A&QT R 2004 (THETA 14)
2004 IEEE-TTTC -International Conference on Automation, Quality and Testing, Robotics
May 13 – 15, 2004, Cluj Napoca, Romania

A BETTER REPRESENTATION FOR CLASS RELATIONSHIPS IN UML USING OFL META-INFORMATION

**Dan Pescaru (*), Philippe Lahire (**), Ciprian Chirila (*),
Emanuel Tundrea (*)**

*(*) "Politehnica" University of Timisoara, Automation and Computer Science Faculty,
Computer Science Department, V. Parvan no. 2, B622, Timisoara, Romania
dan@cs.utt.ro, chirila@cs.utt.ro, emanuel@emanuel.ro
(**) University "Sophia Antipolis" Nice, I3S Laboratory (UNSA/CNRS),
Les Algoritmes, bat. Euclide B 2000, Route des Lucioles BP121,
F-06903 Sophia Antipolis CEDEX, France
Philippe.Lahire@unice.fr*

ABSTRACT

In the last decade software industry exists a general opinion about the evident gap between object-oriented modeling languages and programming languages with a great impact on products reliability, testability and maintenance. Many companies do not use yet Unified Modeling Language (UML), which is the Object Management Group (OMG) standard of object-oriented modeling languages since many years. Indeed, even they use UML in the analyzing phase, they prefer to jump over implementation model for application. Instead they are using to have only an ad-hoc model that resides directly in implementation. First explanation consists in contradiction between generality of UML and specificity of application model after implementation in a programming language. The reaction of OMG against these critics was the definition of UML Profiles as standard means to adapt the UML to some domain-specific needs. In this framework, this paper propose a precise representation of programming language class relationships that can be included in a language specific Profile. This goal is achieved using meta-information about the programming language described in a meta model named OFL.

Keywords: UML, application modeling, OFL, meta-programming

1. INTRODUCTION

The Unified Modeling Language (UML) [1] is a standard introduced by OMG . It is used in a wide area of contexts, by people coming from different cultures, many of them considering (more or less justified) their case special and asking for a deviation from the standard in the form of a particular tuning of UML. A hard-coded UML precise semantics would preclude the existence of these tunings and thus would be practically unacceptable. Considering this, the OMG proposed a definition for UML Profiles as standard means to adapt the UML to some domain-specific needs.

The goal of this research is to define construction belonging to specific Profiles that bring closer object oriented programming languages and UML. The problem appears when the UML is used to create an implementation model. After the implementation of this model, the application will contain itself an intrinsic model. Because a programming languages has a more precise semantic than UML, this two models will be different. If the specification change the problems will appear at reengineering phase.

If we think at UML Profile solution, the problem is how to specify this profile in order to fill the gap. This problem is harder if we think in terms of number of existing programming languages, each of them with different versions and flavors. The approach presented here tries to use meta-information about a programming language described in a meta-meta model called OFL [2, 3].

2. UML AND UML PROFILES

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML is, as its name implies, a modelling language and not a method or process. UML is made up of a very specific notation and the related grammatical rules for constructing software models.

UML in itself does not prescribe or advise on how to use that notation in a software development process or as part of an object-oriented design methodology. It describes the notation for classes, components, nodes, activities, work flow, logical, objects, states and how to model relationships between these elements. UML also supports the notion of custom extensions through stereotyped elements.

Any modeling language need support for application constraints as assertions. In UML they are modeled in Object Constraint Language OCL [4].

An UML Profile consists of a set of UML extensions (stereotypes, tagged values, constraints) and is supplemented by specifications of the mappings of the domain concepts to those extensions, and specifies additional well-formedness rules (expressed in OCL or in natural language). Each particular profile is described through its Virtual Meta-model.

The general UML Profile mechanism is discussed in [5]. It presents how specific domains, which require a specialization of the general UML meta-model, can define an UML profile. The goal is to focus UML to describe more precisely the considered domains.

Even as concrete UML profiles have started to emerge [6, 7], use of the profiling mechanism is still discussed [8].

3. THE OFL MODEL

OFL is the acronym for Open Flexible Languages [2, 3] and the name of a meta-model for object oriented programming languages based on classes. It is developing in France at University ”Sophia Antipolis” of Nice. It relies on three essential concepts of object oriented languages: the descriptions that are a generalization of the notion of class, the relationships such as inheritance or aggregation and the languages themselves. OFL provides a customization of these three concepts in order to adapt their operational semantics to the programmer’s needs. It is then possible to specify new kind of relationships and classes that could be introduced in an existing programming language in order to improve its expressiveness, its readability and its capabilities to evolve.

Rather than allowing redefining language behaviors thanks to algorithms, OFL propose a set of parameters. At first reading the OFL approach can be summed up as the search for a set of parameters whose value determines the operational semantics of an object language based on classes. Parameters represents the main features of the behaviors of these three important notions that are called concept-relationship, concept-description and concept-language. For instance, concerning the concept-relationship, the value of the Cardinality parameter allows to specify if it is simple or multiple. The operational semantics of each concept must adapt to the value of its parameters. This is achieved thanks to a set of action's algorithms whose execution depends on these values. This paper consider the original model extended through modifiers [9].

Figure 1 presents the OFL Architecture in context of a very basic application. It is organized on three levels: OFL (concepts and atoms), OFL-Components and OFL-Application.

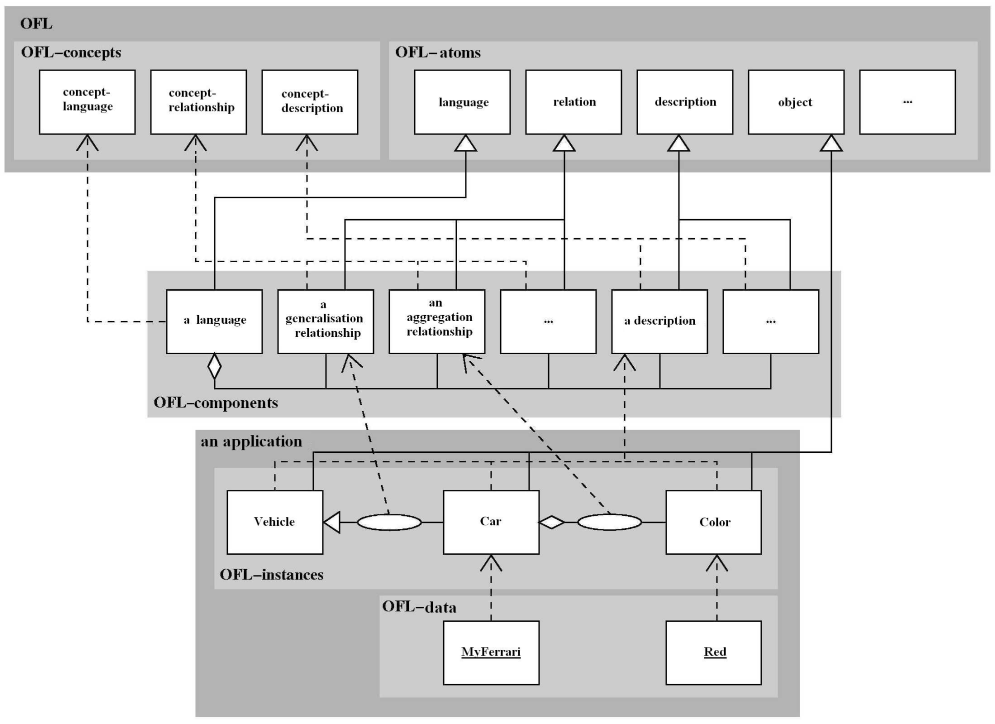

Fig. 1. The OFL Architecture

4. DEFINITION FOR VIRTUAL META-MODEL ELEMETS.

A virtual meta-model is a formal model of a set of UML extensions, expressed in UML. Consequently, this section defines elements regarding relationships representation that have to be included in Profiles designed for object oriented languages. These Profiles will be named generic as OFL-ML Profiles. According to OFL architecture, the Stereotypes introduced in the Virtual Meta-model corresponds to two kinds of relationships: OFL-ImportRelationship and OFL-UseRelationship. It also adds the necessary TaggedValues, Constraints, and Common Model Elements to complete the Profile.

These stereotypes could be used in modeling tools to generate corresponding instances of OFL elements and to fill them with appropriate information.

4.1. Representation of OFL-ImportRelationships

The OFL-import relationship is a generalization of the inheritance mechanism found in object oriented languages. The meta-programmer has responsibility to create an OFL relationship component for each import relationships existing in the modeled language. The Profile will contains all necessary elements in order to represents all these components.

The abstract stereotype <<OFLImportRelationship>> is the base for all the concrete stereotypes representing OFL ImportRelationhip components of the considered language. The name of the generated stereotypes are the same as the name of the OFL components with ”Component” prefix removed (ex. for a component ”ComponentJavaExtends”, a stereotype named <<JavaExtends>> will be created).

All relationships stereotyped as specialization of <<OFLImportRelationship>> will have associated a set of tagged values. Values of these elements correspond to some OFL-AtomRelationship characteristics. These tagged values are presented in Table 1. In addition, one tagged value will exists for each modifier associated with a relationship component.

Table 1. OFL-ML Tagged Values for OFLImportRelationhip

Tagged-Value Name	**Tagged-Value Value**	**Comment**
abstractedFeatures	string (list of feature names)	list of concrete methods that are abstracted
effectedFeatures	string (list of feature names)	list of abstract methods that are effected
hiddenFeatures	string (list of feature names)	list of features that are hidden
redefinedFeatures	string (list of feature names)	list of features that are redefined
renamedFeatures	string (list of feature names)	list of features that are renamed
removedFeatures	string (list of feature names)	list of features that are removed
shownFeatures	string (list of feature names)	list of features that pass the relationship unchanged

All modifiers constraints defined at the level of relationship components will be added. Transformation rule will translate all characteristics of relationships components into corresponding tagged values:

(1) *self.relationshipCharacteristic->forall(f:Feature|f.modifiers->includes('modifier_name'))*

translated into:

```
(1a) self.stereotype.taggedValue->forall(t:taggedValue |
        ( t.name = 'relationshipCharacteristic' and t.values->includes(feature_name) )
                imply
            self.parent.features->forall(f:Feature | f.name = feature_name
                    imply
                f.stereotype.taggedValue->select(name = 'modifier_name')->size = 1))
```

Additionally, several OFL Parameters have to be considered when constraints are designed. The considered parameters are: cardinality, repetition, circularity, feature_variance, abstracting, effecting, masking, redefining, renaming, removing and showing. Also the characteristic AtomLanguage:: validRelationships have relevance in this context.

Considering *ConceptRelationship::cardinality* parameter, it specify the cardinality of relationship as an integer value n in the meaning of cardinality 1-n. As an example, for simple inheritance n = 1 and the cardinality is 1-1. Constraint related with this parameter will check conformance with cardinality specification. If cardinality is ∞ no constraint is necessary.

```
Rule context: cardinality ≠ ∞
context ComponentRelationhip(OFLImportRelationship)
inv: self.child.generalization->select( gen |
        gen.isStereotyped('ComponentRelationship')
                and
        gen.child = self.child)->size = n)
```

4.1. Representation of OFL-UseRelationships

The OFL-Use relationship is a generalization of the aggregation mechanism found in object oriented languages. The meta-programmer has responsibility to create an OFL relationship component for each kind of use relationships existing in the modeled language.

The abstract stereotype <<OFLUseRelationship>> is the base for all the concrete stereotypes representing OFL UseRelationhip components of the considered language. As for import relationships presented in the section above, the name of the generated stereotypes are the same as the name of the OFL components with ”Component” prefix removed (ex. for a component ”ComponentJavaAggregation”, a stereotype named <<JavaAggregation>> will be designed). Also, same way as for import relationship, all use relationships stereotyped as specialization of <<OFLUseRelationship>> will have associated a set of tagged values that corresponds to some OFL-AtomRelationship characteristics: *hiddenFeatures*, *renamedFeatures*, *removedFeatures* and *shownFeatures*.

All associations that correspond to an OFL use relationship must have exactly two ends that correspond to source and target of relationship.

```
context ComponentRelationhip(OFLUseRelationship)
inv:  self.allConnections->size = 2
```

Some constraints regarding parameters of OFL-concept-relationship generated for import relationships are valid also for use relationships. In this context, the OFLUseRelationship stereotype will replace OFLImportRelationship as ancestor of ComponentRelationship stereotype. Also, UML-associations attribute will replace the UML-generalization. This attribute is a set that contains all association relationships in which considered classifier is involved. Considering parameter ConceptRelationship:: cardinality, transformed constraint will be the following:

Rule context: cardinality ≠ 1

```
context ComponentRelationhip(OFLUseRelationship)
inv: self.child.associations->select( assoc |
        assoc.isStereotyped('ComponentRelationship')   and
        assoc.child = self.child)->size = n
```

The list of parameters that are significant in context of an use relationship is: cardinality, repetition, circularity, masking, renaming, removing and showing. Constraints will consider all these values in context of target language.

5. CONCLUSION AND FUTURE WORK.

This paper present an approach for describing UML Profiles for object oriented programming languages modeled by OFL. It is focused on describing a detailed model for class relationships. It enrich original UML elements with features that allow a better representation of these relationships. The main achieved goal is to fill the gap between programming language expressivity and modeling language semantics.

The future work include a better models for class entities and the integration of these elements into several profiles designed for commercial languages like C++, Eiffel, Java or C#.

6. REFERENCES

[1] Object Management Group OMG (2003) - "Unified Modeling Language Specification, version 1.5", 1st ed., *http://www.omg.org*

[2] P. Lahire, P. Crescenzo, and A. Capouillez (2002) - "Le modele OFL au service du m´etaprogrammeur - application a Java", *proceedings of LMO 2002,* Montpellier, France

[3] P. Crescenzo and P. Lahire (2002) - "Customisation of Inheritance", Springer Verlag, *LNCS series, ECOOP'2002 (The Inheritance Workshop) and Proceedings of the Inheritance Workshop at ECOOP 2002,* University of Jyvskyl, Finland

[4] R. Hennicker, H. Hussmann, and M. Bidoit (2002) - "Object Modeling with the OCL: The Rationale behind the Object Constraint Language", *Springer Verlag, LNCS series,* volume 2263

[5] P. Desfray (1999) - "White Paper on the Profile Mechanism", *OMG document ad/99-04-07, http://www.omg.org*

[6] Object Management Group OMG (2001) - "UML Profile for EJB Specification" , *Version 1.0, http://www.omg.org*

[7] Object Management Group OMG (2002) - "UML Profile for CORBA Specification", *Version 1.0, http://www.omg.org*

[8] C. Atkinson and T. Kuhne (2000) - "Strict profiles: Why and how", *Springer Verlag, LNCS series, volume 1939, UML 2000 Third International Conference, University of York, UK*

[9] D. Pescaru and P. Lahire (2003) - "Modifiers in OFL: An Approach for Access Control Customization", *The 9th International Conferences on Object-Orinted Information Systems - OOIS'03, WEAR workshop, Geneva, Swizerland*