Journal of Theoretical and Applied Information Technology
31st May 2016. Vol.87. No.3
© 2005 - 2016 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TRANSFORMING XML SCHEMA CONSTRAINING FACETS AND XML QUERIES TO OBJECT CONSTRAINT LANGUAGE (OCL)

[1]TOUFIK FOUAD, [2]BAHAJ MOHAMED
[1] PhD, LITEN Laboratory, University Hassan I, FSTS Settat, Morocco
[2] Professor, LITEN Laboratory, University Hassan I, FSTS Settat, Morocco
E-mail: [1]toufik.fouad@gmail.com, [2]mohamedbahaj@gmail.com

ABSTRACT

Unified Modeling Language UML became the main part of software development including web applications that use XML for exchanging structured data. That's why there is a need for modeling XML elements with UML.

Design Recovery or Reverse Engineering allows us to get conceptual schema which helps developers to understand systems and to ease its maintenance.

A lot of XML Schema mapping methods focus only on getting the structural part (elements, complex types and attributes …) without giving importance to constraints and restrictions and also XML Queries. In this sense our goal is to represent the mapping and the transformations rules from XML elements and queries to UML/OCL.

Keywords: *UML, OCL, XML Schema, XQuery, Transformation.*

1. INTRODUCTION

UML(Unified Modeling Language) has become the standard language for designing and representing object oriented system, it provide a graphical notation and elements to construct an application model which describe the different components of the system, there structure and behavior. Furthermore UML contains various diagrams to model the static aspect of the system (use case, class diagrams,) and also the dynamic aspect (sequence, state diagrams,)

At the same time with the evolution of the use of web applications, XML (eXtensible Markup Language) play an importance role in exchanging structured data and transporting information over the web. XML is a text-based format that provides a mechanism for describing document structures using markup. XML has Document Type Definition (DTD) which defines the document structure using a set of rules for describing elements and other markup components. But DTD has some limitation when we describe high structured data, in DTD there are no constraints imposed on the kind of character data allowed, so data typing is not possible, there is no support for namespaces, that's why the World Wide Web Consortium (W3C) propose other solutions. The best alternative is XML Schema; XML Schemas are themselves XML document. They include most basic programming type such as Integer, String and floating point number, and also provide an object oriented approach to define the format of an XML documents. XML Schema present the static part of XML so the UML diagram which has the same properties, and static in the same time, is the Class diagram, here we can map and transform XML elements and attributes, to classes and properties with the same characteristics (data type, relationship, generalization, …), in conceptual level, it is necessary to generate conceptual models to illustrate data structure and relationships in XML Schema [1,2]. Here much work has been done to represent different approach and methods for mapping between XML Schema and UML. [3,4] present a formalization of transformation rules from XML Schema to UML Class Diagram, the authors in [5] propose a method of reverse engineering from XML to generate Software Requirement Specification (SRS) in a document manner. In [6] the author extends UML and adds some stereotypes to describe SOX Schema used by commerceOne elements. In [7] Bird, Goodchild and Halpin have proposed an approach that uses a conceptual language of role object model to generate XML Schema. In [8] the authors presented a solution for mapping UML object-oriented model to XML Schema, and then to relational database table schema, [9] provide an evolution framework by which the xml schema and documents are incrementally updated according to the changes in the conceptual model (expressed as a UML class models), in [10] the authors present a set of rules using XSLT styleSheets that transform the UML class diagram into an adequate document type definition (DTD), in [11] the authors present correspondence rules for generating DTDs from UML diagrams. From the other side the authors in [12] developed an algorithm for constructing UML class diagram from XML Schema, [13] provides a comparison of several approaches which automatically create a platform specific model for XML Schemas, based on a comprehensive set of transformation patterns supporting creation of UML model that is as concise and semantically expressive without losing XML Schema information, the authors in [14] present method transformation from textual XML Schema to graphical UML to facilitate understanding of XML Schema.

Since XML document are considered as source of information or a database, it is necessary to get data from this source. XQuery is to XML what SQL is to database. XQuery is W3C standard language for retrieving data from XML document.

All the previous work focuses only on the structural part without transform restriction component or what we call in XML Schema the Facets. The same thing for transforming and mapping XML queries, most research dismiss this part.

In this paper we focus on the restriction and facets component, to transform this component we use OCL Object Constraint Language, after that we will transform constraining generalization/specialization(partition and mutual exclusion constrains) and finally we will catch the part of XML queries transformation, using OCL collections.

2. MAPPING BETWEEN CONSTRAINING FACETS AND OCL

Many works have been done to represent XML Schema in UML. In detail, researchers convert different elements and attributes to classes, properties and relationships (Generalization, Composition, Aggregation …), but the representation of constraining facets has not been done yet, UML diagram does not provide all relevant aspects of a specification, that's why it is necessary to describe additional constraints about the object in the model. Constraint specify invariant conditions that must hold for the system being modeled, constraint are often described in natural language and this always result in ambiguities, then we need to use a formal language to express constraints, easy to read and write, this language is OCL (Object Constraint Language) [15].

OCL contains 12 constraining facets; we present here some constraint with the correspondent OCL clause.

2.1 Length

We use this restriction to limit the length of a value in an element. The example above defines an element called password with a restriction, the value must be exactly eight characters.

```
<xs:element name=”password”>
  <xs:simpleType>
    <xs:restriction base=”xs:string”>
         <xs:length value=”8” />
     <xs:restriction/>
   <xs:simpleType>
</xs:element>
```

We suppose that the account element is the parent node of password element, our context here is the account class, the OCL invariant correspondent is:

```
        context Account inv:
                self.password.size()=8
```

2.2 minLength and maxLength

We use this restriction to define the minimum and the maximum length. The example above defines an element called password with a restriction, the value must be minimum five characters and maximum eight characters.

```
<xs:element name=”password”>
  <xs:simpleType>
    <xs:restriction base=”xs:string”>
         <xs:minLength value=”5”/>
         <xs:maxLength value=”8”/>
     <xs:restriction/>
   <xs:simpleType>
</xs:element>
```

The Account class is the context, here we use the operator AND for type Boolean to combine the two conditions, and the OCL invariant is as follow:

```
context Account inv: self.password.size()>=5
                     and self.password.size()<=8
```

2.3 Pattern

We use this restriction to limit the content of an XML element to define a series of numbers or letters that can be used. The example above defines an element called name with a restriction. The acceptable value is zero or more occurrences of lowercase letters from (a to z).

```
<xs:element name=”name”>
  <xs:simpleType>
    <xs:restriction base=”xs:string”>
       <xs:pattern value=”([a-z])*” />
    <xs:restriction/>
  <xs:simpleType>
</xs:element>
```

We suppose that the Person element is the parent node of name element, we suggest also adding a method “pattern (…)” to OCL String type, the returned value of this method is a Boolean type, equals TRUE if the attribute respect the pattern parameter. The correspondent OCL clause is:

```
context Person inv:
     self.name.pattern(“[a-z]*”) ==TRUE
```

2.4 Enumeration

We use this restriction to limit the content of an XML element to a set of acceptable values. The example below defines an element called "country" with a restriction. The only acceptable values are: Morocco, Algeria, and Tunisia.

```
<xs:element name=”name”>
   <xs:simpleType>
     <xs:restriction base=”xs:string”>
          <xs:enumeration value=”Morocco” />
          <xs:enumeration value=”Algeria” />
          <xs:enumeration value=”Tunisia” />
     <xs:restriction/>
    <xs:simpleType>
</xs:element>
```

We suppose that the parent node of country element is address element, the correspondent OCL invariant is:

```
context Address inv:
   self.country = Country::Morocco Or
   self.country = Country::Algeria Or
   self.country = Country::Tunisia
```

2.5 minInclusive and maxInclusive

We use this restriction to control the value of an element; the follow example defines an element called age. The value of age cannot be lower than 0 or greater than 120.

```
<xs:element name=”password”>
   <xs:simpleType>
      <xs:restriction base=”xs:string”>
           <xs:minInclusive value=”18”/>
           <xs:maxInclusive value=”120”/>
       <xs:restriction/>
   <xs:simpleType>
</xs:element>
```

We suppose that the parent node of age element is person element, then the context is the class Person, the correspondent OCL invariant is as follow:

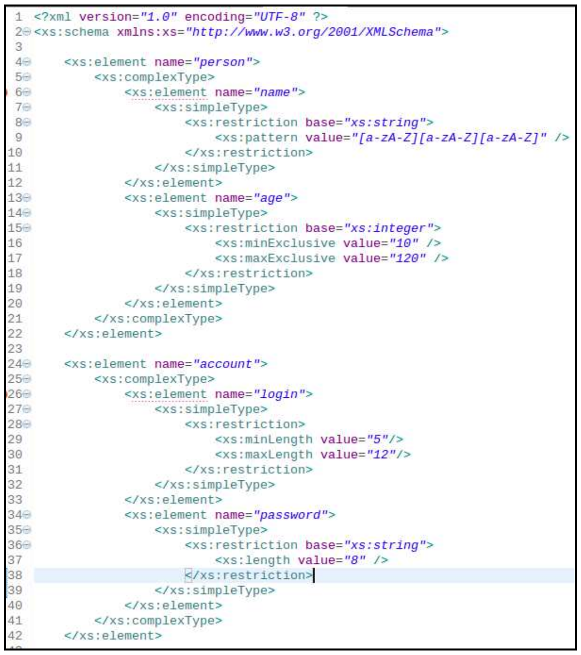
```
context Person inv:
     self.age >= 18 and self.age <= 120
```

3. IMPLEMENTATION AND VALIDATION

To demonstrate the validity of our approach, a tool has been developed (figure 2). This tool take as input an XML Schema file, then we normalize the xml file to have good structure of different nodes and elements. After that we transform all constraining facets to OCL clause.

To develop our prototype, we used java as a programming language, and to parse the XML file we used DOM XML Parser. DOM parser parses the entire XML document and loads it into memory; then models it in a TREE structure for easy traversal and manipulation.

After loading the XML file, we search and store all constraining facets (restrictions).the second step is getting the attribute and the object context, finally we transform the constraining facet (restriction) to OCL clause. In (Figure 1) we present the tested XML Schema file.

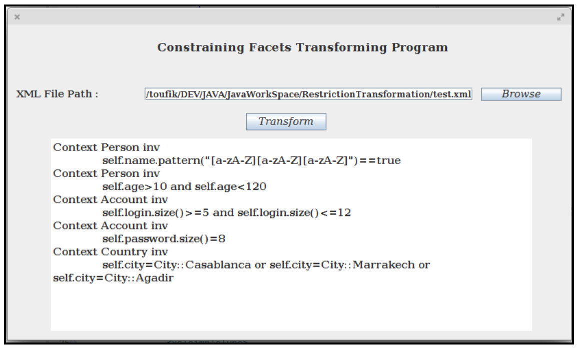


```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="person">
        <xs:complexType>
            <xs:element name="name">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="age">
                <xs:simpleType>
                    <xs:restriction base="xs:integer">
                        <xs:minExclusive value="10" />
                        <xs:maxExclusive value="120" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:complexType>
    </xs:element>

    <xs:element name="account">
        <xs:complexType>
            <xs:element name="login">
                <xs:simpleType>
                    <xs:restriction>
                        <xs:minLength value="5"/>
                        <xs:maxLength value="12"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="password">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:length value="8" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:complexType>
    </xs:element>
```

Figure 1: XML Schema File, Restrictions Example



/toufik/DEV/JAVA/JavaWorkSpace/RestrictionTransformation/test.xml

```
Context Person inv
        self.name.pattern("[a-zA-Z][a-zA-Z][a-zA-Z]")==true
Context Person inv
        self.age>10 and self.age<120
Context Account inv
        self.login.size()>=5 and self.login.size()<=12
Context Account inv
        self.password.size()=8
Context Country inv
        self.city=City::Casablanca or self.city=City::Marrakech or
self.city=City::Agadir
```

Figure 2: Constraining Facets Transformation Program

4. XML SCHEMA GENERALIZATION/ SPECIALIZATION TRANSFORMATION

The author in [16] introduces different mechanism to present generalization specialization in XML Schema:

4.1 Derived types which contains tow xml element construct

- A simple type or a complex type can be derived from a base type by using Restriction. The transformation of Restrictions or constraining facets has been done in the previous section.
- The second element extends an existing simpleType or complexType element by using Extension. The example above shows us an extension of address element to USAddress element.

```
<xs :complexType name=”address”>
  <xs :sequence>
    <xs :element name=”street” type=”xs:string” />
    <xs :element name=”city” type=”xs:string” />
  </xs :sequence>
</xs :complexType>

<xs :complexType name=”USAddress”>
  <xs:complexContent>
   <xs:extension base=”address”>
     <xs:sequence>
      <xs :element name=”state” type=”xs:string” />
     </xs:sequence>
   </xs:extension>
  <xs:complexContent>
</xs :complexType>
```

4.2 Substitution groups & abstract elements and types

We focus on two cases of generalization/specialization constraints, we begin with the partition constraint which can be represented in the figure above using c- xml [17].

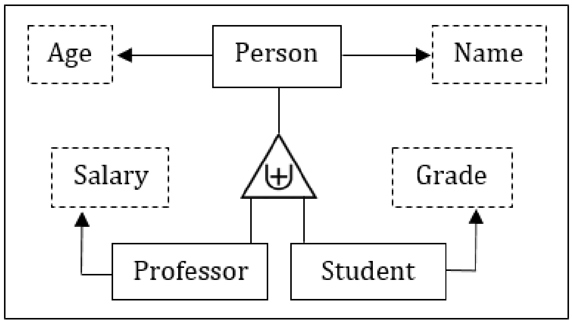
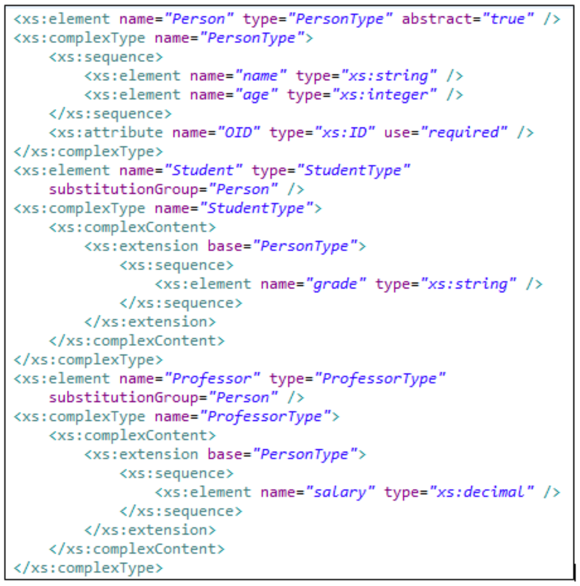


Figure 3: Generalization/Specialization Partition Constraint in C-XML

C-XML is a conceptual model consisting of object sets, relationship sets, and constraints over these object and relationship sets. In the notation of c-xml, boxes represent object sets dashed if lexical and not dashed if nonlexical, in figure 1 the set of objects in professor and student is a subset of the set of objects in person, c-xml give us the possibility to add constraint to generalization by writing a symbol inside the triangle of generalization/specialization, in our case we represent the partition constraint by adding the symbol $(\uplus)$ in the triangle, we transform our c-xml example to xml schema(figure 4) by following the mechanism in [16].



```
<xs:element name="Person" type="PersonType" abstract="true" />
<xs:complexType name="PersonType">
    <xs:sequence>
        <xs:element name="name" type="xs:string" />
        <xs:element name="age" type="xs:integer" />
    </xs:sequence>
    <xs:attribute name="OID" type="xs:ID" use="required" />
</xs:complexType>
<xs:element name="Student" type="StudentType"
    substitutionGroup="Person" />
<xs:complexType name="StudentType">
    <xs:complexContent>
        <xs:extension base="PersonType">
            <xs:sequence>
                <xs:element name="grade" type="xs:string" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:element name="Professor" type="ProfessorType"
    substitutionGroup="Person" />
<xs:complexType name="ProfessorType">
    <xs:complexContent>
        <xs:extension base="PersonType">
            <xs:sequence>
                <xs:element name="salary" type="xs:decimal" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Figure 4 XML Schema Translation of C-XML in Figure 3

In set terminology, we say that **Student** U **Professor** = **Person** and **Student** ∩ **Professor** = {}. Student and Professor are mutually exclusive, to ensure that the sets are disjoint, we use a unique identifier OID for each element, which is defined in the parent abstract element Person, the OID play the same role of primary key in RDB. To transform the uniqueness of OID and to ensure that the sets of Student and professor are disjoint, we present the equivalent OCL clause.

```
Context Student inv
  Student.allInstances()->forAll(s1,s2 | s1 <> s2
                              Implies
                               s1.oid <> s2.oid);
  Let professorColl : Set = Professor.allInstances();
Student
     .allInstances()->forAll(Student s |
                  profColl->forAll(Professor p | s<>p
                       implies
                  s.oid <> p.oid))
```

The second case is mutual-exclusion constraint, in c-xml we replace $(\uplus)$ symbol with + (figure 5), by following the mechanism of transformation from c-xml to xml schema in, we keep the same previous xml schema code and we change abstract value to false of Person element, we still have **Student** ∩ **Professor** = {}, but no longer **Student** U **Professor** = **Person**, we have **Student** U **Professor** ⊆ **Person**. Now we have the possibility to create instance of Person and add some invariants to present our OCL clause.

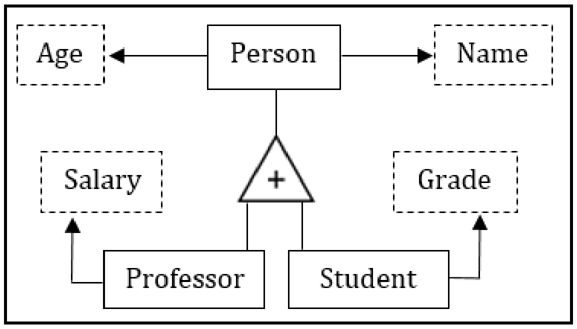


Figure 5: Generalization/Specialization Mutual Exclusion Constraint in C-XML

```
context Person inv
 Person.allInstances()->forAll(Person p1,Person p2
                        | p1 <> p2 implies
                         p1.oid <> p2.oid)
```

Person.allInstances() allow us to get all instances of Person, Student and Professor and check the uniqueness of OID quickly, unlike in partition constraint when we use nested collections (Student, Professor) to verify the differences between subsets.

In this part we have presented the transformation of two straightforward cases of generalization/specialization constraints (partition and mutual exclusion) using the notion of substitution group.

The transformation process is not entirely satisfactory for some cases, like unconstrained generalization/specialization and generalization/specialization with only a union constraint. This tow cases are more difficult to handle.

In the previous examples we presented simple generalization/specialization using the substitution groups. We can't transform multiple generalization/specialization because we have no way to specify in XML schema that an element is a member of two substitution groups.

5. MAPPING BETWEEN XML QUERIES AND OCL

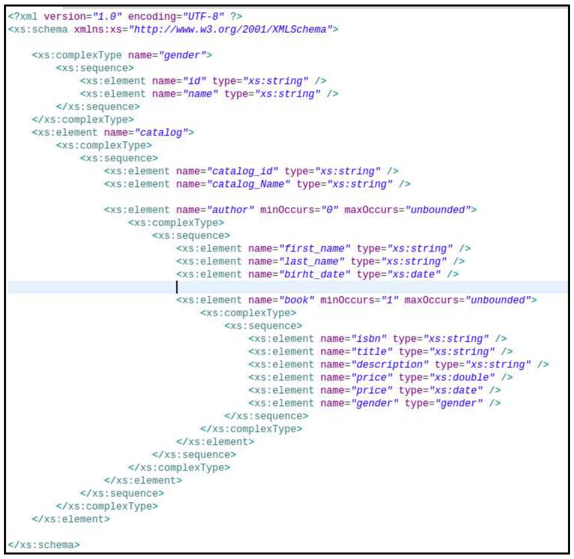
XQuery provide query mechanisms for data extraction from web based document, it is a query and programming language for processing XML documents and data, it is a language to select subsets and substructures from a large set of XML files.

In XQuery, path expression are used to locate nodes, such as element, attributes and text nodes, in XML data, the result of path expression is an ordered list of unique nodes. A path expression consists of a series of steps. Each step represents movement through a document in a particular direction, and each step can apply one or more predicates to eliminate nodes that fail to satisfy a given condition. The result of each step is a list of nodes that serves as a starting point for the next step [15].

Object Constraint Language (OCL) provides a mechanism of navigation so we can navigate an association on the class diagram to refer to other objects and their properties. To do so, we navigate the association by using the opposite association-end: object.roleName.

Both language, OCL and XQuery have the movement mechanism which help us to retrieve data, from this point we suggest to transform XML Queries to OCL clauses.

To understand our mapping between XML Queries and OCL, we represent in Figure 6 an example of an XML Schema definition, and the correspondent UML Class Diagram in Figure 7.


```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:complexType name="gender">
        <xs:sequence>
            <xs:element name="id" type="xs:string" />
            <xs:element name="name" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
    <xs:element name="catalog">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="catalog_id" type="xs:string" />
                <xs:element name="catalog_Name" type="xs:string" />

                <xs:element name="author" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="first_name" type="xs:string" />
                            <xs:element name="last_name" type="xs:string" />
                            <xs:element name="birht_date" type="xs:date" />

                            <xs:element name="book" minOccurs="1" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="isbn" type="xs:string" />
                                        <xs:element name="title" type="xs:string" />
                                        <xs:element name="description" type="xs:string" />
                                        <xs:element name="price" type="xs:double" />
                                        <xs:element name="price" type="xs:date" />
                                        <xs:element name="gender" type="gender" />
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```
Figure 6 XML Schema File Example

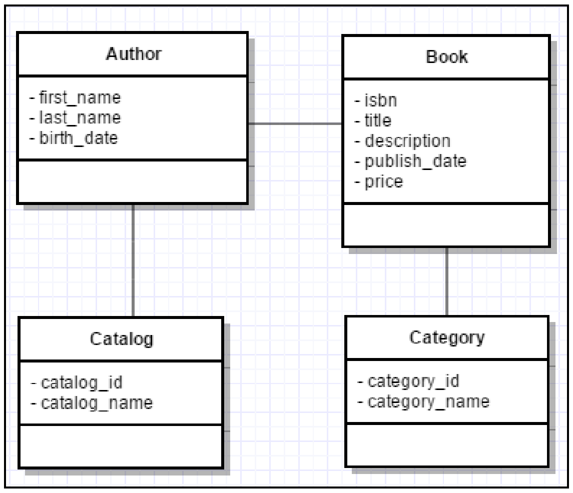


Figure 7 UML Class Diagram Example

5.1 Path Expression

Xquery Path expressions are used to locate nodes, such as element, attribute, and text nodes, in XML data. A path expression consist of a series of one or more steps, separated by slash "/" or double slash "//". Every step evaluates to a sequence of nodes. We take the example of listing all Books title of all Authors:

("books.xml")/catalog/author/book/title.

Using the mechanism of navigation between Objects in OCL, we can retrieve all Books title of all Authors. In our UML Class Diagram, we have One to Many association between Author class and Book class, and also One To Many association between Catalog and Author class, so the Catalog class contains collection of Authors and the class Author contains collection of books. The OCL expression is as follow:

Catalog.getAuthors().getBooks().getTitle().

5.2 Predicates

Predicates are used in a path expressions to filter the result by applying a specified test, it retain some items and discard others. XQuery uses predicates to limit the extracted data from XML document. The following predicate is used to select all the book elements that have a price element with a value less than 50:

("books.xml")/catalog/authors/book[price<50]

The equivalent OCL clause is:

```
  Context Book inv :
          Book.allInstances()->select(b Book |
                                  b.price<50)
```

The Book.allInstances() is the set of all books and is of type Set(Book). It is the set of all books that exist in the system at the time that the expression is evaluated.

We use select() method to specify a subset of a collection, the result is a list that contains all the elements from collection for which the condition on the price evaluates to true

5.3 Let and Sequence Expressions

The Let expression allow us to define a variable and use it more than one time. To use a list of items which may be very similar to each other or they may be of different type, XQuery provide Sequences. XQuery support operators to construct, filter, and combine sequences of items. Sequences are never nested.

The example above gets all books with a price less than 100 of the last author in our catalog.

```
Let $maxPrice :=100
Let $firstAuthor=
("books.xml")/catalog/author[fn:last()]
Let $books=$firstAuthor/book[price<$maxPrice]
```

The equivalent OCL clause is:

```
Context Catalog Inv:
  Let maxPrice : Integer =100
  Let firstAuthor : Author = self.getAuthors()
                                   ->first()
  Let books : Sequence : firstAuthor.getBooks()
                              ->select(b : Book | b.price <
                                  maxPrice).asSequence()
```

5.4 Arithmetic Expressions

Arithmetic expressions perform operations that involve addition, subtraction, multiplication, division and modulus (+,-,*, /), the result of an arithmetic expression is a numeric value, an empty sequence, or an error.

The example above shows a simple arithmetic operation in Xquery.

```
Let $somme=5+9
```

The equivalent OCL clause is:

```
Let somme: Integer=5+9
```

5.5 Comparison Expression

Comparison expressions are used to compare values; there are three kinds of comparison expressions: general, value and node.

5.5.1 General Comparison

General comparison are used for comparing atomic value or nodes that contain atomic value, and also can operate on sequences of more than one item, as well as empty sequences. The general comparison operators are (=, !=, <, <=, >, >=). The result of a general comparison that does not raise an error is always true or false.

The comparison expression above returns true if at least the price of one book is less than 150.

```
("books.xml")/catalog/authors/book/price < 50
```

The equivalent OCL clause is:

```
Context Book inv:
  Book.allInstances()->exists(b Book | b.price <50)
```

The exist() method check if there is at least one element in a collection for which a constraint holds.

5.5.2 Value Comparison

Value comparison differs fundamentally from general comparison in that they can only operate on single atomic value. They use eq (equal to), ne (not equal to), lt (less than), le (less than or equal to), gt (greater than), and ge (greater than or equal to). The example above return true if there is only a single element "author" which "last_name" value equal to "Hugo".

```
("books.xml")/catalog/authors/last_name
                                    eq"Hugo"
```

The equivalent OCL clause is:

```
Context Author inv:
  Author.allInstances()->exists(a Author|
        a.getLastName()->size()=1
        and a.lastName = "Hugo")
```

5.6 Logical Expression

A logical expression is either an and-expression or an or-expression. If a logical expression does not raise an error, its value is always one of the boolean values true or false. The example above gets all books of category "technology" or "Science Fiction".

```
(“books.xml”)/catalog/authors/book[category_name
            eq  “Technology”
                        or
              category_name eq “Science Fiction” ]
```

The equivalent OCL clause is:

```
Context Book inv:
   Book.allInstances()->select(b Book |
   b.getCategoryName().equals(“Technology”)
                         or
   b.getCategoryName().equals(“Science Fiction”))
```

5.7 FLWOR Expressions

XQuery provides a feature called a FLWOR expression that supports iteration and binding of variables to intermediate results. This kind of expression is often useful for computing joins between two or more documents and for restructuring data. FLWOR stands for “for, let, where, order by, return”.

The “for” and “let” clauses in a FLWOR expression generate an ordered sequence of tuples of bound variables, called the tuple stream. The optional “where” clause serves to filter the tuple stream, retaining some tuples and discarding others. The optional “order by” clause can be used to reorder the tuple stream. The “return” clause constructs the result of the FLWOR expression. The example above gets the title of all books which have price greater than 200:

```
    For $b in (“books.xml”)/catalog/authors/book
            Where $b/price>200
            Order by price
            return  $b/title
```

The equivalent OCL clause is:

```
Context Book inv
   Book.allInstance()->select(b Book | b.price>200)
                          ->collect(title)->asSequence()
```

When we want to specify a collection which is derived from some other collection, but which contains different objects from the original collection, we can use a “collect” operation.

5.8 Constructors

XQuery provide constructors that can create dynamically new XML node (elements, attributes, text …) within a query, and include theme in our result. The example above adds new element “additionalInfo”, which contains elements (“rating” and “priceCategory”).

```
For $b in (“books.xml”)/catalog/authors/book
Where $b/price>200
Return  <additionalInfo>
              <rating>1</rating>
      <priceCategory>Expensive</priceCategory>
            </additionalInfo>
```

The equivalent OCL clause is:

```
Context Book
 Def additionalInfo: Set(TupleType(rating: Integer,
                     priceCategory: String))=
          Book.allInstnaces()->select(b Book |
          b.price>200)->Tuple{
          b.rating=1, b.priceCategory=”Expensive”}
```

“Def” expression enables the definition and the reuse of variables/operations over multiple OCL expressions.

“TupleType” combines different types into a single aggregate type; the parts of a TupleType are described by its attributes, each having a name and a type.

5.9 Conditional Expressions

XQuery supports a conditional expression based on the keywords if, then, and else. The value of a conditional expression is defined as follows: If the effective boolean value of the test expression is true, the value of the then-expression is returned. If the effective boolean value of the test expression is false, the value of the else-expression is returned.

```
Let $technologyBook= count(
         (“books.xml”)/catalog/authors/book[
        category_name=’Technology’ ])
If $ technologyBook >0 then
                    Return $technologyBook
Else
Return “Technology Book List Empty”
```

The equivalent OCL clause is:

```
Let result : String
Let technologyBook: Integer=
                              Book.allIsntance()
                              ->count(“Technolgy”)
If technologyBook then
        Result = technologyBook
Else
        Result=“Technology Book List Empty”
```

```
End if
```

5.10 Quantified Expressions

A quantified expression determines whether some or all of the items in a sequence meet a particular condition. The value of a quantified expression is always true or false. A quantified expression begins with a quantifier, which is the keyword some or every, followed by one or more in-clauses that are used to bind variables, followed by the keyword satisfy and a test expression.

The first xquery expression return true if there is at least one book of the category "Math".

```
Some $book in ("books.xml")/catalog/authors/book
        Satisfy $book/category_name = "Math"
```

The equivalent OCL clause is:

```
Context Book inv
  Book.allInstances()->exists(b Book
                  | b.getCategory()
                  .getCategory_name= "Math")
```

The exists() operation in OCL allows specifying a Boolean expression that must hold for at least one object in a collection.

The second example return true if all book category is "Math"

```
Every $book in ("books.xml")/catalog/authors/book
        Satisfy $book/category_name = "Math"
```

The equivalent OCL clause is:

```
Context Book inv
  Book.allInstances()->forAll(b Book |
    b.getCategory().getCategory_name()="Math")
```

The forAll() operation in OCL allows specifying a Boolean expression, which must hold for all objects in a collection.

5.11 Instance of and Cast Expressions

To determine whether a sequence of one or more items matches a particular sequence type, we use an instance of expression. The instance of expression does not cast a value to the specified sequence type. It simply returns true or false, indicating whether the value matches that sequence type.

```
Let $book =("books.xml")/catalog/authors
                        /book[fn:last()]
$book instance of xs:Integer
```

The equivalent OCL clause is:

```
Context Book inv
    Let book : Book = Book.allIsntances()
                  ->asSequence()->last()
    Book.ocllsTypeOf(Integer)
```

Casting is the process of changing a value from one type to another. The cast expression can be used to cast a value to another type. XQuery provides a cast expression that creates a new value of a specific type based on an existing value. A cast expression takes two operands: an input expression and a target type.

```
Let $price = ("books.xml")/catalog/authors/
                        book[fn:last()]/price
$price cast as  xs:String
```

The equivalent OCL clause is:

```
Context Book inv
    Let price : Integer= Book.allIsntances()
                        ->asSequence()
                        ->last().getPrice()
                        .oclAsType(String)
```

6. CONCLUSION

In this paper we presented in the first step a set of rules of transformation, from XML Schema constraining facets to OCL (Object Constraint Language) expressions. To validate our approach, we provide a tool developed in java which take an XML Schema file as input, after extracting different restrictions using Java and DOM XML Parser, we transform theme to OCL expressions, after that we presented the transformation from constraining generalization(partition and mutual exclusion constraints to OCL clauses.

In the second step we catch XQuery expressions, the query language for XML documents, we have shown the equivalent OCL clauses for different XQuery Expressions. Our next goal will be focused on implementing a complete reverse engineering tool, which takes a XML Schema file and XQuery file as input to generate after that UML conceptual diagram enriched with OCL clauses.

REFRENCES:

[1] C. Haitao, "A Survey to Conceptual Modeling for XML", Proc, 2010 3rd International Conference on Computer Science and Information Technology, Volume 8, 2010, pp 473-477.

[2] M. Necasky, "Reverse Engineering of XML Schemas to Conceptual Diagrams", Proceedins 6th Asia Pacific Conference on Conceptual Modeling, Volume 96, 2009, pp 117-128.

[3] Aman, H., Ibrahim, R XML Schema Reverse Transformation: A Case Study, in D. Taniar, C. Torre, computational Science and Its Application – ICCSA 2015, Volume 9158 of Lecture Note in Computer Science , Springer 2015, pp 575-586.

[4] Aman, H., Ibrahim, R.: Formalization of transformation rules from XML schema to UML class diagram. International Journal Software Engineering and it Application 8(12), 75–90 (2014)

[5] Aman, H., Ibrahim, R.: Reverse engineering: from XML to Uml for generation of software requirement specification. In: 2013 8th International Conference on Information Technology in Asia - Smart Devices Trend: Technologising Future Lifestyle, Proceedings Of CITA 2013 (2013)

[6] Grady Booch, Magnus Christerson, Mathew Fuchs, Jari Koistinen; "UML for XML Schema Mapping Specification"; Rational Software Corp. and CommerceOne

Inc., December 1999.

[7] BIRD, L., GOODCHILD, A. and HALPIN, T. (2000): Object Role Modeling and XML Schema. Proc. International Conceptual Modeling Conference, Salt Lake City, USA, 309-322, Springer.

[8] I-Chen Wu, Shang-Hsien Hsieh; An UML-XML-RDB Model Mapping Solution for Facilitating Information Standardization and Sharing in Construction Industry Proceedings. National Institute of Standards and Technology, Gaithersburg, Maryland. September 23-25, 2002, pp. 317-321

[9] Evolution of XML schemas and documents from stereotyped UML class models: A traceable approach Information and Software Technology, Volume 53, Issue 1, January 2011, Pages 34-50

[10] Thomas Kudrass, Tobias Krumbein, Rule-Based Generation of XML DTDs from UML Class Diagrams in L. Kalinichenko,R. Manthey ,B. Thalheim,U. Wloka, Advances in Databases and Information Systems, Volume 2798 of Lecture Notes in Computer Science, Springer,2003, pp. 339-354

[11] E. Kuikka, A. Eerola, A Correspondence between UML Diagrams and SGML/XML DTDs in P. King, E. V. Munson, Digital Documents: Systems and Principles, volume 2023 of Lecture Notes in Computer Science; Springer, 2004, pp. 161-175

[12] Mikael R. Jensen, Thomas H. Moller, Torben Bach Pedersen. Converting XML Data to UML Diagrams for Conceptual Data Integration. DIWeb'2001. pp.17~31

[13] M. Bernauer, G. Kappel, G. Kramler, Representing XML Schema in UML - A Comparison of Approaches, in: N. Koch, P. Fra-ternali, M. Wirsing (Eds.), Web Engineering, Vol. 3140 of Lecture Notes in Computer Science, Springer, 2004, pp. 440-444.

[14] F.Salim, R. Price, M. Indrawan, S. Krishnaswamy, Graphical Representation of XML Schema, in Xuemin Lin, Hongjun Lu, Yanchun Zhang, Advanced Web Technologies and Applications, volume 3007 of Lecture Notes in Computer Science, Springer, 2004, pp. 234-245

[15] Object Constraint Language Specification v 2.4 OCL.2.4

URL:http://www.omg.org/spec/OCL/2.4/PDF/

[16] R. Al-Kamha, D. W. Embley, and S. W. Liddle. Representing Generalization/Specialization in XML Schema. In EMISA, 2005

[17] David W. Embley, Stephen W. Liddle, and Reema Al-Kamha. Enterprise Modeling with Conceptual XML. In Proceedings of the 23rd International Conference on Conceptual Modeling (ER2004), pages 150–165, Shanghai, China, November 2004.