Using XML as an Object Interchange Format

G.M. Bierman
Department of Computer Science
University of Warwick

May 17, 2000

Abstract

In the ODMG standard for object databases [1], a specification language is defined to dump and load the current state of ODMG-compliant databases. In this paper we propose an alternative language, OIFML, based upon XML.

1 Introduction

The Object Data Management Group (ODMG) have given a number of specifications for the persistence of object-oriented programming language objects in databases. These specifications form an industry standard for object data management systems (ODMSs), which has been published as a book [1] (hereafter referred to simply as the Standard).

The Standard has four main components.

1. An object model.
2. Object specification languages.
3. Object query language (OQL).
4. Programming language bindings (currently for Java, C++ and Smalltalk).

The Standard [Chapter 3] defines two object specification languages: Object Definition Language (ODL) and Object Interchange Format (OIF). ODL is used to specify object types that conform to the ODMG object model. OIF is a specification language used to dump and load the current state of an ODMG-compliant ODMS.

In this paper we are especially interested in OIF. XML is fast becoming the standard for data exchange, particularly on the Internet. Rather than use the ODMG's language, we shall show in this paper how to use XML as an object interchange format. We define a new XML document type, OIFML, and show how it can be used to specify ODMG-objects.

2 A brief introduction to XML

XML is a powerful language to describe documents. Documents typically have both structure and content, and XML provides a means for separating one from the other in an electronic document. For example, a memo typically consists of a number of elements: a "from" element, a "to" element, a "subject" element, and finally a "body" element. Here's an example of such a memo, written in XML.

```
<memo>
 <to>     W3C</to>
 <from>   Gavin</from>
 <subject>Names</subject>
 <body>   What about nesting of names in XML?</body>
</memo>
```

The structure in this document is given by the text between the angle brackets, these are called *tags*. Notice that tags always come in pairs. If an XML document does not have matching tag pairs, then it is considered to be *ill-formed*. Tags are sometimes referred to as markup, and sometimes as metadata. The information between the matching tags is known as the *content* of the element.

XML elements are permitted to have additional *attributes* (which we'll refer to as XML-attributes when discussing object databases in the next section). Attribute values are given in the start tag, for example:

```
<person age="21" height="200">Gavin</person>
```

An element may have attributes but no content. XML provides a shorthand for such elements as follows.

```
<security level="classified"/>
```

XML documents can also contain a description of their logical structure, which is called a *document type declaration (DTD)*. This is declared in the beginning (the *prolog*) of an XML document, either directly or by giving a URL where it can be found. The intention is that should a DTD be given, the document is checked to see that it adheres to the DTD (it is then said to be *valid*).

We have now covered most of the features of XML necessary to understand the rest of this paper. Further details on XML can be found on the W3C website[1] or, for example, in the book by Goldfarb and Prescod [2]

3 OIFML

In this section, we demonstrate how to use an XML-based language, OIFML, as a specification language for ODMG-objects. The DTD for OIFML is given in Appendix A. We assume in places that the reader is familiar with ODL (some helpful examples are given in [4]).

This section follows exactly the same structure as the Standard [§3.3]—we consider the same features, give the same examples, and even use the same headings. We hope this will help readers familiar with the ODMG Standard.

3.1 Basic structure

An OIFML file contains a number of object definitions. Its basic structure is as follows.

```
<?xml version="1.0"?>
<!DOCTYPE oif_file
  SYSTEM "http://www.dcs.warwick.ac.uk/˜gmb/oifml.dtd">
<oif_file>
```

[1] http://www.w3.org/TR/REC-xml

```
...

</oif_file>
```

The first line of the prolog simply states which version of XML we use (1.0, for now). The second line states that the contents should adhere to the DTD given at the particular URL. (This DTD is given in Appendix A.)

3.2 Object definitions

The following is a simple example of an object definition in OIFML.

```
<odmg_object oid="Jack">
 <class>Person</class>
</odmg_object>
```

This defines a instance of the class `Person`, with the *unique* identifier `Jack`. Notice that its attributes are left undefined.

3.2.1 Physical clustering

It is possible to specify that when an object is loaded in, it be placed *physically near* another object. (Obviously the notion of nearness is implementation dependent.) Such clustering is specified using an (optional) XML-attribute `proximity`.

For example, the following specifies an instance of the class `Engineer`, with identifier `Paul`, which is to be placed physically near the object with identifier `Jack`.

```
<odmg_object oid="Paul" proximity="Jack">
 <class>Engineer</class>
</odmg_object>
```

3.3 Attribute value initialisation

When specifying an object, an arbitrary subset of its attributes can be initialised explicitly. A tag `contents` is used to specify these attributes. An attribute is represented with the tag `attribute`, which has a compulsory XML-attribute `name`, which gives the name of the attribute. We use the tag `value` to specify the associated value.

For example, assume the following ODL definition.

```
interface Person {
    attribute string Name;
    attribute unsigned short Age;
};
```

The following specifies an instance of the Person class, with the value "Sally" for the attribute Name, and value 11 for the attribute Age.

```
<odmg_object oid="Sally">
 <class>Person</class>
 <contents>
  <attribute name="Name">
   <value><string val="Sally"/></value>
  </attribute>
  <attribute name="Age">
   <value><unsignedshort val="11"/></value>
  </attribute>
 </contents>
</odmg_object>
```

3.3.1 Short initialisation format

We are also permitted to simply list the values, and not specify the attributes.[2] Here the values are assumed to initialise the attributes in the order they appear in the ODL definition. For example, here is our earlier example in such a shortened form.

```
<odmg_object oid="Sally2">
 <class>Person</class>
 <contents>
  <value><string val="Sally"/></value>
  <value><unsignedshort val="11"/></value>
 </contents>
</odmg_object>
```

3.3.2 Copy initialisation format

It is often the case that several objects are to be initialised with the same set of attribute values. A tag shared_value_object is provided for this purpose. For example, the following specifies an instance of the Company class which is physically near the object with identifier McPerth, and initialised with the same attribute values.

```
<odmg_object oid="McBain" proximity="McPerth">
 <class>Company</class>
 <contents>
  <shared_value_object ref="McPerth"/>
```

[2] It is not clear how valuable this facility is in an interchange language. We have included it for completeness.

```
 </contents>
</odmg_object>
```

3.3.3 Boolean literals

We can define a boolean literal using the `bool` tag. This has a (compulsory) XML-attribute, `val`, which takes the values either `true` or `false`.

```
<bool val="true"/>
<bool val="false"/>
```

3.3.4 Character literals

We can define a character literal using the `char` tag. This value is given using the (compulsory) XML-attribute, `val`.

```
<char val="h"/>
<char val="i"/>
```

3.3.5 Integer literals

We provide a number of different tags, corresponding to the different sorts of integers in the ODMG object model. Again the value is given using an XML-attribute, `val`. Here are some examples.

```
<short         val="-3"/>
<long          val="2147483648"/>
<unsignedshort val="3"/>
```

3.3.6 Float literals

We provide the tags `float` and `double` to specify float literals. Here are some examples

```
<float  val="8.88"/>
<double val="10e5"/>
```

3.3.7 String literals

We provide the tag `string` to specify string literals. For example:

```
<string val="hello"/>
```

3.3.8 Initialising attributes of structured types

We allow attributes of structured types to be initialised in OIFML. For example, assume the following ODL definition.

```
struct PhoneNumber{
    unsigned short CountryCode;
    unsigned short AreaCode;
    unsigned short PersonCode;
 };

struct Address{
    string      Street;
    string      City;
    PhoneNumber Phone;
 };

interface Person{
    attribute string  Name
    attribute Address PersonAddress
 };
```

We provide a tag `struct` to specify a structured value. The components of the structured value are specified using the tag `field`. This tag has a (compulsory) XML-attribute called `name`. We use the tag `value` to specify the associated value.

For example, the following specifies an instance of this class `Person`, which initialises some attributes of structured types.

```
<odmg_object oid="Sarah">
 <class>Person</class>
 <contents>
  <attribute name="Name">
   <value><string val="Sarah"/></value>
  </attribute>
  <attribute name="PersonAddress">
   <value><struct>
           <field name="Street">
            <value><string val="Willow Road"/></value>
           </field>
           <field name="City">
            <value><string val="Palo Alto"/></value>
           </field>
           <field name="Phone">
```

```
            <value><struct>
                    <field name="CountryCode">
                     <value><unsignedshort val="1"/></value>
                    </field>
                    <field name="AreaCode">
                     <value><unsignedshort val="415"/></value>
                    </field>
                    <field name="PersonCode">
                     <value><unsignedshort val="1234"/></value>
                    </field>
                   </struct>
            </value>
           </field>
         </struct>
   </value>
  </attribute>
 </contents>
</odmg_object>
```

3.3.9 Initialising multidimensional attributes

An attribute is allowed to have a dimension greater than one—such an attribute is essentially a fixed-size array. For example, assume the following ODL definition.

```
interface Engineer{
    attribute unsigned short PersonID[3];
 };
```

We provide a tag `array` to specify attributes with dimensions. This tag has an XML-attribute `size`, which is used to specify the size of the dimension. The elements of the array are specified using the `element` tag, which has a (compulsory) XML-attribute `index`, which is used to specify which element is being initialised. Any elements which are not specified remain uninitialised.

For example, the following specifies an instance of the class `Engineer`, where the first and third elements of the attribute `PersonID` are initialised (arrays are assumed to be indexed starting from zero).

```
<odmg_object oid="Jane">
 <class>Engineer</class>
 <contents>
  <attribute name="PersonID">
   <value><array size="3">
           <element index="0">
            <value><unsignedshort val="450"/></value>
           </element>
           <element index="2">
            <value><unsignedshort val="270"/></value>
           </element>
          </array>
   </value>
  </attribute>
 </contents>
</odmg_object>
```

We permit a shorthand for specifying that a contiguous sequence of an array is initialised (again starting from zero). For example, assume the following ODL definition.

```
interface Sample{
    attribute unsigned short Values[1000];
 };
```

The following specifies an instance of the class `Sample`, with the first four elements defined (starting with the element indexed at zero).

```
<odmg_object oid="T1">
 <class>Sample</class>
 <contents>
  <attribute name="Values">
   <value><array size="1000">
           <value><unsignedshort val="450"/></value>
           <value><unsignedshort val="23"/></value>
           <value><unsignedshort val="270"/></value>
           <value><unsignedshort val="22"/></value>
          </array>
   </value>
  </attribute>
 </contents>
</odmg_object>
```

3.3.10 Initialising collections

We provide a tag `collection` to enable attributes to be initialised with a collection. This tag has a compulsory XML-attribute, `type`, which takes the value `set`, `bag` or `list`, as appropriate. For example, assume the following ODL definition.

```
interface Professor:Person{
    attribute set<string> Degrees;
 };
```

The following specifies an instance of the class `Professor`, where the collection type attribute (a set) is initialised.

```
<odmg_object oid="Feynman">
 <class>Professor</class>
 <contents>
  <attribute name="Degrees">
```

```
   <value><collection type="set">
           <value><string val="Masters"/></value>
           <value><string val="PhD"/></value>
          </collection>
   </value>
  </attribute>
 </contents>
</odmg_object>
```

We also permit *dynamic* arrays. For example, assume the following ODL definition.

```
struct Point{
    float X;
    float Y;
 };

interface Polygon{
    attribute array<Point> RefPoints;
 };
```

Thus the attribute `RefPoints` contains an array of unspecified size. We can specify such arrays by simply dropping the XML-attribute, `size`, used in the previous section. For example, the following specifies an instance of the `Polygon` class where two of the elements of the dynamic array are initialised.

```
<odmg_object oid="P1">
 <class>Polygon</class>
 <contents>
  <attribute name="RefPoints">
   <value><array>
           <field index="5">
            <value><struct>
                    <field name="X">
                     <value><float val="7.5"/></value>
                    </field>
                    <field name="Y">
                     <value><float val="12.0"/></value>
                    </field>
                   </struct>
            </value>
           </field>
           <field index="11">
            <value><struct>
                    <field name="X">
                     <value><float val="22.5"/></value>
                    </field>
                    <field name="Y">
                     <value><float val="23.0"/></value>
                    </field>
                   </struct>
            </value>
           </field>
          </array>
   </value>
  </attribute>
 </contents>
</odmg_object>
```

It is perfectly acceptable to have fixed-size arrays containing dynamic arrays, as demonstrated by the following example.

```
<odmg_object oid="P2">
 <class>PolygonSet</class>
 <contents>
  <attribute name="PolygonRefPoints">
   <value><array size="10">
           <field index="0">
            <value><array>
                    <field index="0">
                     <value><float val="9.7"/></value>
                    </field>
                    <field index="1">
                     <value><float val="8.98"/></value>
                    </field>
                   </array>
            </value>
           </field>
           <field index="10">
            <value><array>
                    <field index="0">
                     <value><float val="22.0"/></value>
                    </field>
                    <field index="1">
                     <value><float val="60.1"/></value>
                    </field>
                   </array>
            </value>
           </field>
          </array>
   </value>
  </attribute>
 </contents>
</odmg_object>
```

3.4 Link definitions

The following sections describe the OIFML syntax for specifying relationships.

3.4.1 Cardinality “one” relationships

We provide a tag relationship to initialise attribute relationships. This tag has a compulsory XML-attribute, name, which takes the name of the relationship. The link to the object which forms the relationship is given using the link tag.

For example, assume the following ODL definition.

```
interface Person{
    relationship Company  Employer
    inverse      Company::Employees;
 };
```

The following specifies an instance of this class Person, where the relationship Employer is initialised with the object with identifier McPerth.

```
<odmg_object oid="Jock">
 <class>Person</class>
 <contents>
  <relationship name="Employer">
   <link to="McPerth"/>
  </relationship>
 </contents>
</odmg_object>
```

3.4.2 Cardinality “many” relationships

We also allow for relationships with cardinality “many”. We use the same relationship tag as earlier, but we provide a new tag links. This takes a *list* of references to objects, and also has a compulsory XML-attribute, type, to specify whether the relationship forms a set, bag or list. For example, assume the following ODL definition.

```
interface Company{
    relationship set<Person> Employees
    inverse      Person    ::Employer;
 };
```

The following specifies an instance of this class Company, and establishes a relationship, Employees, between this instance and the objects Jack2, Joe, and Jim.

```
<odmg_object oid="McPerth">
 <class>Company</class>
 <contents>
  <relationship name="Employees">
   <links to="Jack2 Joe Jim" type="set"/>
  </relationship>
 </contents>
</odmg_object>
```

4 Conclusions

In this paper we have shown how ODMG-objects can be encoded in a new XML-based language, OIFML. XML is fast establishing itself as the standard for electronic data interchange. It seems prudent therefore to respect this standard when defining means for the interchange of object databases, rather than defining yet another ad-hoc language.

A nice consequence of using the XML standard is that immediately our language, OIFML, is supported by a large number of tools. A wealth of parsers, editors, browsers are already available (For example, the CD-ROM attached to the XML Handbook [2] contains 125 XML software packages!).

As well as providing a means for seeding new object databases, OIFML can be used to seed special semi-structured databases, such as Lore [3], with ODMG-compliant databases.

Acknowledgements

I am grateful to Ken Moody for his comments on an earlier draft.

References

[1] R.G.G. CATTELL ET AL. *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann, 2000.

[2] C.F. GOLFARB AND P. PRESCOD. *The XML Handbook (second edition).* Prentice-Hall International, 2000.

[3] J. MCHUGH, S. ABITEBOUL, R. GOLDMAN, D. QUASS, AND J. WIDOM. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[4] J.D. ULLMAN AND J. WIDOM. *A First Course in Database Systems.* Prentice-Hall International, 1997.

A DTD for OIFML

```
<!-- DTD for Object Interchange Format   -->
<!-- for ODMG compliant object databases -->
<!-- Copyright (c) G.M. Bierman          -->
<!--               Univ. of Warwick      -->
<!--               May 2000              -->

<!ELEMENT oif_file    (odmg_object)*>

<!ELEMENT odmg_object (class, (contents | shared_value_object))>

<!ATTLIST odmg_object oid       ID    #REQUIRED
                      proximity IDREF #IMPLIED>

<!ELEMENT class (CDATA)>

<!ELEMENT contents   (attribute | value | relationship)* >

<!ELEMENT attribute  (value)>
<!ATTLIST attribute   name CDATA #REQUIRED>

<!ELEMENT value (bool | short | long | longlong | unsignedshort | unsignedlong
               | float | double | string | char | collection | struct | array)>

<!ELEMENT bool          (EMPTY)>
<!ATTLIST bool          val (true|false) #REQUIRED>
<!ELEMENT short         (EMPTY)>
<!ATTLIST short         val CDATA #REQUIRED>
<!ELEMENT long          (EMPTY)>
<!ATTLIST long          val CDATA #REQUIRED>
<!ELEMENT longlong      (EMPTY)>
<!ATTLIST longlong      val CDATA #REQUIRED>
<!ELEMENT unsignedshort (EMPTY)>
<!ATTLIST unsignedshort val CDATA #REQUIRED>
<!ELEMENT unsignedlong  (EMPTY)>
<!ATTLIST unsignedlong  val CDATA #REQUIRED>
<!ELEMENT float         (EMPTY)>
<!ATTLIST float         val CDATA #REQUIRED>
<!ELEMENT double        (EMPTY)>
<!ATTLIST double        val CDATA #REQUIRED>
<!ELEMENT string        (EMPTY)>
<!ATTLIST string        val CDATA #REQUIRED>
<!ELEMENT char          (EMPTY)>
<!ATTLIST char          val CDATA #REQUIRED>

<!ELEMENT collection (value)* >
<!ATTLIST collection type (set|bag|list) #REQUIRED>

<!ELEMENT struct (field)+>
<!ELEMENT field  (value)>
<!ATTLIST field   name CDATA #REQUIRED>

<!ELEMENT array  (element* | value*)>
<!ATTLIST array  size CDATA #IMPLIED>
<!ELEMENT element  (value)>
<!ATTLIST element  index CDATA #IMPLIED>

<!ELEMENT shared_value_object (EMPTY)>
<!ATTLIST shared_value_object ref IDREF #REQUIRED>

<!ELEMENT relationship (link | links)>
<!ATTLIST relationship name    IDREF #REQUIRED>
<!ELEMENT link  (EMPTY)>
<!ATTLIST link  to IDREF #REQUIRED>
<!ELEMENT links (EMPTY)>
<!ATTLIST links to IDREFS #REQUIRED  type (set|bag|list) #REQUIRED>
```