*Journal of Universal Computer Science, vol. 9, no. 8 (2003), 956-969*
*submitted: 24/2/03, accepted: 30/5/03, appeared: 28/8/03* © *J.UCS*

# XOCL – an XML Language for Specifying Logical Constraints in Object Oriented Models

**Franklin Ramalho**
(Universidade Federal de Campina Grande, Brazil
Universidade Federal de Pernambuco, Brazil
franklin@dsc.ufcg.edu.br, fsr@cin.ufpe.br)

**Jacques Robin**
(Universidade Federal de Pernambuco, Brazil
jr@cin.ufpe.br)

**Roberto Barros**
(Universidade Federal de Pernambuco, Brazil
roberto@cin.ufpe.br)

**Abstract:** In this paper, we present XOCL, an XML-based language to represent OCL (Object Constraint Language) constraints in UML models. XOCL was designed in two steps from the UML meta-model and OCL EBNF grammar published by OMG: (1) construction of a simple OCL meta-model and (2) derivation of an XML Schema for this meta-model. XOCL applications include full interoperability among UML modelling tools as well as finely grained structured input for automatic behavioral code generation and model checking.

**Keywords:** OCL, UML, XML, Meta-Modelling, XMI, XMLShema.
**Categories:** D.1.5, D.1.6, D.2.12, D.3.3

## 1 Introduction

UML (Unified Modeling Language) [Booch et al. 1998] has emerged as the main standard language for pre-code software engineering artifacts for several reasons:

- It is able to model all aspects of a system: structural, behavioral, distributed, etc.
- It provides a simple and concise graphical notation;
- It provides mechanisms that allow its extension for several problem domains;
- It is supported by a wide variety of CASE tools.

With its use progressively outgrowing its original high-level object-oriented modeling purpose, new requirements emerged for UML. One of them is the ability to specify integrity constraints and application domain rules within models in a detailed, structured, unambiguous and semi-formal way. To address this requirement, OMG incorporated from version 1.4 of the UML standard the semi-formal textual model annotation language OCL (Object Constraint Language) [Warmer and Kleppe 1999].

Specifying constraints and rules as OCL expressions instead of natural language notes brings numerous benefits:

1. It avoids ambiguity and therefore misunderstandings between developers;
2. It allows for much more thorough automatic correctness and consistency model checking [Brucker and Wolff 2001];
3. It greatly facilitates automated behavioral code-generation from models [Mellor and Balcer 2002];
4. It widens the applicability of UML beyond software engineering to new fields in which detailed, rigorous constraint or rule specification is crucial. These fields include data and metadata integration in data warehouses [Poole et al. 2001], disparate federated information systems [Purvis et al. 2000], semantic web ontologies [Cranefield and Purvis 1999], knowledge engineering [Schreiber 1999] [Devedzic 2001] and multi-agents systems engineering [Bergenti and Poggi 2000].

Several UML modeling tools such as Poseidon [Poseidon 2003] and Rose [Rose 2003] already support OCL constraints. However, each of these tools relies on an internal UML model storage format that: (1) is proprietary, and (2) mixes model content with its graphical layout. This virtually prevents reuse and integration of models developed with different tools. However, such reuse is a key aspect of the most modern trends in software engineering, information systems, databases and artificial intelligence.

To address this problem, OMG put forward XMI [OMG 2003], an XML based standard for representing UML models in textual format. XMI defines one XML element or attribute for each element of the UML graphical notation. Therefore, XMI encodes UML model content and structure separately from their visual presentation, much like XML does with web pages. Being based on XML also makes the XMI codification of an UML model both legible by humans and an appropriate input to model exchange, model checking and code generation software.

Although XMI has elements and attributes for all UML diagrams, its current version does not cover OCL constraints. This gap has prevented UML, OCL and XMI to reach their full potential as an integrated standard set for a variety of emergent applications. These applications include reuse of pre-code development artifacts in modern software engineering processes based on component markets, product lines, design patterns, frameworks and software architectures [Atkinson et al. 2002], automated model checking and fully automated code generation.

In this paper we propose XOCL, an XML language for detailed, fully structured encoding of OCL constraints and rules that completes the OMG and W3C standard puzzle to support such advanced applications. An XMI document with embedded XOCL expressions captures the complete structure and semantics of a UML model detailed with OCL constraints in a format that is adequate for both interchange and sophisticated processing.

Consider for example an automated behavioral code generator. An OCL constraint might serve as the basis for generating exception-raising code to call upon violation of the constraint. With current UML editors, the OCL constraint will appear in the XMI document exported by the editor and input to the code generator as a long, unstructured string inside a single *<constraint>* tag. Before tackling its generation task, the code generator will first need to parse this string into semantically relevant syntactic constituents and hence incorporate an OCL syntax grammar. But this defeats the very purpose of XML mediation among components, namely to avoid encapsulating knowledge about upstream component languages into downstream ones to achieve a more modular and extensible task distribution.

## 2 Technological Background

### 2.1 OCL

OCL permits the specification of three kinds of constraints: invariants, which are static, and pre and post-conditions, which are dynamic. An invariant is associated to a class, an interface or a type. It specifies a condition that must be true for all instances of the associated class, interface or type, at any time. In contrast, a pre or post-condition is associated to an UML operation and specifies a condition that needs to be verified only immediately before or immediately after its execution (respectively).

Figure 1 shows an excerpt from the class diagram of a simple company UML model. The diagram includes two OCL constraints. The first one, *managerConstraint* is an invariant that requires all instances of the *Company* class that constitutes its context to have its *manager* association filled with an instance of the *Person* class with the *age* integer attribute value within the [18,65] interval and the *isUnemployed* boolean attribute set to *false*. The second one, *resultOkConstraint* is a post-condition that requires the method *income* of the *Person* class that constitutes its context to always return a value greater than *5000*.

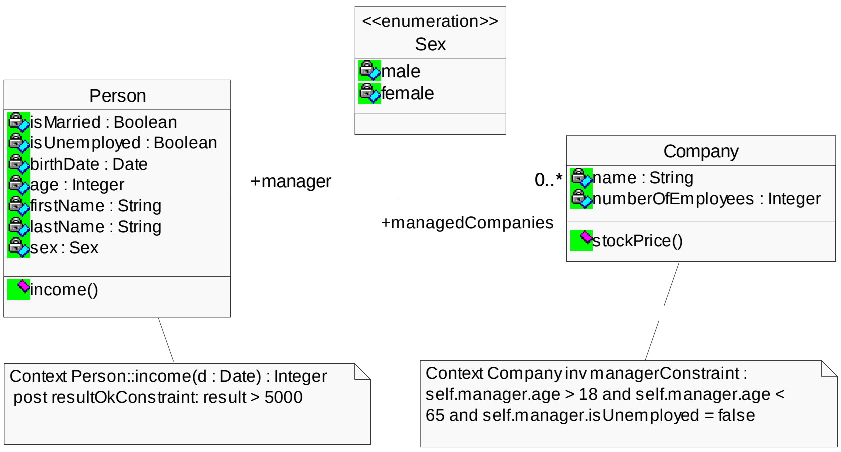


*Figure 1: Example of an UML model with OCL constraints*

These two example constraints illustrate the design rationale underlying OCL: to achieve the most practical balance between formal precision and intuitiveness for the widest possible user base.

### 2.2 XML and XMLS

XML (eXtensible Markup Language) [W3C 2003] was initially created for publishing documents on the Web. Its main goals were: (1) divide semantic structure of document content (in XML) from its visual rendering by the browser (in HTML); and (2) codify the semantic structure in a format adequate for being used as both documentation (human legible) and data (input prepared for automated processing). These two features combined with the fact that XML is an open and general purpose standard, have progressively turned XML the choice for interoperability among databases, programming languages and intelligent agents in heterogeneous distributed applications. In these multiple roles, XML has blurred the distinctions between documents, concepts, data, information and knowledge and between areas such as information systems, distributed system, databases and artificial intelligence.

This new role for XML as a universal platform for interoperability made necessary to address two new requirements: (1) schema specification (or metadata) for XML data and (2) data and metadata distribution among multiple web sites. The W3C put forward XML Schema (XMLS) to address the first of these requirements and XML Name Spaces (XMLNS) to address the second one.

The *<xmlns>* element of an XML document generally indicates the URL where to find the schema that specifies the structure pattern to which the document must conform. This use of namespaces and schemas allows establishing, in a non-centralized way, a common understanding about the validity and the meaning of the XML data interchanged between components of an open system.

XMLS allows defining:

- The elements that can appear in a XML document;
- The attributes that can appear in each of these elements;
- How these elements can be nested;
- How these elements must be ordered at a specific nesting level;
- The data types of the elements and attributes, as well as its default or fixed values.

Before the XMLS definition, an XML document schema could be specified only using a DTD (Document Type Definition). Although handy for the initial, restricted, document representation purpose of XML, the DTD formalism revealed too limited for adequately specifying database schemas and the intentional part of knowledge bases. XMLS improves on DTD for such purposes thanks to the following features:

- Support for a richer set of built-in data-types than DTDs, including byte, date, integer, and SQL and Java primitive data-types;
- Use of an XML-based syntax, thus preventing the user to have to learn yet another syntax (that of DTD) and allowing reuse of all tools and languages available to create and manipulate XML documents to work with XMLS documents;
- Extensibility through XMLS constructs that allows reusing of parts of schemas in other schemas;
- Object-oriented user-defined data types with inheritance of element, attribute, and data-type definitions;
- Integration with XML namespaces.

### 2.3 Models, meta-models, XML and UML

Three trends recently emerged in computational language design. The first one is the use of meta-circularity to specify language parts or extensions using the language itself. The relation between XML and XMLS exemplifies this trend: XML provides a syntactic standard for XMLS, whereas XMLS provides a structural standard for any documents, data or knowledge about a specific application domain codified in XML syntax. The second trend is the use of UML class diagrams together with OCL constraints to specify object-oriented models for representing basic types and language constructors. Such diagram is frequently called a language meta-model [Carlson 2001]. The third trend is the use of XMI for representing such meta-models in textual format with XML syntax. This approach permits using XLST [Kay 2000] to declaratively implement software that process artifacts written in the projected language (in our case, XOCL). Declarative programming within the XML paradigm allows faster prototyping and maintenance for a wide range of services such as automated code-generation, language translation, compilation and compression [OMG 2003].

Cooperation between such services in an open, component-based, heterogeneous system requires the definition of a common XMI meta-model schema. For a language meta-model is essentially a conceptual ontology of what the language can express, the use of a DTD to specify a meta-model schema is as problematic as its use to specify an artificial intelligence application knowledge base. XMI meta-model schemas are thus preferentially defined in XMLS.

## 3 XOCL Design Methodology

The three trends mentioned in the previous section can be combined in a two step language design methodology: (1) define an object-oriented meta-model of the language using a visual CASE tool that supports UML and OCL; and (2) codify in XMLS the schema for the XMI textual format of this meta-model. This is the approach we followed to design XOCL. We first defined a meta-model of OCL using UML and OCL itself. From this meta-model, we then specified an XML Schema for XOCL expressions to be embedded in XMI encoding of domain application UML models.

### 3.1 An OCL Meta-Model

Our OCL meta-model consists UML class diagrams with OCL constraints that represent the OCL constructs and their associated UML constructs. Although being much simpler than the current draft meta-model under construction by OMG [OMG 2003], our meta-model covers all OCL constructs. We developed it from three sources: (1) the OMG UML 1.4 meta-model [OMG 2003] to guarantee its integration with the latest UML standard; (2) the OCL EBNF (Extended Bacchus-Naur Form) grammar [Reisner 1984], and (3) the OMG natural language OCL specification.

Figure 2 shows a small excerpt of our proposed OCL meta-model. Our OCL construct classes are shown in gray whereas the reused UML OMG meta-model classes are shown in white. An OCL constraint (*Constraint* class) consists of one or more expressions (*Expression* class) and is associated to a context (*Context* class).

This context either is structural (*StructuralContext* class) for invariant constraints or OCL variable definitions (respectively *Invariant* and *Definition* subclasses of the *StructuralConstraint* class), or behavioral (*BehavioralContext* class) for pre and post-condition constraints (respectively *PreCondition* and *PostCondition* subclasses of the *BehavioralConstraint* class)[1]. In addition, a structural constraint applies to either a *Feature* or an *AssociationEnd*, two classes reused from the UML meta-model, while a behavioral constraint only applies to a *BehavioralFeature* another UML meta-model class that subsumes UML methods and operations.

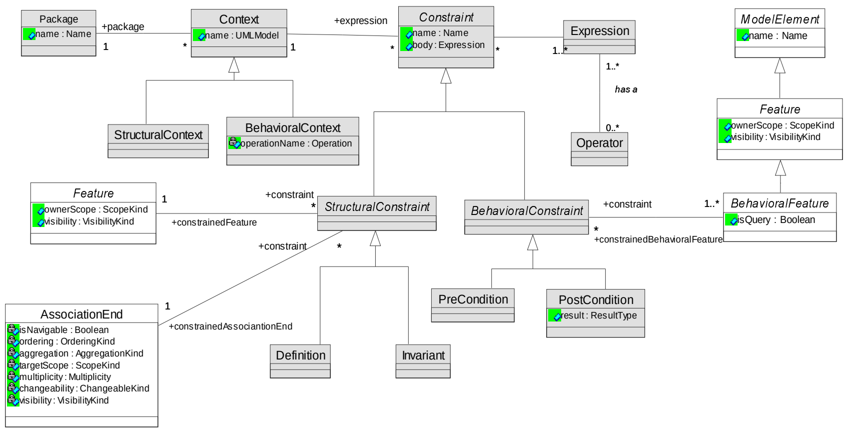


*Figure 2 : Part of the OCL meta-model*

### 3.2 XOCL Schema

With this schema, applications can interchange XOCL messages with the guarantee that: (1) the receptor will have the same understanding about the data sent; (2) these data will not present any syntactical error; and (3) any piece of OCL code included at the XML document will not be incomplete.

Figure 3 shows an excerpt from the XOCL schema written in XMLS. This schema specifies that an XOCL constraint be identified through an <OCLConstraint> root element (line 1). According to the meta-model showed in section 3.1, a constraint is classified as StructuralConstraint or as BehavioralConstraint. A StructuralConstraint can be further classified as a Definition or an Invariant, whereas a BehavioralConstraint can be classified as a PreCondition or as a PostCondition. An <Invariant> element, specified between the lines 13 and 24 in Figure 3, has two mandatory child elements: <Context> (line 17) and <Body> (line 19) in order to

---

[1] This cross association between constraint subtypes and their respective associated context subtype is actually represented in the complete OCL meta-model as an OCL constraint that we omitted from Figure 2 to avoid cluttering it.

validate only XOCL constraints with a specified context and a specified OCL code body.

```
<xs:element name="OCLConstraint" type="xocl:ConstraintType"/>
<xs:complexType name="ConstraintType" abstract="true"/>
<xs:complexType name="StructuralConstraintType" abstract="true">
   <xs:complexContent>
      <xs:extension base="xocl:ConstraintType"/>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="BehavioralConstraintType" abstract="true">
   <xs:complexContent>
      <xs:extension base="xocl:ConstraintType"/>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="InvariantConstraintType">
   <xs:complexContent>
      <xs:extension base="xocl:StructuralConstraintType">
         <xs:sequence>
            <xs:element name="Context"
                        type="xocl:StructuralContextType"/>
            <xs:element name="Body" type="xocl:ExpressionType"/>
         </xs:sequence>
         <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="PreConditionConstraintType">
   <xs:complexContent>
      <xs:extension base="xocl:BehavioralConstraintType">
         <xs:sequence>
            <xs:element name="Context"
                        type="xocl:BehavioralContextType"/>
            <xs:element name="Body"
                        type="xocl:ExpressionType"/>
         </xs:sequence>
         <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

*Figure 3: XOCL elements definition in XMLS*

Each OCL construct of our complete OCL meta-model has a corresponding XML construct, either an element or an attribute, in XOCL. In addition to define such constructs, the XOCL XMLS also specifies valid nesting among these constructs. There are many different possible ways to encode such constructs and nesting constraints in XMLS. The most straightforward codifications turned out to be very verbose and thus hard to understand by human users. There are several XMLS design issues involved making XOCL a concise and clear language. They are discussed in the next section.

## 4 Issues in Mapping Object-Oriented Models to XML Schema

The design of XOCL raised many issues concerning the relative conciseness, genericity and extensibility of alternative codifications and the resulting compromise between human legibility and machine processing simplicity. For lack of space, we focus our discussion here on only two key issues: (1) usage of inheritance and (2) usage of XMLS grouping.

### 4.1 Usage of Inheritance

In XMLS, inheritance is provided by the <*extension*> constructor, which allows a type to extend another already declared. This is illustrated in Figure 3: <*OCLConstraint*> is declared as a *ConstraintType* (line 1). This type is complex, in the sense that it may have at least one child element or attribute, and abstract, in the sense that it cannot be instantiated as an XML element. Two others abstract types that extend *ConstraintType* are *StructuralConstraintType* (line 3) and *BehavioralConstraintType* (line 8). While *InvariantConstraintType* (line 13) extends *StructuralConstraintType*, it is not abstract and can type any <*OCLConstraint*> element. *PreConditionConstraintType* (line 25) similarly extends *BehavioralConstraitnType*. Given this type hierarchy, declaring in a XML document that an OCL constraint is an invariant, only involves adding the single element: <*OCLConstraint xsi:type="InvariantConstraintType"*> [2]. Using inheritance, one of the features that make XMLS more expressive than DTD, allows an XOCL specification to be concise without leaving out details. This carries over to the code of processing applications taking as input XMI documents with embedded XOCL. For example, an XSLT application can have different rules for processing invariants and for processing pre-conditions. Such differentiated processing would not be straightforward were all constraints identified only through the <*Constraint*> element.

As mentioned in section 3.1, while the context of an invariant is only an UML classifier (e.g., a class), the context of a pre or post-condition consists of a pair <UML classifier, UML operation> (e.g., a class and one of its methods). In our OCL meta-model, this is accounted for by a constraint context type hierarchy that reflects the constraint type hierarchy. How XMLS type inheritance allowed us to encode this directly in XOCL is shown in Figure 4. The StructuralContextType has just two child elements: (1) <Package> (line 6) that identifies the package where the UML artefact is; and (2) <UMLClassifier> (line 8) that identifies the UML artifact whose constraint is applied. On the other hand, the BehavioralContextType has three additional child elements: (1) <OperationName> (line 22) that identifies the method name where the pre or post-condition is applied; (2) <FormalParameterList> (line 23) that contains a list of all parameters of the method identified by <OperationName> tag; and (3) <ReturnType> (line 25) that declares the return type of the method.

---

[2] The xsi:type attribute is a XML keyword for typing the current element, in the case, <*OCLConstraint*>.

```
<xs:complexType name="ContextType" abstract="true"/>
<xs:complexType name="StructuralContextType">
   <xs:complexContent>
      <xs:extension base="xocl:ContextType">
         <xs:sequence>
            <xs:element name="Package" type="xocl:PackageType"
                        minOccurs="0"/>
            <xs:element name="UMLClassifier"
                        type="xocl:UMLModelType"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="BehavioralContextType" abstract="true">
   <xs:complexContent>
      <xs:extension base="xocl:ContextType">
         <xs:sequence>
            <xs:element name="Package" type="xocl:PackageType"
                        minOccurs="0"/>
            <xs:element name="UMLClassifier"
                        type="xocl:UMLModelType"/>
            <xs:element name="OperationName" type="xs:string"/>
            <xs:element name="FormalParameterList"
                        type="xocl:FormalParameterListType"/>
            <xs:element name="ReturnType" type="xocl:Type"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

*Figure 4: Type inheritance in XOCL schema*

### 4.2 Usage of Element Groups

In our XOCL schema, all operators are defined through the <xs:group> XMLS constructor. This allows defining element groups to be reused in various complex type definitions, so as to avoid repetitions of the same element definitions within several other different complex type definitions. An example of the combined usage of element grouping and type inheritance in XOCL is given in Figure 5. An expression type hierarchy first distinguishes between atomic and compound expressions. For example, 'A > B' is a compound expression of subtype comparative. Its components are the comparative operator '>' and its operand A and B, both atomic expressions. Lines 8-23 in Figure 5 define the group of comparative operators (ComparativeOperators group). This group can then be referenced anywhere as a substitute for the repetition of the all the comparative operators definition. Lines 3-5 contain an example of such reference. Figure 6 shows the XOCL <GreaterThan> element encoding the 'A > B' expression following the Schema fragment of Figure 5.

```
<xs:complexType name="ComparativeExpressionType">
   <xs:complexContent>
      <xs:extension base="xocl:CompoundExpressionType">
         <xs:group ref="xocl:ComparativeOperators"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:group name="ComparativeOperators">
   <xs:choice>
      <xs:element name="Equal"
             type="xocl:ComparativeOperatorType"/>
      <xs:element name="GreaterThan"
            type="xocl:ComparativeOperatorType"/>
      <xs:element name="LessThan"
            type="xocl:ComparativeOperatorType"/>
      <xs:element name="GreaterOrEqualThan"
            type="xocl:ComparativeOperatorType"/>
      <xs:element name="LessOrEqualThan"
            type="xocl:ComparativeOperatorType"/>
      <xs:element name="NotEqual"
            type="xocl:ComparativeOperatorType"/>
   </xs:choice>
</xs:group>
<xs:complexType name="ComparativeOperatorType">
   <xs:sequence minOccurs="2" maxOccurs="2">
      <xs:choice>
         <xs:sequence>
            <xs:element name="Exp"
                  type="xocl:ArithmeticExpressionType"/>
         </xs:sequence>
         <xs:sequence>
            <xs:element name="Exp"
                  type="xocl:AtomicExpressionType"/>
         </xs:sequence>
      </xs:choice>
   </xs:sequence>
</xs:complexType>
```

*Figure 5: ComparativeOperator group and ComparativeOperatorType composed type specification*

```
<Exp xsi:type="ComparativeExpressionType">
      <GreaterThan>
         <Exp xsi:type="StringLiteralExpressionType">
            <Literal value="A"/>
         </Exp>
         <Exp xsi:type="StringLiteralExpressionType">
            <Literal value="B"/>
         </Exp>
      </GreaterThan>
</Exp>
```

*Figure 6: Use of <GreaterThan> element*

## 5 Example of OCL constraint codified in XOCL

Having discussed some aspects of the XOCL schema, let us now look at an example XML element that follows this schema. Such element is given in Figure 7. It codifies in XOCL the first example OCL constraint of section 2.1 reproduced below:

**Context** Company **inv** managerConstraint:

self.manager.age > 18 **and** Self.manager.age < 65 **and** self.manager.isUnemployed = false

In Figure 7, the *<OCLConstraint>* root element, at line 1, just identifies the element as specifying an OCL constraint. Lines 3 to 5 declare the context associated with the constraint. Line 4 specifies that this context is an UML class identified through an xsi:type attribute of the <UMLClassifier> element. At line 4, the class associated to the constraint is identified via a name attribute, in this example with value “Company”.

Next comes the <Body> element that contains the OCL expressions specified in lines 6-51. Line 6 indicates that the constraint body expression is of type NaryLogicalExpressionType. In this example, the expression is a Boolean conjunction encoded by an <And> element in line 7. This element has three operands, each one being a comparative expression (indicated by an <Exp> element). They are respectively encoded in lines 8-20, 21-34 and 35-49.

The first comparative expression specifies the “self.manager.age > 18” constraint consisting of the ‘>’ operator and its two operands: ‘self.manager.age’ and ‘18’. This constraint is encoded as a <GreaterThan> element with one child element per operand. The first one, in lines 9-15, is an <Exp> element, with the xsi:type attribute filled with “UMLModelPropertyCallExpressionType” because it references a role (manager) and an attribute (age) of a class (Company). <PropCall> elements appear as child elements which nesting reflects the construct nesting of the OCL expression. Therefore, we have one <PropCall> element (line 10) for the ‘self’ OCL keyword, another (line 11) for the ‘manager’ role and another (line 12) for the ‘age’ attribute. The second operand, in lines 16-18, is an expression element of type StringLiteralExpressionType with one child <Literal> element that contains the operand value specified through the value attribute.

The two other comparative expressions of the constraint, “self.manager.age < 65” and “self.manager.isUnemployed = false" that appear next as child elements of the <And> element are encoded following the same conventions.

## 6 Conclusions

In this paper we presented XOCL, an XML language to codify OCL constraints. XOCL elements can be embedded within XMI documents to provide the same detailed XML structuring for constraints as for the other parts of a UML model. To design XOCL, we started by constructing a complete, but simple OCL meta-model using UML and OCL itself. From this meta-model we then specified an XOCL document schema using XMLS. Our main concern when faced with alternative design

```
<OCLConstraint xsi:type="InvariantConstraintType"
               name="managerConstraint">
 <Context>
   <UMLClassifier xsi:type="ClassType" name="Company"/>
 </Context>
 <Body xsi:type="NaryLogicalExpressionType">
   <And>
      <Exp xsi:type="ComparativeExpressionType">
         <GreaterThan>
            <Exp xsi:type="UMLModelPropertyCallExpressionType">
               <PropCall name="self" type="Object">
                  <PropCall name="manager" type="Association">
                     <PropCall name="age" type="Attribute"/>
                  </PropCall>
               </PropCall>
            </Exp>
            <Exp xsi:type="ArithmeticalLiteralExpressionType">
               <Literal value="18"/>
            </Exp>
         </GreaterThan>
      </Exp>
      <Exp xsi:type="ComparativeExpressionType">
         <LessThan>
            <Exp xsi:type="UMLModelPropertyCallExpressionType">
               <PropCall name="self" type="Object">
                  <PropCall name="manager" type="Association">
                     <PropCall name="age" type="Attribute"/>
                  </PropCall>
               </PropCall>
            </Exp>
            <Exp xsi:type="ArithmeticalLiteralExpressionType">
               <Literal value="65"/>
            </Exp>
         </LessThan>
      </Exp>
      <Exp xsi:type="ComparativeExpressionType">
         <Equal>
            <Exp xsi:type="UMLModelPropertyCallExpressionType">
               <PropCall name="self" type="Object">
                  <PropCall name="manager" type="Attribute">
                     <PropCall name="isUnemployed"
                                            type="Attribute"/>
                  </PropCall>
               </PropCall>
            </Exp>
            <Exp xsi:type="BooleanLiteralExpressionType">
               <Literal value="false"/>
            </Exp>
         </Equal>
      </Exp>
   </And>
 </Body>
</OCLConstraint>
```

*Figure 7 – Example of code in XOCL language*

options for this schema has been to make XOCL elements as concise as possible to facilitate their understanding by both human readers and software applications. Concision also save communication bandwidth for model interchange over the network.

The methodology we outlined in this paper for the design of XOCL is general. It can be used to design XML codification of any language. This is an interesting contribution given the ubiquitous need for such codifications triggered by the growing success of XML mediated multi-language development process and platform such as the MDA architecture [OMG 2003] and .net.

The XOCL language presented in this paper provides a key missing piece of the OMG and W3C standard puzzle, that includes XML, XMLNS, XMLS, XSLT, UML, OCL, XMI and CWM, to support advanced applications and development practices within such open, heterogeneous paradigms. We intend to submit it as a proposal to the appropriate OMG working group. It is available for inspection and feedback on the web at www.cin.ufpe.br/~fsr/xocl.

The purpose of XOCL is to fill a gap in XMI. To be useful in practice, it needs to be integrated XOCL with XMI. This not an immediate step because the current XMI specification provided by OMG is encoded as a DTD. For the same reasons discussed in sections 2.2 and 4 which motivated specifying XOCL as an XMLS, the expressive limitations of DTD turn this XMI specification under-constrained and the documents that conform to it extremely verbose. To address this problem, an OMG working group is currently drafting an XMLS for XMI.

XMLS suffers himself from some limitations. For example, it does not allow expressing validity constraints that involve elements located in different branches of an XML document viewed as a tree. To address this limitation and others, Schematron [Schematron 2002] and XCSL [XCSL 2001] have been proposed to complement XMLS to encode more sophisticated XML document schemas. Theses languages are based on arbitrary document fragment matching and transformation patterns using the W3C standards XPath [Kay 2000] and XSLT. They can be easily integrated with XMLS (*e.g.,* Schematron expression can even be embedded within an XMLS <*appinfo*> element). As future work, we intend to choose one of these constraint specification languages to refine and more concisely and generically express the most complex constraints of our XOCL Schema.

**Acknowledgements**

This research was supported by grants from CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) of the Brazilian Federal Government.

## References

[Atkinson et al. 2002] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B. Wust, J. and Zettel, J.: "Component-based product line engineering with UML"; Addison-Wesley, Reading/MA (2002).

[Bergenti and Poggi, 2000] Bergenti F. and Poggi A.: "Exploiting UML in the Design of Multi-Agent Systems"; Proc. ECOOP - Workshop on Engineering Societies in the Agents' World, Sophia Antipolis and Cannes (2000), 96-103.

[Booch et al. 1998] Booch, G., Jacobson, I. and Rumbaugh, J.: "The Unified Modeling Language User Guide"; Addison-Wesley, Reading/MA (1998).

[Brucker and Wolff 2001] Brucker, A. D. and Wolff, B.: "Checking OCL Constraints in Distributed Systems Using J2EE/EJB", Technical Report 157, Albert-Ludwigs-University at Freiburg, Germany (2001).

[Carlson 2001] Carlson, D.: "Modeling XML applications with UML: practical e-buisiness applications"; Addison-Wesley, Reading/MA (2001).

[Cranefield and Purvis 1999] Cranefield and Purvis, M.: "UML as an ontology modelling language"; Proc. Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence, Stockholm (1999).

[Devedzic 2001] Devedzic, V.: "Knowledge Modeling State of the Art"; Integrated Computer-Aided Engineering, 8, 3 (2001), 257-281.

[Poseidon 2003] Poseidon for UML, May 2003, http://www.gentleware.com

[Kay 2000] Kay, M.: "XSLT Programmer's Reference"; Wrox Press, (2000).

[Mellor and Balcer 2002] Mellor, S.J. and Balcer, M.J.: "Executable UML: a foundation for the model driven architecture"; Addison-Wesley, Reading/MA (2002).

[OMG 2003] OMG - Object Management Group; Link: http://www.omg.org

[Poole et al. 2001] Poole, J., Chang, D. Tolbert, D. and Mellor, D.: "Common Warehouse Meta-model: an introduction to the standard for data warehouse integration"; Wiley & Sons (2001).

[Purvis et al. 2000] Purvis, M.K., Cranefield, S., Bush, G., Carter, D. McKinlay, B., Nowostawski, M. and Ward, R.: "The NZDIS Project: An Agent-based Distributed Information Systems Architecture"; Proc. Hawaii International Conference on Systems Sciences, Big Island (2000).

[Reisner 1984] Reisner P.: "Formal grammar as a tool for analyzing ease of use"; Proc. Human Factors in Computing Systems, (Thomas J.C. & Schneider M.L.; eds.), New York (1984), 53-78.

[Rose 2003] Rational Rose, May 2003, http://www.rational.com/uml

[Schematron 2002] Schematron Assertion Language, Version 1.5, October 2002, http://www.ascc.net/xml/resource/schematron/Schematron2000.html

[Schreiber 1999] Schreiber, G.: "Knowledge engineering and management: the CommonKADS methodology"; MIT Press, Cambridge/MA (1999).

[Warmer and Kleppe 1999] Warmer J. and Kleppe A.; " The Object Constraint Language: Precise Modeling with UML", Object Technology Series, Addison-Wesley, Reading/MA (1999).

[W3C 2003] W3C – World Wide Web Consortium; Link: http://www.w3.org/

[XCSL 2001] XML Constraint Specification Language, Version 2.2, May 2001, http://www.di.uminho.pt/~jcr/PROJS/xcsl-www/