Mapping Rules to Convert from ODL to XML-SCHEMA

Artur Afonso de Sousa¹, José Luis Pereira² and João Álvaro Carvalho²

¹ Escola Superior de Tecnologia de Viseu, Departamento de Informática, Portugal
ajas@di.estv.ipv.pt

² Universidade do Minho, Departamento de Sistemas de Informação, Guimarães, Portugal
{jlmp, jac}@dsi.uminho.pt

Abstract

As new standards for technology specifications related to XML are unveiled, and stable tools to implement them become available, the widespread usage of XML as an universal format for data exchange between heterogeneous systems (using the Internet), will increasingly become a reality. Therefore, in order to deal efficiently with the large amounts of XML data that will be generated in the near future it is imperative to find efficient alternatives for the storage and management of this special kind of data. For that matter, object-oriented databases seem to be a good alternative.

In this paper, we begin with an analysis of the different alternatives available to store and manage XML data. After that, we concentrate our attention on the Object-Oriented (OO) database approach and, in that context, we present a first set of rules that allow the translation from a simple database schema specified in ODL (Object Definition Language) into a schema specified in XML-SCHEMA.

In the near future we intend not only to expand this set with new rules that will handle other, more complex, OO modeling constructs, but also to create another set of transformation rules to work the other way round (i.e. from XML-SCHEMA to ODL). Our aim is to cover all the possible modeling situations one may come across in OO database schemas specified in ODL.

1. Introduction

Everyone agrees that, despite its infancy, XML promises to revolutionize the future of the WWW. Due to its flexibility and extensibility, some even consider that XML has all the qualities to become the “lingua franca” of the Internet. Indeed, the expectations around XML are increasingly high, particularly in the area of data interchange between heterogeneous systems connected to the Internet. Therefore, in order to deal efficiently with the large amounts of XML data that, certainly, will be generated and exchanged between systems in the near future, it is essential to analyze the different options available regarding the storage and management of this special kind of data.

In the database community there are different opinions about the merits of each database model in dealing with XML data. There are those who claim that the relational (and object-relational) model is still the best available option, due to its maturity and widespread use in organizations. There are others who argue against the relational model, because of its over simplicity and limited semantic power, and rather prefer the object-oriented model. An increasing number of people are embracing the very recent semi-structured data models and the native systems as the best alternative regarding this matter. So, the point is, everybody claims that “their” model is the right one to store and manage XML data.

With this paper we hope to contribute to this discussion, clarifying a little bit more what is involved with the storage and management of XML data using object-oriented databases. With this purpose in mind, we present a first set of nine rules that allow the translation from a simple database schema specified in ODL (Object Definition Language) into a schema specified in XML-SCHEMA [11, 26, 5], which, from a conceptual point of view, seems to be a particularly adequate option.

Regarding the structure of this paper: the second section deals with the storage and management of XML data; in the third section we present a set of transformation rules to convert an ODL schema to a XML-SCHEMA schema; finally, the fourth section concludes the paper.

2. Storage and management of XML data

Nowadays, there are several alternatives to the storage and management of XML data, namely relational (and object-relational) databases, object-oriented databases, semi-structured databases, native XML databases, and even the file system.

XML is quickly becoming the standard for data interchange in the WWW. Therefore, if we take into account that today a large amount of electronic data is stored in relational and object-relational databases, it
should be clear why database vendors are increasingly expanding their systems with functionalities regarding the efficient storage and management of XML data. In fact, most of the DBMSs available in the market already support some kind of XML data manipulation. First, XML documents are translated into a suitable format, which can be stored in those systems; then, using the well-known SQL, those documents may be easily and efficiently manipulated and queried. Additionally, query results may be delivered in the XML format.

Since the ODMG standard [8], which is now in the 3rd version, has already achieved a sufficient level of maturity, one may consider that object-oriented DBMSs (OO DBMSs) are, in the present, valid alternatives to store XML data. Indeed, regarding this matter, one expects that the universal adoption of XML may, very soon, increase the acceptance of object-oriented databases, whose market is still very small when compared with that of the relational and object-relational ones. With that purpose in mind, object-oriented database vendors are quickly expanding their products with facilities to store and manage XML data in an efficient manner.

However, it is important to notice that, to store XML documents in conventional databases (relational, object-relational and object-oriented), a certain amount of translation work is inevitable. The reason is that the structure of an XML document is different from the data models of conventional database systems and, therefore, translation rules have to be applied. Depending on the specific database model in use, there are situations in which the translations may be easily accomplished. However, there are other cases that imply very complex translation procedures. In any case, to store XML data in these systems it is necessary that the XML documents be highly structured and their structure be compatible with that of the database. Nevertheless, regarding this last point, the use of XSLT style sheets makes it possible to transform the structure of an XML document in order to facilitate its translation to a specific database schema.

Due to the nature of the information in the web and the flexibility of the XML, we may expect that a great part of the XML data will exhibit the characteristics of semi-structured data [21]. Therefore, several researchers claim that the research results obtained in the area of semi-structured data models may be applied in the management of XML data. Initially, the LORE DBMS was developed to store and manage semi-structured data in the OEM format [20]. This DBMS has been recently redefined with the purpose of storing and managing XML data [14, 15]. The LORE system uses a very powerful declarative query language – Lorel. This system has mechanisms that allow the creation of indexes to accelerate data access and a schema (called dataguide), which is dynamically updated and, so, correctly represents the state of the data at every moment.

Another possibility that has recently emerged is the utilization of native XML DBMSs. Some examples, among others, are eXcelon\(^1\), Tamino [25] and Natix [17]. These systems have a major advantage: it is not necessary to do any translation because the data is directly stored in the database in XML format. Additionally, these are real DBMSs, which deliver many of the functionalities that are normally present in conventional databases. However, one has to take into consideration that this is a very immature technology, which will take some time to become stable and robust.

The file system is also considered a simple approach and a viable alternative to the storage of XML data. In a very simplistic way, one may consider an XML document as a database and its underlying grammar (whether it is defined as a DTD, or in the XML-SCHEMA format) as the database schema. Therefore, using an XML processor (for instance, which implements the API DOM) to access the document structure and making use of a programming language (java, perl, or C#, among others) to manipulate/query the content of the documents, we may implement rudimentary databases. However, that limits the kind of queries that may be formulated in the database. Those queries are restricted to the possibilities that are offered by the XML processor, or by the programming language used. In [16], the reader may observe the implementation of several examples that follow this methodology.

Obviously, the creation of a standard query language to interrogate XML data, such as Xquery, will certainly accelerate the utilization of this approach. For instance, instead of using structured files to store their data, new applications can use XML text files as their storage support. Additionally, they should allow the formulation of ad hoc queries (for example, through Xquery) on these data. Since XML is a universal standard, those applications that have their data in this format are immediately ready to exchange data with other applications that also “understand” XML.

In an attempt to replace the solutions based on proprietary DBMSs to store and manage XML data, the PDOM (Persistent DOM) system has been recently proposed. This system’s main aim is to add persistent characteristics to the object models created with the API DOM, from W3C. There are tools already available which implement the PDOM system, and which have an engine to process XML queries. In http://xml.darmstadt.gmd.de/xql/ it is possible to download a tool, called GMD-IPSI XQL, which implements this approach. This tool has been developed in

\(^1\) See http://www.exceloncorp.com.
java and has two main components: an implementation of the PDOM system and an engine to process XQL queries [22]. XML documents are processed through the API DOM and are stored on binary files in the PDOM format. After that, they may be easily and efficiently consulted through the XQL query language. Moreover, PDOM files may also be manipulated through the DOM interface. Notice that, with this tool, it is also possible to create PDOM files, via programming, using the methods available in the API DOM. The major advantage of this solution, when compared to the utilization of conventional DBMSs, is the possibility to store and interrogate XML documents, which have an arbitrary structure, without additional translations.

In [13, 23, 24], the storage and management of XML data through RDBMSs is analyzed and some possible translation rules, and the corresponding performance results, are presented. Additionally, the advantages and disadvantages of this approach to store and manage XML data are discussed. In [1, 2, 3, 27], different storage strategies for XML data, namely relational, object-relational, object-oriented and semi-structured DBMS and the file system, are analyzed. [9] describes an application that uses the OO-DBMS O2 [4] to store and manipulate XML data. A C++ program reads an XML document and creates the corresponding tree in the database. After that, the users may interrogate the database through OQL queries. According to Chaudhri and Zicari, the resulting application was surprisingly easy to develop and it had a very good performance.

3. Mapping rules to convert an ODL schema to an XML-SCHEMA schema

The ODMG 3.0 standard has already achieved a sufficient level of maturity and one expects that the universal adoption of XML may, very soon, increase the acceptance of object-oriented databases. Moreover, the XML-SCHEMA language is much more powerful than the previous document type declarations concerning the representation of data types and the specification of semantic restrictions, which are fundamental in conventional databases. Therefore, we believe it is particularly relevant to develop work in the sense of providing a bi-directional translation between XML documents and OO databases.

Now, let's consider a simple example of a possible conversion between an OO database schema, specified in ODL, and a schema specified in XML-SCHEMA. Figure 1 illustrates the corresponding UML Class Diagram.

The choice of ODL to specify the OO database schema is due to the fact that this language is a standard proposed by the ODMG that allows the portability between OO database schemas. As a result, a database schema defined in ODL can, theoretically, be implemented in any OO-DBMS that follows the standard.

![Figure 1 - An example Class Diagram](image)

The ODL specification corresponding to the Class Diagram above is represented in table 1 (keywords in bold):

```
Class book
(extent books key isbn )
{ attribute string isbn;
   attribute string title;
   attribute unsigned short year;
   attribute float price;
   relationship set<author> writtenby
     inverse author::write;
   relationship publisher publishedby
     inverse publisher::publish;
};
Class author
(extent authors key code )
{ attribute string code;
   attribute string name;
   attribute string surname;
   attribute list<string> email;
   relationship set<book> write
     inverse book::writtenby;
};
Class publisher
(extent publishers key pubid )
{ attribute string pubid;
   attribute string name;
   attribute string headquarter;
   attribute string homepage;
   relationship set<book> publish
     inverse book::publishedby;
};
```

Table 1 - ODL representation for the Class diagram of figure 1.

Reminding some aspects about ODL: the keyword `extent` is used to define a mechanism that represents all the instances of a class; the keyword `key` is used to specify the attribute or attributes whose values uniquely identify an instance of a class; the keyword `list` is used in the definition of the attribute `email` of the class `author` to specify that an author may have several email addresses; the keyword `relationship` is used to specify a relationship between two classes; and the keyword `set`, used in the definition of some relationships (writtenby, publish and write), indicates the cardinality of those relationships. In the following, we describe a set of seven transformation
rules to convert a simple ODL database schema into an XML-SCHEMA schema:

**Rule 1** – Since every XML document must have a root element, such an element has to be created with, for example, the database name. One should also associate with that root element an anonymous complex type that includes a special choice element with a maxOccurs attribute equal to unbounded.

**Rule 2** – Each top-level class in the ODL schema is converted into an element with the same name, which is included in the special choice element created above (rule 1). One should also associate with each of these elements a complex type with a special sequence element. The contents of each one of these special sequence elements are derived from the next rules.

**Rule 3** – The attributes with basic data types (string, short, date, float, etc.) are translated into atomic elements with the same name. These elements are included in the corresponding special sequence element that has resulted from rule 2, and their data types should, if possible, be the same, although users could indicate another data type.

**Rule 4** – Regarding **key** attributes, a special **key** element has to be declared as a sub-element of the element that has resulted from rule 1. This special element should also have a **name** attribute which value should be equal to the name of the class that the key attribute belongs to, plus the letter K (Key) appended to it. The xpath attribute of the **selector** (sub) element belonging to that special key element should contain a Xpath expression like: 
//class_name_which_the_key_attribute_belongs_to.
Nevertheless, users should have the possibility to define the value of the Xpath expression. The xpath attribute of the (sub) element **field** should contain an Xpath expression like: **keyattribute_name**.

**Rule 5** – Each relationship that contains the keywords **set, list or bag**, is converted into an element with the same name. This element is included in the special sequence element that has resulted from rule 2 and, since the cardinality of the relationship is greater than one, one should explicitly associate the attribute maxOccurs with it, with value unbounded. This element should include an anonymous complex data type with a special element **complexContent**. Moreover, this special element should include a special restriction element that contains an attribute with the name of the key attribute of the referred class. Additionally, a special keyref element has also to be declared as a sub-element of the element that has resulted from rule 1. This special element should also have a name attribute which value should be like: class_name_which_the relationship_belongs_to/Ref_class_name_referred_to. This special element should also include a refer attribute with the value: key_element_name_of_referred_class. The xpath attribute of the selector (sub) element belonging to this special element should contain a Xpath expression like: /class_name_which_the_relationship_belongs_to/relatio_nship_name. Nevertheless, users should have the possibility to define the value of the Xpath expression. The xpath attribute of the (sub) element *field* of this special element should contain a Xpath expression like: @key_attribute_name_of_the_referred_class.

**Rule 6** – Each relationship that does not contain any of the keywords set, list or bag, is converted into an element with the same name. The translation is similar to that of the previous rule, with three exceptions: the attribute maxOccurs (with value unbounded) should not be associated with the created element. The user should define its data type, although a default data type may be used for this matter; finally, in the declaration of the special keyref element, the xpath attributes of the selector and field (sub) elements will have another values. So, the xpath attribute of the selector element should contain a Xpath expression like:

class_name_which_the_relationship_belongs_to.
Nevertheless, users should have the possibility to define the value of this Xpath expression. The xpath attribute of the field element should contain a Xpath expression like: 
//relationship_name.

**Rule 7** – Each attribute of a class that contains the keyword list in its definition is converted into an element with the same name, which is included in the special sequence element that has resulted from rule 2. This element should include an anonymous simple data type. Moreover, the simple data type should include a special list element with an itemType attribute, whose value could be indicated by users, or defined by default (for instance, string).

Regarding rule 1, in the example below we use an element bibdb to represent the entire database. Notice that we use a special choice element with a maxOccurs attribute equal to unbounded to specify that the element bibdb might contain several book, author or publisher (sub) elements, without any specific order. Thus, if we apply the transformation rules described above to the previous ODL schema, we should obtain the XML-SCHEMA schema illustrated in table 2:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>XML Schema derived from an ODL schema</xsd:documentation>
  </xsd:annotation>
<xsd:element name="bibdb">
  <xsd:complexType>
    <xsd:sequence>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```
Table 2 - Resultant XML-SCHEMA

In the XML-SCHEMA example above, some aspects are worth noting. For instance, the specification of data types for elements, using the attribute `type`; the specification of data restrictions; the use of special elements and attributes, belonging to the XML-SCHEMA vocabulary, such as `key`, `keyref`, `choice`, `list`, `selector`, `field`, `refer`, `xpath`, etc. Through the definition of the special element `key`, named `publisherK`, we have established an identity constraint, in which the element `pubid` is a key of the element `publisher`. Moreover, the definition of the special element `keyref`, named `bookRefpublisher` indicates that the (sub) element `publishedby`, of the `book` element, points to a `publisher` element, whose value of its (sub) element `pubid` is equal to the value of the (sub) element...
publishedby. Hence, contrarily to the document types declarations, when one uses XML-Schema it is possible to specify rigorously what is included in a restriction. In other words, we can specify that the (sub) element publishedby, of a book element, point exactly to a publisher element whose value of its (sub) element pubid is equal to the value of de (sub) element publishedby of that element book. In addition, one can use the special attributes minOccurs and maxOccurs to define restrictions regarding the number of instances.

The seven rules presented above can only handle the most basic situations regarding the translation from an ODL database schema into an XML-Schema schema. Now, let’s consider another translation rule that deal with the cases in which there are classes with complex type attributes. For that matter let’s modify the specification of the publisher class described earlier (see table 3):

```java
Class publisher
(extent publishers key pubid )
{
    attribute string pubid;
    attribute string name;
    attribute string headquarter;
    attribute address publisheraddress;
    attribute string homepage;
    relationship set book::publishedby;
}

Class address
{
    attribute string street;
    attribute string city;
    attribute string zip_code;
    attribute string country;
}
```

**Table 3 - New definition of the class publisher**

In this new definition of the class publisher, another attribute (publisheraddress) of complex data type address has been included. So, let’s introduce another rule:

**Rule 8** – Each attribute with a complex data type is converted into an element with the same name. This element should be included in the special sequence element that has resulted from rule 2. However, this special element should include an anonymous complex type with a special sequence element in which the elements corresponding to the conversion of the attributes integrated in the class (that defines the complex type of the attribute being converted) are declared. Of course, according to the data type of these attributes we should apply the corresponding rules. For instance, if the data type is simple then we should apply rule 3 again; if it is complex, we should apply rule 8 once more, etc.

Using this new rule, the ODL schema above would be translated to XML-Schema, as illustrated in table 4.

```xml
<xsd:element name="publisher">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="pubid" type="xsd:string"/>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="headquarter" type="xsd:string"/>
            <xsd:element name="publisheraddress">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="street" type="xsd:string"/>
                        <xsd:element name="city" type="xsd:string"/>
                        <xsd:element name="zip_code" type="xsd:string"/>
                        <xsd:element name="country" type="xsd:string"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

**Table 4 - New XML-Schema representation of the class publisher**

To present the last rule of our basic set of translation rules, let’s consider the ODL data type enum. This is used to restrict the set of allowed values that an attribute may represent. As an example, let us suppose that the class author is redefined as illustrated in table 5.

```java
Class author
(extent authors key code)
{
    attribute string code;
    attribute string name;
    attribute string surname;
    attribute enum title {Bachelor, Master, PhD};
    attribute list<string> email;
    relationship set book::writtenby;
}
```

**Table 5 - New definition of the class author**

In XML-Schema it is also possible to restrict the set of permitted values of a simple data type. So, let’s introduce the nine rule:

**Rule 9** – Each attribute (of a class) that contains the keyword enum in its definition is converted into an element with the same name, which should be included in the special sequence element that has resulted from rule 2. This element should include an anonymous simple data type. Moreover, this simple data type should include a special restriction element with a base attribute, whose value could be indicated by users, or defined by default (for instance, string). The restriction element should also include, for each one of the values of the enum data type,
a special *enumeration* element with the same value in its *value* attribute.

Using this new rule, the ODL schema above (definition of the class *author*) would be translated to XML-SCHEMA, as illustrated in table 6:

```xml
<xsd:element name="author">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="code" type="xsd:string"/>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="surname" type="xsd:string"/>
      <xsd:element name="title" minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Bachelor"/>
            <xsd:enumeration value="Master"/>
            <xsd:enumeration value="PhD"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Table 6 - New XML-SCHEMA representation of the class *author*

Now, considering all the extensions applied to the initial ODL schema, and the corresponding translation to XML-SCHEMA, there is an example, in table 7, of a possible instance of that schema.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bibdb xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:\Conf\SCCC 2002\Camara Ready\schema.xsd">
  <book>
    <isbn>1-55860-452-9</isbn>
    <title>O-R DBMSs Tracking ... Great Wave</title>
    <year>1999</year>
    <price>50</price>
    <writtenby code="A1"/>
    <writtenby code="A2"/>
    <publishedby/>
  </book>
  <author>
    <code>A2</code>
    <name>Paul</name>
    <surname>Brown</surname>
    <email>pb@hotmail.com</email>
    <write isbn="1-55860-452-9"/>
  </author>
  <author>
    <code>A1</code>
    <name>Michael</name>
    <surname>Stonebraker</surname>
    <title>PhD</title>
    <email>ms@hotmail.com</email>
    <write isbn="1-55860-452-9"/>
  </author>
</bibdb>
```

Table 7 - Example of a XML document

Notice that we can use the *minOccurs* and *maxOccurs* attributes to restrict the occurrences of elements and attributes in the XML instance document. The example above, despite its simplicity, clearly demonstrates the versatility of the XML (and XML-SCHEMA) in the representation of complex data structures. The nesting of elements in an XML document may have an arbitrary number of levels. Furthermore, both the ODMG object model and the XML-SCHEMA specification offer many other facilities to deal with the representation of complex data. However, a more complete example is beyond the purposes of this paper.

In the near future we intend not only to expand this set with new rules that will handle other, more complex, OO modeling constructs, but also to create another set of transformation rules to work the other way round (i.e. from XML-SCHEMA to ODL). Our aim is to cover all the possible modeling situations one may come across in OO database schemas specified in ODL.

In [10] the translation of SGML documents into the OO database system O2 is described. In [18, 19], some suggestions to translate document type declarations into object-relational schemas are presented. In [6, 7] is presented a system prototype to publish the content of object-relational databases in the XML format. In [12], the architecture and the functioning mode of a tool to convert the content of relational databases to XML documents is described.
4. Conclusion

XML is revolutionizing the WWW. Today, more and more people agree that XML has all the qualities to become the “lingua franca” of the Internet, and so the expectations around XML are increasingly high. In particular, due to its characteristics of flexibility, extensibility and high level of automation, it is expected that XML will soon achieve a major role in the exchange of data and interoperability between heterogeneous computing systems.

Databases and XML seem to be complementary technologies. In fact, while XML offers a flexible and easy approach to data interchange between heterogeneous systems, databases support the efficient storage and management of data in those systems. Nowadays, there are essentially five approaches regarding the storage and management of XML data: object-relational (and relational) databases; object-oriented databases; semi-structured databases; native databases and the file system itself (together with the proper APIs, programming languages, and XML query languages).

In this paper we have discussed some of the issues involved in the storage and management of XML data through object-oriented databases. In that context, we introduced a first set of rules that allow the translation between a database schema specified in ODL (Object Definition Language) and a schema specified in XMLSCHEMA. The reason behind the choice of XMLSCHEMA to describe the structure of the data in the resulting XML document is straightforward. XMLSCHEMA is a much more powerful and expressive language than the earlier document type declarations and thus, it’s more adequate to the job. In fact, some XMLSCHEMA facilities, which are basic needs in the database context, such as data type specifications and several restrictions about data, are simply not available in document type declarations.

In a next paper we plan to extend the set of rules described here with new rules to handle more complex ODL modeling constructs. Our aim is to cover all the possible situations one can come across when translating an ODL schema to a schema specified in XMLSCHEMA.

References


