove to refine transformations of patterns. Smith and Stotts in [3] use a formal language to express design patterns; this specification can be used to analyze the source code and high level of abstraction too.

Florijn, Meijers and Van Winsen in [5] present a tool for using patterns in different ways. The tool allow, between other things, to check whether the occurrence of a pattern is consistent with the invariants governing this pattern.

Lauder and Kent in [6] apply a visual notation. They divide the specification of patterns into three models: 1) an abstract model which catches the pattern's essence, 2) a model which refines the abstract model (adding domain specific constraints), and 3) a model which documents a concrete deployment.

In [7] Le Guennec, Sunyé and Jézéquel modificate UML 1.3 metamodel. They use meta-level collaborations and OCL constraints (instead of parameterized collaborations). France, Kim and Song in [8] define a metamodeling language (using UML and OCL constraints), specifying perspectives of design patterns.

OMG considers in [9] that rules that govern a business concept can be represented with a pattern with constraints. First templates with constraints are defined for representing a pattern, later it is bound in a model using collaborations.

In [10] and [11] we showed that defining a profile for each pattern is an option for the specification of design patterns. Many are the advantages of defining profiles for particular domains; such as to add information to the model to transform it to other models or to code, and extending semantic of the UML metamodel. Other is that if UML tools are profile aware is not required to define a specific tool for patterns. Although profiles were often used for defining specific domains, they can be used for general domains as the pattern definition is. If stereotypes are introduced into models, software developers can see clearly the pattern that they use, to improve communication with their colleagues and to establish a common vocabulary.

3.1. Defining design patterns with UML profiles

As was explained in the paragraph, the specification of design patterns using UML profile was proposed in [10] and [11]. The specification of each pattern is incorporated within architecture for patterns.

A general structure for defining patterns is give by Profiles. A hierarchy between levels of profiles (see Figure 2) is imposed allowing the reuse of definitions. In the bottom level is the "Profile Package" of OMG standard, following with a "Design Pattern Framework Profile" (DPFP) and a level of "classes of patterns". Definition of particular patterns is on the top.

Common features to all pattern profiles are specified in DPFP and the level of "classes of patterns" is inspired in one of the most popular catalogs of design pattern (see [12]).

If a new pattern wants to be included, a new profile should be defined. All patterns will respect the same generic structure of some of "classes of patterns" but each profile will have its own particular semantic.

To define a profile for a design pattern, four steps should be followed. 1) to identify the main participants of the pattern, 2) to identify its responsibilities, 3) begin the

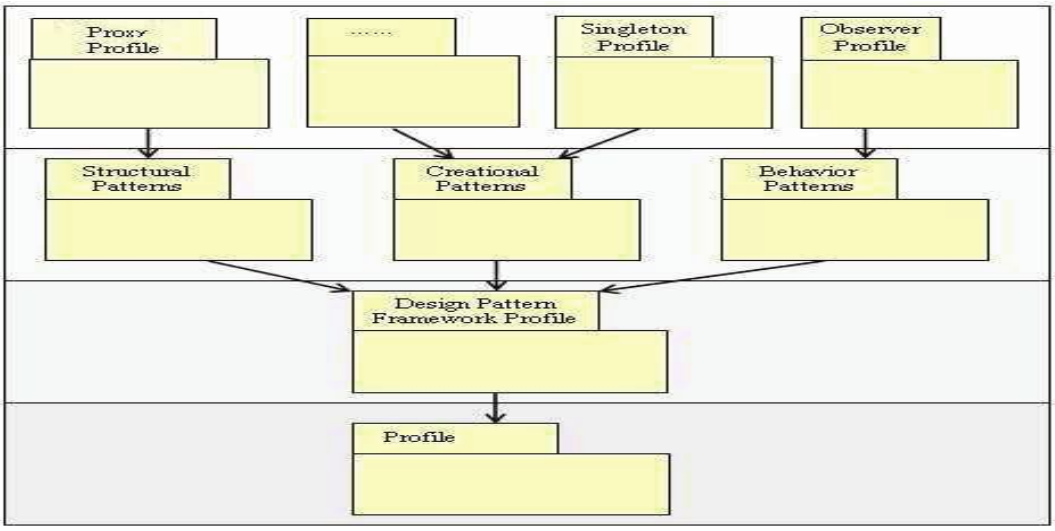

Figure 2. Architecture for patterns using UML profiles

profile specification, including a stereotype for each participant of the pattern, 4) to associate OCL constraints to each stereotype.
OCL constraints could impose too restrictions to definition of a pattern. If this happen, the benefits of using pattern could be loss because it is not could be instantiated in different situations. In next section a proposal for this problem is presented.
In the next section the complete propose of this work is shown. It illustrates a way in which OCL constraints are used in a profile of pattern**.** The definition of OCL constraints for proxy design pattern is shown as an example of our advance.

4. Defining OCL constraints in "Proxy Pattern Profile"

Suppose that the "Proxy pattern"[1] is instantiated for solve a design problem. A document editor requires the capability of including graphics in a document. The creation of graphic objects is an expensive operation, but the opening of a document should be fast. A solution can be to define a "Graphic" superclass, an "Image" and an "ImageProxy" class, and use lazy initialization. Image proxy creates the real image only when the document editor needs to display it (invoking its "Draw" operation). It is possible to introduce stereotypes in the model, defining a profile for the proxy pattern (see Figure 3).
UML profile Proxy Pattern is explained following. The main participants of the pattern are the class "Proxy", the class "Subject", the class "RealSubject" and the rol "realSubject" in the association between "RealSubject" and "Proxy". About responsibilities of participants; "Proxy" contains a reference (through the rol "realSubject") for accessing to "RealSubject", "Subject" defines interface for "Proxy" and "RealSubject". Also "RealSubject" specifies the real object for "Proxy" represented. For beginning "Proxy Profile" specification, a stereotype for each participant should be defined. According to responsibility's pattern, stereotype is associated with a UML metaelement and the corresponding semantics constraints are imposed.
In the example Figure 3, Proxy pattern is used defining an interface ("Subject"), and "RealSubject" and "Proxy" implement it. But "Subject" may be declared as superclass, and "Proxy" and "RealSubject" as it children. Then "Subject" can be an interface or a superclass. Because this reason, stereotype "Subject" is associated with the classifier metaelement, UML profile corresponding to Proxy Pattern is shown in Figure 4,

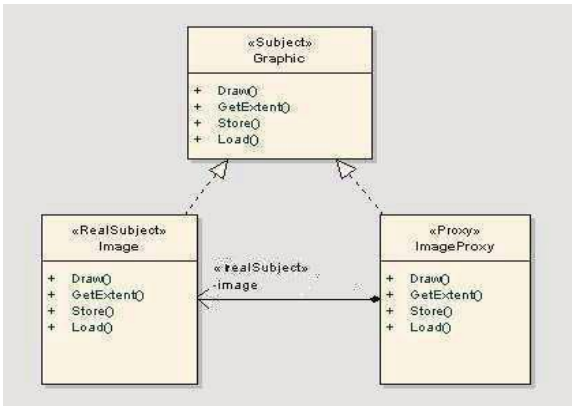

Figure 3. A model using stereotypes of Proxy Pattern

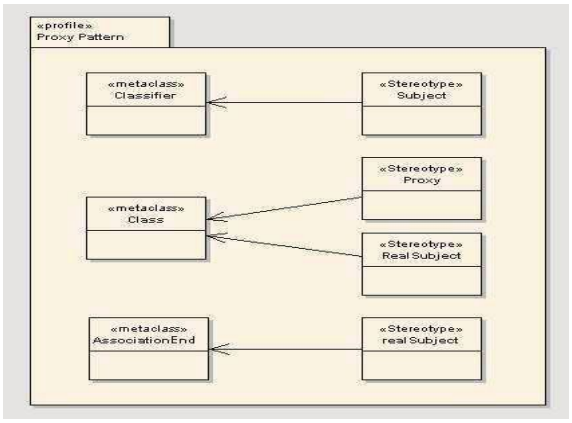

Figure 4**.** Proxy Profile

OCL constraints over stereotypes are established under it. In the following paragraphs, there are semantic constraints of the profile Proxy Pattern.

-- 1 "Subject" is interface or abstract classes

```
context Subject inv:
      (self.base.isTypeOf(Class) and self.base.isAbstract)
      or   (self.base.isTypeOf(Interface))

    --  If "Subject" is interface
```

-- 2 "Proxy" class implements an interface stereotyped "Subject"

```
context Proxy inv:
      (self.base.isTypeOf(Interface)) implies
 Interface.allInstances->exists (i|
 i.isStereotyped("Subject") and
                                  self.base.ownedOperation
->includesAll(i.ownedOperation) )
```

[1] Gamma E., Helm R., Johnson R., Vlissides J., "Design Patterns. Elements of Reusable Object-Oriented Software", pag. 207.

```
-- 3 “RealSubject” class implements an interface stereotyped “Subject”

context RealSubject inv:
    (self.base.isTypeOf(Interface)) implies
            Interface.allInstances->exists (i|
                    i.isStereotyped(“Subject”) and
                     self.base.ownedOperation
                    ->includesAll(i.ownedOperation) )

     -- If “Subject” is abstract class
--  4  “Proxy” have a father stereotyped “Subject”

 context Proxy inv:
                (self.base.isTypeOf(Class) and
      self.base.isAbstract) implies
                self.base.superClass
->exists( p|p.isSterotyped (“Subject”) )

--  5 “Proxy” defines all methods, which are defined in its father (“subject”).

 context Proxy inv:
                (self.base.isTypeOf(Class) and
      self.base.isAbstract) implies
            self.base.superClass->
                select( c|c.isSterotyped(“Subject”)
      ).allMethods
            -> forall( m|m.specification.isAbstract=true)
implies
    self.base.allMethods->exists( m2|
                m2.isAbstract=false and m2.name=m.name
and
 m.ownedParameter->intersection(m2.ownedParameter)
  ->isEmpty
            )
-- 6 “RealSubject” have a father stereotyped “Subject”

context RealSubject inv:
                (self.base.isTypeOf(Class) and
      self.base.isAbstract) implies
                        self.base.superClass->exists(
p|p.isSterotyped (“Subject”) )

-- 7  “RealSubject” defines all methods, which are defined in its father (“subject”).

context RealSubject inv:
                (self.base.isTypeOf(Class) and
      self.base.isAbstract) implies
          self.base.superClass->
                  select ( c|c.isSterotyped(“RealSubject”)
      ).allMethods
            -> exists ( m|m.specification.isAbstract=true)
implies
    self.base.allMethods->exists ( m2|
                m2.isAbstract=false and m2.name=m.name
and
                m.ownedParameter->
intersection(m2.ownedParameter)->isEmpty   )

-- 8  “realSubjet” is a role relation between a stereotyped “Proxy” element and a stereotyped

context realSubject inv:
  self.base.participant.isSterotyped(“RealSubject”)   and
  self.base.participant.association.participant
    ->exist(c|c.isStereotyped(“Proxy”)                  and
c.aggregation=#composite)
```

It is possible to check these constraints on elements of a particular model. It will use stereotypes of the profile. Therefore, the model is correct according the syntactic and semantic rules by the profile established.

The verification of OCL constraints with a UML tool can be done combining check techniques. 1) Delay consistency check, 2) on-line consistency check. The first refer about OCL specifications that they may be performed after the model is completed. The seconds talk about OCL specifications should be checked while the stereotypes are introduced in a model. For example, constraint 8, can be checked while stereotype is introduced. But the constraints 2 and 3 may be better performing them after the model is finished.

Pattern could loss generality if OCL constraints impose restrictions too. An approach for this may be as following. An interaction between users and UML tool is performed. When the tool verifies a constraint which is not fulfilled, it warns to user showing the violated constrain. User decides if the constraint is accepted (adding more restrictions to model and giving less generality to pattern) or not accepted. OCL constrains are used by UML tool for explicating to software engineer when pattern is not fulfilled.

A main constraints set will give to pattern its essential spirit; these minimum semantic requirements will be the more abstract pattern. If new constraints may be verified, more restrictions to patterns are added, so it is less general. A constraint within these main constraints is an OCL operation in the DPFP level. The “isElementsOfPattern” is defined to check if all the required stereotypes of a particular profile are in a design model.

```
context
Core::Profile::isElementsOfPattern(m:ModelElement):Bo
olean
isElementsOfPattern =
   self.ownedStereotype->forAll(s | m.model->
     exists( comp| comp= s.name) ) )
```

In the same way as "Proxy Pattern" has been defined, others profiles of patterns can be defined too. All this profiles will make a library of patterns.

5. Conclusions

Although UML profiles were often used for defining specific domains, they can be used for general domains as the pattern definition is. Stereotypes can be introduced in UML models to see clearly a pattern used and to establish a common vocabulary between software developers.

If OCL constraints are attached to a stereotype, all model elements associated it must verify its restrictions. This work shows how OCL constraints are used to define semantic restrictions over stereotypes. The definition of OCL constraints for proxy design pattern is shown as an example of our proposal. Previously, specification of design patterns with UML profiles is explained.

We show a technique for the verification of OCL constraints. A delay consistency check and on-line consistency check technique are combined. Also, we describe an approach for not losing the essential spirit of the pattern when constraints are imposed. OCL constrains are used by UML tool for explicating to software engineer when pattern is not fulfilled.

6. References

[1] UML 2.0 Infraestructure Specification, http://www.omg.org/technology/ documents, August 2006.

[2] UML 2.0 OCL Specification, http://www.omg.org, August 2006.

[3] K. Lano, J. Bicarregui, S. Goldsack, "Formalising Design Patterns", *In BCS Northern Formal Methods Workshop, EWIC Series*, 1997.

[4] J. Smith, D. Stotts, "Elemental Design Patterns: A Formal Semantics for Composition of OO Software Architecture", *In 27th Annual IEEE/NASA Software Engineering Workshop*, December 2002.

[5] G. Florijn, M. Meijers, P. Van Winsen, "Tool Support for Object-Oriented Patterns", *In ECOOP'97, 11th European Conference on Object-Oriented Programming*, Vol. 1241 LNCS, pp. 472-495, June 1997.

[6] A. Lauder, S. Kent, "Precise Visual Specification of Design Patterns", *ECOOP'98, 12th European Conference on Object-Oriented Programming*, Vol.1445 LNCS, pp. 230-236, July 1998.

[7] A. Le Guennec, G. Sunyé, J. Jézéquel, "Precise Modeling of Design Patterns", *In UML 2000*, Vol. 1939 LNCS, pp. 482-496, 2000.

[8] R. France, D. Kim, E. Song, "A UML-Based Pattern Specification Technique", *IEEE Transactions on Software Engineering*, Vol.30(3), pp.193-206, 2004.

[9] OMG, "UML Profile for Patterns Specification". http://www.omg.org/technology/ documents, 2004.

[10] N. Debnath, A. Garis, D. Riesco, G. Montejano, "UML Profiles for Design Patterns", *ISCA 20th Internacional Conference on Computers and their Applications*, pp 435, March 2005.

[11] N. Debnath, A. Garis, D. Riesco, G. Montejano, "Defining Patterns using UML Profiles". *ACS/IEEE International Conference on Computer Systems and Applications*, IEEE Press, www.ieee.org, March 2006.

[12] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns, Elements of Reusable Object-Oriented Software", *Addison-Wesley*. 1995.