CISE Magazine March/April 2002
Geoffrey Fox

XML and the Importance of being an Object?

In the beginning of time, I was a postdoc traveling with decks of cards from job to job and from one vendor's FORTRAN compiler to another. I found it hard to remember the details of what would now be called the meta-data for many jobs and in particular could never remember what the sixth input parameter in my (3I2,A4,2X,E12.4,I6,5F10.4) read statement was! Thus I adopted FORTRAN namelist statements and input parameters with some syntax like:

$ioparm niters="6", couple="2.0" model="6" $end. (1)

Not finding this supported uniformly, I wrote (in FORTRAN of course) my own namelist package. This undoubtedly increased my productivity as I sat through many midnight shifts on the Lawrence Berkeley CDC 6600. Some 25 years later, I diligently supported the same functionality in a multitude of web configuration files but now using Perl and an email-like syntax with attribute name and value separated by colons as in

…….
niters:6
model:6 (2)
couple:2.0
……..

Today XML has swept the world and everybody will write this in a format like:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<ioparm xmlns="http://www.BlahBlah.org/schemas/foxparms.xsd" >
<niters>6</niters> (3)
<couple model="6" >2.0</couple>
</ioparm>
```

So the skeptic can wonder if really the progress in 30 years has been that striking and why XML is greeted with such euphoria especially as it doesn't run on the CDC 6600. Maybe I could use my long practiced skill with overlays to port J2ME (Java for personal digital assistants and other tiny machines) to the small memory of those titans of the past.

However this is day dreaming – we should return to XML. Originally this data structure specification was released with a rather clumsy method (called DTD's) to specify the allowed elements, attributes and other features of an XML instance like (3) above. Here *niters* and *couple* are elements and *model* is an attribute. Equation (3) can be thought of as defining the allowed properties and their relationship for an "instance" of an *ioparm* object. Recently the web consortium W3C has released the important XML Schema specification [http://www.w3.org/XML/Schema] which is a very elegant and

powerful ways of expressing the general object structure of an XML data instance. Schema essentially replace DTD's and use an XML syntax to specify object structure. For those familiar with object oriented languages like Java, schema play a similar role to classes. One can nest attributes and elements in any fashion you like with instances looking like:

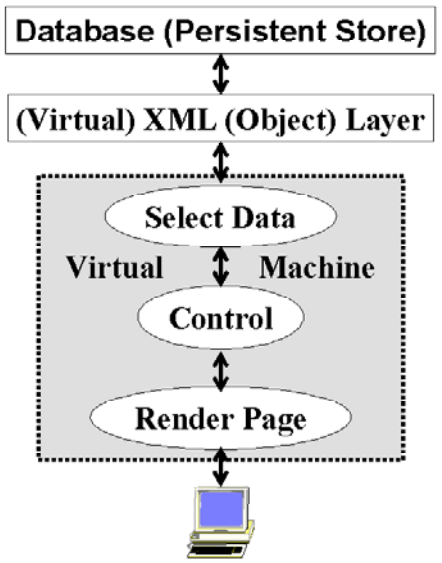
```
…..
<couple model="6" >
<comment>Just a Test<author>Fox</author></comment>
2.0
<units>Fermi**-2<units>
</couple>
```

This emphasizes the difference between XML and namelist or email metadata – XML can specify objects with complex structure; the previous technologies collections of (name,value) pairs.

Now one can argue for ever whether FORTRAN or Java or C++ or Python is a better language and whether object based languages are important. However it appears that the sophisticated data structures allowed by XML are helpful in expressing information in many fields and that the many communities developing XML-based standards are exploiting the object structures allowed by XML. Examples are the learning object standards from IMS and ADL [http://www.imsproject.org], the Geography markup language GML [http://opengis.net/gml/01-029/GML2.html], and XSIL (eXtensible Scientific Interchange Language) developed by Roy Williams at CalTech [http://www.cacr.caltech.edu/SDA/xsil]. From these, we will generate yet more complex objects – for instance Marlon Pierce has extended XSIL to define resources – software and computers -- used in our Gateway computing portal [http://www.gatewayportal.org]. For example the ANSYS installation on a computer Modi4 at NCSA could be denoted:

```
<XSIL Name="Modi4 Type="csm.parseXMLHost">
 <Param Name="HostName">modi4.ncsa.uiuc.edu</Param>
<Param Name="QueueType">LSF</Param>
<Param Name="ExecPath">/usr/apps/fe/bin/ansys57</Param>
<Param Name="WorkDir">/scratch</Param>
<Param Name="QsubPath">/usr/local/bin/bsub</Param>…</XSIL>
```

Now suppose that all data is specified in XML – this has the interesting consequence that all data are objects and these objects are not specified in a traditional language but in a simple XML Schema. In a web or Grid computing application, one will see some sort multi-layer architecture such as that shown on the left. Data at the top is fed through a processing engine (in grey) and rendered onto a client machine. Now XML is neither a natural Fortran binary file and nor is it a bunch of tables as in the relational model. Thus neither SQL (the database access standard) nor Fortran I/O is the natural way to access information



stored in XML. So we represent the challenge by the virtual XML layer shown in the figure. We term this "virtual" because although our standard may be specified in structure by an XML Schema, it is often impractical to represent it as a stream of characters implied by the simple XML syntax. Rather we assume in many cases a different and more efficient representation will be given. This we will discuss in a later article but here we just stress that XML implies both a different way of specifying data structure and a different of accessing Data. This is most naturally with some XML Query syntax and not with SQL or any traditional language input command. The September 11 tragedy highlighted the importance of reliable storage but here we see that future storage will become intelligent – a stream of data-structures with embedded Schema – that implies we need much enhancements of both our storage and our processing capability.

We can consider further implications of XML specified objects – let us suppose your enterprise invests an enormous amount in nifty XML Schema for the corporate data crown jewels and your job is to design software to manipulate it. It would probably be unwise to have separate specifications for the data in XML and the control code – rather we would want the Java or C++ code to automatically generate their data structures from XML Schema. There are now many ways of doing this – we have had good success with Castor [http://castor.exolab.org/] and we can expect substantial new ideas and technology in this area. This then has implications for teaching computing – we should learn the web in elementary school; XML in middle school and control software (Java) for XML objects in high school. Computer languages need to recognize this and separate the specification of information more clearly from its processing.

Now you say that well XML is interesting but its only data – real objects have properties (data) and methods. But what is a method – it is specified by the subroutine/method name and the list of input or output parameters and this is all just "data". In fact, the Gateway system mentioned above -- largely developed by Akarsu, Haupt, Pierce and Youn – has always used XML to specify methods in a fashion like:

```
<interface name="submitJob" extends="BeanContextChild">
    <method return="void" name="test"></method>
    <method return="string" name="execLocalCommand">
        <arg in="string">command</arg>
     </method>
     <method return="string" name="execRemoteCommand">
       <arg in="string">host</arg>
       <arg in="string">user</arg>
       <arg in="string">command</arg>
       <arg in="string">carrier</arg>
     </method>
     <method return="string" name="copyFileFromBackend">
     …….. </method>
     <method return="string" name="copyFileToBackend">
     ……. </method>
</interface>
```

Such specifications can be converted into (remote) method calls and implemented with Java or CORBA. We can hide the particular syntax used for method and the language expression by specifying all interfaces in XML. This idea has been made far more powerful with the Web Services Definition language WSDL which uses XML to specify a single interface to multiple languages and transport protocols

[http://www.w3.org/TR/wsdl]. This has been developed by Microsoft and IBM and we will discuss it later.

So we have come a long way from FORTRAN namelists. XML is very important and data are objects.