XML: Its Role in TCP/IP Presentation Layer (Layer 6)

Judith Myerson

BREAKING THE BARRIER
Integration between applications should be viewed, at least, as a communication problem. Like the parties on the telephones, the applications should speak the same language to effectively communicate with one another. If the sending application uses one language, and the receiving application speaks in another language, they would need a translator, just like a human counterpart, to understand one another as long as both agree on the semantics and syntax of the language.

Once the applications speak the same language independently or through a translator, there should be a mechanism to support the exchange of messages in a standard way. One such instance is XML, which has become the de facto standard for communication between applications. Using XML allows human agents to send all messages of self-describing texts between applications. This makes it easier for both humans and machines to understand the messages on a common ground. While it nearly achieves the concept of human–machine interaction, it falls short in the packaging of these messages. XML scripts in text format

PAYOFF IDEA
XML, obviously, has not literally been endorsed as the TCP/IP presentation layer, nor is it literally implemented as a layer complete with headers, trailers, PDUs, and all that stuff. While TCP/IP has proven to be a very potent architecture, its usefulness has been limited by the lack of a standardized way of associating meaning with blobs or bits on the wire. XML, however, is much more powerful in other ways when applied to TCP/IP applications. It is positioned to help applications break out and move well beyond that barrier and is creating the potential for networked applications the likes of which one can only begin to dream about, such as product integration, translation for all browsers, dynamic XML servers, XML mapping, natural language dialogue, and even Universal XML.
can be quite a bit larger than their binary representation of the same information.

There are three aspects of inter-application communication:

1. transport: refers to how the information gets across the wire
2. protocol: refers to how to package the information sent across that wire
3. message: refers to the information itself

The transport is usually a lower-level network standard such as TCP/IP. On top of such transports are CORBA, DCE, and DCOM. This means each transport uses a different protocol to communicate: CORBA uses IIOP while electronic mail uses SMTP. One can package, for example, a message, specify a destination, and send the message to the destined location.

While XML does have SOAP as a lightweight XML protocol, it relies on other protocols as well. Using SOAP allows one to use various synchronous and asynchronous mechanisms to send messages based on whatever the appropriate protocol one chooses. However, this protocol seems constrained and has not overcome the barrier imposed by the TCP/IP model scheme. What this means is that some protocols are not as easily extended as XML. Flexibility and extensibility are the norm for XML. They easily lead to standardization of definitions, semantics, schemas, and templates in the exchange of information between products, applications, and vendors.

XML documents contain meta-information about the information being transmitted and can easily be extended. One advantage of using XML is that both humans and computers can read the documents. A disadvantage is that XML is less efficient than transmitting information in binary format. This efficiency is overcome, in part, with the latency of Web-based applications, so the overhead is not as large as it seems. Any protocol can be used to send XML messages.

To send or receive an XML message, one must enable an application to do so — independent of the protocol used. Once one gets applications or organizations to agree on the meanings of XML message, one can send a package of these messages to its intended destination. The meanings must be exact and unambiguous. A data dictionary is suggested to list the XML vocabularies, their meanings, and their associated schemas. These schemas define document structure for specific industries. This allows industry-specific information to be exchanged as XML and would serve as input into the development of XML templates that organizations can use to send and receive messages.

For example, FPML defines an XML schema for the financial industry to exchange information about financial products. ebXML uses an XML schema for the electronic commerce to exchange information in busi-
ness-to-business and business-to-customer scenarios. adXML targets advertising agencies, while CIML (Customer Identity Markup Language) is useful for information on customer relationship management. More than 200 XML schemas for specific industries, groups, and programs are listed at www.xml.org.

These schemas, along with others, are the foundation for building reusable templates of schemas. More templates are on the way as organizations see the merits of standardizing meta-information contained in XML messages and applications.

To get a message to where it is supposed to go, HTTP is a natural choice. As XML documents do not have the ability to listen to a port like HTTP does (port 80), they must be translated into HTML formats — via XSL, for example. This protocol generates HTTP requests/responses as an application (the other example is FTP) at the application layer (the top) of the TCP/IP model.

Once translated, the application moves to the next layer down the TCP/IP road — the presentation layer. Here, this layer formats the data so that it is recognizable or readable by the receiver. It provides services such as encryption, text compression, and reformatting to provide a standardized interface. It is also concerned with the data structures used by programs and therefore negotiates data transfer syntax for the receiving application layer. When the HTTP requests/responses get the data down to the wire, they take a ride on the highway, get off it, and enter the "welcome" door of a receiving host’s TCP/IP tower.

PRODUCT INTEGRATION
One of the great benefits of XML is the ease of integration of products. A good example of the power of XML in this area comes from the CiscoWorks 2000 Service Level Management Solution that determines the impact on various enterprise resources and the degree of success in moving an application or its data from a failed server, node, cluster, or any other network component to an operational one.

By providing XML interfaces via an SDK, Cisco has allowed partner products to integrate more closely into CiscoWorks 2000. Extending the benefits of the Cisco Management Connection, the XML interface allows other products to access Cisco information at the transport and network layers of the OSI model and other information on remaining layers, including the application and presentation layers, and present it to the user under one seamlessly integrated display.

XML integration also allows an XML partner’s product to pass control information and data to CiscoWorks 2000. The interface allows CiscoWorks 2000 to perform actions to a group of routers, for example, thus providing a one-to-many capability for the partner’s product. This is one of the ways XML standardizes the definitions between products — accomplished through the presentation layer.
TRANSLATION FOR ALL BROWSERS

While various tools are available to translate XML documents into HTML using XSL, Microsoft’s XSLISAPI lets users enable XML on all browsers. The transformation occurs entirely on the server and enables a browser to convert XML documents into HTML requests/responses for processing by the presentation layer.

XSLISAPI is a self-extracting executable and one can obtain it from the MSDN Online Downloads Site. It currently works on Windows 2000 Server or Win2K Advanced Server if one installs MSXML 2.4 or higher. It comes with restrictions when it is used on the Windows NT 4.0 Server. XSLISAPI may change significantly to fit into the ASP+ and .NET architecture.

This tool automatically chooses different stylesheets based on a client’s browser. Downloading the file will not complete the installation. One needs to take additional steps:

1. Enter at the command line prompt: regsvr32 xslisapi2.dll.
2. Right-click the Default Web Site node (if right-handed) in the IIS administration Control Panel applet.
3. Select Properties and then the ISAPI Filters tab.
4. Add the new filter to the list. After closing and reopening the Properties dialog box, a green arrow will appear next to the new filter name.
5. Create a new virtual directory — Xslisapi. Do not forget to point this directory to the folder where xslisapi.exe was expanded.
6. Set the Run Script permissions on the virtual directory.

To associate an XSL file with an XML document, do the following in an XML processing instruction:

```xml
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl"
 server-config="sampleA-Config.xml"
 href="sampleA-IE5.xsl" ?>
```

`href` takes precedence over `server-config`. If `href` is not included, the `server-config` attribute points to an XML file in the same folder as the XML document. The XML file contains information about which XSL stylesheet to use for a given browser. One example of a code snippet from a possible `server-config` file is:

```xml
<server-styles-config>
 <device browser="IE" version="5.0">
  <stylesheet href="IE5.xsl"/>
 </device>
 <device browser="Netscape" version="4.5">
  <stylesheet href="NN45.xsl"/>
 </device>
</server-styles-config>
```
From the server-config file, the filter points to NN45.xsl as a XSL stylesheet to use and loads it. Next, the filter transforms the XML code into HTML, as illustrated in the following example:

```csharp
origPath = Request.ServerVariables("HTTP_SSXSLSRMFILE:" unlaw); 
ServDoc.URL = origPath; 
ServDoc.UserAgent = Request.ServerVariables("HTTP_USER_AGENT:" unlaw); 
requestPath = Server.MapPath(origPath); 
ServDoc.Load(requestPath); 
ServDoc.Transform(Response); 
```

**DYNAMIC XML SERVERS**

Not all XML servers are the same. They are generally grouped into repository and dynamic. Repository servers have been around for a while. They hold XML documents — those documents that already have been encoded in XML. Too many, however, can consume enormous resources with the presentation layer on their way over to an XML server and eat up precious disk space with such a server.

Enter dynamic XML servers as a partial solution. Rather than storing the source information as an XML document, they collect information in a traditional data source or in a live application. What this means is that one can pull data out of a traditional database on one server, pass its more streamlined format through the presentation layer, and transform it into an XML document on another server.

One can also use the dynamic server as an XML interface for, say, the existing acquisition tracking component of the much larger E-commerce system. One example function of this interface is to receive XML documents from prospective bidders at a Web site and extract the XML information from those documents. After passing through the presentation layer, the interface stores it in a database, gets it to generate XML documents of another kind, and feeds it to the live acquisition tracking application on the receiving server. The dynamic interface can be accomplished through protocols such as CICS and JavaBeans.

One advantage of using the dynamic XML server is to make publicly available the XML parsers, Java, or a scripting language and make calls to a database given the appropriate access rights and privileges. A disadvantage is that one may not have the expertise to handle the load balancing problems, database connection pooling, and setting the limits that the cache pages and memory can reach. One might be better off with a ready-made dynamic XML server. When considering a server, keep in mind that vendors offer different storage capabilities and methods, espe-
cially when they store and retrieve data from various sources, assign XML
tags, and distribute them to applications.

XML MAPPING
Mapping between XML and relational databases is more complicated
than mapping between XML and objects. There are additional joins in
SQL queries needed to create the XML, while the XML and the objects are
very similar. In parsing XML, one will find a one-to-one relationship be-
tween each object and the XML.

Recognizing the problems regarding XML mapping, Oracle offers a
database with hybrid capabilities that can store XML natively. Its SQL
syntax has been extended with XML Query Language. As they provide a
more natural XML mapping, some products are being marketed as XML
databases created from the ground up (Tamino from Software AG) or re-
designed (eXcelon Corp). While each provides an XML Query Language,
it has not been standardized. The World Wide Web Consortium (W3C) is
currently working on a XML Query Language. By itself, this language will
access XML files as if they were databases.

NATURAL LANGUAGE DIALOGUE
In March 2000, Voice Forum (www.voicexml.org) released VoiceXML 1.0.
Two months later, W3C accepted it as the basis for developing a W3C di-
ologue markup language that could be used to provide voice interfaces
on traditional Interactive Voice Response (IVR) platforms. The initial ver-
sion of the language included support for basic state-based dialogue ca-
pabilities, using a design with simple form-based natural language
capabilities that leaves room to grow as the technology evolves.

While VoiceXML reuses many concepts and designs from HTML, the
differences between visual and voice interactions should be noted. When
an HTML document is fetched from a network resource specified by a
uniform resource identifier, it is presented to the user all at once. A
VoiceXML document, in contrast, contains a number of dialogue units
(menus or forms) presented sequentially — only if the user is talking to
or listening to one other person. This difference is due to the visual me-
dium’s ability to display a number of items in parallel, while the voice
medium is inherently sequential.

The field of spoken interfaces is not nearly as mature as the field of
visual interfaces; thus, standardizing an approach to natural dialogue is
more difficult than designing a standard language for describing visual
interfaces such as HTML. VoiceXML allows applications to give users
some degree of control over the conversation — in a standard way. The
data used in voice interfaces is negotiated in the presentation layer for
transfer to the receiving application.
UNIVERSAL XML
When standards organizations ratify key standards for XML and implement them, one will see a new trend in the market. Within two years, XML is destined to be universally supported such that separate XML products will not be necessary. When standards are in place, the market will offer general-purpose dynamic XML server products. The developers will no longer think in terms of low-level details of XML syntax and semantics when they develop applications. In addition, one may see an extension to the presentation layer based on the powerful capabilities of XML. This extension would be the first attempt in associating the data transfer syntax with bits on the wire.

CONCLUSION
The powerful capabilities of XML when applied to TCP/IP applications appear to be unlimited. Universal XML will help make the move easier beyond the traditional TCP/IP model that has been constrained by lack of a standardized way of associating data with blobs and bits on the wire.

Judith M. Myerson is the author of numerous articles on technology, standards, and protocols. She has published in the fields of database management, application development, network management, server administration, distributed systems, and component-based technology. She can be reached at jmyerson@bellatlantic.net.