Towards Complete Mapping between UML/OCL and XML/XQuery

Sherif Sakr[1] and Ahmed Gaafar[2]

[1] Department of computers and information science, Konstanz univercity
Sherif.Sakr@inf.uni-konstanz.de
[2] Faculty of Computers and Information, Cairo Univercity
Ahmed.Gaafar@cu.edu.eg

Abstract. The Unified Modeling Language (UML) is extended to model web applications. Meanwhile, Web technology becomes heavily relies on XML documents. The structure of XML documents, namely the XML Schema or Document Type Definition (DTD) for these documents can be modeled using UML data structures. UML tools are usually concerned with the generation of the structure and behavior of the system that is captured by models in their equivalents in the selected platform. In this paper we introduce a novel approach for the integration between UML and XML families of technologies. We model the structure of XML using UML class diagrams and based on this, we study how queries for XML documents, namely XQuery expressions can be described using UML techniques. We show that modeling of XML documents and its queries represented by XQuery expressions is possible using the querying capabilities of UML Class diagram and the Object Constraint Language (OCL). As a result, we see how these two technologies compare, what the advantages of both technologies are and how they can be combined.

1 Introduction

The eXtensible Markup Language XML [6] is rapidly becoming the standard format to exchange structured data over the web. The Document Type Definition (DTD) language, (which has traditionally been the most common method for describing the structure of XML instance documents) lacks enough expressive power to properly describe highly structured data. The W3C realized limitations of DTDs and has been busy to work on alternatives. Nowadays XML Schema become the most common method for defining and validating structured XML documents as it solves the limitations of DTD and provides a much richer set of structures, types and constraints for describing data. Since XML documents is viewed as a source of storing and exchanging information in XML format, it is logical to pose queries against that XML documents[1]. XQuery is the current W3C standard language for querying XML documents. At the same time, the Unified Modeling Language UML [20] is the OMG's standard language for object oriented analysis and design. UML defines modelling languages that span a range from functional requirements and activity workflow models to class structure design and component diagrams. These models, and the development process that uses them, improve and simplify communication among an application's many diverse stakeholders. Class diagram is a UML diagram that shows the static structure of the domain abstraction (classes) of the described model. It describes the types of objects in the system and various kinds of static relationships that exist among them. It shows the attributes, operations and the constraints of each class in the model. XML Schemas are static by nature so the most appropriate UML diagram to express them is the class diagram. A UML class diagram is very suitable to be used as a tool for visual representation of the elements, relationships, and constraints of an XML vocabulary. Class diagrams allow complex XML vocabularies to be understood by non-technical business stakeholders. There was a lot of work done in this area to introduce different approaches for graphically and visually representing XML document structure in the form of DTD or XML Schema using UML Class diagram and other different models [13][14]. Although UML sustain many aspects of software engineering, it does not provide explicit facility for writing queries. UML provides a textual Object Constraint Language OCL which can be used to express detailed aspects about the modelled system. OCL [16] was originally designed for expressing constraints on a UML Model. However its ability to navigate through the model types has lead to attempts for using it as a query language [15]. We believe that OCL can play for UML the same role which XQuery plays for XML. There are many similarities between OCL and XQuery which motivates us that both of them can play the same role in its world. Based on this, in this paper, we propose a novel approach for integrating XML and UML families of technologies. Figure 1 shows our proposed framework for the integration between UML/OCL and XML/XQuery. Our integration proposal is based on the idea that, each XML document is an instance of XML schema. Each XML schema can be graphically modelled using UML class diagram notations. Queries over the XML document usually represented using XQuery language and our target is to find a complete mapping schema that enable us to represent each query expressed by XQuery language as an OCL expression over the UML class diagram. Achieving this target is out of scope of this paper as in this paper, we only introduce our proposal and show the possibility of this mapping as according to our knowledge, there is no work had been

done in this point of research before. We also show that the current capabilities of OCL can cover and satisfy most of the XQuery expressions which motivates us to continue in building our complete mapping scheme. The paper proceeds as follows. In section 2, we briefly give an overview of XML, XML schema, XQuery and OCL. In section 3, we show the similarities between XQuery and OCL which motivated us for such mapping and integration framework. In section 4 we represent our case study and our idea for the possibility of representing XQuery expressions using OCL. Finally, in section 5, we conclude the paper and outline issues for further research.

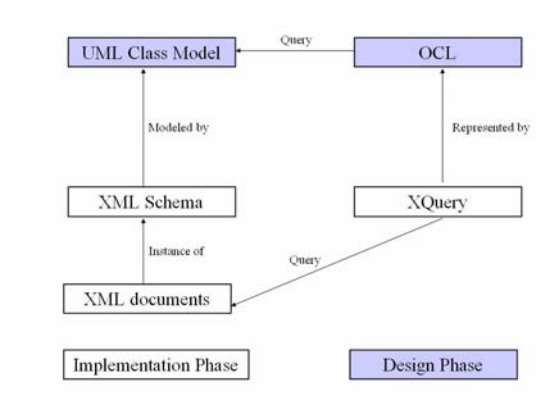

Fig1: UML/OCL, XML/XQuery mapping framework

2. Background: XML, XML Schema, XQuery and OCL

2.1 XML

XML has become the most popular format for marking up all kinds of data, from web content to data used by applications. XML is a mark-up language for structured documentation. XML provided a simple and general facility, which is useful for data interchange. It is intended to make it easy and straightforward to use SGML on the web and to make it easy to define document types and easy to transmit and share them across the web. XML enables independent computer systems to exchange, interpret, and act on data, even if those systems run on different hardware and are programmed in different languages [6].
It is a Meta mark-up language, which can be used to describe the logical structure of a wide variety of documents and data in different ways according to the application. It does not truly understand the content of document, what the tags actual conceal is entirely up to the user and application. XML specifies neither semantics nor a tag set. It only provides a facility to define tags and the structural relationship between them. All of the semantics will be defined either by the application that process the documents or by style sheets.
The basic construct of an XML document is the element. Elements can be nested at any depth and can contain other elements (sub-elements). An element contains a portion of the document delimited by two tags: the start tag, at the beginning of the element, of the form <tag-name>, and the end tag, at the end of the element, of the form </tag-name>. Empty elements of the form <tag-name/> are also possible.
A list of attributes can also be specified for an element. Attributes are of the form name = value, where name is a label and value is a quoted string, and are listed within the start tag of the element. Attributes can have different types allowing one to specify an element identifier (attributes of type ID often called id), links to other elements of the document (attributes of type IDREF, referring to a single target, or IDREFS, referring to multiple targets), or additional information about the element. Figure 2 shows an XML document example.

```
<bookstore>
      <topic>
           <name> Database Systems </name>
           <book>
              <title>... </title>
              <author>... </author>
              <isbn>... </isbn>
           </book>
     </topic>
</bookstore>
```

Fig 2: XML Document example.

2.2 XML Schema

An XML document is essentially a structured medium for storing and exchanging of information. In order to assess the validity of an XML document, we need to establish exactly to which structure the information within the document must adhere. This is accomplished with a schema, which is a model used to describe the structure of information within an XML document. Schemas are used to model a class of data. Once a data model is in place for a particular class of data, we can create structured XML documents that adhere to that model [7][18].
A schema describes the arrangement of markup and character data within a valid XML document. In other words, an XML document must adhere to a schema in order to be valid. We can think of a schema as an agreement between an XML application and the XML document on which it is based on. There exits two approaches for modeling XML documents: Modeling XML documents with DTD (Document Type Definition) and Modeling XML documents with XML Schema language. We chose XML schema to be our alternative for modeling XML document structure for the following reasons: XML Schema is the current standard of W3C to model the XML document structure, XML Schema has more expressive power and modeling capabilities than DTD, And the most important reason for our work is mapping XML schema to the UML class diagram is more preserving for the XML document semantics rather than mapping DTD to the class diagram. XML Schema is a new approach defining the schema for XML documents that uses an XML vocabulary (XML-Data). XML Schema is used to establish the schema of a class of documents[7][8]. XML Schemas describe the elements and their content model so that documents can be validated. However, XML schemas go several steps further than DTDs by allowing associating data types with elements. This allows XML processor to perform data validation, which is an extremely significant benefit of using XML Schema instead of traditional DTD. One of the most interesting aspects of XML schemas is that they are expressed in XML syntax. This means that we create an XML schema as an XML document. So, the familiar tag approach to encoding XML documents is all we need to code an XML schema. XML Schema also provides the facility to make description of the content models and to reuse the elements via inheritance. Also XML Schema supports namespace integration and attributes groups which allow to logically combining attributes. Figure 3 shows an XML Schema example.

```
<schema>
  <element name="bookstore" type="bookstoreType"/>
  <complexType name="bookstoreType">
   <sequence>
    <element name="name" type="xsd:string"/>
    <element name="topic" type="topicType" minOccurs="1"/>
   </sequence>
  </complexType>
  <complexType name="topicType">
   <element name="name" type="xsd:string"/>
   <element name="book" type="bookType" minOccurs="0"/>
  </complexType>
  <complexType name="bookType">
   <element name="title" type="xsd:string"/>
   <element name="author" type="xsd:string"/>
   <attribute name="isbn" type="isbnType"/>
  </complexType>
  <SimpleType name="isbnType">
   <restriction base="xsd:string">
    <pattern value="\[0-9]{3}[-][0-9]{3}[-][0-9]{3}"/>
   </restriction>
  </simpleType>
</schema>
```

Fig 3: XML Schema example.

2.3 XML Query Language: XQuery

Since XML documents is viewed as a source of storing and exchanging information in XML format, so it is logical to pose queries against that XML documents. This is the basic reason why a query language for XML data is extremely important. The World Wide Web Consortium (W3C) provides two textual languages to formulate XML queries and express document transformations, XQuery and XSLT. XQuery is designed to be a language in which queries are concise and easily understood, and to be flexible enough to query a broad spectrum of information sources, including both databases and documents[1] [3]. XQuery is defined in terms of XQuery1.0 and XPath 2.0 data model. The Query data model [9] represents XML data in the form of nodes and values, which serve as the operands and results of the XQuery operators. XQuery is closed under the Query data model, which means that the result of any valid XQuery expression can be represented in this model. In the Query data model, every value is an ordered sequence of zero or more items. An item can be either an atomic value or a node. An atomic value has a type, which is one of the atomic types defined by XML Schema or is derived from one of these types by restriction. A node is one of the seven kinds of node defined by XPath, called

document, element, attribute, text, comment, processing instruction, and namespace nodes. Nodes have identity, and an ordering called document order is defined among all the nodes that are in scope. XQuery is a functional language and instead of executing commands as procedural languages do, every query is an expression to be evaluated, and expressions can be combines quite flexibly with other expressions to create new expressions so the basic building block of XQuery is the expressions [1]. In XQuery, Several types of expressions are possible: *Primary expressions, Path expressions, Sequence expressions, Arithmetic expressions, logical expressions, Comparison expressions, Conditional expressions, Quantified expressions, FLOWR expressions, Element Construction expressions, Validate expressions and Unordered expressions.* We will present all these types of expressions with more details in section 4.

2.4 OCL

The Object Constraint Language [16] is a textual specification language, designed especially for the use in the context of diagrammatic specification languages such as UML.OCL was always used to add well-formedness rules on both the model and metamodel levels within UML. OCL is tightly connected to UML diagrams, as it is used as textual addendum within the diagrams, e.g. to define pre- and post-conditions, invariants, transition guards. OCL also uses the elements defined in the UML diagrams, such as classes, methods (side effect free) and attributes. The language is based on types. Each OCL expression evaluates to a type either predefined by the language or defined by the model on which the expression is built. Composing an expression comes out through the concept of navigation. Navigation in OO modeling means to follow links from one object to locate other object(s). Navigation in OCL is one of the following forms [16] [19]:

- *Navigating from an object to a property:* an example of this is accessing the value of an object's attribute or method like
  ```
  context Person inv:
  Self.age > 25.
  ```
  The size of the result in this type of navigation is always of maximum 1.
- *Navigating from type to type:* this happens when the expression moves from one object type to another through associations. For example
  ```
  context NaturalPerson inv:
  Self.address->size > 0.
  ```
  In this expression we started at the type represented by the variable self which is NaturaPerson, through the "." a navigation through the association took place and the result of the expression at this moment is a set containing all address object that matches the self object. In this case also the type of result is compound on the following form Collection (destination object type). In our case it will be the following Collection (Address). Since Collection is an abstract type in the OCL metamodel, we have to refine the type by selecting from one of three types Set, Sequence, or Bag. The selection of the correct type depends mainly on the semantics of the association between the source and destination types. By default the type is Set. If the association is labeled with {ordered} then the collection is of type Sequence. If it is possible that navigation will generate duplicate elements then the collection type is Bag.
- *Operations over collections:* this is a special type of navigation in which a Boolean predicate is examined against collection elements, either for testing that all, some, none of the elements match this predicate. An example on this navigation:
  ```
  context NaturalPerson inv:
  self.Address->select(attrStateOfapartment = #main)-> size =1.
  ```
  In this constraint an operation is made on the set of addresses a person might have to select only those addresses that are the main residence for the person. The purpose of this constraint is to state that a person must have only one main address. It is clear that within one expression the three different types of navigation can occur (which is very common with complex expressions).

Almost all uses of OCL before were as a constraint language. Constraints expression are subset of the OCL expressions that evaluate to the type Boolean. In this paper, we are about to use the querying features of OCL and its ability to represent XQuery expressions.

3. Motivations for the mapping between OCL and XQuery

The purpose of OCL is to specify constraints on UML Model Elements and to limit the possible system states. OCL can be used also to query the models [17]**,**[15]**.** In [15] OCL was not considered as a complete query language because of lacking expressions that evaluate to tuple types. But with the emergence of OCL 2.0, tuples are now possible to be expressed using OCL. On the other hand the purpose of XQuery is to provide flexible and powerful query facilities to extract data from a collection of real and virtual XML documents. In XQuery

everything is an expression that evaluates to a value. We believe that OCL can play for UML models the same role that XQuery plays for XML documents. We justify our belief by showing the following similarities between OCL and XQuery.

- *Path Expressions vs. Navigation*
  Both languages support the notion of moving between types (in OCL) and nodes (in XQuery) [10]. The three different types of OCL navigation expressions we showed in section 2.4 with the set of its predefined operations over collections can cover the different XQuery expressions that we cover in detail in section 4.
- *Type based Expressions*
  XQuery is strongly typed language, meaning that the types of values and expressions must be compatible with the context in which the value or expression is used. Types can be imported from one or more XML Schemas that describe the input documents and the output document. XQuery language can then perform operations based on these types[5]. For example, this expression raises a type error because when isbn attribute is defined as string and then compared with an integer value.

  /book[isbn] = 1234

  In the same way OCL expression must evaluate to a type that is either predefined in the language or defined in the model to which the expression is attached.
- *Declarative nature of queries*
  Queries of both languages are declarative in nature where we specify *what* we need rather than specifying *how* to reach it.
- *Common constructs*
  Both of the languages are able to represent different control, and logic constructs like if statements, looping, arithmetic operations, and logical comparison, etc.

4. Mapping between XQuery and OCL

We are looking for complete mapping scheme between XQuery and OCL. Achieving this target is out of the scope of this paper. In this paper we only open the minds for the possibility of this mapping. In section 4.1 we represent a case study that describe our suggested scenario of integration between XML, XQuery, UML and OCL. In section 4.2 we give an overview over the different types of expressions supported by XQuery, We represent our idea of the direct mappings and matches exist between some XQuery expressions and OCL. For the expressions which are not possible to be mapped directly, that will be our future work to make some extensions in OCL to make it possible.

4.1 Case study

In this section, we represent our case study according to the proposed framework of XML/XQuery and UML/OCL integration scenario represented in figure 1. Figure 4 show the XML schema of our example. Figure 5 shows an example of XML document instantiated according to the XML schema represented in figure 4. Figure 6 shows the representation of our sample XML Schema using UML class diagram notations. There are many advantages for using UML class diagram to model the XML document structure: UML is a popular method for designing software and has proven to be valuable for data modelling, UML syntax is generally more widely understood that XML Schema syntax, It is a lot easier to grasp a general picture from UML diagrams and UML is independent of the XML schema language will be used on the implementation level[22]. But the most important advantage of using UML is *stereotype*, an extension mechanism of UML in which a set of tagged values with known semantics are defined and used to mark classes or other elements of UML diagrams to have properties implied by the stereotype. There are different approaches for representing XML document using UML class diagrams [21][22] [23]. in our case study we have chosen simple one that represent each simple element type contains only text as an attribute for the class and each complex element type contain elements as separate class. We chose the class diagram association relationship to represent the relationship between the elements of our classes. Our idea of representing XQuery using OCL is independent from this step of representing XML Schema using UML class diagram so any other approach can be used without affecting our idea and our results.

```
<Schema name="OrderSchema">
  <ElementType name="firstName" content="textOnly" dt:type="string"/>
  <ElementType name="lastName" content="textOnly" dt:type="string"/>
<ElementType name="customerID" content="textOnly" dt:type="string"/>
  <ElementType name="addressID" content="textOnly" dt:type="string"/>
  <ElementType name="city" content="textOnly" dt:type="string"/>
  <ElementType name="state" content="textOnly" dt:type="string"/>
  <ElementType name="orderID" content="textOnly" dt:type="string"/>
  <ElementType name="quantity" content="textOnly" dt:type="float"/>
  <ElementType name="shipDate" content="textOnly" dt:type="date"/>
  <ElementType name="productName" content="textOnly" dt:type="string"/>
  <AttributeType name="price" dt:type="float"/>
  <ElementType name="product" content="mixed">
   <element type="productName" minOccurs="1" maxOccurs="*"/>
   <attribute type="price"/>
  </ElementType>
  <ElementType name="Customer" content="eltOnly">
   <description>
    This element type represents the customer information
   </description>
   <element type="firstName" minOccurs="1" maxOccurs="1"/>
   <element type="lastName" minOccurs="1" maxOccurs="1"/>
   <element type="custID" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="Address" content="eltOnly">
   <description>
    This element type represents the address information of an order invoice
   </description>
   <element type="addressID" minOccurs="1" maxOccurs="1"/>
   <element type="street" minOccurs="1" maxOccurs="1"/>
   <element type="city" minOccurs="1" maxOccurs="1"/>
   <element type="state" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="Shipto" content="eltOnly">
   <element type="Customer" minOccurs="1" maxOccurs="1"/>
   <element type="Address" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="order" content="eltOnly">
   <description>
    This element type represents the order information
   </description>
   <element type="orderID" minOccurs="1" maxOccurs="1"/>
   <element type="quantity" minOccurs="1" maxOccurs="1"/>
   <element type="product" minOccurs="1" maxOccurs="1"/>
   <element type="shipDate" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="orderInvoice" content="eltOnly">
   <element type="Shipto" minOccurs="1" maxOccurs="1"/>
   <element type="order" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```

Fig 4: Case study XML Schema example

```
<orderInvoice>
  <Shipto>
    <Customer>
        <firstName>Folkert</firstName>
        <lastName>Wilken</lastName>
        <custID>123C</custID>
    </Customer>
    <Address>
        <addressID>212</addressID>
        <street>206 Soflingert Street</ street >
        <city>ULM</city>
        <state>Baden-Wutenberg</state>
    </Address>
  </Shipto>
  <order>
        <orderID>ord123</orderID>
        <quantity>3</quantity>
        <product price="40">PC Monitor</product>
        <shipDate>12.12.2004</ shipDate >
  <order>
</orderInvoice>
```

Fig 5: Instantiated XML document

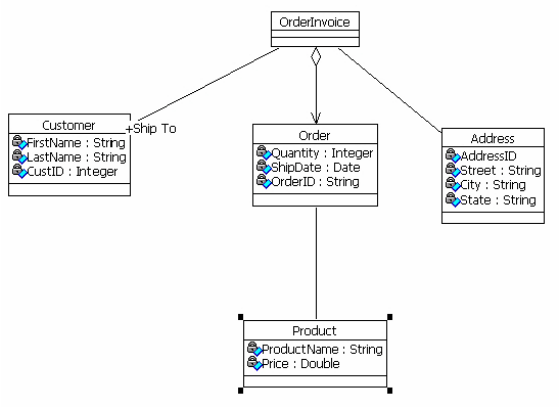

Fig 6 : Class model representing our sample XML Schema

4.2 Direct Mapped Expressions

- **Path expression**

Path expressions are used to locate nodes in XML data. Consist of a series of one or more steps separated by "\" or "\\". Every step is evaluated to a sequence of nodes. Each operation E1\E2 is evaluated as follows: E1 is evaluated and its node sequence result serves in turn to provide an inner focus for an evaluation for E2.

```
Ex1: List the first names of all customers.
("Sample.xml")/orderinvoice/shipto/customer/firstName.
```

This expression can be represented in OCL using the navigation between object types. If we refer to figure 6 and we try to query the names of customers for whom invoices have been issued; then the start of the navigation will be the "OrderInvoice**"** class. Navigation flows through the association link **"**Shipto" now the expression "OrderInvoice.Shipto" will evaluate to a *collection of objects* (*sequence of nodes in XML document*) of type "Customer". To reach the information we need (list of first names) we need to make one further navigation from the type "Customer" to the attribute "FirstName". At the end of this navigation we have reached the information we need from this query. The equivalent OCL expression is

```
OrderInvoice.Shipto.Firstname
```

- **Positional Predicates**

Predicates are Boolean conditions that select a subset of the nodes computed by a step expression.

```
Ex2: Return the first Customer in the document
("Sample.XML")/OrderInvoice/ShitpTo/Customer[1].
```

To return a certain element at certain position in a collection of objects we need to use the subsequence operation defined in OCL document [12]**.** The equivalent OCL expression is

```
OrderInvoice.Shipto-> asSequence->subsequence(1,1)
```

- **FLOWR expression**

XQuery provides a feature called a FLOWR expression that supports iteration and binding of variables to intermediate results. It is similar to the SELECT-FROM-WHERE statements in SQL. The name FLOWR is an acronym, standing for the first letter of the clauses that may occur in FLOWR expression. *for and let clauses*, generate a sequence of bound variables called the tuple stream. *Where clause* serves to filter the tuple stream retaining some tuples and discarding the others. *Order by clause* improves an ordering on the tuple stream. *Return clause* evaluated once for every tuple in the tuple stream.

```
Ex3: Return the ID of all orders which have quantity greater than 10
For $X in ("Sample.xml")/orderInvoice/order
  Where $X/quantity > 10
Return  $X/orderID
```

To represent a query where some objects are filtered according to a given selection criteria we can use the Select operation from OCL to make this filtration. The navigation starts from "OrderInvoice**"** using the association to the class "Order OrderInvoice.Order**"** the expression now evaluates to a set of objects of type "Order**"** on this set we need to select only orders which have quantity greater than 10. the expression now becomes `OrderInvoice.Order-> Select(quantity > 10)`. This expression also evaluates to set of orders but for which the expression `quantity > 10` holds. To reach the information we need to make one more navigation to the attribute "OrderID". The complete expression will be.

```
OrderInvoice.Order-> Select(quantity >10).OrderID
```

- **Conditional expression**

XQuery's conditional expressions are used in the same way as conditional expressions in other languages. It is a Very well known and common type of expressions. *IF* test expression *Then* then-expression *Else* else-expression.

```
Ex4: Return the ID of all orders which have quantity greater than 10 (rewriting of Ex3 using of conditional expression format instead of where clause format)
For $X in ("Sample.xml")/orderInvoice/order
   If $X/quantity > 10  then
      Return  $X/orderID
   Else
      ()
```

OCL has a direct representation for the conditional expressions like all other language, The equivalent OCL expression is:

```
if OrderInvoice.Order.quantity > 10 then
    OrderInvoice.Order.OrderID
Else
    NULL
End if
```

- **Arithmetic expression**

XQuery support the arithmetic operators +,-,*, /, div, mod. Each operand of the arithmetic expression should be represented by sequence with length exactly equal to one and then normally apply the expression operators over the sequence values. The result of arithmetic expression is a sequence with exactly one element.

```
Ex5: let X = 3 + 4
```

The equivalent OCL expression is:

```
Let  X = 3 + 4
```

- **Quantified expression**

Quantified expressions support existential and universal quantification. The value of a quantified expression is always true or false. If the quantifier is *some* then the quantified expression is true if at least one evaluation of the expression is true. If the quantifier is *every* then the quantified expression is true if every evaluation of the expression is true.

```
Ex6: this expression returns true if there exist any instance of order that have quantity greater than 10
Some $X in ("Sample.xml")/orderInvoice/order Satisfy  $X/quantity > 10.
```

This expression can be represented in OCL using "Exists" operation defined for collection types. This expression evaluates to the Boolean type

The OCL expression will be

```
OrderInvoice.Order-> Exists (Quantity > 10)
```

- **Logical expression**

A logical expression is either "and-expression" or an "or-expression". Its value is always one of the Boolean value true or false.

```
Ex7: return all products which have prices in the range between 10 and 20
("Sample.xml")/orderInvoice/order/product[@price > 10 and @price < 20].
```

The equivalent OCL expression will be

```
OrderInvoice.Order.Product-> Select( Price > 10 and Price < 20)
```

- **Sequence expression**

XQuery supports operators to construct and combine sequence of items. Sequences are never nested. Also XQuery provides the *union*, *intersect* and *except* operators for combining sequences of nodes

```
Ex8: let s1=(1,2,3)
        let s2= (4,5,6)
        let s3 = (s1,s2)
       for $X in s2
       return $X
```

The equivalent OCL expression is:

```
let  s1 : sequence(Integer)  = (1,2,3)
let s2: sequence(integer) = (4,5,6)
let s3: sequence(integer) = s1->union (s2)
for I = 1 to s3->size()
```

```
{
      s3->subsequence(I,I)
}
```

- **Value comparison expression**

Value comparison expressions are intended for comparing single values. Each operand must contain exactly one atomic value. The result of the expression is true if the value of the first operand satisfy the comparison operation (eq – ne – lt – le – gt - ge)to the value of the second operand otherwise the result of the comparison is false.

```
Ex9: (“Sample.xml”)/orderInvoice/Shipto/customer/firstName eq “Ahmad”
```

Representation of value comparison expressions can be represented using OCL with a navigation path that ends at an attribute and then the value of this attribute is compared to the value. The OCL expression for this is `OrderInvoice.Shipto.Firstname = “Ahmad”`. The XQuery value comparison necessitate that the expression evaluates to a single node and its value is compared. The previous OCL expression lacks this restrictions as it will evaluate to true only if all the “FirstName” value for all objects are equivalent to the value “Ahmad”. In the following we add the condition that the navigation results in only a single object which “FirstName” value is equivalent to “Ahmad”.

```
OrderInvoice.Shipto.firstname-> Size = 1 and OrderInvoice.Shipto.firstname
= “Ahmad”
```

- **General comparison expression**

General comparison expressions are existentially quantified comparisons that may be applied to operand sequences of any length. The result of the expression is true if there is a pair of atomic values, one belonging to the first operand and the other belongs to the second operand which are satisfying the comparison operation (= , !=, < , <=, >, >=) otherwise it is false

```
Ex10: (“Sample.xml”)/orderInvoice/Shipto/customer/firstName = “Ahmad”.
```

The equivalent OCL expression is

```
OrderInvoice.Shipto-> Exists(Firstname = “Ahmad”)
```

- **Unordered function**

XQuery expressions return sequences that have well-defined order. The *unordered* function takes any sequence of its argument and returns the same sequence of items in a nondeterministic order. A call to the *unordered* function is permission for the argument expression to be materialized in whatever order the system finds most efficient.

```
Ex11: unordered((“Sample.xml”)/orderInvoice/Shipto/customer/firstName)
```

The equivalent OCL expression is

```
OrderInvoice.Shipto.FirstName -> asSet()
```

- **Built-in functions**

XQuery has a set of built-in functions and operators including many that are familiar from the other languages and some that are used in customized XML processors. These built-in functions include (min() – max () – count() - … etc)

```
Ex12: return the number of orders in this document
Count((“Sample.xml”)/orderInvoice/order).
```

The count function in XQuery has a direct equivalent in OCL which is the operation **Size** defined for collection types. The equivalent OCL expression is

```
OrderInvoice.Order-> Size()
```

- **Element construction expression**

XQuery provides constructors that can create XML structures with in a query. Constructors are provided for evry kind of node in XQuery data model.

```
Ex13 :
For $X in (“Sample.xml”)/orderInvoice/order
          Where $X/quantity > 10
         Return  <bigQuantity> <OrderID>$X/orderID</OrderID>
            <quantitv>$X/quantity<quantity></bigQuantity>
```

Such expression was not possible in OCL versions prior to Version 2.0. with the addition of tuple types to OCL it is capable of modeling the previous query where the expression evaluates to a tuple or a collection of tuples that have elements each of which belongs to possibly different type. In the equivalent OCL expression we first need to define the tuple type for this query, then we define the expression which evaluates to this type.

```
Def:
```

```
bigQuantity: Set(Tupletype(OrderID: String, Quntity: Integer))=
OrderInvoice.Order->Select( quantity > 10)->Tuple[OrderID = OrderID,
Quantity = Quantity
```

4.3 Unmapped Expressions

- **Node Comparison (is and isnot operators)**

Each operand must be either single node or an empty sequence. A comparison with the *is* operand is true if the two operands have the same identity otherwise it is false. A comparison with the *isnot* operand is true if the two operands have different identities otherwise it is false.

```
Ex14: (“Sample.xml”)/orderInvoice/Shipto/customer/firstName isnot
(“Sample.xml”)/orderInvoice/Shipto/customer/firstName
```

In OCL, there is no direct mapping for this expression because this type of XQuery expressions has the access to the Node Identity not its value which is not explicitly represented in the class model and its instances.

- **Order comparison (<< and >> operators)**

Each operand must be either single node or an empty sequence. A comparison with the << operand is true if the first operand node is earlier than the second operand node in document tree otherwise it returns false. A comparison with the >> operand is true if the first operand node is later than the second operand node in document tree otherwise it returns false.

```
Ex15: (“Sample.xml”)/orderInvoice/Shipto/customer/firstName <<
(“Sample.xml”)/orderInvoice/Shipto/customer/firstName
```

There is no direct mapping for this expression because it is not currently possible to represent the precedence between objects like it is possible to represent the precedence between nodes in XML tree.

5. Conclusion and Future work

In this paper, we have shown and discussed the possibility of representing XQuery expressions using OCL. Our proposal in this paper can be regarded as the first step of our work towards a complete framework of integration between XML and UML both worlds. It is therefore our plan in the future work to design a complete framework of integration between XML/XQuery on one hand and UML/OCL on the other. In this work UML and OCL will play the role of modelling the data structure and the queries in the design. The modelled data structure and queries will be mapped to an XML schema and XQuery expressions in the XML implementation layer. We believe that as a side effect of this work the XML layer can be replaced with any other implementation layer (Relational Database and SQL – Object Oriented Database and OQL -…). In our work we chose to use the XML as an implementation target layer because the current widespread of usage of XML and to introduce new line for using UML in the web environment. The use of OCL has been addressed e.g. in the work of [17] [15]. The major drawback there was the inability of OCL to represent queries that return tuples and outcome of join operations but With OCL 2.0, tuple types are definable and so such problem is solved. In section 4.4 we have shown that some expressions are not possible to map without extra information. This stems from the difference in the architectural model between OCL and XQuery. We expect our work to continue and evolve in the future. The authors are currently planning to build a prototype tool that represent their proposed framework of integration and which can implement their future complete mapping scheme. Also another point for future work will be to build extra metamodels for the XQuery and the XML documents to let UML and OCL understand the tree structure which makes mapping of the currently unmapped expressions possible.

References

1. Denise Draper, Peter Fankhauser, Mary F. Fernandez, Ashok Malhotra, Kristoffer Rose, Michael Rys and Philip Wadler. :XQuery 1.0 and XPath 2.0 Formal Semantics. Technical Report W3C Working Draft, World Wide Web Consortium, May 2003.
2. S.Alagic.: Type-Checking OQL Queries In the ODMG Type Systems". Transactions on Database Systems, 24(3), 1999.
3. J.Robie.: An Introduction to XQuery. in XQuery from the Experts. A Guide to the W3C XML Query Language edited by Howard Katz, Addison-Wesley, 2003.
4. D.Chamberlin: Influences on the Design of XQuery. in XQuery from the Experts: A Guide to the W3C XML Query Language, edited by H. Katz, Addison-Wesley, 2003.
5. M. Fern´andez, J. Simon, P.Wadler.: Static Typing in XQuery. in XQuery from the Experts: A Guide to the W3C XML Query Language, edited by H. Katz, Addison- Wesley, 2003.
6. Tim bray, Jean Paoli, C.M.Sperbrag, Ere Maler.: eXtensible Markup Language (XML) 1.0 specification. W3C Recommendation. October 2000.
7. Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn.: XML Schema Part 1: Structures. W3C Recommendation. May 2001.
8. Paul V. Biron, Ashok Malhotra.: XML Schema Part 2: Datatypes. W3C Recommendation, May 2001.
9. XQuery 1.0 W3C Working Draft, May 2003.
10. XPath 2.0 data model, W3C Working Draft, May 2003.
11. OMG, editor. The Common Warehouse Metamodel Specification. OMG, 2000.
12. Rational Software Corporation: The Object Constraint Language specification,*Version 1*.4. 1999.
13. S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, L. Tanca, .: XML-GL: a Graphical language for Querying and Restructuring XML Documents. computer networks 31, 1999.
14. Enrico Augurusa, Daniele Braga, Messandro Campi, Stephano Ceri. :Design and implementation of a graphical interface to XQuery. SAC2003.
15. D.H. Akehurst, B. Bordbar.: On Querying UML Data Models with OCL. UML 2001 - The Unified Modeling Language.Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 2001, Proceedings, 2001
16. J. Warmer, A. Kleppe, T. Clark, A. Ivner, J. H.ogstr. om, M. Gogolla, M. Richters,H. Hussmann, S. Zschaler, S. Johnston, D. S. Frankel, and C. Bock.: Object Constraint Language 2.0. Technical report, Submission to the OMG, 2001
17. Martin Gogolla and Mark Richters. :On constraints and queries in UML. In Martin Schader and Axel Korthaus, editors, The Unified Modeling Language -- Technical Aspects and Applications, pages 109--121. PhysicaVerlag, Heidelberg, 1998.
18.Sherif Sakr, Mokhtar Boshra.: Using relational metadata to generate enhanced XML document structure. In proceedings of INFOS 2001, Cairo University, November 2001.
19. Ali Hamie, John Howse, Stuart Kent.: Navigation Expressions in Object-Oriented Modeling Proceedings Fundamental Approaches to Software Engineering, 1st International Conference, 1998.
20. OMG, Unified Modeling Language Specification, version 1.4, 2001.
21. John Heintz and W. Eliot Kimber.: Using UML to define XML document types. isogen international, 2000.
22. Juho Tikkala.:Modeling W3C XML Schemas using UML. 2003
23. Rainer Conrad, Dieter Scheffner, J.Christof.:XML Conceptual Modeling using UML. 2003