Proc. 6th Australian Workshop on Requirement Engineering (AWRE'2001),
University of New South Wales, UNSW Printer Room, Sydney, Aybuke Aurum (Ed.)

Using OCL for Defining Precise, Domain-Specific UML Stereotypes

Martin Gogolla

University of Bremen, Computer Science Department
Postfach 330440, D-28334 Bremen, Germany
EMail gogolla@informatik.uni-bremen.de
Web www.db.informatik.uni-bremen.de
[Phone/Fax] ++49-421-218-[7469/3054]

Abstract: This paper studies the extension mechanisms of the Unified Modeling Language UML. It introduces a new appproach for defining UML stereotypes and shows its usefulness by considering domain-specific requirements for databases. The stereotypes introduced possess a precise meaning by translating them into expressions formulated in the Object Constraint Language OCL. Apart from proposing a general framework for stereotype definition, this work is also one step in the development of a domain-specific requirements language for information systems.

Keywords: Domain-specific requirement, database, stereotype, UML metamodel, Object Constraint Language OCL.

1 Introduction

The Unified Modeling Language UML [8] allows for three extension mechanisms: tagged values, constraints, and stereotypes. This paper concentrates on the latter two and applies these concepts to meet the need for domain-specific requirement languages.

The basic idea of the paper is to use UML stereotypes for the formulation of domain-specific requirements and to employ the Object Constraint Langauge OCL [12, 10, 5] to define a precise meaning for the introduced stereotypes. This rather general approach can be used in various domains. We show its usefulness by defining stereotypes for concepts of relational databases. The paper points out how to use the UML metamodel to formally define such stereotypes. Interestingly, up to now there seems to be little, if no work on the formal definition 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 object diagrams in order to deal with situations where generic definitions are needed.

Relational database issues are probably not spectacular but on the other hand important for everyday practical work. And this field is well-suited for our pupose because the fundamental topics of relational databases are quite well and formally understood. Thus apart from a general approach to precise stereotype definiton the paper defines stereotypes for typical relational databases requirements.

This definition of relational database requirement is a case study for the formulation of general domain-specific requirements with UML and OCL. If the concepts, the formal description of the respective domain and the relationship to notions in UML and OCL are clear, any set of requirements can be formulated in the way as we do here describe the database domain. A prerequisite of our approach is that the concepts and formal definitions have to be given as extensions of the UML metamodel, and the respective stereotypes have to be expressible analogously to the relational requirements which we use for demonstration purposes. For example, in the relational database domain we have notions like relation and key condition, and these have correspondent notions in UML under the names class and invariant. We think that the richness of concepts in UML supports a large variety of domains.

Our approach has connections to other related work. Metamodels of object-oriented languages have not only been studied for UML [8], but also for example for OML [7]. General considerations and classifications of UML stereotypes have been discussed in [2, 4]. UML-related questions concerning application-dependent stereotypes [11] and application-dependent development have been studied for example in [6, 3, 1, 9].

The structure of the rest of the paper is as follows. Section 2 discusses the basic idea of the paper by an example. Section 3 points out on which part of the UML metamodel our approach relies. Sections 4 and 5 explain the technical details of our ideas by considering the stereotypes <<singleKey>> and <<relation>>. Section 6 shortly touches further stereotypes which can be handled in the same way. The paper closes in Sect. 7 with some final remarks.

2 The Basic Idea

The aim of this paper is to provide domain-specific UML language constructs. We study such constructs by considering concepts frequently occuring in the relational database field. Let us explain the basic idea of our approach by one such important issue, namely the notion of a key for a relation. A key is a set of attributes uniquely identifying tuples in a relation. UML does not directly support attributes being keys. Thus this concept must be newly introduced. The UML language feature we use to formulate keys are stereotypes.

In Fig. 1, we have shown on the left a relation called Person with three attributes. The relation is presented in form of a UML class, and two of the attributes are marked with the stereotype <<key>>. Thus the key for relation Person consists of these two attributes. On the right of Fig. 1, we show how to resolve the <<key>> stereotype. The class Person with the stereotype is transformed into a class without stereotypes but with an additional OCL invariant attached to the class. This OCL constraint is the implementation of the stereotype and formulates the condition that, if two tuples in relation Person coincide in their key attributes, then the tuples must be identical. This can be said also the other way round: Two distinct tuples (p1<>p2) must possess at least one key attribute for which the key attributes in the tuples are distinct (p1.firstName<>p2.firstName or p1.lastName<>p2.lastName).

The overall idea of the paper is to show how to define presice, domain-specific UML stereotypes and demonstrate the general approach by providing a set of such stereotypes like <<key>> for concepts of the relational database world. As for any domain-specific requirements, this set of concepts should make the development in the domain (relational database schemas) with UML easier, because the stereotypes are chosen to present typical application constraints (typical relational database integrity constraints). We will also explain the technical details which are needed for the transformation of UML class diagrams with stereotypes to pure UML classes with explicit OCL constraints. The approach is general enough in order to be applied not only to relational database systems, but also to other domain-specific areas like for example the design of web pages.

3 Excerpt from UML Metamodel

In Fig. 2, that part of the UML 1.4 [8] metamodel is pictured which deals with stereotypes and constraints. We use UML 1.4 throughout the paper. Important for our pupose are the classes ModelElement, Stereotype, and Constraint and the three associations between them. Let us have a closer look at these elements.

Classes: ModelElement is a general metaclass in the UML metamodel having most other metaclasses as specializations. Constraint allows to restrict the semantics of a model element in textual form, for example by giving an OCL expression. Stereotype is designed to specialise a subclass of ModelElement and to give it a restricted interpretation. Name and BooleanExpression are categorized in UML as data types. They are used here for attribute domains.

Associations: The association on the left with role names extendedElement and stereotype

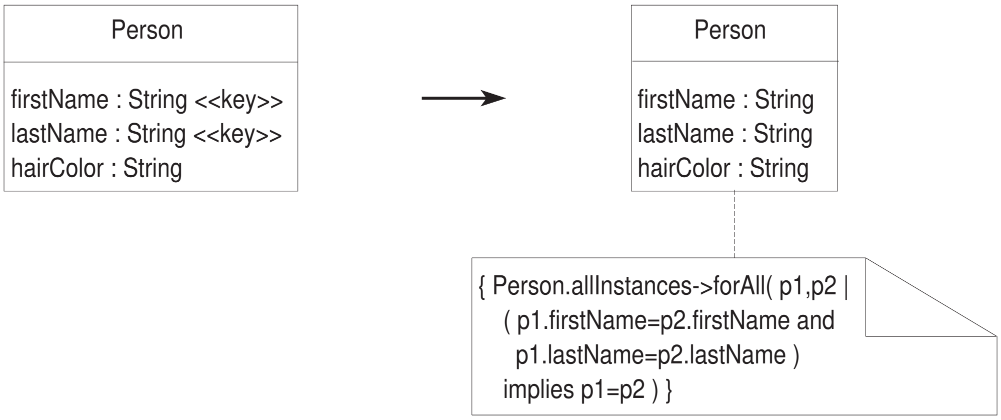

Figure 1: Basic Idea of `<<key>>` Stereotype

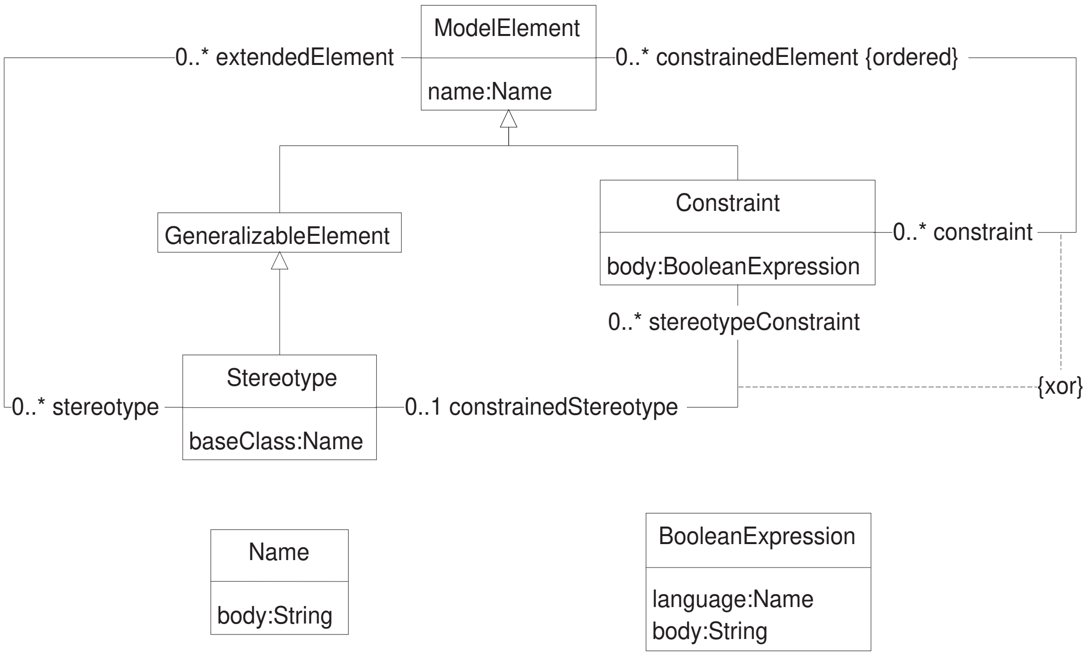

Figure 2: Relevant Part of the UML Metamodel Concerning Stereotypes

describes that a ModelElement object can be annotated with Stereotype objects. The association in the middle with role names constrainedStereotype and stereotypeConstraint expresses that a Stereotype object can be connected to a Constraint in order to specify precisely the meaning of the Stereotype. The association on the right with role names constrainedElement and constraint allows for connecting a Constraint to the ModelElement objects which are to be restricted. The xor restriction requires that a Constraint is connected either to a Stereotype or to a ModelElement but not to both.

Attributes: Each ModelElement possesses a name. The name of the ModelElement is unique within a corresponding name space. A Stereotype refers to a baseClass. This is the name of the metaclass the stereotype extends. A Constraint has a body giving the restricting BooleanExpression which again has a body represented as a String. This string must be a valid expression in the language specified by the language attribute.

Figure 2 contains important elements of Figs. 2-5, 2-10, 2-11, and 2-12 of UML 1.4. Concerning the different versions of UML, we observe that the extension mechanism part of the UML metamodel seems to be a bit unstable because the major versions of UML, namely UML 1.1, UML 1.3, and UML 1.4 all differ in subtle details. But the basic idea is of course the same.

4 Stereotype <<singleKey>>

This section explains the first technical details of stereotype definitions with OCL by means of an example. The stereotype we define is the <<singleKey>> stereotype. This stereotype can only be used for relations with exactly one key attribute. This restriction will be droped later by the <<key>> stereotype. We decided however to introduce the <<singleKey>> stereotype first in order to explain the general approach by a simple example. In the left part of Fig. 3 an example for the use of <<singleKey>> is shown, and in the right part of Fig. 3 the example class is pictured as an object diagram of the UML metamodel. This diagram shows that Class objects possess Attributes as features and Attributes are owned by Classes.

The precise definition for the stereotype <<singleKey>> as an object diagram of the UML metamodel is pictured in Fig. 4. This definition is explained best by considering the objects defined there.

singleKey: The definition of the stereotype <<singleKey>> has first to define an object of class Stereotype with the name of the stereotype to be defined, namely singleKey. The <<singleKey>> stereotype is designed to be applied to attributes and therefore its attribute feature baseClass (which is of type Name which in turn possesses a body attribute) is set to "Attribute". This Stereotype object is connected via the extendedElement role name to the ModelElement which is restricted as for example the attribute isbn in Fig. 3. The singleKey object is further described by two Constraint objects.

applicabilityConstraint: This first constraint restricts the applicability of the singleKey object. As specified by the language attribute the constraint is an OCL constraint. The body of the constraint is an OCL expression given as a String value. This constraint defines with a let construct a local variable extElems representing the collection of objects which the <<singleKey>> stereotype is supposed to extend. It requires that each such element is an attribute[1] and that there is only one such attribute because that was the pupose of the <<singleKey>> stereotype. The main reason for the restriction is however that the following constraint keyConstraint becomes relatively simple compared to the general case with a key consisting of an arbitrary number of attributes. As a remark explaining the attribute types in the object rectangles, we note that applicabilityConstraint.body

[1] One might wonder why it is neccessary at all to require the extended elements to be attributes since this seems to be said already by giving the baseClass. However, the baseClass returns only a Name object, and the UML metamodel does not pose any well-formedness rule on the class of the extended elements. See UML 1.4, Section 2.6.3.

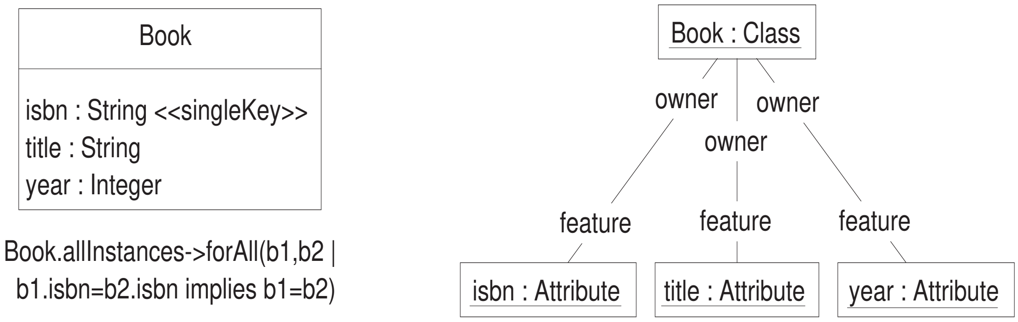

Figure 3: Example for Stereotype <<singleKey>>

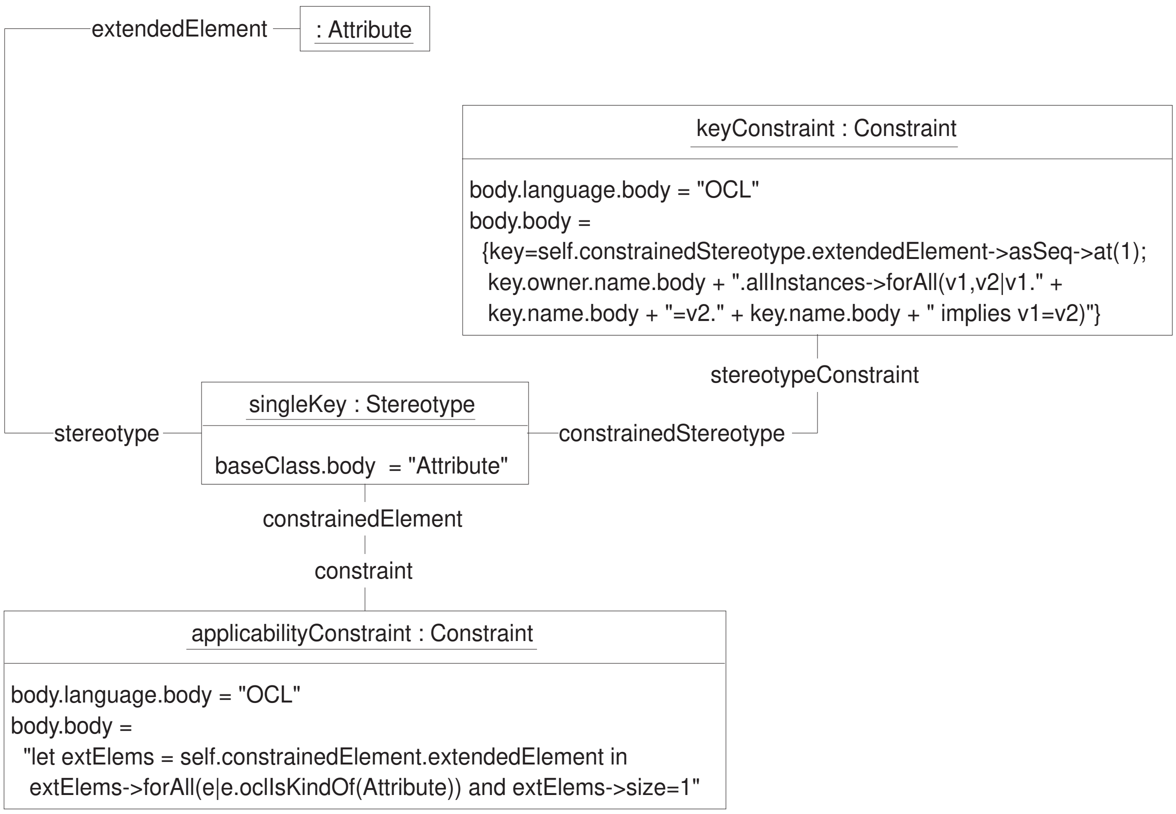

Figure 4: Metamodel Definition of Stereotype <<singleKey>>

is of type BooleanExpression, applicabilityConstraint.body.language is of type Name and applicabilityConstraint.body.body and applicabilityConstraint.body.language.body possess type String.

keyConstraint: This second constraint defines the actual key condition. Its value is a constraint like the one shown in Fig. 3. There is however one technical complication here in contrast to the applicabilityConstraint. One cannot just give the constraint as a String constant, because not all subexpressions are known in advance. One has to navigate from the constraint and the stereotype to the attribute where the stereotype was written down and from there to the class which is the owner of the attribute. The variable key is an auxiliary variable which holds the attribute where the <<singleKey>> stereotype was written down.[2] The construction of the constraint then retrieves the name of the owning class with the expression key.owner.name.body and the name of the key attribute with expression key.name.body. These values are concatenated by the operator + in the correct order with appropriate String constants intertwined.

The content of the constraint is given by the last two lines of the object rectangle representing a string expression. Once again, we emphasize: One cannot give a single String constant in this place for the constraint, but the constraint has to be constructed by using String expressions resulting from navigation in the metamodel object diagram and by using appropriate String constants.

Let us finally discuss the different roles the applicabilityConstraint and the keyConstraint play. The applicabilityConstraint is a constraint which is attached to the stereotype as a ModelElement as revealed by the role names. It controls the correct application of the stereotype. The keyConstraint is a constraint which is attached directly to the stereotype and constitutes the main content of the stereotype. The intention is that applicabilityConstraint is checked first to see whether the stereotype is applied correctly and afterwards the keyConstraint cares about the correct constraint generation.

[2]On the right side of the assignment for variable key the item asSeq stands for the OCL operation asSequence. Throughout the paper, we abbreviate the OCL keyword Sequence by Seq.

5 Stereotype <<relation>>

This section explains the second part of the technical details for precise stereotype definition. In Fig. 5 we show an application of the stereotype <<relation>>. This stereotype is applied to a class and requires that two objects of the respective class must be distinct by at least one attribute value. This is analogous to the requirement that a relation consists of a set of tuples, i.e. two tuples are distinguishable by their components. This requirement does not hold in general for objects, i.e. in general, two distinct objects with identical attribute values are allowed.

The metamodel definition of stereotype <<relation>> is displayed in Fig. 6. The object relation of class Stereotype is the actual object of interest. As before, the two attached constraints constitute the central definition.

applicabilityConstraint: This constraint controls correct application. It requires that this stereotype is applied to exactly one Class object for which all attributes have data types as result types and the class object possesses at least one attribute.

relationConstraint: For this constraint, yet another complexity aspect comes in because the stereotype is intended to be applied to a class with an arbitrary number of attributes. Thus the generated constraints like the one given in Fig. 5 will differ in size according to the number of attributes. This means that the constraint generation process must take the number of attributes into account and must use a loop accessing each single attribute. We now consider the variables used in the relationConstraint object. The variable attrs holds the sequence of attributes of the class on which the stereotype <<relation>> was written down. Variable relName refers to the name of this class as a String value. attrNum captures

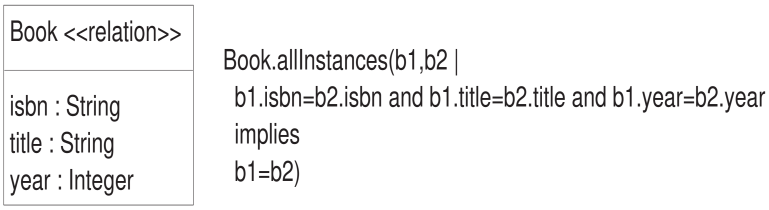

Figure 5: Example for Stereotype <<relation>>

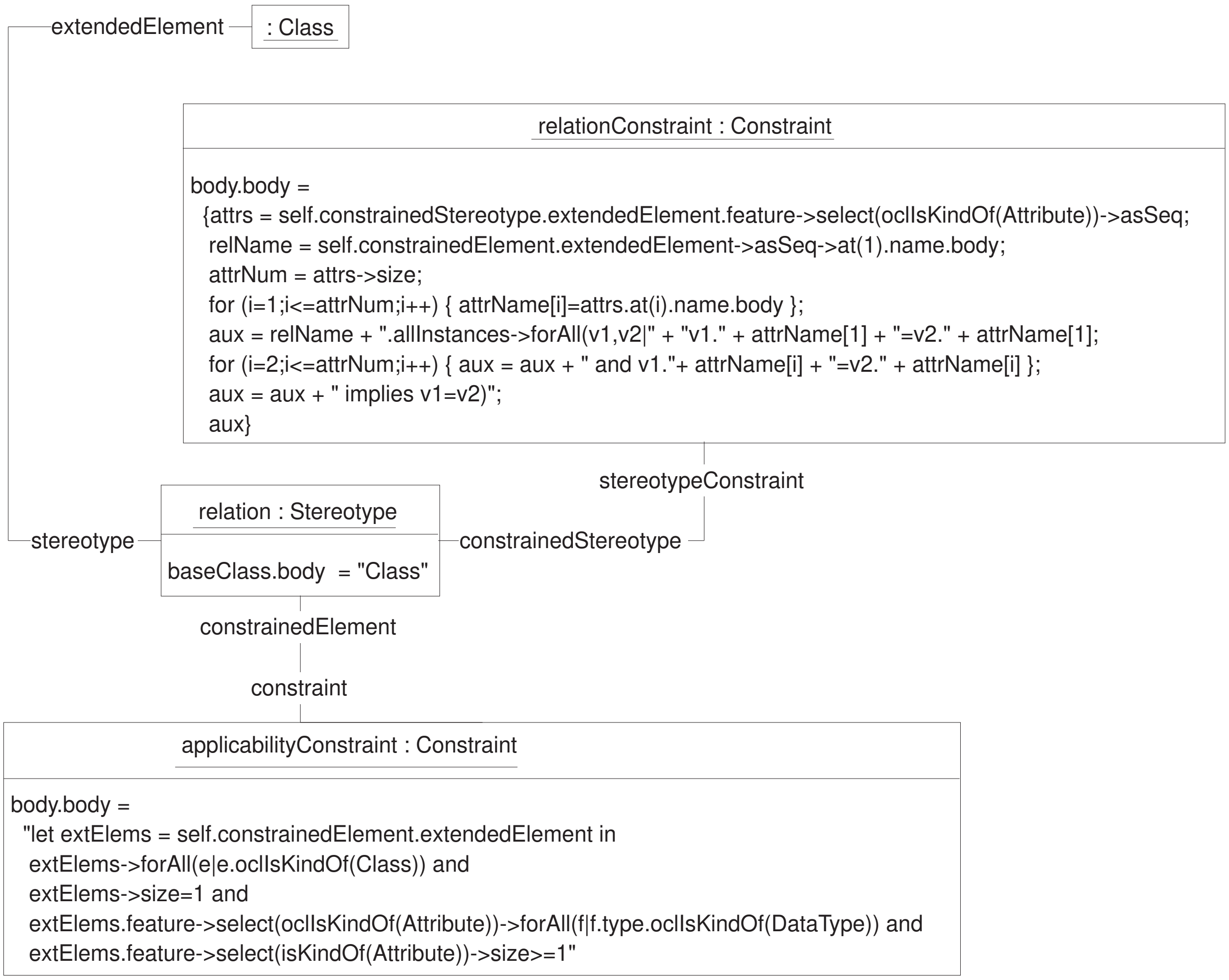

Figure 6: Metamodel Definition of Stereotype <<relation>>

the number of attributes. attrName is an array holding the names of the attributes. The variable aux is an auxiliary variable used for the construction of the constraint. The value of this variable is the value of the constraint body. We emphasize, that we need a sort of programming language to define the resulting constraint. It is not sufficient to deal only with simple String constants.

6 Further Stereotypes

We now come back to the promised definiton of a general <<key>> stereotype. An example for its use has already been presented in Fig. 1. The technical definition is given in Fig. 7 completely analogously to the stereotypes defined before. The applicabilityConstraint requires that the stereotype is applied to attributes which are owned by a common class. The keyConstraint defines with the help of some auxiliary variables the actual contraint. This construction is very similiar to the construction in the <<relation>> stereotype.

The reader might have noticed that writing down <<relation>> is equivalent (in the sense that the generated user model constraints are equivalent) to notating <<key>> on all attributes of the relation. Furthermore, if one specifies <<key>> for at least one attribute, this implies the implicit stereotype <<relation>> for the class.

In exactly the same way as we have defined above the stereotypes <<singleKey>>, <<relation>>, and <<key>> we can define stereotypes for other important notions of the relational database field. In Fig. 8, we show the use of an <<inclusionDependency>>. In this case, the baseClass of the stereotype would have been given as Dependency. The respective constraint would require that the attribute values indicated in the source relation also appear in the target relation. Further typical relational integrity constraints which could be handled within our approach include functional and join dependencies. We refrain from presenting the technical details since they are analogous to the above defined stereotypes.

7 Conclusion

In this paper we have used OCL to define precise, domain-specific stereotypes. The general approach for the definition was illustrated by examples from relational database integrity constraints. Our appoach is based on the UML metamodel and consists of the following steps.

1. A stereotype definition is given by a generic object diagram referring to the part of the UML metamodel relevant for stereotypes as depicted in Fig. 2. This object diagram mainly introduces the following three objects.

2. An object nameForStereotype of class Stereotype gives a name to the stereotype and specifies its base class.

3. An object applicabilityConstraint of class Constraint is restricting the syntactic applicability of the stereotype by giving an OCL expression.

4. An object actualConstraint of class Constraint is generated for inclusion in the user model and is defining the semantics of the stereotype. This constraint may be constructed by using programming language-like features as for example variables and loops.

The application of the defined stereotype is initiated by writing down the respective stereotype in the concrete syntactical representation of a UML user model, for example by notating the stereotype close to the element which is to be restricted. This UML user model, which is annotated with the stereotype, must then be translated into the abstract syntax, i.e. an object diagram according to the UML metamodel. In this step, a new stereotype object has to be introduced for each occurrence of the stereotype, and the stereotype object is linked to the model element where the stereotype was notated. Following this, applicabilityConstraint checks for proper application, and, if this was successful, the program in the actualConstraint generates the constraint to be inserted in the user model.

Definitions for the stereotypes like the ones we have given in the object diagrams will also appear

Figure 7: Metamodel Definition of Stereotype <<key>>

Figure 8: Example for Stereotype <<inclusionDependency>>

in a (meta-)case tool which allows to define stereotypes. Thus this work can also be seen as a contribution to achieve concepts for the implementation of stereotypes in (meta-)case tools.

References

[1] John Artim, Mark van Harmelen, Keith Butler, Jan Gulliksen, Austin Henderson, Srdjan Kovacevic, Shijian Lu, Scott Overmyer, Ray Reaux, Dave Roberts, Jean-Claude Tarby, and Keith Vander Linden. Incorporating work, process and task analysis into commercial and industrial object-oriented systems development. *ACM SIGCHI Bulletin*, 30(4):33–36, 1998.

[2] 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.

[3] Maurits C. Blok and Jacob L. Cybulski. Reusing UML specifications in a constrained application domain. In *Proc. 5th Asia-Pacific Software Engineering Conf. (APSEC '98)*, pages 196–202. IEEE Computer Society, 1998.

[4] 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.

[5] 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 64–91. Springer, Berlin, LNCS, 2001.

[6] M. Gong, L. Scott, and R. Offen. MetaBuilder: A generic CASE tool builder. In *Proc. 4th Asia-Pacific Software Engineering and Int. Computer Science Conf. (APSEC'97)*, pages 435–444. IEEE Computer Society, 1997.

[7] 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.

[8] OMG, editor. *OMG Unified Modeling Language Specification (1.4)*. OMG, 1999. `http://www.omg.org`.

[9] Björn Regnell, Per Runeson, and Claes Wohlin. Towards integration of use case modelling and usage-based testing. *Journal of Systems and Software*, 50(2):117–130, February 2000.

[10] 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 38–63. Springer, Berlin, LNCS, 2001.

[11] 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.

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