Editor: Geoffrey Fox, gcf@indiana.edu

XML AND THE IMPORTANCE OF BEING AN OBJECT

By Geoffrey Fox

IN THE BEGINNING OF TIME, I WAS A POST-DOC TRAVELING WITH DECKS OF CARDS FROM JOB TO JOB AND FROM ONE VENDOR'S FORTRAN COMPILER TO THE NEXT. I FOUND IT HARD to remember the details of what would now be called the metadata 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 a syntax similar to

```
$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 with Perl and an email-like syntax that had attribute name and value separated by colons, as in

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

Today, XML has swept the world, and everybody writes Example 2 like this:

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

XML

The skeptic might wonder if 30 years' progress has really been that striking or why XML is greeted with such euphoria, especially because it doesn't even 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 daydreaming—let's get back to XML.

Originally, this data structure specification was released with a rather clumsy method (called data-type definitions or DTDs) to specify the allowed elements, attributes, and other features of an XML instance. In Example 3, `niters` and `couple` are elements, and `model` is an attribute. Think of Example 3 as defining the values of the allowed properties and their relationship for an "instance" of an `ioparm` object.

Recently, the Web consortium released the XML Schema specification (www.w3.org/XML/Schema), which is an elegant and powerful way of expressing an XML data instance's general object structure. Schema essentially replace DTDs and uses an XML syntax to specify object structure. For those familiar with object-oriented languages such as Java, Schema plays a similar role to classes. You can nest attributes and elements in any fashion you like, with instances that resemble this:

```
.....
<couple model="6" >
<comment>Just a Test<author>Fox</author>
  </comment>
2.0
<units>Fermi**-2<units>
</couple>                                       (4)
```

Example 4 emphasizes the difference between XML and `namelist` or email metadata. XML can specify objects with complex structure, whereas the previous technologies can only specify collections of `(name,value)` pairs.

We can argue forever whether Fortran, Java, C++, or Python is the best language and whether object-based lan-

 1521-9615/02/$17.00 © 2002 IEEE

guages are important. However, the sophisticated data structures XML allows are helpful in expressing information in many fields, and many communities developing XML-based standards are exploiting the object structures XML permits. Examples include the learning object standards from IMS (www.imsproject.org), the Geography Markup Language (http://opengis.net/gml/01-029/GML2.html), and the eXtensible Scientific Interchange Language, developed by Roy Williams at the California Institute of Technology (www.cacr.caltech.edu/SDA/xsil). From these, we can generate even more complex objects—for example, we could denote the Ansys installation on the National Center for Supercomputing Applications' Modi4 computer as

```
<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>                    (5)
```

XML object architecture

Suppose that all data is specified in XML, which has the interesting consequence of turning this data into objects. These objects are not specified in a traditional language but in a simple XML Schema. In a Web or Grid computing application, we would see some sort of multilayer architecture (see Figure 1). XML is neither a natural Fortran binary file, nor is it a bunch of tables, as in a relational model. Thus neither SQL (the database access standard) nor Fortran I/O is the natural way to access information stored in XML. Figure 1 shows this challenge in the virtual XML layer—"virtual" because although an XML Schema could specify the structure of our information, it is often impractical to represent our information as a stream of characters. XML implies both a different way of specifying data structure and a different way of accessing it. Access (search) is most natural with a certain XML query syntax rather than with SQL or any traditional-language input command. Future data storage will have to become intelligent—a stream of data structures with embedded Schema—which means we need improvements of both our storage and processing architectures.

Specifications and methods

Let's look at further implications of XML-specified objects. Suppose your enterprise invests enormously in nifty XML Schema for the corporate data crown jewels, and your job is to design software to manipulate them. Having separate specifications for the data in XML and the control code would probably be unwise; rather, you would want the Java or C++ code to automatically generate data structures from XML Schema. There are many ways to do this—I have had good success with Castor (http://castor.exolab.org), and I expect substantial new ideas and technology in this area. Computer language designers will hopefully recognize this development and separate the specification of information more clearly from its processing. This could have further implications for teaching computing. Students might learn the Web in elementary school, XML in middle school, and control software (Java) for XML objects in high school.

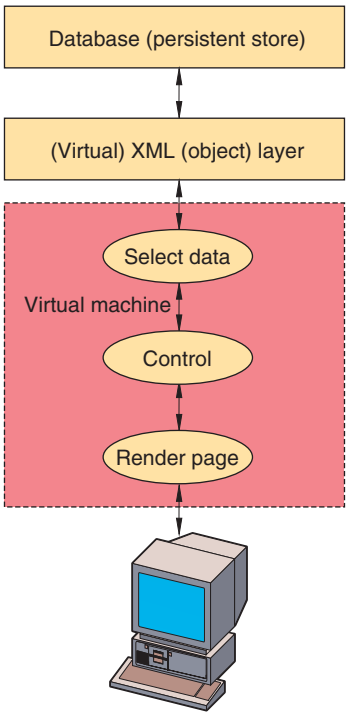

Figure 1. A multilayer XML architecture. Data at the top is fed through a processing engine (in pink) and rendered onto a client machine.

You might say that, well, XML is interesting, but it's only data; real objects have properties (data) and methods. But what is a method? It is just specified by the subroutine–method name and the list of input or output parameters, all of which are just "data." The Gateway portal system (www.gatewayportal.org) has always used XML to specify methods in this manner:

```
<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>                                    (6)
```

Such specifications can be converted into remote method calls and implemented with Java or Corba. We can hide the particular implementation and language used for a method by specifying all interfaces in XML. The Web Services Definition Language, which uses XML to specify a single interface to multiple languages and transport protocols (www.w3.org/TR/wsdl), has made this idea far more powerful.

We've come a long way from Fortran `namelists`.

Geoffrey Fox is director of the Community Grids Lab at Indiana University. He received a PhD in theoretical physics from Cambridge University. Contact him at gcf@indiana.edu; www.communitygrids.iu.edu/IC2.html.