A Classification of Stereotypes for Object-Oriented Modeling Languages

Stefan Berner[1], Martin Glinz[1] and Stefan Joos[2]

[1]University of Zurich, Winterthurerstr. 190
CH-8057 Zurich, Switzerland
{berner, glinz}@ifi.unizh.ch
[2]Robert Bosch GmbH, Postfach 30 02 20,
D-70469 Stuttgart, Germany
stefan.joos@de.bosch.com

Abstract. The Unified Modeling Language UML and the Open Modeling Language both have introduced stereotypes as a new means for user-defined extensions of a given base language. Stereotypes are a very powerful feature. They allow modifications ranging from slight notational changes up to the redefinition of the base language. However, the power of stereotypes entails risk. Badly designed stereotypes can do harm to a modeling language. In order to exploit the benefits of stereotypes and to avoid their risks, a better understanding of the nature and the properties of stereotypes is necessary.

In this paper, we define a framework that classifies stereotypes according to their expressive power. We identify specific properties and typical applications for stereotypes in each of our four categories and illustrate them with examples. For each category, we discuss strengths and weaknesses of stereotypes and present a preliminary set of stereotype design guidelines.

1 Introduction

Since about 1990 a broad variety of object-oriented modeling languages have been developed [1][2][3][4][9][11][13][14][17]. These languages are used to describe the requirements and the design of a software system. Since 1996, various attempts have been made to unify different methods and languages. As a result of this endeavor, two languages have been developed: the Unified Modeling Language UML [12] and the Open Modeling Language OML [6]. Both UML and OML introduce a distinctive new feature: they allow users to extend or even to modify the base language in order to adapt the language to specific situations or needs. The language construct that is used to implement this feature is called a *stereotype*.

In the context of object-oriented modeling, the notion of stereotypes was introduced before by Rebecca Wirfs-Brock [16]. Her principal idea is to provide a secondary clas-

This work was partially supported by the Swiss National Science Foundation under grant Nº 20-47196.96.

sification for objects: stereotypes classify objects according to their use, independently of the primary classification by classes and class inheritance. This classification helps to better organize a model and improves system understanding. Wirfs-Brock uses a fixed number of stereotype classes, namely: domain-, design-, control-, delegation-, structure-, service-, and interface-objects as well as information objects.

UML and OML both generalize Wirfs-Brock's notion of stereotypes from a secondary classification to a concept that allows for general extensions of the base language. A stereotype in UML and OML can add new properties to elements of the underlying language or can modify existing ones. (In the following, we will call the underlying language that is being stereotyped the *base language.*) For example, a stereotype can *extend* UML classes to include a property that designates a class to belong to the model, the view or the controller in a model-view-controller pattern. On the other hand, in older versions of UML an Actor was expressed using a stereotype that redefined the language element Class.

In this paper we discuss the UML/OML kind of stereotypes in a general context of object-oriented modeling languages. However, as the notion of stereotypes is not limited to object-oriented approaches, we define a stereotype independently of object-orientation as follows:

> DEFINITION. A *stereotype* in a modeling language is a well-formed mechanism for expressing user-definable extensions, refinements or redefinitions of elements of the language without (directly) modifying the meta-model of the language.

Stereotypes provide language users with limited metamodeling capabilities without giving them (direct) access to the metamodel of the language. This is a very powerful mechanism. However, as is frequently the case with powerful features, stereotypes have both a bright and a dark side. On the bright side, stereotypes can lead to modeling languages which are more flexible and expressive and which are better adaptable to specific problem types and application domains. On the dark side, unsystematic or excessive use of stereotypes can lead to a proliferation of incompatible dialects of a language and can make a language both difficult to handle and to understand. Thus, unconsidered use of stereotypes can do more harm than good.

Like language design in general, designing good stereotypes is a difficult and demanding task. ‘Good’ in this context means:

- Every stereotype is properly defined. The definition balances formality and understandability in an optimal way. Intuitive semantics and formal definition of semantics (if existing) are congruent. The stereotype does not introduce inconsistencies into the language.
- Every stereotype is useful. That means (1) it provides a concept or feature that the base language does not have and (2) it eases the application of the language in a given context or application domain.
- The set of all stereotypes is consistent. A stereotype must not be inconsistent with other stereotypes, unless a stereotype is explicitly declared to be mutually exclusive with other, contradictory ones.

- The set of stereotypes is orthogonal. When a stereotype introduces a new distinctive feature or concept into a language, then this feature should not be provided by another stereotype, too.

The task of designing good stereotypes would become considerably easier if we had a proper design methodology or at least a set of design guidelines for their creation. However, neither a methodology nor guidelines presently exist. In order to develop guidelines (and finally arrive at a methodology), a deeper understanding of the nature of stereotypes and of the implications of their use is necessary.

In this paper, we contribute to the solution of the stereotype design problem. We introduce a classification of stereotypes according to their expressiveness – that means according to their potential to alter the syntax and semantics of the base language. Every class represents a related set of purposes for using stereotypes and has specific stereotype design requirements associated with it. For each class, we give a first-cut set of guidelines for designing and defining stereotypes of this class.

To our knowledge, there is almost no related work on classification and design of stereotypes. Eriksson and Penker [5] classify the standard (i.e. predefined) stereotypes of UML according to the language concept they are applied to. For user-defined stereotypes, however, there is no classification. In the literature on UML and OML (e.g. [12][5][6]), the design and definition of stereotypes is treated in a very vague and superficial fashion only. In an earlier paper in German [10], we have introduced the idea of our classification framework along with some examples and a preliminary discussion of weaknesses and benefits.

The rest of the paper is organized as follows: Following the introduction we present our classification and illustrate it with examples. In the next section, we identify strengths and weaknesses of stereotypes in each of our four categories. In section 2, we demonstrate how a rigorous definition of a non-trivial stereotype should look like and present a preliminary set of guidelines for the design of stereotypes. The paper concludes with some remarks about our achievements and future research directions.

2 A Classification of Stereotypes

The extent to which stereotypes alter a language ranges from mere notational variations to a complete redefinition of the language.

Simple stereotypes typically change the notation (i.e. the concrete syntax and/or visual representation) of a language element and/or introduce new features that serve as a kind of ‘structured comment’. Powerful stereotypes, on the other hand, impose semantic restrictions on the added language elements or even redefine the semantics of language elements. This can go up to a complete syntactic and semantic redefinition of the base language. We classify stereotypes according to their expressiveness into four categories:

- *Decorative stereotypes* vary the concrete syntax of a language.
- *Descriptive stereotypes* extend the syntax of a language such that additional information can be expressed.

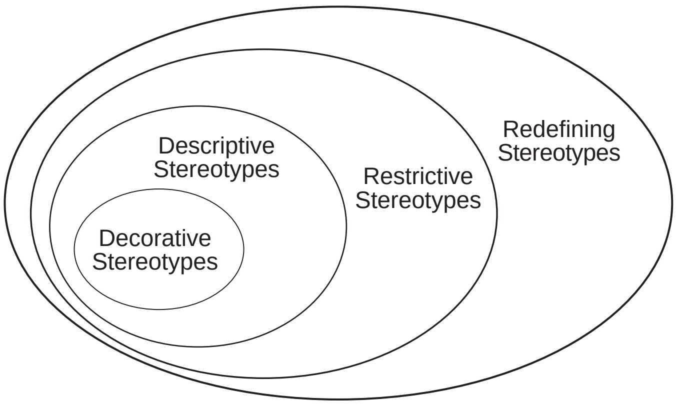

Figure 1. Classification of stereotypes according to their expressive power into four categories and their inclusion hierarchy

- *Restrictive stereotypes* extend the syntax of a language and impose semantic restrictions on these extensions.
- *Redefining stereotypes* modify the (core) semantics of a language element.

Note that our classification forms an inclusion hierarchy, not a partition. The more powerful categories include all the potential of the less powerful ones (see Figure 1). In the following subsections, we describe the four categories more in detail and provide examples.

2.1 Decorative Stereotypes

DEFINITION. A *decorative stereotype* modifies the concrete syntax of a language element and nothing else.

Thus, decorative stereotypes vary the way in which a language element is visually represented. They do not introduce any essential additional information or new concepts into the base language. The represented model and the essence of the language that expresses the model remain unchanged.

A decorative stereotype improves the understandability of a model in the same way that a good illustration improves the understandability of a text. However, it can also worsen the understandability in the same way that an erratic illustration does. As the interpretation of signs depends on the personal and cultural background of the viewer, it can become quite tricky to decide whether a decorative stereotype eases or hampers understandability. Some form of usability testing can act as a decision-maker here. But the potential benefit of the new stereotype – remember it is 'just' a decorative one – must be promising enough to justify the testing effort.

Decorative stereotypes are typically used to adapt the notation of a language or of some of its elements to some given standard or to personal preferences.

Examples

- In textual representations, a decorative stereotype can be used to change key words (like ENTITY_TYPE instead of CLASS, etc.), and if accessible, text/character attributes (font, style, size, color, etc.) or other formats (indents, alignments, line spacing, etc.).
- In graphic representations, a decorative stereotype can change graphic symbols or other attributes like colors, line width, etc. Figure 2 depicts a typical application of a decorative stereotype: In UML the graphic notation of the class symbol is replaced by that of OOAD [1]. Note that this is a pure cosmetic change. Both the abstract syntax and the semantics of UML classes remain untouched.

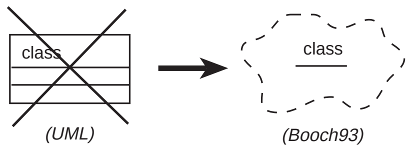

Figure 2. An example for a decorative stereotype

2.2 Descriptive Stereotypes

DEFINITION. A *descriptive stereotype* extends or modifies the abstract syntax of a language element and defines the pragmatics of the newly introduced element. The semantics of the base language remains unchanged. Additionally, a descriptive stereotype may modify the notation (the concrete syntax) of the stereotyped language element.

Thus, descriptive stereotypes are on a pure syntactic level. They do not impose any semantic restrictions on the extended or modified syntax. The persons who use a descriptive stereotype must rely on the description of the stereotype pragmatics in order to use and interpret the stereotype properly.

As a descriptive stereotype can also modify the concrete syntax of the stereotyped language element, it includes the expressiveness of a decorative stereotype. When compared with simple comments, descriptive stereotypes have the advantage of a well-defined syntactic structure, which makes some formal checking and analyses possible.

Secondary classifications (in the sense of Wirfs-Brock's stereotypes [16]) and standardized annotations are typical applications of descriptive stereotypes. Standardized annotations have the form keyword <value> where <value> must conform to a given type.

Examples

- A classification of objects according to Jacobson's OOSE [9] into Entity-, Interface-, and Control classes or distinguishing different kinds of relationships (association, usage, part-of, …) in a language which provides only 'primitive' relationships are examples of secondary classifications.

- An extension of the definition of a UML class with a stereotype ConfigInfo that consists of Author : string, Created : date, LastModified : date is a standardized annotation which allows to specify the author and the creation/modification dates of a class in a controlled way. Note that string and date are static data types that can be checked on a purely syntactic level. Semantic restrictions would lead to restrictive stereotypes (see section 2.3).
  It should also be noted that a *Tagged Value* in UML is equivalent to a simple descriptive stereotype. So, in UML an alternative to this stereotype ConfigInfo would be the usage of three user-defined tags: author, created and lastModified.

2.3 Restrictive Stereotype

DEFINITION. A *restrictive stereotype* is a descriptive stereotype that additionally defines the semantics of the newly introduced element.

Typically, the semantics impose compulsory structural restrictions on the newly introduced language element – hence the name restrictive stereotype. A restrictive stereotype does not change the semantics of the base language – it only extends it by the semantics of the stereotype.

The concept of restrictive stereotypes allows for a fully formal definition of the stereotype. However, in practice the definition will frequently be semi-formal only. This is no surprise when considering that most contemporary specification languages (including UML) do not have a completely formal definition of their semantics.

Restrictive stereotypes are first-class members in the language they are added to. They have the same expressive power and can be defined with the same degree of rigor as the elements of the base language themselves. Restrictive stereotypes are typically used to add missing features to some elements of a language, to strengthen weak features or to introduce a metalanguage on top of a given language.

Examples

- A class category construct (borrowed from OOAD [1]) can be introduced into UML in order to strengthen UML's rather weak package construct. In order to behave like the original class categories of OOAD, the stereotype must fulfill the following restrictions:
  - A class category must only contain classes or other class categories.
  - Any relationship between two class categories must be a uses-relationship.
  - Inside a class category a class may be tagged for being exported using instances of another stereotype, namely a descriptive stereotype export on classes. Classes outside a class category may have relations only with those components of a class category that are tagged for being exported.
- The support for requirements tracing is very weak in UML. The existing Trace dependency is quite unspecific. Its use is optional and lacks detailed semantics

([12] p. 56). Thus, mandatory requirements tracing that is enforced by a tool is almost impossible using Trace. A significantly improved requirements tracing feature can be introduced into UML by first introducing a restrictive stereotype requirementId : int with a restriction unique. Using this stereotype, every element of an UML requirements model can be tagged with a unique number identifying it. Now we introduce another restrictive stereotype traceBackToRequirements : array of int with the restriction that these integers must be existing requirement identifiers. Using this stereotype, we can trace design elements back to their requirements by attaching a set of requirement identifiers to every element of an UML design model. Note that neither a descriptive stereotype nor the UML Trace would suffice to accomplish this task, because they would also accept values other than valid requirement identifiers.

- A meta-language for the identification of design patterns [7] in a model can be introduced on top of a modeling language with restrictive stereotypes: the elements of a design model can be tagged with the pattern name and the role they play in this pattern (see Figure 4). Structural restrictions that the pattern imposes on the elements that instantiate the pattern can be enforced by defining restrictions for the pattern language stereotypes accordingly (see Figure 3).

The last example from the above list also demonstrates a situation where it is clearly preferable to define a feature with stereotypes instead of including it in the base language: patterns evolve and are frequently application-dependent. Thus, the set of those patterns that can be used and have to be documented in a model should be definable on the level of projects or organizations, and, hence, not be a part of the base language. Restrictive stereotypes are the appropriate means for doing so.

2.4 Redefining Stereotype

DEFINITION. A *redefining stereotype* redefines a language element, changing its original semantics. Concerning syntax, a redefining stereotype behaves in the same way as a restrictive one.

With decorative, descriptive and restrictive stereotypes, instances of the stereotype remain valid instances of the stereotyped language element. For redefining stereotypes, this is no longer true. A redefining stereotype can introduce a new language element that is no longer related to the element of the base language that it stereotypes.

Using redefining stereotypes, deep and radical changes can be imposed to a language. New language concepts can be introduced. In its extreme, redefining stereotypes can embed another language in a given base language.

Examples

- In earlier version of UML, some model elements that do not belong to the core of the language were defined as redefining stereotypes, e.g. Use Case and Actor were stereotypes of Class.

- A subset of the specification language Z [18] can be embedded in UML using a redefining stereotype Scheme for UML classes. Instances of Scheme are no longer classes, but valid Z schemes.

3 Strengths and Weaknesses of Stereotypes

In this section, we discuss the pros and cons of stereotypes both in general and for our four categories in particular.

The main general advantage of stereotypes is that they make a language *flexible* and *adaptable*. When used properly, they improve a modeling language, making models easier to express and to understand.

On the other hand, there are two general drawbacks and risks.

- Working with stereotypes requires effort for designing and maintaining them, and for training all the users and readers of a stereotyped language how to use and interpret the stereotypes.
- Badly designed stereotypes and the use of an excessive number of stereotypes both turn the potential benefit of stereotypes into its contrary: they harm a language, making it more difficult to use and to understand.

The potential benefit of stereotypes as well as the drawbacks and risks grow with increasing power of the stereotypes. In the sequel, we identify and discuss specific strengths and weaknesses of the stereotypes in the four categories of our classification.

Decorative Stereotypes. The definition of decorative stereotypes is easy and requires little effort. Furthermore, it is easy to incorporate the features required for the definition and use of decorative stereotypes into a tool. When used to adapt the language syntax to a given standard, decorative stereotypes improve the understandability of the language for all persons working with this standard.

On the other hand, decorative stereotypes threaten the very purpose of any language – that is, communication among people. When everybody uses her or his own decorative stereotypes, we quickly end up in the same situation as the ancient people of Babel when attempting to build their tower... Furthermore, a decorative stereotype does not add any real power to a language, it is mere ‘syntactic sugar’.

Descriptive Stereotypes. Descriptive stereotypes enrich a language in a controlled way by introducing structured annotations (comparable to tagged values in UML) and secondary classifications. This is clearly better than providing the same information with comments only, because descriptive stereotypes can be made mandatory and are amenable to syntactic checking and analyses. The definition of descriptive stereotypes is easy and requires little effort: only the syntax and a short natural language statement on the pragmatics of the stereotype are required. These properties make descriptive stereotypes a favorite candidate for adapting a language to company-wide or project-specific documentation standards.

On the negative side, the power of descriptive stereotypes is very limited, because they are purely syntactical. This is frequently insufficient, in particular for structured annotations. As soon as we want to impose semantic restrictions on the application of a stereotype and/or on the values of its variables, a descriptive stereotype no longer works and a restrictive one is required instead. Thus, a descriptive stereotype cannot improve bad or weak features of the base language. Neither can it supply a real feature (that means one with significant semantics) that is missing in the base language. Finally, in the same way that too many comments can 'drown' the code of a program, using too many descriptive stereotypes can yield clumsy and hardly readable models, where the essence of the model disappears in a flood of annotations.

Restrictive Stereotypes. Restrictive Stereotypes add real power to a language. Their capabilities go far beyond those of descriptive or decorative stereotypes. Not only can they add structured annotations with semantic restrictions, but also essential features that are missing in the base language. Restrictive stereotypes also can improve bad or weak features of the base language. Another powerful feature of restrictive stereotypes is their capability for defining a metalanguage on top of the base language (see the pattern examples in sections 2.3 and 4). In contrast to descriptive stereotypes, it is possible to check models for their compliance not only with syntax rules, but also with given semantic restrictions. When these restrictions are formally defined, checking can be automated and built into a tool.

Finally, a well-defined language forces its users to adhere to a basic set of modeling and design principles that the language is founded upon. Restrictive stereotypes can do the same for the features they introduce into the base language.

On the negative side, restrictive stereotypes have the following disadvantages and limitations.

- Designing restrictive stereotypes is expensive. The designers require a profound knowledge of: the desired properties of the stereotype to be designed, the base language, the general principles of good language design, and the metalanguage that is used to specify the semantics of the stereotype. Ignoring these requirements leads to bad (i.e. incomprehensible, contradictory or simply wrong) stereotypes. Badly designed restrictive stereotypes damage the base language instead of improving it.
- In order to define restrictive stereotypes properly, the metamodel of the base language should provide a mechanism for a formal specification of the semantics of the stereotype. This is a demanding requirement, which, for example, is only partially met by UML. Only a formal specification makes it possible to automatically check restrictions defined by a stereotype (of course, only those that can be automatically checked at all). Without this capability, restrictive stereotypes lose much of their power and finally become similar to descriptive ones.
- Defining too many restrictive stereotypes has the same disadvantages as described for descriptive stereotypes.

Redefining Stereotypes. Redefining stereotypes give the users full control over the base language. In contrast to restrictive stereotypes, a redefining stereotype does not merely extend the semantics of the base language, it modifies it. Instances of a redefining stereotype do no longer need to be instances of the stereotyped base language element. This capability has two major advantages:

- The base language can be extended by new features that have nothing to do with the element of the base language that is being stereotyped. It is even possible to (virtually) delete the stereotyped element from the base language by restricting the allowed number of non-stereotyped instances to zero. Of course it is impossible to really delete an element from the base language, because a stereotype cannot delete elements from the metamodel. However, by restricting the number of allowed instances of a language element to zero accomplishes the same effect as a real deletion.
- Starting from a common general-purpose language, highly specialized languages can be derived for every specific problem or application domain. This can help to keep the base language simple an clean.

On the negative side, all the disadvantages and limitations listed above for restrictive stereotypes also apply to redefining stereotypes, but to an even stronger degree. Three problems deserve special attention.

- The introduction of every redefining stereotype creates a new dialect of the base language that is semantically different. If everybody creates her or his own redefining stereotypes, this is a deadly threat to the very essence of any language: enabling communication between people that have to share information.
- Among all kinds of stereotypes, redefining stereotypes require the highest effort for creation, training and maintenance.
- Making redefining stereotypes is language design, a task requiring special knowledge and experience that typical users of a language do not have. Letting these people nevertheless do language design bears a high risk of failure. And, by the way, modelers are usually employed for creating models, not modeling languages.

4 Defining Stereotypes

If we want to use stereotypes properly, we must have them properly defined first. However, this is not quite easy. The UML and OML reference manuals [12][6] are both considerably vague about the definition of stereotypes. The UML metamodel [15] is somewhat more precise. According to this model, the definition of a stereotype consists of a name, the host (language element(s) being stereotyped), an (optional) new graphic symbol and optional sets of UML tagged values and constraints. However, the UML metamodel still gives the stereotype designer (too) many degrees of freedom: Arbitrary notations may be used for the definition of tag data and constraints. Thus, anything goes, ranging from a primitive (name, host) declaration up to a highly sophisticated definition of the syntax and semantics of a stereotype.

The support for the definition of stereotypes in current tools tends towards the minimum: it is mostly restricted to the modification of graphical symbols – that is to say, to the definition of decorative stereotypes.

In order to define stereotypes of all categories properly, the base language should provide a framework for the definition of stereotypes that is well adapted to these categories. The required elements are shown in Table 1.

Table 1. Elements of a proper stereotype definition

	Decorative Stereotypes	Descriptive Stereotypes	Restrictive Stereotypes	Redefining Stereotypes
SYNTAX:				
Unique name for the stereotype	yes	yes	yes	yes
Host (the language element(s) that can be stereotyped)	yes	yes	yes	yes
Concrete syntax: graphic symbol, etc.	yes	yes	yes	yes
Pragmatics (concept behind the stereotype, for what it shall be used), given with text	(yes)	yes	yes	yes
Syntactic properties of the stereotype (a set of: keyword type [initial value])	no	yes	yes	yes
SEMANTICS:				
Restrictions that have to be fulfilled by all model elements that are instances of the stereotype*	no	no	yes	yes
Formally defined semantics, including restrictions for model elements that are not instances of the stereotype	no	no	no	yes

* Restrictions that shall be formally analyzable and checkable must be formally specified, for example with a constraint language like OCL [8]. Any other restrictions may be stated semi-formally or informally.

In Figure 3, we give a short example how the definition of a restrictive stereotype should look like. We define a stereotype Observer which supports the application of the observer pattern [7].

```
stereotype Observer {                              /* the name of the stereotype */
    host Class;                     /* Classes can be stereotyped by this stereotype */
    properties {                             /* declaration of the syntactic properties
        String id;                                       the stereotype introduces */
        ['Observable', 'Observer'] role;
    }
    restrictions {                        /* restrictions for a restrictive stereotype */
        ( role = 'Observable' ) implies   /* Observable/Subject must have specific methods */
```

```
                ( ( exists a, d, n in self.operations |
                        (a = “attach”) and (d = “detach”) and (n = “notify”) )
                and ...         )
        and
        ( role = ‘Observer’ ) implies                      /* Observer must have update method and ... */
                ( ( exists u in self.operations |
                        u = “update”)
                and ( self.subject -> notEmpty )   /* there must be an observable for this observer ... */
                and ( exists c in self.subject |                 /* ... this means, an associated class ... */
                        exists s in c.stereotypes |                /* ... with a matching stereotype ... */
                                (s = Observer) and (s.role = ‘Observable’) and (s.id = id) )
                and ...         )
    }
}
```

Figure 3. Example for the definition of a restrictive stereotype Observer. When this stereotype is assigned to a class, an id and a role must be specified. The restrictions prescribe that depending on the role the class must have some specific methods and that an observer must have a corresponding observable/subject.

Figure 4 gives an example where the stereotype Observer is applied to three classes. It should be noted that the latest released version of UML Semantics ([15] p. 53) does allow a model element to be assigned more than one stereotype, whereas the Language Reference Manual [12] forbids this, so that the definition of class B in Figure 4 would be *illegal* according to [12] but *legal* according to [15]. The authors of [12] motivate their decision with the argument of simplicity ([12] p. 450). In our view, this is a bad decision that seriously restricts the applicability of stereotypes.

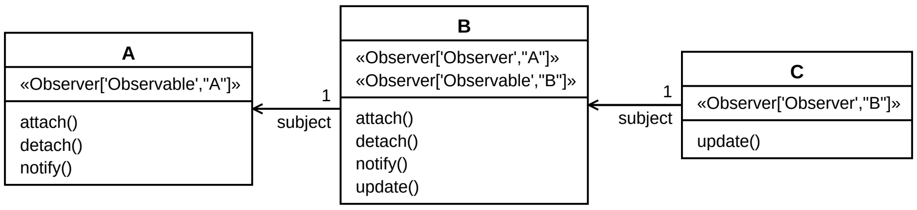
A
«Observer['Observable',"A"]»
attach()
detach()
notify()
1
subject
B
«Observer['Observer',"A"]»
«Observer['Observable',"B"]»
attach()
detach()
notify()
update()
1
subject
C
«Observer['Observer',"B"]»
update()

Figure 4. Example for the application of the stereotype Observer. In this example, Class B observes Class A and Class C observes Class B. Thus, Class B is acting both as an observable/subject and as an observer.

5 Guidelines for Stereotype Design

As mentioned earlier, designing stereotypes is a demanding task and the potential benefit of stereotypes heavily depends on taking the right design decisions.

From our experience with stereotypes we have assembled a preliminary set of guidelines for stereotype design.

General Advice

- Define a stereotype policy and enforce it: who (identify roles) has the right to define stereotypes of which category (e.g. according to our classification) for which purpose and with which scope (e.g. individual, project, department-wide, and company-wide).
- Make sure that every stereotype is properly defined and documented.
- Have every stereotype definition reviewed prior to using it.
- Avoid the creation of stereotypes when its scope is below the level of a project.
- Whenever you define a new stereotype, make sure that you will be able to maintain it in the scope and for the duration of its use.
- Make the stereotype definitions available to all people who need to know them and train these people how to apply and how to interpret them, respectively.
- Define less stereotypes and apply the existing ones more uniformly and with a wider scope.

Guidelines for Decorative Stereotypes

- Thoroughly examine the need for a decorative stereotype. Does the new symbol increase the semiotic value of a language element significantly? If you are not sure that there is a benefit in terms of increased understandability, then do not waste time with decorative stereotypes.
- Do not try to be an artist. Do not ever create a decorative stereotype because you have the feeling that a model looks better then.
- Use decorative stereotypes only to adapt the concrete syntax of a notation to a compulsory company or customer standard.

Guidelines for Descriptive Stereotypes

- Before defining a stereotype for an annotation or secondary classification, thoroughly determine that this is a required standard information with some concept behind it. A model style guide helps to decide.
- Decide whether it is better to include an annotation in the model or to document it separately. If the latter is true, do not define a stereotype.
- When annotations or classifications neither are a required standard nor have a clear concept behind them, use *notes* or other commenting constructs instead. *Notes* indicate that the interpretation of the given information is completely up to reader whereas a descriptive stereotype points out that there is a specific syntax and pragmatics in place.

- Do not try to cure symptoms. Descriptive stereotypes will not cure an unstructured model or the wrong choice of language element, diagram type, etc.

Guidelines for Restrictive Stereotypes

- Thoroughly investigate and discuss the need for the language extensions or modifications that shall be introduced with a restrictive stereotype.
- Never define restrictive stereotypes on the fly.
- Leave the definition of restrictive stereotypes to language and method specialists. For example, state in your stereotype policy (see above) that restrictive stereotypes may only be defined by your software methods group and that a formal validation and approval process has to be employed.
- Take care that a restrictive stereotype is really a restrictive stereotype and not a redefining one. State the semantics of the stereotype as precisely and formally as possible.

Guidelines for Redefining Stereotypes

- Do not do it. If you must do it, do it with extreme care and let only experienced language and method specialists do it.
- Explicitly forbid the definition of redefining stereotypes by individual engineers or within projects.
- Organize the definition of a redefining stereotype as a project of its own. Impose a rigorous design, validation and approval process.

6 Summary and Conclusions

Stereotypes are powerful, but care and experience is required to harness this power. Our classification helps to better understand the nature of stereotypes and to control their application. Every category in this classification represents a typical kind of applications for stereotypes. Hints for proper definition of stereotypes and preliminary design guidelines have been presented.

The discussion in section 3 and the guidelines in section 5 make clear that using decorative and redefining stereotypes both is highly problematic. Variations of the concrete syntax or the style of representation as well as a fundamental redefinition of the semantics of the base language should be done very restrictively only. Hence, descriptive and restrictive stereotypes are the most important ones in practice. Stereotypes from these two categories are especially useful to:

- make models more expressive by augmenting them with additional information in a standardized way

- compensate for deficits and weaknesses in a given modeling language in order to make it better adapted to some classes of problems or to given domains.

Stereotypes are no silver bullet. Their application does not automatically result in 'better' models. They always increase the complexity of the base language and introduce overhead for definition, training and maintenance. So, before introducing language extensions or modifications on the basis of stereotypes, it should be made sure that these are clearly beneficial.

Our further research on stereotypes will be directed towards empirical work on the usefulness of our classification, on better capabilities for the definition of stereotypes and on improving and validating the design guidelines. Finally, this endeavor should end up in a sound methodology for stereotypes.

References

[1] Booch, G. (1994): *Object-Oriented Analysis and Design with Applications,* 2nd ed. Redwood City, Ca.: Benjamin/Cummings.

[2] Champeaux de, D., Lea, D., Faure, P. (1993): *Object-Oriented System Development*. Reading, Mass., etc.: Addison-Wesley.

[3] Coad, P., Yourdon E. (1991): *Object-Oriented Analysis*. Englewood Cliffs, N. J.: Prentice Hall.

[4] Embley, D.W., Kurtz, B.D., Woodfield, S.N. (1992): *Object-Oriented Systems Analysis*. Englewood Cliffs, N. J.: Prentice Hall.

[5] Eriksson, H.-E., Penker, M. (1998): *UML Toolkit*. New York: John Wiley & Sons.

[6] Firesmith, D., Henderson-Sellers, B. H., Graham, I., Page-Jones, M. (1998): *Open Modeling Language (OML) – Reference Manual*. SIGS reference library series. Cambridge, etc.: Cambridge University Press.

[7] Gamma, E. (1995): *Design Patterns: Elements Of Reusable Object-Oriented Software*. Reading, Mass., etc.: Addison-Wesley.

[8] IBM et al. (1997): *Object Constraint Language Specification*; Version 1.1. [http://www.software.ibm.com/ad/oc]

[9] Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G. (1992): *Object-Oriented Software Engineering – A Use Case Driven Approach*. Reading, Mass., etc.: Addison-Wesley.

[10] Joos, S., Berner, S. Glinz, M. (1998): Stereotypen und ihre Verwendung in objektorientierten Modellen – eine Klassifikation [Stereotypes and Their Usage in Object-Oriented Models – A Classification (in German)]. In: K.Pohl, A. Schürr, G. Vossen (eds.): *Proceedings of the GI workshop Modellierung'98*. TR 6/98-I, University of Münster, Germany. 111-115.

[11] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. (1991): *Object-Oriented Modeling and Design*. Englewood Cliffs, N. J.: Prentice Hall.

[12] Rumbaugh, J., Jacobson, I., Booch, G. (1998): *The Unified Modeling Language Reference Manual*. Reading, Mass., etc.: Addison-Wesley.

[13] Selic, B., G. Gullekson, P. T. Ward (1994): *Real-Time Object-Oriented Modelling*. New York: John Wiley & Sons.

[14] Shlaer, S., Mellor, S.J. (1988): *Object-Oriented Systems Analysis: Modelling the World in Data*. Englewood Cliffs, N. J.: Prentice Hall.

[15] Unified Modeling Language Specification (1998): *UML Semantics v1.1*. Framingham, Mass: Object Management Group. [http://www.omg.org/techprocess/meetings/schedule/Technology_Adoptions.html#tbl_UML_Specification]

[16] Wirfs-Brock, R., Wilkerson, B., Wiener, L. (1994): Responsibility-Driven Design: Adding To Your Conceptual Toolkit. *ROAD* **1**, No. 2; (July-August 1994), 27-34.

[17] Wirfs-Brock, R., Wilkerson, B., Wiener, L. (1993): *Designing Object-Oriented Software*. Englewood Cliffs, N. J.: Prentice Hall.

[18] Wordsworth, J.B. (1992): *Software Development with Z*. Reading, Mass., etc.: Addison-Wesley.