Proc. 5th Conf. Unified Modeling Language (UML'2002).
J.-M. Jezequel, H. Hussmann, S. Cook (Eds.), Springer, Berlin, LNCS.

Analysis of UML Stereotypes within the UML Metamodel

Martin Gogolla Brian Henderson-Sellers

University of Bremen University of Technology, Sydney

Abstract. Stereotypes are a powerful and potentially expressive extension mechanism in the Unified Modeling Language (UML). However, it seems that stereotypes are difficult to handle because using stereotypes needs an understanding of the UML metamodel and, in particular, an understanding of OCL constraints. Stereotypes are often applied in a wrong or at least sloppy way without proper declaration. There are also differences between the various versions of UML with respect to subtle details in the stereotype part. A graphical syntax for stereotypes including examples has been introduced only late in UML 1.4. Other difficulties are that constraints are used in the stereotype context in two completely different ways and that no full support of stereotypes is yet offered by tools. The paper points out these difficulties in detail, analyses the UML metamodel part dealing with stereotypes, and makes various suggestions for improving the definition and use of stereotypes.

1 Introduction

The Unified Modeling Language or UML [OMG99,OMG01] allows for three extension mechanisms: tagged values, constraints and stereotypes. This paper concentrates on the latter two. The basic idea of the paper is to study UML stereotypes in detail. We want to point out some difficulties in defining and using stereotypes and employ the Object Constraint Language OCL [WK98,RG01,GR01] to define a precise meaning for the introduced stereotypes. Interestingly, up to now there seems to be little, if any, work on the formal definition and analysis of stereotypes, especially in connection with OCL. Our discussion also reveals that the features of the UML metamodel itself are not sufficient for the comprehensive definition of such stereotypes. One additionally needs a language to define transformations on UML metamodel diagrams in order to deal with situations where generic definitions are needed.

Our ideas have connections to other related work. In the context of object-orientation, the name stereotype was first invoked in [WB94] where stereotypes were used to help in thinking about and creating objects. Metamodels of object-oriented languages with stereotypes have not only been studied for UML [OMG01], but also for example for OML [HSFG97]. General considerations and classifications of UML stereotypes have been discussed in [BGJ99,DSB99]. A general criticism of UML stereotypes can be found in [AKHS00,HS02]. UML-related questions concerning application-dependent stereotypes have been studied in [SW01]. The use of stereotypes covers a wide range; among the many

application areas we only mention web design [Con99], geographical information systems [PL01], multimedia [SE99] and telecommunications [KMP+98]. The present paper takes up ideas from [Gog01] where stereotypes have been introduced for relational database design.
The structure of the rest of the paper is as follows. Section 2 discusses the stereotype part of the UML metamodel. Section 3 analyses the possibilities of that metamodel by studying selected examples. For each example, the impact on the UML metamodel is discussed and suggestions for improvement are made. The paper closes in Sect. 4 with some final remarks.

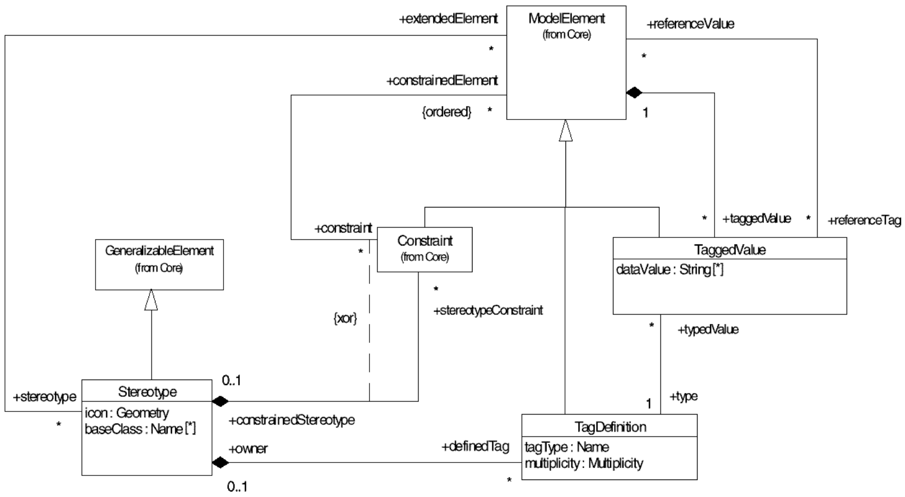

Fig. 1. Abstract Syntax for Stereotypes in UML 1.4

2 Notable Technical Details for UML 1.4 Stereotypes

This subsection concentrates on stereotypes in UML 1.4 as shown in Fig. 1, the latest approved version of UML and the one which we use for our forthcoming examples. Readers interested now in an object diagram belonging to the metaclass diagram in Fig. 1 may already have a look at Fig. 5 that will be explained below.

2.1 Attribute baseClass

First, the type of the attribute baseClass is Name, which essentially means that baseClass gives a String value. It would be better if the attribute does not only hold a name of a metaclass but the metaclass itself. It is, however, not possible to express this with the current UML features.
Second, if, as is implicit in the textual semantics in the OMG documents, the attribute baseClass is required to refer to a class name of the UML specification, a metaclass, one could additionally give a constraint requiring this:

```
context Stereotype baseClassRefersToAMetaclass:
  baseClass="Abstraction" or ... or baseClass="Object" or ... or
  baseClass="VisibilityKind"
```

This means that this constraint must list exactly the names of the classes in the UML metamodel.

Third, the above constraint is merely a restriction for the baseClass attribute. This does not guarantee that the model elements linked to the stereotype belong to the correct class. If one wants to formally assure in an OCL-like way that the model elements belong to the proper class, one can do the following trick:

> Assume we have a Stereotype object s42 with s42.baseClass being the name of a UML metaclass. Then we can generate an OCL constraint which assures the desired restriction. The constraint looks like:
>
> ```
> "context Stereotype s42IsLinkedToCorrectModelElements:" +
>   "s42.extendedElement->forAll(e|" +
>     "e.oclIsKindOf(" + s42.baseClass + "))"
> ```

The expression above constructs a String representing an OCL constraint. It consists of String constants notated between quotation marks " and one string expression s42.baseClass accessing the UML metamodel. The Stereotype object s42 must already exist in order to do that construction. To draw an analogy from progamming languages, this constraint is more like a run-time generated constraint than a design- or compile-time determined one. When a Stereotype object exists, we can and must generate a respective constraint.

2.2 Associations between ModelElement, Constraint and Stereotype

There are two possibilities for links between stereotypes and constraints. One possibility is that the links belong to the association between Stereotype and Constraint. The other possibility for such a link is the association between ModelElement and Constraint, because each stereotype is a model element. Consequences of this construction will be discussed in detail in Sect. 3.3.

2.3 Association between ModelElement and TaggedValue

Tagged values are needed for tag definitions, which, in turn, are relevant for Stereotypes. The textual description in UML 1.4, page 2-80, lines 12–13, states that

> *Any tagged value must have one or more reference value links or one or more data values, but not both.*

This means that an xor constraint is probably missing in the class diagram of Fig. 1 between the association with role names referenceValue and referenceTag and the association with role names modelElement (modelElement is the implicit OCL role name for this side of the second association) and taggedValue. Furthermore, the multiplicities in the class diagram do not match the intended meaning. On the ModelElement side of the association with role name taggedValue there should be a multiplicity 0..1 instead of the multiplicity 1. The wording also suggests that the role name dataValue should be present on the ModelElement side of that association.

Summary: The attribute of metaclass Stereotype could be further restricted by a constraint requiring that the attribute refers a UML metaclass. A constraint for the declaration of the stereotype and a match to its use should be generated after the stereotype has been introduced. An xor constraint should be added between the association with role names referenceValue and referenceTag and the association with role names modelElement and taggedValue.

3 UML Stereotype Analysis by Selected Examples

This section gives detailed examples for UML stereotypes in the graphical and tabular notation proposed in the UML notation guide and studies properties of the UML stereotype definition. Examples worked out in detailed form are hard to find in the literature. The section also shows how these stereotypes, in particular their declaration and use, are represented as object diagrams corresponding to the UML 1.4 metamodel as shown in Fig. 1. This approach allows conclusions to be drawn on the technical advantages and drawbacks of the UML metamodel.

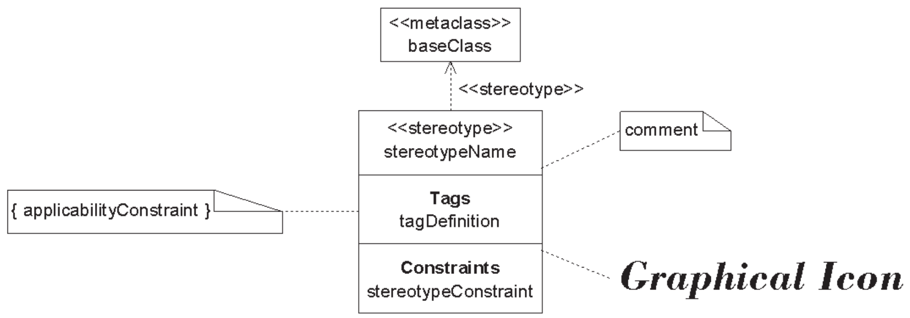

Fig. 2. General Notation of a Stereotype Declaration

Figure 2 shows the general notation of a stereotype declaration. The following items must be or can be provided: a mandatory name for the stereotype shown as stereotypeName, the mandatory base metaclass of the stereotype shown as baseClass, zero or more optional definitions of stereotype tags shown as tagDefinition, zero or more optional stereotype constraints shown as stereotypeConstraint being constraints that are attached to the model element to which the stereotype is applied, and zero or more optional constraints shown as applicabilityConstraint that restrict the applicability of the stereotype. In addition, optional notes containing comments and a graphical icon may be present.

3.1 UML Stereotype ≪persistent≫

Figure 3 shows the declaration of the stereotype ≪persistent≫ in the way it is proposed in the UML notation guide. Although the UML already knows the stereotype ≪persistent≫ as a standard stereotype, we use a variation of it because it is a good example for a general application- and domain-independent stereotype.
The UML metaclass Class and the stereotype ≪persistent≫ are shown within class rectangles. A dependency from stereotype ≪persistent≫ to metaclass Class displays the fact that the base class for stereotype ≪persistent≫ is the UML metaclass Class. The stereotype ≪persistent≫ owns a boolean, single-valued tag indicating the form in which the storage is supposed to be performed. Two comments explain the intended meaning in textual form.

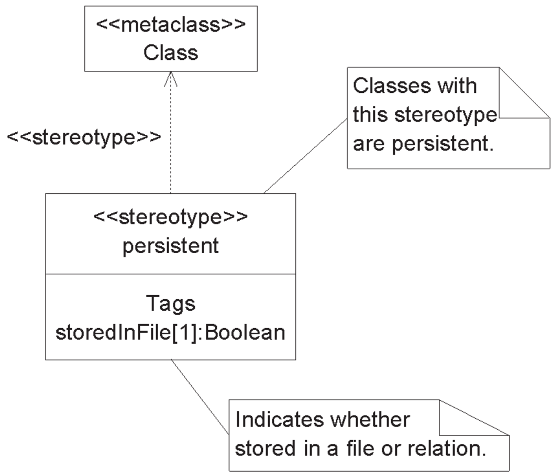

Fig. 3. Declaration of the Stereotype ≪persistent≫

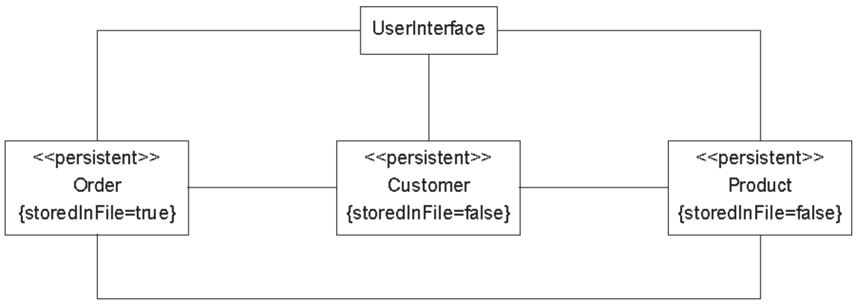

Fig. 4. Use of the Stereotype ≪persistent≫

Figure 4 gives an example for using the stereotype ≪persistent≫. It shows four classes, where three of them are persistent. Objects of class Order have to be stored in a file, whereas objects of classes Customer and Product are supposed to be stored in a relation. The class UserInterface is not a persistent class.

Figure 5 now combines the facts from Fig. 3 and Fig. 4 in the form of an object diagram belonging to the UML 1.4 metamodel as shown in Fig. 1. We show the figure to illustrate an instantiation of the UML metamodel for a particular use. The upper part above the dashed line with the objects belonging to the classes Stereotype, TagDefinition and Comment reflects the declaration of Fig. 3. The lower part below the dashed line with the objects belonging to classes TaggedValue and Class represents the use of the stereotype in Fig. 4. The Class objects with names Order, Customer, Product and UserInterface are of course classes in the class diagram in Fig. 4.

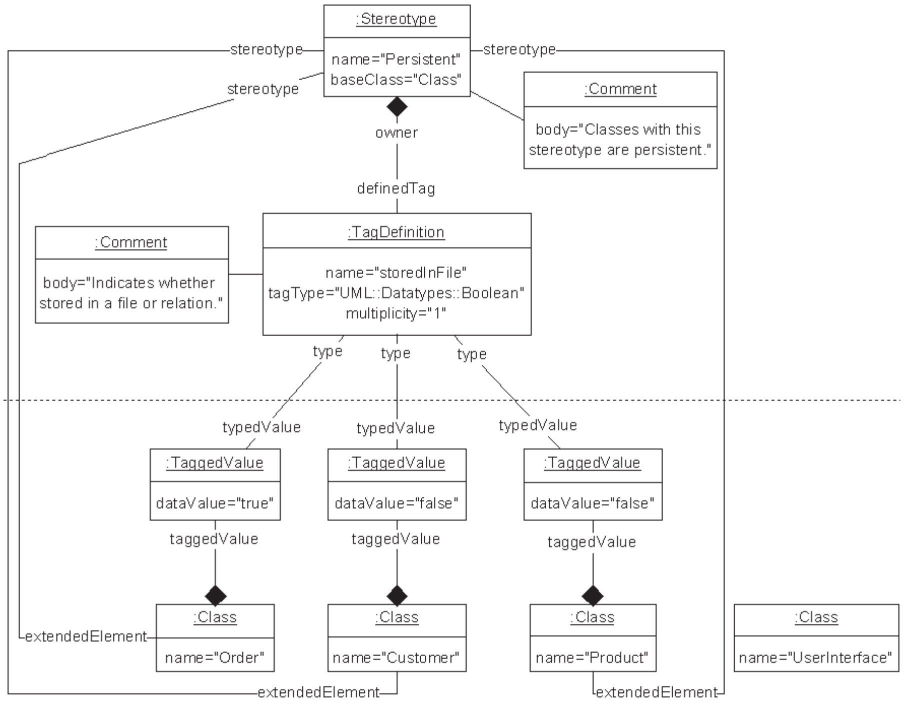

Fig. 5. Metamodel Representation of the Stereotype ≪persistent≫ and its Use

Summary: Stereotypes must be declared and can then afterwards be applied. It is not sufficient to simply use the stereotype without a definition. This is analogous to declaring a variable or defining a procedure in a programming language. Without full declaration the item cannot be used properly because its details are unknown. It seems that in the literature stereotypes are frequently applied without a proper declaration (in fact, usually without any declaration).

3.2 UML Stereotype ≪important≫

This section studies an example for a stereotype which is syntactically correct according to the UML metamodel, but where the defined stereotype communicates no real message. The declaration in Fig. 6 introduces the stereotype ≪important≫ in the form proposed in the UML notation guide. The purpose of this stereotype is to mark (or brand or classify) certain elements in a UML model as being important in order to draw special attention to these model elements. The stereotype uses the metaclass ModelElement as its baseClass. It has no constraints and no tag definitions. It only consists of its name and a comment.

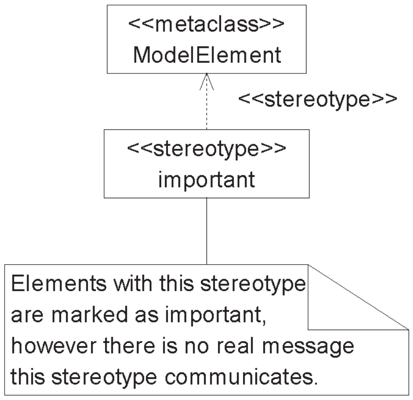

Fig. 6. Declaration of Stereotype ≪important≫ Following the UML Notation Guide

Summary: The UML metamodel allows the declaration of stereotypes without tag or constraint content. Thus stereotypes can be used for simply marking model elements. Therefore, not every syntactically allowed stereotype involves a constructive and good modelling contribution. If the stereotype has no tag or constraint, the same modelling effect could probably have been achieved by connecting the stereotyped model elements with a single comment. A more general observation is that stereotypes should be avoided if an equivalent modelling can be achieved by using more modest model elements. Altogether, these considerations lead to some observations and consequences concerning the UML metamodel.

The attribute baseClass is not restricted, because it can be set equal to ModelElement (the top of the metahierarchy in UML) meaning that the defined stereotype can be applied to all kinds of model elements. However, it seems hard to decide which metaclasses should be allowed for the baseClass attribute and which not. A simple solution would be to exclude the declaration of stereotypes for general ModelElements and add a well-formedness rule forbidding the metaclass ModelElement as the baseClass Attribute. This guarantees that stereotypes can be only declared for subclasses of ModelElement.

```
context Stereotype forbidModelElementAsBaseClass:
  baseClass<>"Foundation::Core::ModelElement"
```

3.3 UML Stereotype ≪singleKey≫

Figure 7 shows the declaration of the stereotype ≪singleKey≫ following the UML notation guide. The intention of this stereotype is to mark an attribute in a class as being the key attribute of the class, i.e. to require that two different objects also differ in the value of this key attribute. The constraints in Fig. 7 require that there is exactly one attribute in a class marked by this stereotype and the attribute values identify an object within its class uniquely. The baseClass of the stereotype is the metaclass Attribute. We would like to draw the attention to some subtle points in the notation. First, the note on the left side of the diagram (within the rectangle with the bent corner) contains a constraint (not only a comment) as indicated by the enclosing braces. Second, there is another constraint present in the diagram, namely the one in the rectangle for stereotype ≪singleKey≫. Third, the difference between these two constraints concerns the objects to which these constraints refer. The constraint in the note concerns the stereotype itself and restricts the way the stereotype may be applied. The constraint in the rectangle for stereotype ≪singleKey≫ concerns the model element to which the stereotype is applied, namely the attribute required to be the key. The UML 1.4 description does not explicitly distinguish between these two usages of constraints. Our notation is a proposal rather than the status quo. The following discussion will illustrate these points in more detail.

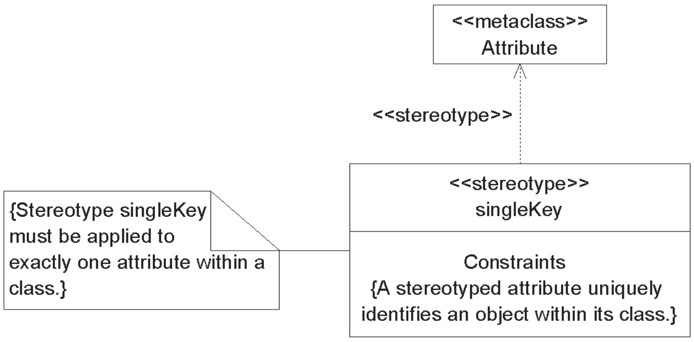

Fig. 7. Declaration of Stereotype ≪singleKey≫ Following the UML Notation Guide

The object diagram in Fig. 8 follows the UML metamodel, takes up the information given in Fig. 7 and extends it with necessary facts. In particular, instead of giving the constraints in textual form, it specifies them formally with OCL. We could have given the OCL description already in Fig. 7, but we prefer to have the informal description there. The diagram links the stereotype ≪singleKey≫ with the two constraints using different associations. The link with the role names constrainedElement and constraint belongs to the association between ModelElement and Constraint. Thus the stereotype ≪singleKey≫ is regarded here as a ModelElement due to the UML 1.4 metamodel generalizations from Stereotype to GeneralizableElement and from there to ModelElement. The link with roles names constrainedStereotype and stereotypeConstraint belongs to the association between Stereotype and Constraint. The constraint attached through the first link establishes a restriction on the stereotype itself, and the constraint attached through the second link establishes a restriction on the model element (in this case an attribute) to which the stereotype is applied.

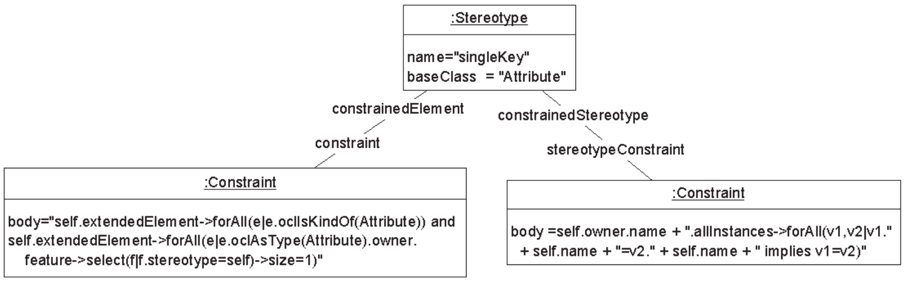

Fig. 8. Declaration of Stereotype ≪singleKey≫ as an Object Diagram

The first constraint is a conjunction divided into a left part (L) and a right part (R). It restricts the possibilities for applying the stereotype by demanding that (L) the model element to which the stereotype is applied is an attribute (or belongs to a submetaclass of metaclass Attribute), and (R) within a class there is exactly one attribute to which the stereotype is applied. The formal requirement (L) is the realization of the textual requirement given in the UML semantics, i.e. that the model element to which a stereotype is applied matches its baseClass. This is necessary, because the UML semantics do not express that textual requirement formally as an OCL constraint. The formal requirement (R) concerns the intended application of the stereotype to attributes identifying objects within a class. It requires that within a single class there is exactly one attribute marked by the stereotype. Formally, the (R) expression navigates from the stereotype to the attribute and to its class, from there to all the features of that class, counts the features which are linked to the stereotype where the navigation started and requires that the number of such features is one.
The second constraint, which is the one that will be attached to the model element to which the stereotype is applied (in this case an attribute within a class), embodies the idea that the values of the stereotyped attribute identify the objects within the class. The general idea of the OCL expression is that two object variables having identical key attribute values also represent identical objects, so that the OCL expression would read like

```
className.allInstances->forAll(o1,o2|
  o1.attrName=o2.attrName implies o1=o2).
```

Alternatively and equivalently, we could write the implication the other way round emphasizing that different objects have different attribute values.

```
className.allInstances->forAll(o1,o2|
  o1<>o2 implies o1.attrName<>o2.attrName)
```

But the problem in both formulations is that the class name and the attribute name are not known in advance, because one does not know to which attribute in which class the stereotype ≪singleKey≫ will be applied. Therefore, the constraint to be attached to the stereotyped attribute uses a generic formulation in which the name of the class and the name of the attribute are retrieved by the UML 1.4 metamodel expressions `self.owner.name` and `self.name`, respectively. These string-valued expressions are navigating through UML 1.4 metamodel object diagrams (like the one to be shown in Fig. 10 for this example) and are concatenated with other appropriate string constants in order to establish the proper OCL constraint. The way this constraint is used will be explained in the following.

In Fig. 9, an application of the stereotype ≪singleKey≫ is shown. There is one class Book with three attributes. One of them is stereotyped with stereotype ≪singleKey≫.

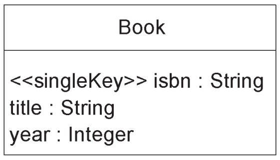

Fig. 9. Application of Stereotype ≪singleKey≫ to Class Book

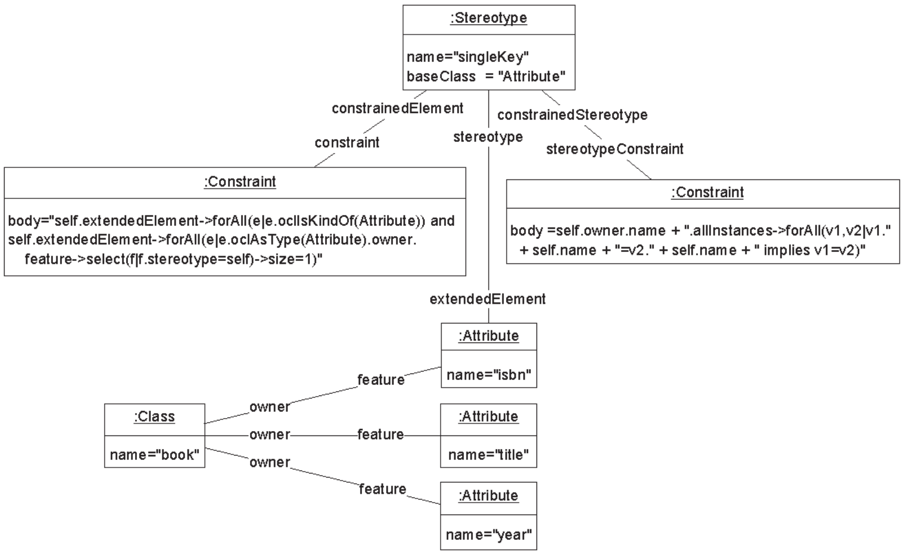

Fig. 10. Metamodel Diagram of Applying ≪singleKey≫ before Attachment

Figure 10 shows how the situation of Fig. 9 is represented as an object diagram of the UML metamodel. It links the stereotyped attribute to the stereotype object.

Before we explain the next step of the stereotype application, let us quote an important part for the current context from the UML 1.4 semantics, section *2.6 Extension Mechanisms*, subsection *2.6.4 Detailed Semantics*, page 2-83.

> *Any constraints on the stereotype are implicitly attached to the model element.*

Figure 10 displays the situation before this implicit attachment is carried out. Figure 11 displays the situation after this attachment has been done. It shows that in order to attach the constraint to the attribute, the body expression of the constraint is evaluated with variable self being the attribute named isbn. Thus the expression self.name evaluates to "isbn", and the expression self.owner.name evaluates to "Book". Then, the final constraint with the substituted actual class name and actual attribute name shows up. We would like to draw attention to another point illustrating the difference between the two constraints because the expression self occurring in both constraints refers to different objects. The expression self in the first constraint references the Stereotype object, in this example, the object with name singleKey. The expression self in the second constraint references the Attribute object, in this example, the object with name isbn.

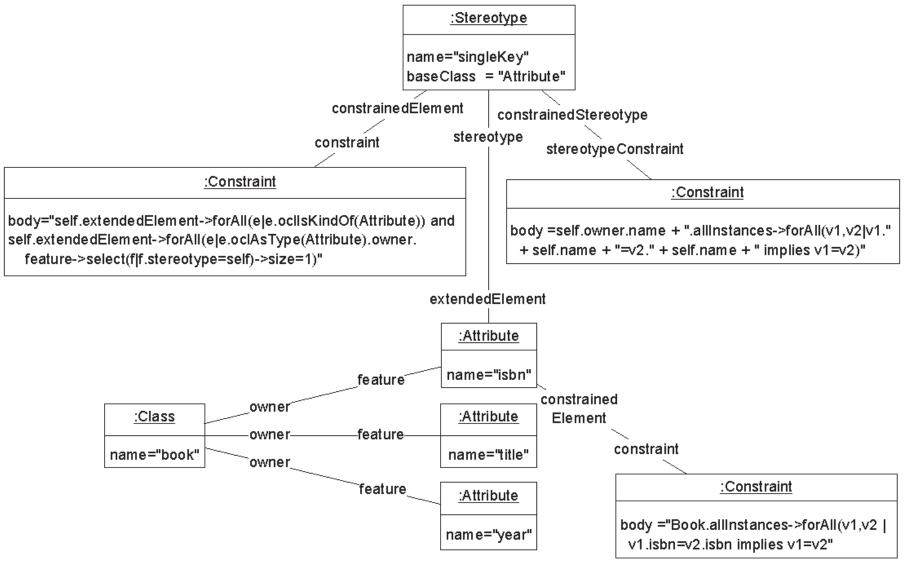

Fig. 11. Metamodel Diagram of Applying ≪singleKey≫ after Attachment

Figure 12 gives examples for incorrect applications of the stereotype. On the left hand side of Fig. 12, the stereotype ≪singleKey≫ is applied to two attributes within a single class. This implies that the first constraint (the constraint on the stereotype itself) will evaluate to false due to failure of requirement (R)

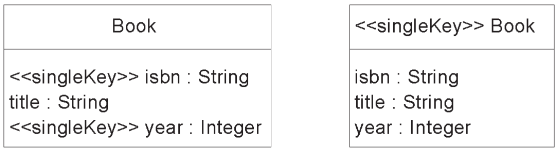

Fig. 12. Incorrect Applications of Stereotype ≪singleKey≫

demanding that stereotype ≪singleKey≫ is applied to one attribute only. This is because the expression

```
self.extendedElement->forAll(e|e.oclAsType(Attribute).
  owner.feature->select(f|f.stereotype=self)->size=1)
```

will evaluate to false, if the extended model element e is equal to the Attribute object named isbn or the Attribute object named year. In particular, in these cases the expression

```
e.oclAsType(Attribute).owner.feature->select(f|f.stereotype=self)
```

will evaluate to a set containing two stereotyped attributes instead of a set with cardinality 1 as required by the constraint. On the right hand side of Fig. 12, the stereotype ≪singleKey≫ is applied to a class which violates requirement (L) from the first constraint. This is because the expression

```
self.extendedElement->forAll(e|e.oclIsKindOf(Attribute))
```

evaluates to false as there is an extended model element, namely the Class object named book, whose metaclass is not Attribute or a submetaclass of Attribute.

Summary: The description of the stereotype semantics in the UML metamodel should be expanded in order to explicitly explain the distinction between the two possible associations between stereotypes and constraints. The first association is the association between ModelElement and Constraint, the second association being the association between Stereotype and Constraint. The description of the stereotype semantics should discuss the implicit attachment of constraints to stereotyped model elements in more detail.

In order to achieve more modelling power, it must be possible to specify constraints for stereotypes that are generic in the sense that these constraints allow parts of itself to refer to the model element to which they are applied. These constraints are not directly attached to the stereotyped model elements without further consideration, but the parts of them referring to the actual stereotyped model element are evaluated before the attachment is carried out. These parts are actualized and substituted with the actual values accordingly. Stereotypes for certain complex but still moderate constraints, like for example key constraints in information systems [Amb99,Gog01], cannot be specified without such a mechanism.

3.4 UML Stereotype ≪statesHaveTransitions≫

Stereotypes can also be used for checking whether the model under development currently has certain properties expressed as application prerequisites for the stereotype. For example, in a state machine, a stereotype could check for unconnected states having no incoming and no outgoing transitions and report on these states. Figure 13 shows the declaration of such a stereotype.

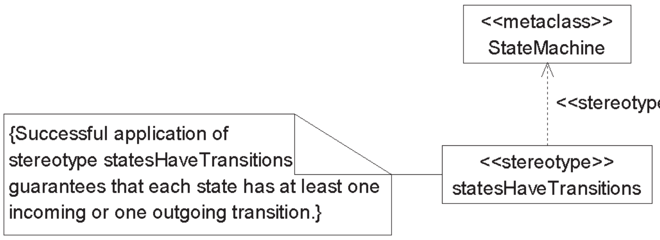

Fig. 13. Graphical Representation of Stereotype ≪statesHaveTransitions≫

In the upper part of Fig. 14 we have given the formal OCL constraint assuring that the states of a stereotyped state machine have transitions. The constraint is based on the abstract syntax for UML 1.4 state machines (UML 1.4 Semantics, page 2-147) and checks for states which have no incoming and no outgoing transition. This stereotype is applied during the modelling process to see whether a developed state machine has this property or not. If the state machine has that property, the application of the stereotype would be OK and the modeller can proceed with the knowledge that the states really have transitions. If the state machine does not have that property, a tool has to report on this. In that case, the lower part of Fig. 14 could be helpful. There, a derived operation for state machines called namesOfUnconnectedStates() is defined. This operation, when applied to a state machine, returns the names of the unconnected states and, by this mechanism, gives feedback to the modeller on how to improve the model. The OCL constraint for stereotype ≪statesHaveTransitions≫ is a rather simple one, because it only searches for single, isolated states. It is possible to formulate more sophisticated constraints; for example, one could check with an OCL constraint that the state machine has exactly one initial and at least one final state and that all states are connected by a transition path starting in the initial state.

Summary: Stereotypes can be used to check whether a certain modelling situation is present or not. If the stereotype itself is equipped with a constraint in order to restrict its application to a special context, the marking of a model element with the stereotype gives feedback on whether that special context is present or not. A tool providing full stereotype support would either report that the application of the stereotype is acceptable, i.e. the proper context is present, or the tool would report that the application of the stereotype is not acceptable, i.e. the proper context is not present and in this case the tool could analyze why

```
self.extendedElement->forAll(sm|
  sm.oclIsKindOf(StateMachine) and
  sm.oclAsType(StateMachine).top.oclIsKindOf(CompositeState) and
  sm.oclAsType(StateMachine).top.oclAsType(CompositeState).subvertex->
    forAll(sv|sv.incoming->notEmpty or sv.outgoing->notEmpty)

context StateMachine namesOfUnconnectedStates():Bag(String)
  self.top.oclAsType(CompositeState).subvertex->
    select(sv|sv.incoming->isEmpty and sv.outgoing->isEmpty)->
      collect(sv|sv.name)
```

Fig. 14. OCL Expressions Handling Unconnected States

the context is not present. Thus, another use of stereotypes is for checking the current state of the model under development with respect to certain criteria.

4 Conclusion

In this paper, we have analyzed the expressibility of UML stereotypes and made some concrete suggestions for the improvement of the UML metamodel. We have used OCL to define precise stereotypes. Stereotypes can be employed for various tasks in the modelling process: marking elements, declaring formal constraints in a user-friendly way, providing graphical icons of the application domain and checking the model under development for certain criteria. We feel that UML stereotypes should be used with greater care than shown in the literature. In particular, they should be carefully declared and used only when the message to be communicated cannot be expressed in any other UML terms. Necessary conditions in order to achieve a better use of stereotypes are that the UML metamodel should be adjusted and tool support should be provided for the complete part of the UML metamodel dealing with stereotypes. We hope future versions of UML will allow the definition and use of stereotypes in a more elegant way than is currently possible.

References

[AKHS00] Colin Atkinson, Thomas Kühne, and Brian Henderson-Sellers. To Meta or not To Meta – That is the Question. *Journal of Object-Oriented Programming*, 13(8):32–35, 2000.

[Amb99] Scott Ambler. Persistence Modeling in the UML. *Software Development Online*, August, 1999. http://www.sdmagazine.com.

[BGJ99] Stefan Berner, Martin Glinz, and Stefan Joos. A classification of stereotypes for object-oriented modeling languages. In Robert France and Bernhard Rumpe, editors, *Proc. Int. Conf. Unified Modeling Language (UML'99)*, pages 249–264. Springer, LNCS 1723, 1999.

[Con99] Jim Conallen. Modeling Web Applications with UML. http://www.conallen.com, Conallen Inc., 1999.

[DSB99] Desmond D'Souza, Aamond Sane, and Alan Birchenough. First-class extensibility for UML-profiles, stereotypes, patterns. In Robert France and Bernhard Rumpe, editors, *Proc. Int. Conf. Unified Modeling Language (UML'99)*, pages 265–277. Springer, LNCS 1723, 1999.

[Gog01] Martin Gogolla. Using OCL for Defining Precise, Domain-Specific UML Stereotypes. In Aybuke Aurum and Ross Jeffery, editors, *Proc. 6th Australian Workshop on Requirements Engineering (AWRE'2001)*, pages 51–60. Centre for Advanced Software Engineering Research (CAESER), University of New South Wales, Sydney, 2001.

[GR01] Martin Gogolla and Mark Richters. Expressing UML Class Diagrams Properties with OCL. In Tony Clark and Jos Warmer, editors, *Advances in Object Modelling with the OCL*, pages 86–115. Springer, Berlin, LNCS 2263, 2001.

[HS02] Brian Henderson-Sellers. Notes on the Use of Subtypes and Stereotypes in the UML Metamodel. *Journal of Database Management*, 13(3):44–52, 2002.

[HSFG97] Brian Henderson-Sellers, Donald Firesmith, and Ian M. Graham. OML Metamodel: Relationships and State Modeling. *Journal Of Object-Oriented Programming*, 10(1):47–51, 1997.

[KMP+98] Mohamed Kande, Shahrzade Mazaher, Ognjen Prnjat, Lionel Sacks, and Marcus Wittig. Applying UML to Design of an Inter-Domain Service Management System. In *Proc. ISN*, 1998.

[OMG99] OMG, editor. *OMG Unified Modeling Language Specification, Version 1.3.* OMG, June 1999. OMG Document `formal/00-03-01`, `http://www.omg.org`.

[OMG01] OMG, editor. *OMG Unified Modeling Language Specification, Version 1.4.* OMG, September 2001. OMG Document `formal/01-09-68` through `formal/01-09-80`, `http://www.omg.org`.

[PL01] Francois Pinet and Ahmed Lbath. Semantics of Stereotypes for Type Specification in the UML: Theory and Practice. In H.S. Kunii, S. Jajoda, and A. Solvberg, editors, *Proc. Entity-Relationship Conf. (ER'2001)*, pages 339–353. Springer, LNCS 2224, 2001.

[RG01] Mark Richters and Martin Gogolla. OCL - Syntax, Semantics and Tools. In Tony Clark and Jos Warmer, editors, *Advances in Object Modelling with the OCL*, pages 43–69. Springer, Berlin, LNCS 2263, 2001.

[SE99] Stefan Sauer and Gregor Engels. UML-based Behavior Specification of Interactive Multimedia Applications. In *Proc. IEEE Symposium Visual Languages (VL'1999)*. IEEE, 1999.

[SW01] Ansgar Schleicher and Bernhard Westfechtel. Beyond stereotyping: Metamodeling approaches for the UML. In Ralph H. Sprague, Jr., editor, *Proc. 34th Annual Hawaii Int. Conf. System Sciences (HICSS'34)*. IEEE Computer Society, 2001.

[WB94] Rebecca Wirfs-Brock. Adding to Your Conceptual Toolkit: What's Important About Responsibility-Driven Design? *ROAD*, 1(2):39–41, 1994.

[WK98] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Precise Modeling with UML.* Addison-Wesley, 1998.