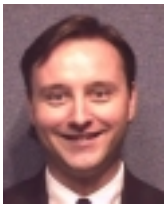
Enabling the Joint Battlespace Infosphere

A Case Study Using XML and Server-Side Processing

Dr. Robert Cherinka, MITRE

Digital information is rapidly becoming integrated into all aspects of military activities, and this presentation highlights work being done for the US Department of Defense. The fundamental requirement addressed by the Joint Battlespace Infosphere (JBI), extracted from the 1998 Scientific Advisory Board report, is "To provide the right information at the right time, disseminated and displayed in the right way, so that decision makers can do the right things at the right time in the right way." The work rests on using emerging web-based technology and tools (XML, XSL, Java, Active Server Pages, and WAP) to provide many of the core services needed to achieve the JBI vision, including user personalization services, publishing services, information transformation services, retrieval and presentation services, and seamless wireless access to information services. Topics to be covered include a brief discussion of the information-centric approach, some background on the JBI, a discussion on using XML to enhance interoperability, and developer details on building an XML-based infosphere.



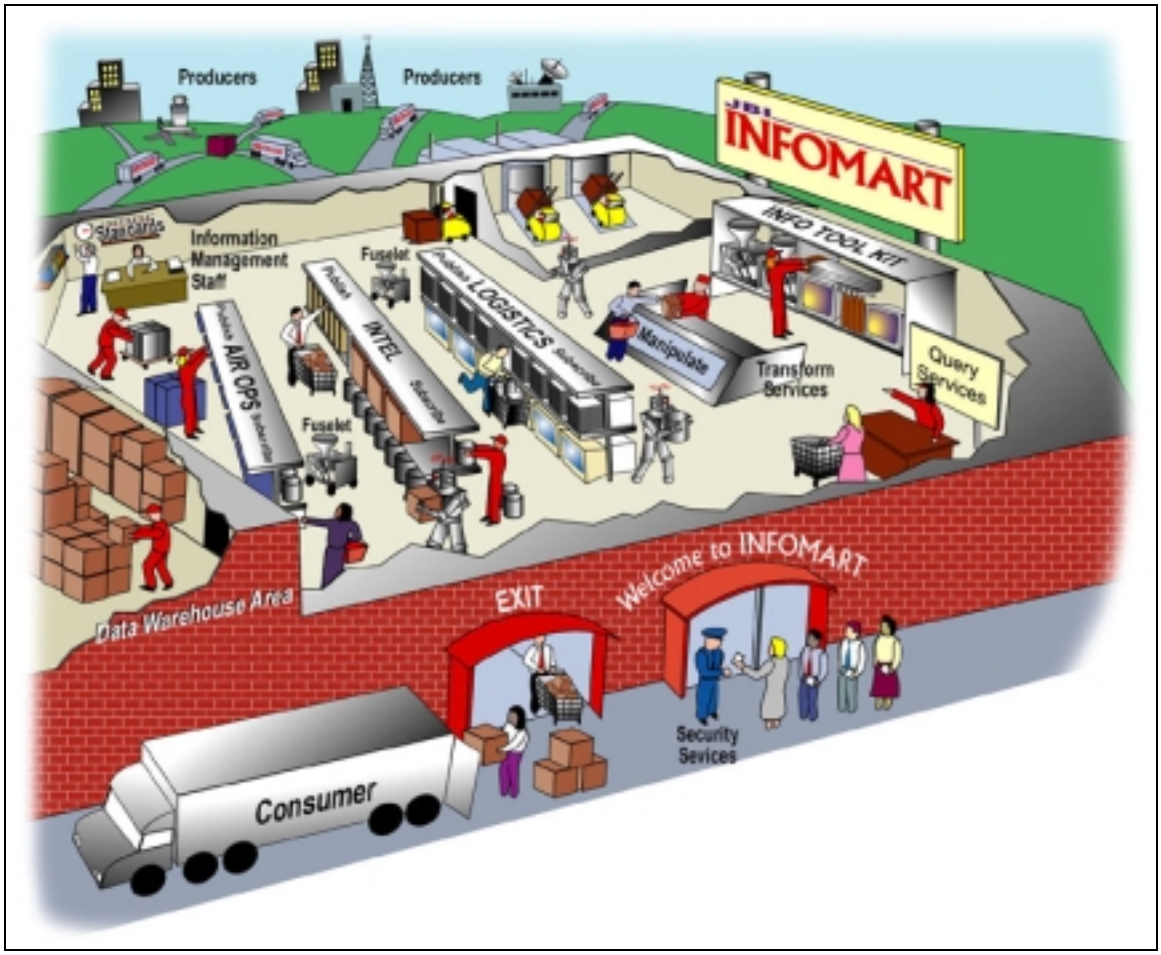
Dr. Robert D. Cherinka is a Lead Information Systems Engineer for the MITRE Corporation, located in Hampton, Virginia. His expertise is in software and process engineering technology. Prior to joining MITRE in 1993, he was an Officer performing software engineering duties in the United States Air Force at the Air Combat Command Computer Support Squadron, Langley Air Force Base, Hampton, Virginia.

Information-Centric Approach

Most previous attempts to "manage" information have been task or system-based. That is, developers made decisions about how to define, organize, manipulate, store, and transport information based on what was optimal for the system under development. What generally results is termed a "stovepipe" system: one that is unable to interoperate effectively with other systems. The real economic result here is increased costs through the inability of systems to work together.

Communities are now recognizing that, for systems to work together effectively, an "information-centric" approach is needed. This approach allows the construction of a loosely coupled network of systems that support a wide variety of information producers and consumers. Interoperability is enhanced through the use of common information objects, such as military Message Text Formats (MTFs) – an Air Base Status Report for example – and mechanisms for their interpretation and understanding. In addition, systems need to be "adaptable" in that they should be able to accept input from older object definitions. It is important to recognize this necessity, as "legacy" systems will still need to communicate with state-of-the-art ones.

The Information Supermarket View



The information-centric approach emphasizes middle tier services. This middle tier represents a capability that brings together all of the information necessary to support the consumers and their missions. To make another analogy, this layer functions as an "information supermarket"

or "infomart", as shown above. Although essentially distributed, the infomart provides one-stop information shopping for consumers and a one-stop information outlet for producers. The goods and services available for exchange are all information-based products.

The infomart is distinct in organization, process, and usage from the communications infrastructure on which it rides and the user application systems which it serves. It is a place where information is brought together under coordinated management, independent of the management of communications and systems.

The infomart must rely on an underlying transport network – the infrastructure through which information is stored and transported. This distributed infrastructure includes telecommunication networks, servers, routers, and the like. The purpose of the infomart is to provide a virtual, centralized market for the exchange of information and service commodities by application systems. These systems include both producers and consumers of these information commodities, some of which do both functions simultaneously.

The infomart should supply certain core information-oriented services through its operation. For example, it should contain or support processing functions to allow for storage, access, retrieval, fusion, and management of information. This is similar to a food supermarket that supplies rows of foodstuffs on shelves (storage and indexing) for easy shopping (access and retrieval) as well as delicatessen services to repackage (filtering and fusing) luncheon meats and cheeses.

The commodities and services of the infomart are based on commonly used information units called information objects. While there is standardization in such an approach, it does not imply that there is only one kind of information object for a particular function, just as we would not expect to find only one kind of soup at the supermarket. Information objects are based on the needs of decision makers and decision aids in various functional domains. Core services (for example, query) and technologies (for example, schemas) are provided to allow application systems to interpret the meaning of these objects and effectively use them. An underlying theme of this vision is the exploitation of emergent web-based commercial off-the-shelf (COTS) technology and tools – such as those mentioned later in the commercial alternatives section – to provide many of these core services, with the accompanying benefits of reduced costs and scalability.

Fundamental to the success of the infomart mission is a universal method for information producers and consumers to share information commodities. Information producers place their information products into the infomart by "publishing" while information consumers receive information products available through the infomart by "subscribing". (Consumers also can receive information products through "query", discussed below.) In general, publishing and subscribing occur through (standardized) information objects rendered via a common representation as discussed above. (It is worth noting that this does not mean there is only one way of producing an information object of given content.) Information producers must be capable of providing their information products via the common representation, and information consumers must be capable of reading such standardized products. Thus producers and consumers must add an interface in order to be able to use the infomart and its services.

Commercial technologies and tools are now becoming available that address these issues. These are actually cheaper than requiring each consumer to devise his or her own interface to each producer's information format. Of course, producers (in particular) and consumers are free to continue to use their proprietary information formats in pre-negotiated, point-to-point information transfers. However, doing so will prevent wider distribution of available information and will not address interoperability concerns.

Information consumers typically find ways to use and combine information in which producers never considered. Consequently, information consumers require the capability to locate and extract only the information they require. It must also be possible for consumers to manipulate retrieved information into more meaningful forms, say to match their internal system representation or to combine it with other information. Consumers will search for and retrieve the necessary information through the infomart's query services. In addition, commercial technologies are now available to assist consumers to reformat and translate retrieved information into a user-specified organization.

Finally, just as a regular supermarket requires a staff to manage and run its operation, the complexity of the infomart requires a trained professional information management staff to keep it operating smoothly. Staff are needed to manage the informational content. For example, the staff might decide which categories of information are required to have on hand (warehouse storage) versus which information would be obtained by links to other available sources based on the particular contingency being addressed. The infomart's management staff also would interact with the communications management staff to identify information bottlenecks and possibly to request restructuring of communications capability to keep the information flowing. The infomart's management staff would address the needs of the user applications that interact with the infomart, solving information related issues such as access privileges, security concerns, information location, information translation, troubleshooting, and so on that may not be handled properly or automatically by its infrastructure.

Joint Battlespace Infosphere

Digital information is rapidly becoming integrated into all aspects of military activity. As a result, operations are becoming increasingly fast-paced and diverse as the management of information moves closer to the center of military power. To provide commanders with the knowledge required to make decisions in this environment, a greatly enhanced command and control (C2) concept for intelligence gathering, dissemination, and visualization is needed, based on revolutionary new information age concepts and technologies.

The Battlespace Infosphere (BI), as described in the December 1998 Scientific Advisory Board (SAB) report "Information Management to Support the Warrior", has been proposed as a vehicle for realizing this goal. More recently, the SAB has referred to this concept as the Joint BI (JBI). The fundamental requirement addressed by the BI, as extracted from the 1998 SAB report, is "To provide the right information at the right time, disseminated and displayed in the right way, so that commanders (and "crew warfighters") can do the right things at the right time in the right way." Included as an objective is to do all this "faster than the other guy, thereby ensuring information superiority." In other words, the JBI could be seen as an infomart.

Using XML to Enhance Interoperability

In 1997, the World Wide Web Consortium (W3C) endorsed the eXtensible Markup Language (XML) as a future standard for data interchange. For the first time ever, we have an internet standard for transmitting data, along with its interpretation criteria, that is platform independent, programming language neutral, both machine and man readable, and is emerging as an industry standard.

The United States Department of Defense (DoD) recognizes the potential XML may have to enhance interoperability and information exchange among Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR) systems. Many initiatives are now underway within the DoD to use XML for this purpose. MITRE, a US government-owned non-profit research and development corporation, has been instrumental in helping the DoD realize the potential of this new family of technologies. In particular, this realization is comprised of three key ingredients:

- XML: An international industry standard for describing and sharing structured information with widespread support by mainstream COTS venders and products
- XML-MTF: An XML Message Text Format application supporting military information exchange requirements, operational procedures, and terminology across Joint and Allied boundaries
- XML-JBI: An enterprise portal for publishing, subscribing, transforming and personalizing information objects

XML

The W3C is attempting to rapidly evolve common specifications for the Web's information space so that organizations from many diverse fields can exploit and build upon it. An area of intense activity is the W3C's work on XML, which began in 1996. Designed to meet the challenges of large-scale electronic publishing, XML is playing an increasingly important role in the exchange of a wide variety of data on the Web.

XML is a data markup language like HTML, using bracketed tags to describe the treatment of the data in the document. With HTML, the tags primarily relate to the formatting and display of the text. With XML, the tag repertoire is extensible; anyone can define a tag to describe some attribute of the text. If applications agree on their use of these extended tag definitions, then they are able to understand the context of the text exchanged between them.

XML-MTF

To address the C2 information interoperability problem, the US and its allies have invested a great deal of time and resources formalizing information standards to reduce the ambiguity of natural language and increase opportunities for automation. One of the most prolific of these standards is the Message Text Format standard, which was begun over 20 years ago. Today, the MTF program governs a significant portion of all structured text information exchanged to support the full spectrum of military operations, including intelligence, air operations, fire support, maritime operations, logistics, medical, etc. It includes an actively managed repository of Information Exchange Requirements (IERs) describing over 600 information products drawing on over 7000 standard data element definitions.

The United States Air Force and MITRE are leading an initiative, called XML-MTF, to drastically improve the quality, capability, and affordability of MTFs. This initiative promotes the adoption of XML to make MTF information available in a critically important industry standard format. Organizations who share a common terminology typically refer to "vocabularies" when describing common "element" or "tag" names in XML. In this case, the MTFs represent this common vocabulary. The XML-MTF initiative capitalizes on the military's investment in IERs and leverages industry standards to improve our ability to find, retrieve, process, and exchange large amounts of information easily across system, organizational, and format boundaries (that is, the right information at the right time in the right format). In addition, it enables us to use low cost, high quality, rapidly evolving, mainstream commercial software for processing military information. The following highlights the XML-MTF effort:

- An initiative to modernize military information standards through commercial technologies
- Capitalizes on 20+ year investment in military information requirements from MTF standard
- Leverages industry standard XML format
- Defines a standard XML mapping for MTF messages

XML-JBI

MITRE is currently working on a prototype to construct and demonstrate what may be thought of as an initial component (block) of the JBI capability. The work rests on using emerging web-based COTS technology and tools to provide many of the core services needed to achieve the JBI vision. In particular, it exploits XML in direct support of common core services, including:

- User Personalization Services
  - Provide user authentication
  - Create and manage user profiles ("my XML-JBI")
- Publishing Services
  - Upload text, XML, XSL documents
- Information Transformation Services
  - Convert to/from XML, HTML, Text, database sources
  - Perform batch processing of static views
  - Perform subscription services
- Retrieval and Presentation Services
  - Download text, XML, XSL documents
  - Provide static "tailored" views
  - Provide dynamic views (for example, queries, filters)
  - Support the creation of new views

Up to now, I have discussed the theory of the XML infosphere concept and its specific use for the JBI. The next section will provide some details for developing an XML infosphere.

Building an XML Infosphere: a Developer's View

This section will attempt to provide developers with some insight into building a similar infosphere approach for their own use. There are a number of technologies used for this and it is assumed that the reader will have a working knowledge of XML, XSL, Java, Visual Basic (VB) and Active Server Pages (ASP). In order to follow the examples discussed here, the reader should make use of the supporting code that accompanies this article. The included source represents a basic set of generic capabilities for implementing the infosphere concept without including any of the proprietary military components. However, it should be sufficient for demonstration purposes.

Setting up the XML Infosphere Software

The following steps will aid the user in setting up the example XML Infosphere software that is presented. It is important to note that several off-the-shelf tools, such as the Java Development Kit, need to be downloaded and installed separately. These tools are listed below.

I. Set up a server with NT 4.0 Server and SP6.0. Install IIS, ASP, Access98, and Outlook98. (Note: the software has been successfully set up on Windows 2000 Server with Access and Outlook 2000.)

II. Download and install JRun 2.3.3, a Java servlet engine by Allaire. (Note: there is a demo version which allows five users to be connected at one time that you can download from their website, www.allaire.com. Although suitable for a development environment, it is not intended for a production environment.) Note that JRun is very unhappy if there are spaces in the `CLASSPATH`. It is best to avoid installing this software in directories like "`Program Files`". The JRun installer requires that you select a Java Virtual Machine (JVM). Use the JVM that comes with JRun.

III. Download and unzip LotusXSL from www.alphaworks.ibm.com/tech/LotusXSL. Note that LotusXSL is very unhappy if there are spaces in the CLASSPATH. It is best to avoid installing this development software in directories like "Program Files".

IV. Download and install the Java 2 Development Kit, known as JDK 1.2 or SDK 1.2 from java.sun.com/products/jdk/1.2/.

V. Download and unzip Rhino 1.4 Release 3 (do not get Rhino 1.5 Release 1 or newer; there is an incompatibility with LotusXSL) from www.mozilla.org/rhino/.

VI. Download and install the Adobe SVG Viewer plug-in from www.adobe.com/svg/. This plug-in is used to provide SVG viewing capabilities within a browser.

VII. Unzip the contents of the xml-jbi.zip. This zip file contains five other zip files, which should be placed on the desktop (or in some other temporary directory). The contents of these five zip files go to different places.

A. xmljbi.zip (not to be confused with xml_jbi.zip) should be unzipped into the c:\ directory, resulting in a c:\xmljbi directory. This is the exact directory that it must appear in. It contains the following items:

1. Membership1.mdb, which is the database used to store login user account and subscription information. In order for logins to work, a DSN must be created. Using the ODBC control panel, create a system DSN pointing to the location of this database. Name the DSN "Membership". In this database is a table called "Members". This table should be empty; if it is not, delete the contents of the table (do not delete the table itself).
2. users folder, which eventually contains a directory for each user. If this folder does not exist, create an empty folder called "users".
3. xmill folder, which contains the xmill win32 executables for compressing and decompressing XML files.
4. config.xml, which contains configuration variables used by xml-jbi. The contents of this file should be modified appropriately. Generally this file should only be edited if you attempt to change the default location of some parts of the xml-jbi software.

B. xml_jbi.zip (not to be confused with xmljbi.zip) should be unzipped into the c:\inetpub\wwwroot directory, resulting in a c:\inetpub\wwwroot\xml_jbi directory. This website contains the ASP pages which provide the GUI, the XML/XSL Repositories, and various other scripts. The directory structure is divided into core (the core services) and data (the data-specific services). Note that in the data directory there are several sample data types. Delete any data type you are not interested in. If you delete a sample data type, it would be a good idea to comment out the reference to this data type in the c:\inetpub\wwwroot\xml_jbi\core\menubar.asp file. Look for the HTML comment that says "comment out unused data types". For any data type that you do not delete, make sure that the c:\inetpub\wwwroot\xml_jbi\data\<name of data type>\backup\invalid directory exists. If it does not exist, create it.

C. Copyit_install.zip should be unzipped into any temporary directory, if you are establishing an email interface into xml-info. Run the setup.exe file and install CopyIT! into c:\program files\CopyIT.

D. Eventhandler_install.zip should be unzipped into any temporary directory. Run *all* of the setup.exe files (setup1.exe, setup2.exe, setup3.exe, etc.) and install all of the event handlers into c:\xmljbi\handlers. The event handlers provide the publishing services for the xml_jbi. The XML_JBI_Dispatcher.exe is the main service which monitors the inboxes, and needs to be running in the system tray in order for the publishing services to work. A shortcut to run this program at startup should be placed in the startup folder – or the program may be run manually each time the server is rebooted.

E. servlets.zip should be unzipped into the c:\Jrun\servlets directory (the xml-jbi was developed using JRun and the "Jrun\servlets" structure is part of JRun). The files in servlets.zip contain the various Java servlets used for server-side XML/XSL processing. The servlet must be given an alias with JRun and an "init" argument for each data type that utilizes a file upload – use the following procedure. To create an alias to the servlet, start the JRun Administrator. On the Services tab, select the Service ID jse. Click the Service Config button. Click the Aliases tag. Click the Add button. Set the Name, the Class Name, and the Init Argument Name and Value (see table below for specific values). Check Pre-load. Repeat for each alias. Save everything. This alias allows the FileUploadServlet to be called using the name fileupload and uploaded files are placed into the inbox directory of the data type. For example:

Name	fileupload_usmtf_1998	fileupload_tam
Class Name	com.livesoftware.servlets.spack1.FileUploadServlet	com.livesoftware.servlets.spack1.FileUploadServlet
Initial Argument Name	uploaddir	uploaddir
Initial Argument Value	c:\inetpub\wwwroot\xml_jbi\data\usmtf_1998\inbox	c:\inetpub\wwwroot\xml_jbi\data\tam\inbox

VIII. Make sure your system CLASSPATH contains the following jar files:
- A. JDK1.2.2 – src.jar, lib\tools.jar, lib\dt.jar.
- B. LotusXSL – lotusxsl.jar, bsf.jar, bsfengines.jar, xalan.jar, xerces.jar.
- C. JRun – lib\servlet.jar, lib\jrun.jar.
- D. Rhino – js.jar, jstools.jar.

IX. JRun has its own CLASSPATH variable, it does not use the system CLASSPATH used by the JDK. The JRun CLASSPATH can be found in the JRun Administrator; select the General tab and then the Java tab. Set it up by adding all the jars mentioned in step VIII. Be aware that JRun bundles several jar files with its software; these are stored in the lib directory. If the JRun CLASSPATH references a jar file in the lib directory that is mentioned in step VIII, delete the existing reference and replace it with the new reference. The JRun CLASSPATH should not reference multiple jar files of the same name, and the ones in the lib directory are likely to be older than ones downloaded directly from the Internet.

To run the system, the server will have to be running both the XML-JBI Dispatcher and CopyIT programs. CopyIT responds to incoming mail messages and transfers files to XML-JBI inboxes. The Dispatcher senses when a file has been placed in one of the XML-JBI inboxes and invokes an appropriate event handler to process it. The two programs can be launched from their shortcuts on the Windows Start menu. These programs need to be running to enable XML-JBI to respond to publish/subscribe requests.

Reboot your machine. Start the XML-JBI Dispatcher if you did not place a shortcut to it in your Startup folder. Point your web browser to http://www.yourserver.com/xml_jbi/default.asp. Create an account for yourself and login.

XML-JBI Infrastructure

Within the XML_JBI web hierarchy, there is a distinct separation between core and data services. The core services represent the ASP pages to interact with the generic user

authentication, profiles, and subscription information. In the core\login folder, login.asp contains the form for logging in users. Within login.asp, the user information is stored and accessed in the MS Access database, membership1.mdb, that can be found in the installed xmljbi folder. The core\profile folder contains the ASP files for maintaining user accounts as well as the subscription profiles for each user, again interacting with the membership1.mdb. Once a user has been authenticated, each ASP contains the following statements to ensure security across all web pages in the xml_jbi:

```
<%
if Session("LoggedIn") = False then
   Response.Redirect("default.asp")
end if
%>
```

The data folders are where the information objects that have been agreed to are stored. There is a template folder in the data folder that illustrates the basic set of capabilities and folders that should be available for each new information object. It is also important to note that, for a given information object type, there can be any number of sub-types. For example, a "USMTF_1998" information object type can have individual message types (such as "ATO" or "ACO") that comprise this overall type. The following folders are used:

- Inbox – this is where new information of this type would get published
- XML Repository – XML (converted or published) information is stored here
- XSL Repository – published XSL transformation scripts are stored here
- Scripts – advanced applications and scripts for this type are stored here
- Backup – copies of published files are archived here
- Views – server-generated static or batch views are stored here
- Outbox – the location for file-based download of resulting transformations

An interesting aspect to the way we implemented the xml_jbi is that we use an XML document, called config.xml, to store important information (such as path and environment variables) necessary for many of the tools used. An example file is:

```
<?xml version="1.0"?>
<xmljbi_config>
 <user_root>c:\xmljbi\users</user_root>
 <cmp_home>c:\xmljbi\mtf2xml</cmp_home>
 <jbi_home>c:\inetpub\wwwroot\xml_jbi\data\usmtf_1998</jbi_home>
 <data_home>c:\inetpub\wwwroot\xml_jbi\data</data_home>
 <tam_home>c:\inetpub\wwwroot\xml_jbi\data\tam</tam_home>
 <dtab_home>1998</dtab_home>
 <config_home>c:\xmljbi</config_home>
 <profile_dir>c:\InetPub\wwwroot\xml_jbi\core\profile\
    subscribe_xsl</profile_dir>
 <sleep_msec>1000</sleep_msec>
 <doc_dirs>
 <doc_dir>c:\inetpub\wwwroot\xml_jbi\data\usmtf_1998</doc_dir>
    <doc_dir>c:\inetpub\wwwroot\xml_jbi\data\gccs_cop</doc_dir>
    <doc_dir>c:\inetpub\wwwroot\xml_jbi\data\webcop</doc_dir>
 </doc_dirs>
 <mtf2xml_dir>c:\xmljbi\mtf2xml</mtf2xml_dir>
 <handler_dir>c:\xmljbi\handlers</handler_dir>
 <xmill_dir>c:\xmljbi\xmill</xmill_dir>
 <smtp_host>lang06.mitre.org</smtp_host>
 <email_from>XML-TEST@lang06.mitre.org</email_from>
 <email_display_name>XML-JBI</email_display_name>
```

```
 <!-- mp_common.jar, jmps.jar, and environment.dll must be in the
    cmp_home directory. This config file must be in c:\xmjbi. -->
</xmljbi_config>
```

This file is subsequently used from within many of the ASP files, servlets, XSL stylesheets, and the EventHandler VB programs. This provides a nice way to minimize the configuration management of this information as users only have to change this in one place.

XML-JBI User Personalization Services

Personalization services are provided by a number of services. To support personalization, membership1.mdb is used to store the user account and subscription information using two tables. For user authentication, the core login.asp and logoff.asp files discussed above handle user access by checking that the account is valid in the database. The profile asp files handle the creation and modification of user accounts that include profile information about each user. For example, the following code is used in several of the profile ASP files to access the user profile from the database:

```
<%
' Edit Profile
' This page lets the user edit the information about themselves
' stored in the database
On Error resume next
Set MembershipConn = Server.CreateObject("ADODB.Connection")
MembershipConn.Open "Membership"

'Pull their record from the database.
' create the SQL string
strSQL = "SELECT * FROM Members "
strSQL = strSQL & "WHERE UserName = '" & Session("UserName") & "'"

' execute the SQL command
Set rsResults = MembershipConn.Execute(strSQL)
If Err.Number > 0 Then
  HandleRuntimeError("Err Executing Query")
End if
' see if there are any records returned
If rsResults.EOF Then
  ' Couldn't find their record.  This is bad.
  HandleRuntimeError("EOF reached.")
Else %>

<!-- Display the form and fill it with the results from the query
-->
```

Once a user account and profile has been generated, and the user has logged on, they will be able to register any subscriptions. Subscriptions allow information consumers to state the information objects, information object sub-types, XSL transformations, and the mechanism for sending the results of the transformations or notifications to the user any time the information object is published. For example, a user can subscribe to information that states whenever a USMTF_1998 ATO XML information object is published, apply the XSL file that transforms it to an HTML view of a mission time line, and place the server-generated view in the views folder for the ATO so that it is ready for the user to view the next time he/she goes to the web site. Or, the user could have the resulting file emailed to them. This is discussed later. The subscribe.asp file in the profile folder handles the collection and manipulation of this subscription information to the database.

XML-JBI Publishing Services

Publishing of information is done via a number of ways, such as file, web, e-mail, and sockets. File-based publishing allows a user to place the information object into the inbox using FTP, Web Folders, or any other basic file access to the inbox folder. This is straightforward.

Web-based publishing is implemented using an upload.asp file that communicates with a Java servlet, as configured in the JRun setup instructions discussed previously. The code for this action is given below for the MTF upload servlet:

```
<%
if Session ("LoggedIn") = False then
  Response.Redirect("default.asp")
end if
%>

<HTML>
<HEAD>
<TITLE>XML-JBI File Upload</TITLE>
</HEAD>

<body>

<H1>XML-JBI File Upload</H1>

<form action="/servlet/MTF_Upload" enctype="multipart/form-data"
method="POST">
What file are you sending? <input type="file" name="filename">
<BR><BR>
<input type="hidden" name="postto"
value="/servlet/com.livesoftware.servlets.spack1.FileUploadResultHandler">
<center><input type="submit" value="Send"></center>
</form>

</BODY>
</HTML>
```

The servlet allows a user to pick a file on their computer through an explorer view available from the web page, and then upload that file to the appropriate inbox for the information object type. This is also a straightforward technique.

For e-mail publishing, the included CopyIt code provides a way for administrators to configure message rules to any POP3 SMTP e-mail server through MS Outlook, which must be installed on the server where the CopyIt program will run (not necessarily the POP3 server). At a user-configurable interval, CopyIT! accesses a mailbox through the Outlook 98 client. If an e-mail is found in the Inbox, CopyIT! compares the publisher's SMTP address, the message subject, and attachment extensions to a set of user definable rules. If a match is found between a rule and corresponding message attributes, the message is processed in accordance with the rule. Once all of the rules have been set up and the CopyIt program is running on the server, then users can email information objects either in attachments using the correct file extension or in the body of the message. CopyIt will then decide which *inbox* it needs to go to and will place it there. The included source code illustrates this capability.

XML-JBI Transformation Services

Once an information object has been published to the *inbox* for that information object, there may be a number of transformations that need to be performed on that object. At a minimum, if non-XML files have been published and a given transformation tool has been identified, then

a conversion to XML needs to be performed. The USMTF_1998 military messages that may be published as ASCII text are a good example of this. The XML-MTF tool would be used to convert them to XML. Alternatively, users can publish in XML directly. Ideally, the schemas for the XML information object type are accessible to validate the XML document. In this prototype, we assume that the user has validated the file already. Once an XML file is published or converted, it is then moved to the `xml_repository` folder for that information object type. A copy of the file that was published (as-is) is also moved to the appropriate backup folder for archive. At this point, the user subscription profiles in the `membership1.mdb` file need to be accessed to determine if there are any batch processing actions to be accomplished. If there are, then, for each subscription, the appropriate XSL transformation needs to occur and the resulting output will be sent to the appropriate location, whether that be the outbox or views folder, or sent to the user via the email or socket interface.

The example Visual Basic program that monitors all of the *inbox* folders and decides when something has been published is the dispatcher, which must be running on the xml-jbi server.

```
Sub MonitorInbox()
    'Main routine for monitoring inbox directories and invoking event handlers
    Dim sDirString As String    'Directory where the config.xml file resides
    'For the present, this, and other executables, must be here also.
    Dim sLongString As String   'Gets the entire contents (text) of config.xml
    Dim sFileName As String
    Dim sDocType As String
    Dim sBase As String
    Dim sHandlerDirSpec As String
    Dim sData_Home As String, sInbox As String, sBackupDir As String, _
        sTrashDir As String
    Dim i As Integer, j As Integer, vTemp As Variant, sTemp As String
    'Array of document types and base directories for the inboxes to monitor
    Dim sBaseInfo() As String
    'Objects for accessing the file system
    Dim oFileSystem As Object   'File System Object
    Dim oFile As Object         'File Object

    bBailout = False: sChildProcess = "
    'Get configuration parameters from config.xml file
    'First ascertain where the config.xml file is hiding
    sDirString = GetUserString("XMLConfigFileDir", _
                               "Directory where configuration file resides: ")
    If Len(sDirString) = 0 Then Stop
    sDirString = Trim(sDirString)
    If Right(sDirString, 1) <> "\" Then sDirString = sDirString & "\"
    'Use this occasion to fire up the system tray code
    Call DispatcherForm.Systray
    'Get configuration file contents
    sLongString = ReadConfigData(sDirString & "config.xml")
    'Here's where we extract whatever configuration info we need
    vTemp = GetMultipleConfigStrings(sLongString, "doc_dir")
    If Not IsArray(vTemp) Then Stop
    sBaseInfo() = vTemp
    sHandlerDirSpec = GetSingleConfigString(sLongString, "handler_dir") & "\"
    iSleeptime = 1000   'Default
    sTemp = GetSingleConfigString(sLongString, "sleep_msec")
    If sTemp <> " Then iSleeptime = CInt(sTemp)


    Do  'Do Forever
        'Check the inbox(es) for files; if one is found, invoke appropriate
        'handler, passing it a pointer to the file.
        For j = 1 To UBound(sBaseInfo, 2)
            sBase = sBaseInfo(2, j)
            sInbox = sBase & "\" & "inbox\"
            sBackupDir = sBase & "\" & "backup\"
            sTrashDir = sBackupDir & "invalid\"
            sDocType = sBaseInfo(1, j)
            sFileName = Dir$(sInbox & "*.*")
            Do While sFileName <> "
                If bBailout Then Exit Do
                sHandlerName = CheckThisFile((sInbox & sFileName), sDocType)
```

```
                If sHandlerName <> " Then
                    vTemp = SpawnAndWait((sHandlerDirSpec & sHandlerName), _
                            sInbox, sFileName, sDirString & "config.xml")
                    'Gets back to here only when handler is done processing the file
                    'or if handler wasn't found
                    If vTemp <> " Then
                        MsgBox "Handler " & CStr(vTemp) & " not found. " &_
                               "Skipping file: " & sFileName & "!"
                        Fecalize sInbox, sFileName, sTrashDir
                    End If
                Else
                    Fecalize sInbox, sFileName, sTrashDir
                End If
                sFileName = Dir$    'Next file in this inbox
            Loop
            If bBailout Then Exit For
        Next
        'To here after having processed all the inboxes
        'Sleep for a while and then do it all again.
        DoEvents
        If bBailout Then Exit Do
        Sleep (iSleeptime)
        DoEvents
        If bBailout Then Exit Do
    Loop

    Set oFile = Nothing
    Set oFileSystem = Nothing
End Sub

Function CheckThisFile(sFileSpec As String, sDocType As String) As String
    'Function to examine file found in an inbox and determine whether it's
    'a valid member of the document type
    Dim sReturn As String, sExt As String

    'Originally, this routine examined the file extension.
    'Now, it's really a null routine,
    'just returning the handler program name as a function of which inbox
    'the file appeared in.
    sExt = Right(sFileSpec, Len(sFileSpec) - InStrRev(sFileSpec, "."))
    'DEBUG NOTE: For JWID we must have three document types: "usmtf_1998",
    '"gccs_cop" and "webcop".
    sReturn = "
    Select Case sDocType
        Case "usmtf_1998"
            sReturn = "usmtf_1998.exe"
            'DEBUG: For testing, if "twing" then return an invalid handler name
            If UCase(sExt) = "TWING" Then
                sReturn = "twingnuts.exe"
            End If
        Case "gccs_cop"
            sReturn = "gccs_cop.exe"
        Case "webcop"
            sReturn = "webcop.exe"
    End Select
    'MsgBox sReturn
    CheckThisFile = sReturn
End Function
```

The code above illustrates the monitor inbox routine that gets the paths of the *inbox* folders from the `config.xml` file discussed earlier. It then continues to monitor for information to be published and eventually calls the `CheckThisFile` routine to determine which event handler to call. There are individual event handlers for each information object type, which would then process the information object as described above. The following code snippet shows how we call the LotusXSL XSLT processor from Visual Basic to perform the requested transformations.

```
'SUBSCRIPTION SECTION
    'Get subscription information from the Access database.

    Call RunLotusXSL(sMTF2XMLDirSpec & "working\", _
```

```
                        sProfileDirSpec, sXML_FileName, _
                        "planned_asset_location_as_xml.xsl", _
                        "planned_asset_location_as_xml.xml")
```

The above code illustrates the call to the RunLotusXSL procedure passing the information object (XML document), XSL file, and output filename as parameters. The routine below then uses the file system object to dynamically create a batch file containing the appropriate Java command line call to LotusXSL. This is done to avoid problems with the maximum command line size of MS-DOS when passing parameters to a batch file through the shell command in Visual Basic.

```
Sub RunLotusXSL(sWorkingDirSpec As String, sXSLDirSpec As String, _
                infile As String, xslfile As String, outfile As String)
    Dim sCurDir As String, sCurDrive As String
    Dim sTargetDrive As String
    Dim sFileName As String
    Dim oFileSystem As Object
    Dim sCommand As String
    Dim oFSO As FileSystemObject
    Dim oText As TextStream

    'First get current drive, directory and save
    sCurDir = CurDir$
    sCurDrive = Left$(sCurDir, 1)
    'Change to working directory (where the infile is)
    sTargetDrive = Left(sWorkingDirSpec, 1)
    ChDrive sTargetDrive
    ChDir sWorkingDirSpec

    'Build a batch file to invoke LotusXSL
    Set oFSO = CreateObject("Scripting.FileSystemObject")
    Set oText = oFSO.CreateTextFile("run.bat", True)
    oFSO.OpenTextFile "run.bat"
    oText.WriteLine "SET TMP_CLASS=%CLASSPATH%;."
    sCommand = "java -classpath %TMP_CLASS% org.apache.xalan.xslt.Process -in "
    sCommand = sCommand & infile & " -xsl "
    sCommand = sCommand & sXSLDirSpec & xslfile & " -out "
    sCommand = sCommand & outfile
    oText.WriteLine sCommand
    oText.Close

    'Invoke Lotus XSL and wait for it to finish
    lProcessID = Shell("run.bat", vbNormalNoFocus)
    lProcessHandle = OpenProcess(PROCESS_QUERY_INFORMATION, False, lProcessID)
    Do    'Wait for it to finish
        Call GetExitCodeProcess(lProcessHandle, lExitCode)
        Sleep iSleeptime
    Loop While (lExitCode = STILL_ACTIVE)

    Call CloseHandle(lProcessHandle)
End Sub
```

XML-JBI Retrieval and Presentation Services

The retrieval and presentation services provide a number of ways for information consumers to:

- Query for information of interest
- Dynamically create views of information based on published data and XSL stylesheets
- Examine views that were pre-generated based on subscriptions
- Perform advanced transformations and viewing based on any applications that have been made accessible through the xml_jbi

As stated previously, users can request through a subscription that a particular XSL transformation should occur when an information object of interest is published, resulting in a

server-side batch processing of a rendered view. The consumer can then access the web site and see the latest update of a given view. The XSL scripts can be generated by end users or developers and published to the XML-JBI ahead of time. The particular event handler calls LotusXSL to generate the appropriate view, and then places the output in the views folder. The following ASP code is then used to represent the latest list of views to the user:

```
<%@LANGUAGE = "VBScript"%>

<HTML>
<BODY>

<!-
This ASP page displays a table containing the names (with hyperlinks), sizes, and dates of all
files in the current directory except itself. It also has a hyperlink to create a new window
and another hyperlink that points to this file.
-->

<%
' Title
response.write "<H1>Current Views</H1>"

dim strURL
dim strPath
dim iBSPosition

' This is complete path to the current file.
strPath = request.ServerVariables("PATH_TRANSLATED")

' What we need is the complete path to the current directory.
' Find the last backslash in the strPath variable
iBSPosition = Len(strPath)
Do While Mid(strPath, iBSPosition, 1) <> "\"
 iBSPosition = iBSPosition - 1
Loop
' Strip off the filename, leaving the complete path the current directory
strPath = Left(strPath, iBSPosition)

' Use File System Object to access the list of all files in the current directory
Set objFSO = CreateObject("Scripting.FileSystemObject")
set objFolder = objFSO.GetFolder(strPath)
set objFolderContents = objFolder.Files

' display name of current URL
strURL = "http://" & request.ServerVariables("SERVER_NAME") &
request.ServerVariables("PATH_INFO")
response.write("<P><B>Current URL is </B>")
response.write("<A href='" & strURL & "' target='_blank'>" & strURL & "</A>")
response.write("</P>")

' display the name of the files in this directory in a table with size and date
response.write("<P><B>Files Available</B></P>")
response.write("<TABLE border='1'>")
response.write("<TH>File Name</TH><TH>Size (bytes)</TH><TH>Date Created</TH>")
For Each File in objFolderContents
 if File.Name <> "files.asp" then
  response.write("<TR>")
  ' make the list of files hyperlinks
  response.write("<TD>")
  response.write("<A href='" & File.Name & "' target='_blank'>" _
                       & File.Name & "</A>")
  response.write("</TD><TD>")
  response.write(File.Size)
  response.write("</TD><TD>")
  response.write(File.DateCreated)
  response.write("</TD>")
  response.write "</TR>"
 end if
Next
response.write("</TABLE>")
%>
</BODY>
</HTML>
```

Another feature is the ability to dynamically select both an XML document and XSL stylesheet that has been published, and have the transformation occur on the server to generate the result on the fly. This is good in the case of XML or HTML output that can be passed back to the client. The HTML option helps to address clients who may not have an XML aware browser available. The dynamic view code found in the scripts folder of each information object type illustrates this server-side processing of XML documents from within an ASP file.

```
dim strPath
dim iBSPosition

' This is complete path to the current file.  What we need is the complete
' path to the current directory.
strPath = request.ServerVariables("PATH_TRANSLATED")
' response.write(strPath)
' Find the last backslash in the strPath variable
iBSPosition = Len(strPath)
Do While Mid(strPath, iBSPosition, 1) <> "\"
  iBSPosition = iBSPosition - 1
Loop
' Strip off the filename, leaving the complete path the current directory
strPath = Left(strPath, iBSPosition)
' response.write(strPath)

Set objFSO = CreateObject("Scripting.FileSystemObject")
set objFolder = objFSO.GetFolder(strpath)
set objFolder = objFSO.GetFolder(objFolder.ParentFolder)
set objFolder = objFSO.GetFolder(objFolder.ParentFolder)
' XML Repository
set objXMLFolder = objFSO.GetFolder(objFolder.SubFolders("xml_repository"))
set objXMLFolderContents = objXMLFolder.Files
set objXMLFolderSubs = objXMLFolder.Subfolders
' XSL Repository
set objXSLFolder = objFSO.GetFolder(objFolder.SubFolders("xsl_repository"))
set objXSLFolderContents = objXSLFolder.Files
set objXSLFolderSubs = objXSLFolder.Subfolders

txtSelection=request("ChosenView")
' possible values should be all, folder, search
```

The above code snippet from the `view.asp` file shows the VBScript for using the file system object to collect the latest list of XML and XSL documents that have been published for an information object. These values are then used to populate list controls on the ASP form to allow the user to pick the XML and XSL documents.

```
<%
if txtSelection = "all" then

 ' response.write txtSelection
 ' Backup for specific message
 set objBackupFolder = objFSO.GetFolder(objFolder.SubFolders("backup"))
 if request("xml") <> " then
  set objBackupFolder = objFSO.GetFolder(objBackupFolder.SubFolders(request("xml")))
 end if
end if
%>

<!-- Call Java servlet to handle XML + XSL -->
<FORM action="/servlet/LotusXSL" target="content">
<!-- <input type="hidden" name="debug" value="true"> -->

<TD bgcolor="#dddddd">
Views:

<SELECT name="xsl">
<%
' get all files in xsl_repository
For Each File in objXSLFolderContents
  response.write "<OPTION VALUE='" & File.Path & "'>" & File.Name
Next
```

```
%>
<OPTION value="mtf">MTF view</OPTION>
</SELECT>
</TD>

<TD bgcolor="#dddddd">
XML Files:

<SELECT name="xml">
<%
' get all files in xml_repository/<subfolders>
For Each Folder in objXMLFolderSubs
  Set objFolderContents = Folder.Files
  For Each File in objFolderContents
   response.write "<OPTION VALUE='" & File.Path & "'>" & File.Name
  Next
Next
%>
</SELECT>
</TD>

<TD bgcolor="#dddddd">

<INPUT type="hidden" name="backupdir" value="<% =objBackupFolder.Path %>">
<INPUT type="hidden" name="username" value="<% =Session("Username") %>">
<INPUT type="submit" value="View XML File">
</TD>

</FORM>
```

The above code snippet shows how the LotusXSL processor is called from the ASP code to perform the requested transformation on the selected XML and XSL documents. The result is sent to the browser.

In addition to the dynamic viewer, there can be a number of advanced applications that can be made accessible for the user. I will not spend a lot of time discussing the applications included in the source – however I would like to highlight some interesting examples.

The first example illustrates the passing of parameters to an XSL stylesheet. The following code snippet shows a call from a Java servlet to LotusXSL that passes not only the typical parameters of XML, XSL, and output documents, but also the user parameters of user directory location, user name, and a target metadata number or "tamnum" key.

```
// Use LotusXSL to transform XML
MyTAMLotusXSLTransformation myTAMLotusXSLTransformation = _
     new MyTAMLotusXSLTransformation();
myTAMLotusXSLTransformation.transform(xml_filename, xsl_filename, _
     output_filename, userdirfileURL, username, tamnum);

// Read the file and return to browser
FileContents myFileContents = new FileContents();
output_html = myFileContents.getFileContents(output_filename);
out.println(output_html);
```

The next code snippet represents the extended Java class `MyTAMLotusXSLTransformation` which uses the parameters in the call to LotusXSL.

```
class MyTAMLotusXSLTransformation extends Object
{

 // Constructor
 public MyTAMLotusXSLTransformation()
 {
  // constructor intentionally empty
 }

 // This method does a transform for a given user.
 // It will insert a <xsl:variable name="userdirectory select="userDirectory"/>
```

```
 // element into the DOM of the stylesheet.

 public static void transform(String xmlSourceURL, String xslURL, String outputURL, String
userDirectoryURL, String userName, String tamnum)
  throws java.io.IOException,
   java.net.MalformedURLException,
   org.xml.sax.SAXException
 {

// org.apache.xalan.xslt.XSLTInputSource xslSheet =
//  new org.apache.xalan.xslt.XSLTInputSource (myDOMDocRootNode);

  // Instantiate an XSLTProcessor.
  org.apache.xalan.xslt.XSLTProcessor processor =
  org.apache.xalan.xslt.XSLTProcessorFactory.getProcessor();

  // Set a param named "userdirectory", that the stylesheet can obtain.
  processor.setStylesheetParam("username","'" + userName + "'");
     processor.setStylesheetParam("userdirectory", "'" + userDirectoryURL + "'");
     processor.setStylesheetParam("tamnum", "'" + tamnum + "'");
  // Create the 3 objects the XSLTProcessor needs to perform the
        // transformation.
  org.apache.xalan.xslt.XSLTInputSource xmlSource =
   new org.apache.xalan.xslt.XSLTInputSource (xmlSourceURL);
  org.apache.xalan.xslt.XSLTInputSource xslSheet =
   new org.apache.xalan.xslt.XSLTInputSource (xslURL);
  org.apache.xalan.xslt.XSLTResultTarget xmlResult =
   new org.apache.xalan.xslt.XSLTResultTarget (outputURL);

  // Perform the transformation.
  processor.process(xmlSource, xslSheet, xmlResult);
 }
}
```

The next example illustrates an XSQL query to an Oracle database to extract the results as an XML document, for use in aggregating information from several legacy databases into a larger XML information object for publishing.

```
<?xml version="1.0"?>
<xsql:query
 xmlns:xsql="urn:oracle-xsql"
 connection="TamDB"
 doc-element="tams"
 row-element="tam">
select  tam_id "tam_id",
  tam_dpp_uri "dpp_filename",
  tam_lat_degrees "latitude_degrees",
  tam_long_degrees "longitude_degrees",
  tam_elev_meters "altitude_meters",
  tam_vector_x "vector_x",
  tam_vector_y "vector_y",
  tam_vector_z "vector_z",
  tam_be "be",
  tam_osuffix "osoffix",
  tam_dmpi "dmpi",
  tam_data "tam_data"
       from tams
       where tam_lat_degrees &lt;= {@NW_lat} and
             tam_lat_degrees &gt;= {@SE_lat} and
             tam_long_degrees &gt;= {@NW_long} and
             tam_long_degrees &lt;= {@SE_long}
    order by tam_lat_degrees, tam_long_degrees
</xsql:query>
```

```
    File tempCorFile = new File(TAM_COR_FILENAME);
    getFileFromURL(tempCorFile.toURL(), corFilename);
    if (debug != null) {
     out.println("Jini did not work.  Getting TAM data from local file.");
     String XSQLQuery =
"http://dev.mitre.org/xml_jbi/data/tam/scripts/tam_viewer/queryTAM.xsql?NW_lat=" + NW_lat +
"&SE_lat=" + SE_lat + "&NW_long=" + NW_long + "&SE_long=" + SE_long;
```

```
//     response.sendRedirect("/xml_jbi/data/tam/scripts/tam_viewer/queryTAM.xsql?NW_lat=" +
NW_lat + "&SE_lat=" + SE_lat + "&NW_long=" + NW_long + "&SE_long=" + SE_long);
     getFileFromURL(new URL(XSQLQuery), corFilename);
```

The first code snippet above shows the XSQL file that executes the SQL statement from the database and produces the resulting XML output. The second code snippet illustrates how to call the XSQL script from within a Java servlet. These simple examples demonstrate some of the power that is available to developers who are using these technologies.

But I Thought XML was Too Big!

There is much concern over the amount of bandwidth available to access information, especially in the case of using XML. While it is true that the size of XML documents is usually much larger than the source text document from which the XML is based, the information-centric approach discussed here helps to address these concerns. For example, in many of today's message passing interfaces, the complete message is usually passed from one system to another. In the information-centric approach this method should be used only when absolutely necessary. Instead, consumers subscribe only to the information they need, which will usually mean that some transformation must occur to produce a subset of the information.

In addition, there is a lot of work going on in compression. XML can be significantly compressed in a number of ways. There are commercial tools, such as Pkzip (everyone's favorite), or the AT&T smart compression tool (xmill). Alternatively, Knowledge Based Compression can be used, a MITRE concept which uses knowledge about message structure to direct compression/decompression. It can be combined with other techniques and could be used for legacy binary transmission. The information can also be signed for added security.

Also, optimized markup languages – for example, Wireless Markup Language (WML) – can be exploited where bandwidth is narrow. In addition, XML is being extended to allow transmission of binary data objects within XML elements. This will allow transmission of compressed messages within an XML wrapper. It will also further enable transmission of encoded and encrypted data and messages within XML elements. This is an important development that is significant for DoD developers

Commercial Alternatives

The XML-Infosphere (XML-JBI) example demonstrated here represents a proof-of-concept work that was undertaken for the US DoD. It was intended to evaluate the JBI concept and the use of XML to enhance information interoperability. The prototype works quite well for small applications but certainly is not intended for production or enterprise use. Instead, there are a number of commercially available tools that should be evaluated for this purpose. However, it is our experience that the skills necessary for implementing the prototype we constructed (for example, ASP, VB, VBScript, Java) are also necessary to implement many of these commercial alternatives. Here are some of the tools we are in the process of evaluating:

- IBM Websphere and MQ Series
- Microsoft Site Server
- Oracle WebDB
- Sequoia XML Portal
- Excelen XML Server
- BEA WebLogic Server
- Apache

Conclusions

Using an XML-Infosphere architecture has a number of advantages:

- Emphasizes information-centric approach (hub and spoke)
  - Information producers publish without knowing needs of consumers
  - Information consumers can:
    - Subscribe to the right data in whatever view they want
    - Do creative things more easily with the data
  - Reduces the number of interfaces ("interoperability for the poor")
- Emphasizes COTS vs. GOTS (Government off-the-shelf)

XML is a fantastic technology with great potential. However, for information interoperability you need:

- Comprehension of your operational processes
- Well understood information exchange requirements
- Agreement on terms, definitions, and information objects
- Effective change management strategies

Effective information management requires services for publishing, subscribing to, transforming, personalizing, and presenting information objects. An XML-Infosphere architecture is an effective approach to achieving information interoperability.