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

Works made in the last decade according to software development show a general agreement about the evident gap between object-oriented modeling languages and programming languages. This gap has 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 for several years. Indeed, even if they use UML during the analyzing process, they prefer to jump to the implementation model for the development of application. Instead they are using only an ad-hoc model that resides directly on implementation. A first explanation is that the lack of semantics of the entities of UML model contrasts with the specificity of the application model after its implementation in a programming language. The reaction of OMG against these critics was the definition of UML Profiles as standard means for adapting the UML to some domain-specific needs. In this framework, this paper propose a precise representation of programming language class relationships that can be introduced 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 communities, many of them considering (even if it is more or less justified) their case as 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 precise

framework for the definition of UML Profiles which would act as a standard way to adapt UML to some domain-specific needs.

The goal of this research is to define construction belonging to specific Profiles in order to make object oriented programming languages and UML closer one from the other. The problem appears especially when UML is used to create an implementation model. After the implementation of this model, the application will contain itself an intrinsic model. Because programming languages has a more precise semantic than UML, these two models will be different. This introduces a serious gap between the model and its implementation [10]. If the specification changes then problems may appear during the reengineering phase.

If we consider the definition and the use of UML Profiles, the main problem is about 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 in this paper proposes to use meta-information dedicated to the description of programming languages, which are described in a meta-meta model called OFL [2, 3], developed at "Sophia Antipolis" University of Nice.

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 suggests it, a modeling language and not a method or process. UML is made up with a very specific notation and 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 needs support for application constraints as assertions. In UML they are modeled with the Object Constraint Language OCL [4].

An UML Profile consists of a set of UML extensions (stereotypes, tagged values, constraints) and it includes specifications dealing with the mapping 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 some specialization of the general UML meta-model, may benefit from the definition of an UML profile. The goal is that UML provides a more accurate description of the considered domains. Even if concrete UML profiles have started to emerge [6, 7], the 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 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 the redefinition of 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. Parameter 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 itself 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 considers the original model extended through modifiers. The extension was a result of a previous work [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.

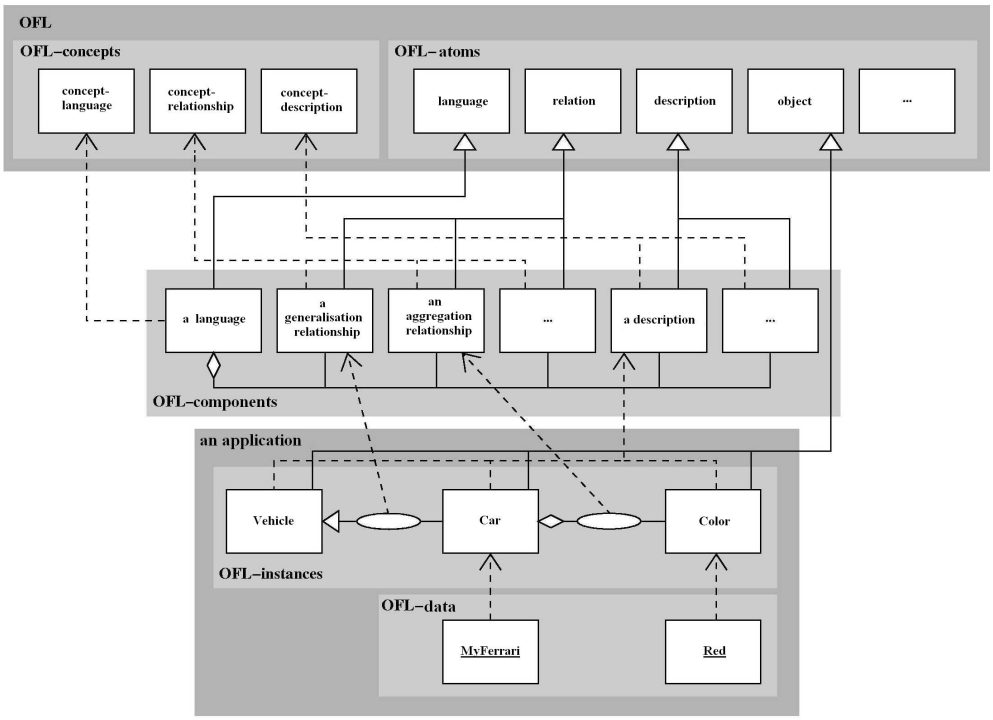

Fig. 1. The OFL Architecture

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 correspond to two kinds of relationships: OFL-*ImportRelationship* and OFL-*UseRelationship*. Virtual model contains also the TaggedValues that are required, the *Constraints*, and the *Common Model Elements.*

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

4.1. Representation of OFL-ImportRelationships

An *OFL-importRelationship* is a generalization of the inheritance mechanism found in object-oriented languages. The meta-programmer has the responsibility to create an OFL relationship component for each import relationships that exists in the modeled language. The Profiles will contain all the elements which enable to represents these components.

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

A set of tagged values will be associated to all relationships which are stereotyped as a specialization of <<*OFLImportRelationship*>>. The values of these elements correspond to some of the OFL-*AtomRelationship* characteristics; they 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 OFL-*ImportRelationhip*

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 rules will translate all characteristics of relationships components into the corresponding tagged values:

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

will be 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))
```

In addition, 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. 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-*UseRelationship* is a generalization of the aggregation mechanism found in object-oriented languages. The meta-programmer has to create an OFL relationship component for each kind of use relationships which is defined in the modeled language. The abstract stereotype <<*OFLUseRelationship*>> is the base for each concrete stereotype which represent an OFL-*UseRelationhip* component within the considered language. As for import relationships, a set of tagged values will be associated to all use relationships which are considered as a specialization of <<*OFLUseRelationship*>>. They correspond to some of the OFL-*AtomRelationship* characteristics: *hiddenFeatures*, *renamedFeatures*, *removedFeatures* and *shownFeatures*.

Some constraints dealing with parameters of OFL-*ConceptRelationship,* which are generated for import relationships, are valid also for use relationships. In this context, the *OFLUseRelationship* stereotype will replace the *OFLImportRelationship* one as ancestor of the ComponentRelationship stereotype. Considering the parameter ConceptRelationship:: cardinality, the constraint will be the following after being transformed:

Rule context: cardinality ≠ 1

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

The parameters that remain significant in the context of a use relationship are: *cardinality*, *repetition*, *circularity*, *masking*, *renaming*, *removing* and *showing*. Constraints will address all these values in the context of the target language.

Figure 2 presents an example of the use of a generated OFL-ML profile to represent a simple Java application.

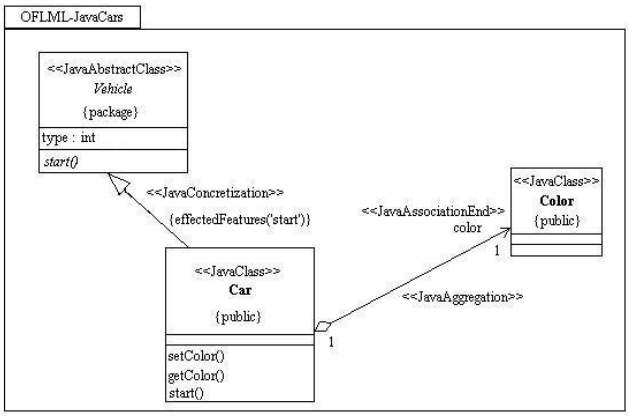

Fig. 2. OFL-ML Profile representation of a Java Application

5. CONCLUSIONS AND FUTURE WORK.

This paper presents an approach for describing UML Profiles for object oriented programming languages modeled by OFL. Because of length constraint, it focused only on the part dealing with class relationships. It extends original UML elements with features that allow a better representation of these relationships. The main issue is to fill the gap between programming language expressiveness and modeling language semantics. Future works include a better modeling of class entities and the integration of these elements into several profiles designed for commercial languages like C++, 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

[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 IC, University of York, UK*

[9] D. Pescaru and P. Lahire (2003) - "*Modifiers in OFL: An Approach for Access Control Customization*", *OOIS'03, WEAR workshop, Geneva, Swizerland*

[10] K. Kaitanen (1999) – "*J-UML Specification*", http://www.vtt.fi, VTT Organization, Finland