Modeling hypergeneric relationships between types in *XML*

Adeline Capouillez, Robert Chignoli, Pierre Crescenzo, and Philippe Lahire[1]

1 Introduction

The definition of hypergenericity given by P. Desfray in [Des94] is the following: "hypergenericity is a mechanism whereby a model is automatically transformed by applying external rules". In [CCL99] we try to introduce hypergenericity in the notions of types or in the notions of links between types in order to propose a framework for the definition of languages which may be different according to well-chosen sets of parameters. The goals look like those of meta-programmed systems but they are not. With our approach we cannot do everything (only the behaviors of links and types may change) but this is done in a very guided and safe way: in the *OFL* model, each concept is parametrized with a set of parameters (a few of them are mentioned in section 3) and a set of actions and controls whose execution is driven by the value of those parameters. There is no meta-programming task but there are choices of parameter values. Of course, the drawback of such an approach is that if the set of parameters is not sufficient or is too specialized, the facilities provided by such a language or a model customization are useless, but they have been designed in *OFL* in order to avoid it.

One related benefit of hypergenericity is the ability to define libraries of concept instances possibly distributed over the Internet [CCL99]. Sharing these instances means that we must provide a widely used and portable data representation associated to a self-explanatory documentation [ABS00]. The aim of this paper is to study whether *XML* [Mic99] may provide such facilities.

In section 2 we first present an overview of the *OFL* model. In section 3 we propose an approach to model the *OFL* entities using *XML*. Finally we give the benefits and limitations of such an *XML* implementation and we propose possibilities of future works.

2 An overview of the *OFL* model

In order to reach the objectives described in the introduction, we propose a model [CCL99, CCCL00b, CCCL00a] which describes the main concepts and semantics of object-oriented languages based on classes. This proposal relies on three basic foundations: relationships (such as aggregation[2] and inheritance), descriptions (such as Java classes and Java interfaces, C++ classes, ...) and languages[3]. Our ambition is two-fold: on one hand, to be general enough to be able to describe the concepts of description and the relationships of industrial object-oriented languages based on classes (*Java* [CMPS97, AG98], *C++* [Str97], *Eiffel* [Mey92], ...); on the other hand, to allow people to experiment new concepts (example: one *relationship* which handles one or several versions of *descriptions*) and to test their validity in order to make programming languages evolve. The three basic entities of the model are described in this section.

2.1 Relationships

The first fundamental concept of the model is the relationship-concept. We use the following terminology: a relationship-concept is an abstraction of a kind of relationship in the object-oriented languages (an example of the relationship-concept: the *Java*-like inheritance relationship `implements`; an example of relationship: the `implements` relationship between a *Java* interface and a *Java* class). For example the model may be used to define the most common relationships of object-oriented

[1]For all Authors: I3S Laboratory (UNSA/CNRS), OCL project, Sophia Antipolis, France. E-Mails: {Adeline.Capouillez | Robert.Chignoli | Pierre.Crescenzo | Philippe.Lahire}@unice.fr

[2]We name *aggregation* the client-supplier relationships which are defined most of the time through an attribute (with referencing). This relationship is also called *client relationship* in the *Eiffel* terminology and *composition by reference* in *UML* [Lai97, BJR98, RJB98, JBR99].

[3]The model describes also the attributes, methods, messages, basic types, statements, control structures ... but this goes beyond the scope of this paper.

languages such as specialization, generalization, aggregation or composition[4]. To be more precise, each relationship-concept could be described by one class in an implementation of the model, and the instances of this class could be relationships. It is the same for the description-concept and the language-concept which will be presented later. We have a special interest in the relationship between types, in particular in the inheritance relationship which we would like to express better in relation to its use. For each relationship we defined the notions of *source-description* (the one which describes the relationship) and of *target-description*. For example, according to inheritance, the source-description is the heir and the target-description is the ancestor; for aggregation, the source-description is the one which declares an attribute (or a method parameter or ...) and the target-description, the one which describes the type of this attribute. Aggregation and composition are "use" relationships. Inheritance and code reuse are considered as "import" relationships.

Also, the model describes relationships between descriptions and objects but our study is limited to the *instantiation* (with its reverse: the *extension*). On the other hand, the relationships between objects do not belong to the scope of our research, though they could be described.

2.2 Descriptions

The modeling of the notion of description is more traditional [KDRB91]. Our objective is to make several description-concepts coexist with different semantics. That means to handle some kind of interoperability (let us imagine a language which provides description-concepts of both *Eiffel* and *C++*) as well as to take into account languages such as *Java* (in which we describe the notions of interface and class as two description-concepts).

Each description-concept defines a set of relationship-concepts it is compatible with, and handles their interactions. It is at this level that are handled possible problems related to the composition of relationships coming from different relationship-concepts. The best example of the problem concerning the composition of relationships is precisely the difficulties standing from the multiple and repeated inheritance uses. For example: *Eiffel* would be described as one description-concept and three relationship-concepts corresponding to client relationships, expansion and (multiple) inheritance.

Finally, let us note the description-concepts which integrate the notion of genericity. Of course, the notion of type is present. Each description describes several or only one type, depending on whether it is generic or not.

2.3 Languages

The language-concept is an important and simple notion. It describes, as is meant implicitly by its name, a modelized language. Each language includes two components:

1. a set of description-concepts
2. a set of relationship-concepts, each of them has to be compatible with at least one of the selected description-concepts.

The set of relationship-concepts can either be deduced from the set of description-concepts, or be defined explicitly; this possibly allows to reduce, at this level, the compatibility rules described within each description-concept. For example, in a language, one may wish to use the *Eiffel* description-concept but without the ability to use the expansion relationship. Of course, only a restriction of the description specifications is allowed, but no extension.

In the end, language modeling will allow to implement and control the interoperability between the objects created by different language-concepts.

[4] The *composition* relationship should be understood as the *composition by value* in *UML* and *expansion* in *Eiffel*. It describes a specific *use* within which the source-description declares that its instances *contain* an instance of the target-description (but not *reference* as we say for aggregation).

3 Elements of the *OFL* model implementation using *XML*

3.1 Basic principles

In *XML* there are entities such as documents and links (*XLink*) [Mic99]; we look at them in order to decide whether it is possible or not to make them correspond to the description-concept and the relationship-concept. One study of *XLink* shows that proposed links (simple and extended links) with their parameters cannot match the *OFL* concepts. To say that one link is “simple” or “extended” means that the link addresses one or several documents. Moreover the parameters (in the sense of *XML*) provide the name, the address(es) of the target(s), their role and some other information mainly related to the loading or to the look of the documents.

We will use *XML* only as a support to the definition of the *OFL* model which was described in the previous section. *XML* which is to be widely used in the near future may implement storage and data exchanges with also a set of capabilities to handle document appearance. All these facilities are mandatory if we aim to manage libraries of reusable concepts reachable through a network.

As we are exclusively interested in data structuring, we mainly use the Data Type Documents (DTD) of *XML*, in order to modelize the *OFL* concepts. In the future we may use stylesheets for documentation purpose and generate a well-presented and straightforward view of the information contained in the model.

In the framework of the handling of large libraries of *OFL*-concepts it could be an interesting issue to use the *RDF* in order to introduce facilities to browse through the libraries of *OFL*-concepts according to accurate criteria. Moreover, if we address a library of *OFL*-concepts mobile and distributed over the network, it can be interesting to use simple or extended links that are implemented through *XLink*; we choose to store each *OFL*-concept described through a DTD in a separate file.

The mechanism associated to namespaces is too limited to be really useful for such a situation, so that the name given to each element of a DTD is unique in the set of DTDs which describe the *OFL* model.

4 The main aspects of the *OFL*-concepts modeling

Hereafter we shortly present how to structure and organize the modeling of the three *OFL* concepts: description, relationship and language. Meanwhile we will give some parameter samples.

4.1 Modeling the description-concept

These are the contents of file “./ofl-description.dtd” which corresponds to the definition of description-concept parameters:

```
<?xml version=''1.0'',encoding='ISO-8859-1' ?>
<!DOCTYPE description-concept [
  <!- Parameters of the OFL concept called DESCRIPTION-CONCEPT ->
  < !ELEMENT description-concept (dp-general, dp-specific) >
  < !ATTLIST description-concept
    name-of-description CDATA #REQUIRED
  >
    <!- Parameters common for all type of uses ->
    < !ELEMENT dp-general EMPTY>
    < !ATTLIST dp-general
      instance-creation (allowed | denied) #REQUIRED
      instance-sharing (allowed | denied) #REQUIRED
      ...
    >
    <!- Parameters specific to one kind of service ->
    < !ELEMENT dp-specific EMPTY>
    < !ATTLIST dp-specific
      ...
    >
]
```

The *instance-creation* parameter intends to control whether one description-concept allows or denies the creation of instances. The *instance-sharing* parameter deals with the ability to give or not an identity to an object: this will allow to control whether an object may be part of another object or may be simply referenced by several other objects.

4.2 Modeling the relationship-concept

These are the contents of file "./ofl-relationship.dtd" which corresponds to the definition of relationship-concept parameters:

```
<?xml version=''1.0'',encoding='ISO-8859-1' ?>
<!DOCTYPE relationship-concept [
 <!- Parameters of the OFL concept called RELATIONSHIP-CONCEPT ->
 <!- Hypergeneric Parameters only for USE relationships ->
 <!- (by default not included) ->
 <!ENTITY %is-a-use-relationship 'IGNORE' >
 <!- Hypergeneric Parameters only for IMPORT relationships ->
 <!- (by default not included) ->
 <!ENTITY % is-an-import-relationship 'IGNORE' >
 <![ %is-a-use-relationship; [
 < !ELEMENT relationship-concept (relationship-parameters, urp-general, urp-specific) >
 < !ATTLIST relationship-concept
   name-of-relationship CDATA #REQUIRED
 >
   <!- Parameters common for all type of uses ->
   <!ELEMENT urp-general EMPTY>
   <!ATTLIST urp-general
     is-dependent (true | false) #REQUIRED
     is-shared (true | false) CDATA #REQUIRED
     ...
   >
   <!- Parameters specific to one kind of service ->
    <!ELEMENT urp-specific EMPTY>
   <!ATTLIST urp-specific
     transaction-policy (program | method | statement | manual) #REQUIRED
     object-locking (program | method | statement | manual) #REQUIRED
     ...
   >
 ] ]
 <![ %is-an-import-relationship; [
 < !ELEMENT relationship-concept (relationship-parameters, irp-general, irp-specific) >
   <!- Parameters common for all type of uses ->
   <!ELEMENT irp-general EMPTY>
   <!ATTLIST irp-general
     variance (co-vaviant | contra-variant | in-variant | no-variant) #REQUIRED
     polymorphism (up | down | both | none) #REQUIRED
     ...
   >
   <!- Parameters specific to one kind of service ->
    <!ELEMENT irp-specific EMPTY>
   <!ATTLIST irp-specific
     ...
   >
 ] ]
 <!- Hypergeneric Parameters common to all types of relationships ->
 <!- (IMPORT, USE, OBJECT-TO-CLASS) ->
 < !ELEMENT relationship-parameters (rp-general, rp-specific) >
   <!- Parameters common for all type of uses ->
   < !ELEMENT rp-general EMPTY>
   < !ATTLIST rp-general
     cardinality CDATA #REQUIRED
     repetition CDATA #REQUIRED
     ...
   >
   <!- Parameters specific to one kind of service ->
   < !ELEMENT rp-specific EMPTY>
   < !ATTLIST rp-specific
     ...
   >
]
```

In an "Import" relationship, it is possible to give the maximum number of imported descriptions through one relationship concept (*cardinality*): for instance '1' for a simple-inheritance link, 'infinite' for a multiple-inheritance link or any other positive value if the number of imported descriptions is limited. Similarly it is also possible to define the number of times one description may be imported through one relationship concept (*repetition*).

In a "Use" relationship we mention parameters such as *is-shared* which allows or not the sharing of an object between all the instances of the description (notion of *class variable*) or such as *is-dependent* which means something like "composition *versus* aggregation". We also mention parameters which are specific to services (persistence and concurrency are, among others, services that may be implemented). For example, it is possible to define the policy to begin/finish a transaction (*transaction-policy*) or to lock/unlock an object (*lock-policy*).

4.3 Modeling the language concept

These are the contents of file "./ofl-language.dtd" which corresponds to the definition of both parameters and constraints of the language-concept:

```
<?xml version="1.0",encoding='ISO-8859-1' ?>
<!DOCTYPE language-concept [
  <!- Parameters of the OFL concept called LANGUAGE-CONCEPT ->
  <!- Description-concept (Declaration of entity) ->
  <!ENTITY % OFL-description SYSTEM "./ofl-description.dtd" >
  %OFL-description
  <!- Relationship-concept (Declaration of entity) ->
  <!ENTITY % OFL-relationship SYSTEM "./ofl-relationship.dtd" >
  %OFL-relationship
  < !ELEMENT language-concept (language-parameters, descriptions, relationships, valid-combinations) >
  < !ATTLIST language-concept
    name-of-language CDATA
  >
    <!- Hypergeneric parameters ->
    < !ELEMENT language-parameters (olp-general, olp-specific) >
      <!- Parameters common for all type of uses ->
      < !ELEMENT olp-general EMPTY>
      < !ATTLIST olp-general
        services-list NMTOKEN #REQUIRED
        ...
      >
      <!- Parameters specific to one kind of service ->
      < !ELEMENT olp-specific EMPTY>
      < !ATTLIST olp-specific
        ...
      >
    <!- RELATIONSHIP and DESCRIPTION concepts associated to LANGUAGE ->
    <!- List of the language description-concept ->
    <!ELEMENT descriptions (description-concept+) >
    <!ATTLIST descriptions
      nb-elements CDATA #REQUIRED
      type (chronological | ordered) "chronological"
    >
    <!- List of the language relationship-concept ->
    <!ELEMENT relationships (relationship-concept*) >
    <!ATTLIST relationships
      nb-elements CDATA #REQUIRED
      type (chronological | ordered) "chronological"
    >
    <!- Valid combination of RELATIONSHIP/DESCRIPTION concepts in LANGUAGE ->
    <!- List of the language description-concept ->
    <!ELEMENT valid-combinations (name-of-source-description-concept, name-of-relationship-concept,
                                  name-of-target-description-concept, is-valid) >
      <!ELEMENT name-of-source-description-concept EMPTY>
      <!ATTLIST name-of-source-description-concept string-value (CDATA) #REQUIRED >
      <!ELEMENT name-of-relationship-concept EMPTY>
      <!ATTLIST name-of-relationship-concept string-value (CDATA) #REQUIRED >
      <!ELEMENT name-of-target-description-concept EMPTY>
      <!ATTLIST name-of-target-description-concept string-value (CDATA) #REQUIRED >
      <!ELEMENT is-valid EMPTY>
      <!ATTLIST is-valid boolean-value (true|false) 'true' >
]
```

In a language it is possible to specify the services that may be activated (for example: persistence or concurrency). This is useful in order to reuse the description and relationship concepts which implement a service, in a language which does not support this service.

4.4 Using *OFL* concepts

To define a new language, i.e an instance of the language-concept, it is necessary to define an *XML* document and to include the DTD associated to the language-concept in it, to create an instance of the language-concept and as many instances of the description-concept and the relationship-concept as needed in the given language, and then to set the values of their parameters.

For instance file "./language-sample.xml" would contain the instances mentioned above and look like:

```
<?xml version="1.0",encoding='ISO-8859-1' ?>
<!DOCTYPE language-concept SYSTEM './ofl-language.dtd' >
< language-concept >
  ...
< /language-concept >
```

5 Conclusion and prospects

In this paper, we described an approach in order to model the organization and the structuring of the parameters described in the *OFL* concepts. The description of meta-information for object systems, such as an open virtual machine or meta-compilers, is not the main issue of languages as *XML*. Yet they provide quite a reasonable way of describing the whole set of generic parameters proposed by our model.

The main benefits obtained through the use of *XML* are the facilities that it provides for the sharing of information over the network. But we need more expressiveness especially to define the constraints between instances of the *OFL* concepts within a language.

An interesting issue in the future could be the definition of a *XML* dialect using the same approach as *RDF* or *XLink*, and providing an easy way to define the constraints between the DTDs.

References

[ABS00] S. Abitboul, P. Buneman, and D. Suciu. *Data on the Web, From relational to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.

[AG98] K. Arnold and J. Gosling. *The Java Programming Language*. The Java Serie... from the Source. Sun Microsystems, 2 edition, 1998.

[BJR98] G Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language User Guide*. The Object Technology Series. Addison-Wesley Publishing Co., October 1998.

[CCCL00a] A. Capouillez, R. Chignoli, P. Crescenzo, and P. Lahire. Gestion des objets persistants grâce aux liens entre classes (à paraître). In *Conférence* Objets, Composants, Modèles *2000*, mai 2000.

[CCCL00b] A. Capouillez, R. Chignoli, P. Crescenzo, and P. Lahire. How to Improve Persistent-Object Management using Link Information? Technical Report 2000-01, Laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis, January 2000.

[CCL99] R. Chignoli, P. Crescenzo, and P. Lahire. Customization of Links between Classes. Technical Report 99-18, Laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis, November 1999.

[CMPS97] G. Clavel, N. Mirouze, E. Pichon, and M. Soukal. *Java : la synthèse*. Informatique. InterEditions, juillet 1997.

[Des94] P. Desfray. *Object Engineering, the Fourth Dimension*. Addison-Wesley Publishing Co., 1994.

[JBR99] I. Jacobson, G. Booch, and J. Rumbaugh. *Unified Software Development Process*. The Object Technology Series. Addison-Wesley Publishing Co., January 1999.

[KDRB91] G. Kiczales, J. Des Rivières, and D. G. Bobrow. *The Art of the MetaObject Protocol*. MIT-Press, 1991.

[Lai97] M. Lai. *UML : la notation de modélisation objet : applications en* Java. InterEditions (Masson), 1997.

[Mey92] B. Meyer. *Eiffel: The Language*. Object-Oriented Series. Prentice Hall, 1992.

[Mic99] A. Michard. *XML Language and Applications*. Eyrolles, 1999.

[RJB98] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. The Object Technology Series. Addison-Wesley Publishing Co., December 1998.

[Str97] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Publishing Co., 3 edition, 1997.