Enterprise Solutions • Content Management • XML Labs

XML JOURNAL

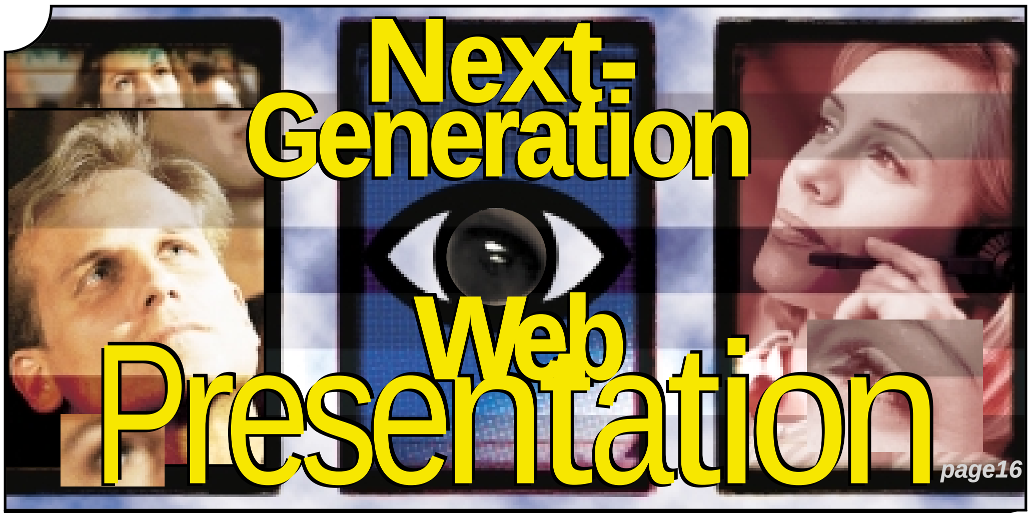
THE ULTIMATE XML ENTERPRISE RESOURCE

July 2003 Volume:4 Issue:7

xml-journal.com

$6.99US $7.99CAN

SYS-CON MEDIA



The future of Web presentation isn't high tech, it's high concept

IBM

ibm.com/websphere/seeit

XML JOURNAL

FOUNDING EDITOR
Ajit Sagar ajit@sys-con.com

EDITORIAL ADVISORY BOARD
Graham Glass graham@themindelectric.com
Coco Jaenicke cjaenicke@attbi.com
Sean McGrath sean.mcgrath@propylon.com
Simeon Simeonov talktosim@polarisventures.com

EDITORIAL
Editor-in-Chief
Hitesh Seth hitesh@sys-con.com
Editorial Director
Jeremy Geelan jeremy@sys-con.com
Managing Editor
Jennifer Van Winckel jennifer@sys-con.com
Editor
Nancy Valentine nancy@sys-con.com
Associate Editors
John Evdemon jevdemon@sys-con.com
Jamie Matusow jamie@sys-con.com
Gail Schultz gail@sys-con.com
Jean Cassidy jean@sys-con.com

PRODUCTION
Production Consultant
Jim Morgan jim@sys-con.com
Art Director
Alex Botero alex@sys-con.com
Associate Art Directors
Louis F. Cuffari louis@sys-con.com
Richard Silverberg richards@sys-con.com
Assistant Art Director
Tami Beatty tami@sys-con.com

CONTRIBUTORS TO THIS ISSUE
Joel Amoussou, Michael Carey, Daniela Florescu, Paul Gubbay, David Kershaw, Nitin Mangtani, Tim Matthews, Eric Pulier, David Quimby, Hitesh Seth, Eric J. Schwarzenbach, Troy Tolle

EDITORIAL OFFICES
SYS-CON MEDIA
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 **FAX:** 201 782-9637
XML-JOURNAL (ISSN# 1534-9780)
is published monthly (12 times a year)
by SYS-CON Publications, Inc.
Periodicals postage pending
Montvale, NJ 07645 and additional mailing offices.
POSTMASTER: Send address changes to:
XML-JOURNAL, SYS-CON Publications, Inc.,
135 Chestnut Ridge Road, Montvale, NJ 07645.
Worldwide Newsstand Distribution
Curtis Circulation Company, New Milford, NJ

An XML Take on Tech·Ed

WRITTEN BY **HITESH SETH**

Microsoft's flagship technology conference, commonly known as Tech·Ed, was held in Dallas in early June. Much has been written elsewhere about this event, but I would like to point out a couple of things I thought were quite significant, from an XML perspective.

Although XML and Web services are well utilized in most Microsoft technologies, a couple of highlights were: InfoPath, Microsoft's XML-based electronic forms technology, part of the next version of Office 2003; XML support in other Office products, particularly Excel and Word; BizTalk Server 2004, a.k.a. Jupiter, the first release of the unified e-business server offering; and SQL Server/XML support.

InfoPath (reviewed in our June issue [***XML-J*** Vol. 4, issue 6]) was definitely a show-stealer. There were about six sessions held on InfoPath itself, and a number of others highlighted the tool as a key collaboration/Web services user interface tool, a.k.a. a smart client. I think Microsoft has made great progress related to the electronic forms creation user interface. What needs to be thought about are the various ways the forms can be filled/submitted. Currently an InfoPath client is required to submit a form; we need to start thinking about how this medium will work across enterprises, in mobile devices, etc.

XML support in Microsoft Word and Excel was the other major spotlight. The XML support in Office applications boils down to (1) the ability to store Office documents as XML documents, which typically have a schema defined by Microsoft; (2) the ability for Office documents to utilize external schemas, and to bind components of the documents to specific elements of a schema. Two applications come to mind: first, a proposal template that can utilize a proposal XML Schema and then have the nicely formatted Word document/template. At runtime a developer can then merge the dynamic XML from a back-end application using the XML Schema–enabled Word template, and generate the proposal on demand and deliver it to the customer. This would be a great asset for application scenarios such as salesforce automation, insurance quotes, etc.

With Excel things become a little more interesting. As a die-hard spreadsheet user, I have come to live with Excel as a very useful tool for ad hoc analysis of data, charting, calculations, pivot tables, etc. XML support for Excel means that all these analytical and reporting features can now be combined with enterprise data. A simplified explanation of the XML support is that Excel 2003 allows the user to map individual cells of the spreadsheet (A1, B2, etc.) with XML Schema elements through a drag-and-drop interface. So an XML-contained purchase order (you know, with Header and Lines), translates into a set of Excel cells containing the customer name and the various order items. This data can come from relational databases (with RDBMS-to-XML conversion utilities available – for instance, SQL Server provides what is known as SQLXML) or from back-end systems, through XML/Web services interfaces and/or wrappers. Excel now becomes a powerful reporting tool, not just for self-contained data, but for external data sources.

Then there were the BizTalk Server 2004 announcements. A first beta of the highly awaited product was made available to conference attendees. A highlight of the new release was the new "Orchestration Designer" interface (a tool that enables business analysts and developers to define and implement business processes). BizTalk Server 2000/2002 utilized a customized version of Visio as the orchestration designer. In BizTalk 2004, elements that make up business orchestrations are now part of the Visual Studio .NET 2003 interface. Another nice addition was the Business Rule Composer, which is geared toward abstracting, defining, and maintaining a common business rules repository. The rules can then be used as decision points and actions within an orchestration. Also, BizTalk 2004 now supports drag-and-drop import of XML Schemas and BPEL (Business Process Execution Language) export functionality.

A key aspect of the show was the introduction of http://techedbloggers.net. This was an ad hoc community established to syndicate feedback about the show. The community consisted of speakers, attendees, staff, and even press, and ended up becoming a single point to collaborate on a central theme, the Tech·Ed conference. Community establishment at technology conferences seems like a natural fit. I won't be surprised if this blogging becomes a trend at future conferences. In fact, I think it makes sense for the conference organizers to build such a platform and provide it as a feature to the attendees.

Watch for ***XML-Journal***'s coverage of JavaOne 2003 in next month's issue.

Author Bio

Hitesh Seth, editor-in-chief of XML-Journal and XML Track chair for the Web Services Edge Conference, is the chief technology officer of ikigo, Inc., a provider of business activity monitoring solutions.

HITESH@SYS-CON.COM

Mindreef

www.mindreef.com

Send Me Your Stylesheet

WRITTEN BY **TIM MATTHEWS**

In these cynical, post-bubble times, most chief information officers are rightfully dismissive about new technologies that promise to boost efficiency or customer service...but once in a while the claims are very true. Stylesheets can make application development cheaper and faster while increasing customer satisfaction. They also allow your customers to take over subsequent application development work – paying the costs along the way.

In terms of costs and efficiency, application development may not seem like a problem zone. After all, the advent of the Web has already revolutionized the process. As the front end of most applications is a Web browser these days, it now may take only months, instead of quarters, to build an application. Once the back-end integration and logic are put into place, and the Web interface is built to satisfaction, presto – the application goes live. Typically only severe problems or a dramatic shift in a business model will cause a major rework.

But this approach doesn't allow for easy customization. Why does that matter? Say your company has an online catalog for industrial supplies, and the competition is coming on strong. How can you stay ahead of the pack? Consider your customer's experience. In order to research, purchase, and track a product shipment, your customer has to open up your catalog and those of all your competitors – each of which has its own Web interface with distinct navigation, flow, and security. That may not sound so cumbersome. But in many manufacturing supply applications, purchasers need to navigate through hundreds of parts and specifications.

If only your online catalog's interface could look just like your customer's own internal Web portal – with everything in the same place – you would save your customer time and irritation. The ease of using your catalog would separate you from your competitors and increase the likelihood of your customer buying from you.

But using the current Web model of development, it would be prohibitive to customize your catalog application for even your top customers. Building screens, customizing navigation, and setting up the security scheme would be a tremendous task. Imagine the cost of your own Web catalog application multiplied by your top 10 customers – hardly a healthy return on investment.

That's where stylesheets come in. Stylesheets are built using Extensible Stylesheet Language (XSL). They are the presentation component of the rapidly growing XML. Both XML and XSL are standards created by the WC3 – the same group that brought us HTML for the Web.

Simply put, XSL stylesheets customize the look of data described in XML. The key is that a single set of XML data can be presented in an infinite variety of ways using XSL, allowing a "faceless" XML application to take on any number of looks, depending on the stylesheet applied.

"...a single set of XML data can be presented in an infinite variety of ways using XSL, allowing a "faceless" XML application to take on any number of looks"

It is not a big adjustment to start employing stylesheets. You simply build your application using XML and then create a stylesheet that makes the application look like your current Web interface. So all you've done is replace an HTML- or Java-coded layout and navigation with something identical built with XML and your XSL stylesheet.

Then you just let your customers know that they can customize the way your online catalog works to make things easier for them. All they have to do is create a custom look-and-feel using an XSL stylesheet and send it to you. When you apply their stylesheet to your XML-based application, voilà – your application appears with their look. Customers start to specify how applications should look by building stylesheets and sending them to you – kind of like EDI, but for presentation.

Applications built this way are also tremendously flexible. A customer, mobilizing his or her workforce using wireless devices, could use a stylesheet that renders your application into the appropriate wireless presentation format, likely WML or XHTML. Web services? No problem. They're already XML-based and hold the potential for even tighter integration.

How will stylesheets save money and improve efficiency? Building customer-facing applications using XML costs less in the first place, since it is much more powerful and sim-

~continued on page 34~

Where Open Minds Meet

- **Learn** how companies have achieved higher profits and increased their productivity by utilizing Linux
- **Participate** in LinuxWorld's **world-class** education program and benefit from interactive training in the **all-new Hands-on Labs!**
- **Discover** the latest innovations and technologies from the hottest companies around
- **Hear** the latest developments and updates on the state of open source at our analyst roundtable discussion

CORNERSTONE SPONSOR

hp invent

PLATINUM SPONSORS

ca Computer Associates · DELL · IBM · intel · Novell · ORACLE · pogo [linux] Merging Business with Linux · redhat · Sun microsystems We make the net work. · VERITAS

GOLD SPONSOR

Hummingbird

SILVER SPONSOR

APPRO HPC Cluster Solutions

LinuxWorld™ Conference & Expo

Conference: August 4-7, 2003
Expo: August 5-7, 2003

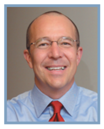
The Moscone Center
San Francisco, CA

Join us this August and see why Linux is thriving! Government agencies and companies in the telecommunication, financial services, retail and manufacturing industries are turning to Linux to save money. Isn't it time you did? **LinuxWorld. Where Open Minds Meet.**

Attend Keynotes *and learn from inspiring visionaries who are devoted to Linux and open source.*

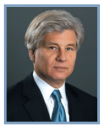
Tuesday, August 5th
10:30am-11:30am
Linux: The Next Step
Peter Blackmore
Executive Vice President
Enterprise Systems Group
Hewlett-Packard

Tuesday, August 5th
1:30pm-2:30pm
Jonathan Schwartz
Executive Vice President
Software Group
Sun Microsystems, Inc.



Tuesday, August 5th
4:30pm-5:30pm
Matthew Szulik
Chairman, Chief Executive Officer and President
Red Hat

Wednesday, August 6th
10:30am-11:30am
Linux and the Evolution of the Internet
Irving Wladawsky-Berger
General Manager
IBM Corporation



Wednesday, August 6th
3:30pm-4:30pm
Charles Rozwat
Executive Vice President
Server Technologies Division
Oracle Corporation

Special Presentation

Open to All Registered Attendees
Tuesday, August 5th
12:00pm-1:00pm
The Golden Penguin Bowl
Host: *Chris DiBona, Vice President – Marketing; Co-Founder, Damage Studios, Inc.*

Analyst Roundtable

Open to All Registered Attendees
Wednesday, August 6th
1:30pm-2:30pm
State of Open Source Roundtable
Moderator: *Larry Augustin, Partner, Azure Capital Partners*
Panelists: *Pierre Fricke, Executive Vice President of Web Application Infrastructure and Product Lifecycle Management (PLM) Infrastructure, D.H. Brown Associates, Inc.; Daniel Kusnetzky, Vice President, System Software Research, IDC; Ted Schadler, Principal Analyst in Software, Forrester; George Weiss, Vice President and Research Director, Gartner*

IDG World Expo

www.linuxworldexpo.com

Security Issues in the Service-Oriented Architecture

WRITTEN BY
ERIC PULIER

Web services security products are ready for prime time

Who doesn't love the service-oriented architecture (SOA)? You get efficiency in your application development, revolutionary ability to interoperate with partners and suppliers, and mastery over change management that was never before possible.

With the technologies available today to take advantage of SOAs in enterprise settings, organizations can quickly find themselves faced with many concurrent decisions and implementations. Your staff is exposing legacy applications as Web services in Visual Studio .NET and JBuilder. Partners are clamoring for access to your systems using Web services. Your boss wants to connect the company to key customers using Web services. No problem? Big problem: security.

SOA security is the two-ton elephant stomping through the data center. According to ZapThink, "Security is the immediate roadblock facing widespread implementation of Web services technologies across the enterprise." Security problems have hampered many organizations in advancing SOA ambitions.

"To be secure, an SOA must have a way to address the security apparatus of a third party"

The good news is that help is on the way. The complicated truth about SOA security is that it is a manageable, straightforward matter if addressed intelligently at the outset. There are a number of promising SOA and Web services security solutions, particularly in the focused Web services management platforms, coming onto the market today that help tackle security issues. With reasonable planning and understanding of the issues involved, it becomes possible to design and implement a successful, secure SOA.

The Issues

The SOA's inherent security problems stem from the ways in which the SOA replaces traditional security parameters with new, open standards. The efficiency of Web services is in interoperability standards, and with that efficiency comes a proportional increase in risk. Imagine exposing your data in a manner that makes it accessible to anyone who just bought *SOAP for Dummies* at the local bookstore. The security problem is twofold in that not only are the new standards completely open – i.e., no one owns them – but they were also developed without security in mind.

Why do SOAs inherently contain security risks? The answer lies partly in the origins of the SOA. Web services were developed over a period of years by industry consensus as a way to, among other things, enable the creation of reusable code, simplify development, and streamline system integration. While these goals were met, the open standards that emerged neglected to address security. Specifically, XML, SOAP, WSDL, and UDDI are open standards that enable the transmission and description of data and procedure calls between systems. However, none of these open standards contain any inherent security aspects of their own. If left alone, they are completely insecure. In fact, Web services were designed to be able to move efficiently through firewalls. Their very openness actually exposes their insecurity all the more.

The nature of the SOA also disrupts the traditional security paradigm in several critical ways:

- ***Machine-to-machine communication:*** Most security infrastructure is geared toward human-to-machine interaction, while Web services involve machine-to-machine interaction. Unauthorized users may find it simple to penetrate and evade detection in SOAs, as compared to traditional architectures, because they are going after exposed application functionality that has no human screening. In addition, these exposed interfaces are offered in a convenient standards-based form. It's similar to posting your home address in the phone book, and then handing out keys to your house with every copy.

- ***Securing unknown third parties:*** By their nature, SOAs may require that an entity authenticate and grant authorization to users from outside its system. To be secure, an SOA must have a way to address the security apparatus of a third party in order to authenticate and authorize use of the service.
- ***Flooding:*** Malicious, unauthorized users can "flood" an SOA with service requests and render it inoperable – a denial of service (DoS) attack.
- ***Unwanted "listening in":*** In contrast to traditional architectures that allow messages to be exchanged either exclusively within the firewall or through private networks, Web services messages may travel across the Internet, where they are vulnerable to eavesdropping by unknown individuals.
- ***Service-level agreement (SLA):*** The SOA has no way of monitoring or assuring the service levels of Web services, crippling the ability of system administrators to identify and respond to security problems in a timely manner.
- ***No logging:*** The unsecured SOA has no message- and transaction-logging mechanism. After a service has been called and used, there is no way to determine who has used the service and from where the request originated. As a result, there is no audit trail that can be used later to investigate possible breaches in security.
- ***No encryption:*** The unsecured SOA has no encryption protocol embedded in its architecture. Messages that are intercepted can be read by anyone.

These challenges can be somewhat intimidating when it comes to transitioning to an SOA. However, it is important not to overlook the incredible cost savings and efficiency that come from an SOA in addition to the inevitability of this computing model. Those looking to implement an SOA should be encouraged that Web services security products, techniques, and standards are fast proving to be ready for prime time.

Example – Supply Chain Management

To illustrate the security problems inherent in SOAs, let's look at a typical supply chain management process that involves a manufacturer and three vendors. Figure 1 represents the traditional business-to-business security environment:

- Each vendor communicates with the manufacturer using a private network.
- Encryption may be used, but the manufacturer and vendor can both be fairly confident that communication is private.
- Authentication is coded into the application, so the manufacturer can be relatively confident that Vendor A is actually Vendor A.
- Authorization is, in addition to being coded into the applications, also handled by the security infrastructure of the entity originating the transmission.

Though secure, this traditional setup is costly and complex to maintain:

- Modifications to the manufacturer's application will automatically require custom revisions to the vendor's application or else they will not be able to communicate.
- Flexibility in extending the functionality of these connected applications is limited to the amount of custom interface development that each trading partner wants to finance.

If the manufacturer and its vendors decide to expose applications as Web services in an SOA, they benefit from greatly increased flexibility, but face security risks. Figure 2 shows what this SOA would look like. Applications developed in this environment have numerous potential functional advantages over the traditional model, including:

- The vendor's ability to create applications that easily interact with the manufacturer's ERP system – "pulling" order data out of the system based on anticipated demand.
- The manufacturer's and vendor's mutual ability to process financial matters between their respective accounting systems.

This is all accomplished without needing any proprietary software to create custom interfaces and remote procedure calls, without requiring a dedicated private network, and without even having to develop any code – it may already exist as a reusable Web service.

Unfortunately, however, the SOA shown in Figure 2 also contains a variety of security risks. Because the messages may travel across public networks, they can be "listened to" by others, they can be intercepted and changed, and they can be rerouted for the purpose of fraud or malicious mischief. In

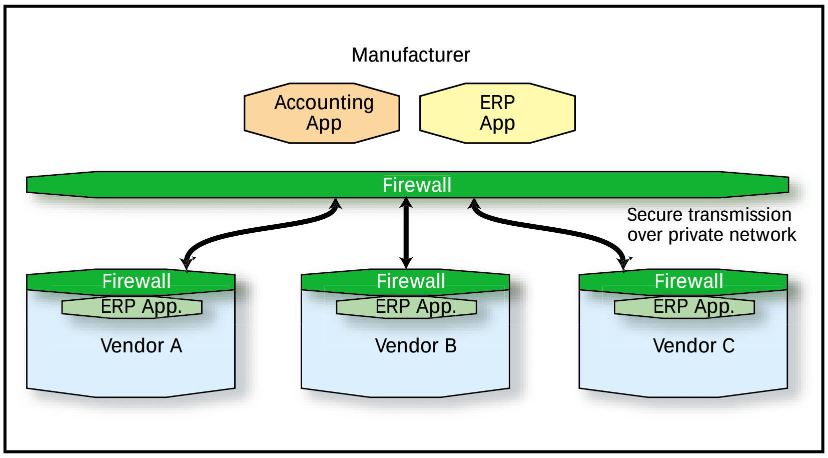


Figure 1 • B2B security environment

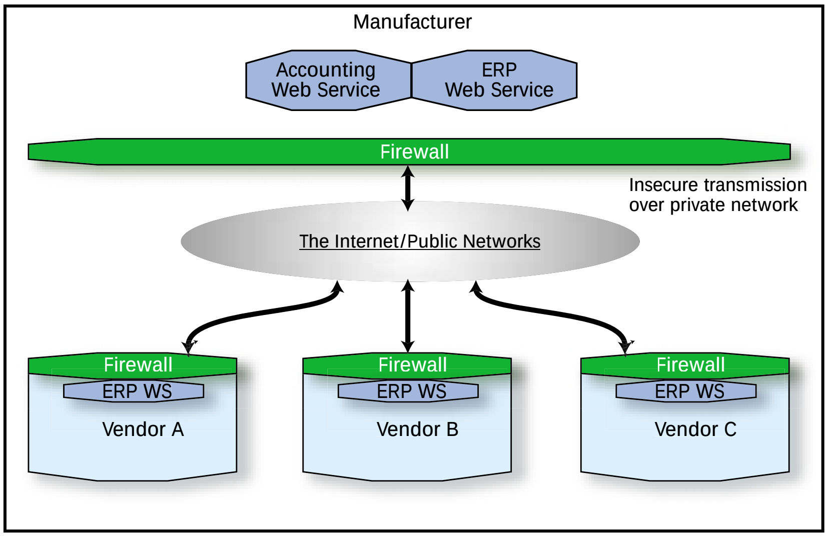


Figure 2 • Security risks inherent in the B2B-to-SOA transition

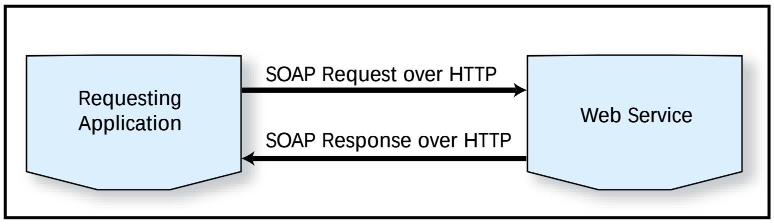


Figure 3 • Standard SOAP request and response

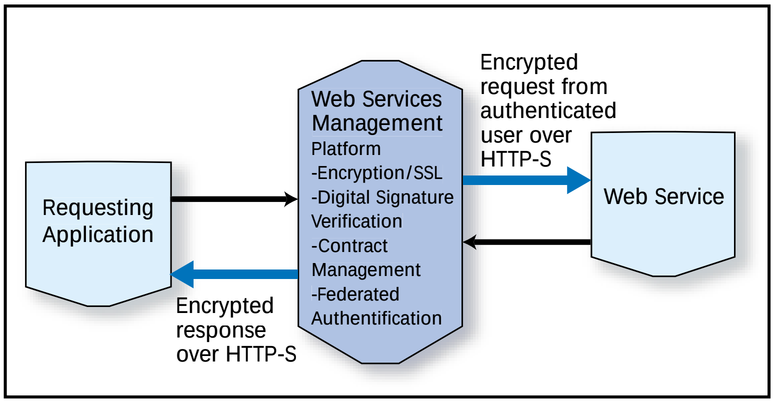

Requesting Application
Web Services Management Platform
-Encryption/SSL
-Digital Signature Verification
-Contract Management
-Federated Authentification
Encrypted request from authenticated user over HTTP-S
Web Service
Encrypted response over HTTP-S

Figure 4 • SOAP request/response filtered through Web services management platform's SOAP message interception application

addition:

- Whatever hard-coded authorization process existed in the traditional model does not exist here.
- Neither the manufacturer nor the vendor knows who is authorized to access specific Web services.
- A hacker could configure a machine to impersonate a vendor's system and make erroneous or fraudulent service calls.
- The SOA can be overloaded by service calls, resulting in a DoS attack, and there is no audit trail to determine who has done what and at what time.

Building Security into Your SOA Through a Web Services Management Platform

To secure an SOA, it is best to identify, acquire, and implement a robust Web services management platform that will intercept SOAP messages and enforce security policies. Ideally, the platform selected will have the capability to connect with any existing security framework currently in use in the enterprise, even if stand-alone implementation is chosen in the first stage.

A suitable Web services management platform will tackle the most pressing security issues in an SOA. It should monitor the SOAP requests, authenticate users, establish authorization, encrypt, provide signatures and certificates, and assure contractually promised service responses. Figure 3 shows the standard Web service request and response transmitted in SOAP over HTTP. In Figure 4, the Web services management platform adds security to this process by authenticating the requesting user and encrypting both the request and the response using an SSL. The result is an SOA that operates in HTTP-S.

"... it is important not to overlook the incredible cost savings and efficiency that come from an SOA"

The following is a checklist of features that must be delivered in the Web services management platform that you select to secure your SOA:

- ***Federated authentication:*** A framework should utilize Security Assertion Markup Language (SAML), Web Service Security (WS-Security), and Java Authentication and Authorization Service (JAAS) open standards to implement a flexible, federated security architecture that can be easily integrated into existing enterprise environments as needed.
- ***Application proxy:*** The platform should provide a secure proxy for the Web services exposed in your enterprise. It should decouple and protect internal Web services from external users and trading partners, as well as proxy-approved external services from users and applications within the organization.
- ***Secure Sockets Layer (SSL):*** The platform should support SSL, thereby facilitating 40-bit or 128-bit secure communications at the HTTP level.
- ***Contract management:*** You should be able to configure the platform to permit access to a service through a service contract. The contract specifies the terms between a service provider and consumer and is associated with an SLA. For example, start date and time, end date and time, and access count.
- ***Digital signing:*** Digital signatures facilitate an implicit trust relationship between a service provider, the service consumer, and the Web services management platform.
- ***XML encryption:*** The platform should enable you to perform XML encryption at the element level in the SOAP message. This allows service providers to increase the level of security for a particular operation within a service.
- ***Replay attack protection:*** The platform should provide Web services within your enterprise with protection from replay attacks by keeping track of SOAP requests and ensuring that the same request is not replayed. This helps eliminate DoS attacks and errors in contract usage and billing.
- ***Auditing:*** It is essential to track and log all service calls to provide a mechanism to audit and reconcile all service transactions.

Conclusion

Though it may seem as if there is a myriad of complex issues surrounding security in the SOA, the bottom line is that you can accomplish most if not all of your security goals with the more robust Web services management platforms on the market today. By managing a vendor-neutral, standards-based architecture in this manner you can take advantage of all the capabilities of existing security vendors (Netegrity, RSA, CA, etc.) in the context of a robust SOA framework. Then, if you work carefully to design security into the SOA from the outset, extending the same level of attention and concern for security that you have used in previous enterprise architecture efforts, you will find that your SOA will be able to operate smoothly and securely, even as it grows.

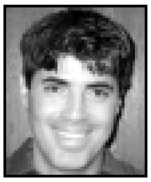
Author Bio

Eric Pulier is CEO for Digital Evolution, Inc., a leading provider of Web services management solutions. With more than 15 years in the software and digital interactive industries, Eric was recently named one of 30 e-Visionaries by VARBusiness. A popular speaker at elite technology conferences around the globe, Eric is a member of the board of directors for the Center for Telecommunications Management and a magna cum laude graduate of Harvard University.

EPULIER@DIGEV.COM

CTIA

www.ctiashow.com



WRITTEN BY **PAUL GUBBAY**

XML and Macromedia Flash MX

A rich new look

Over the last few years XML has become the standard for moving data across the Web. Relational databases, application servers, and operating systems such as .NET make XML a key part of their infrastructure and an integral part of their business plan for the future.

As companies standardize on XML, it makes sense to invest in technologies that focus on the presentation of XML to the end user. Up until now, most of these technologies have been server-based application frameworks that typically convert XML into HTML to deliver a presentation layer to the user that allows for primitive interaction with the data.

Macromedia Flash is an immersive presentation layer that is also prevalent throughout the Web. Macromedia added extensions to Macromedia Flash 5 so developers could work directly with XML using its built-in language (ActionScript). With the release of Macromedia Flash MX, Macromedia again extended the environment with the introduction of components (building blocks) such as EditBox, ComboBox, CheckBox, and DataGrid to make it easier for developers to quickly build rich user interfaces. During this time, Macromedia introduced the concept of rich Internet applications (RIA), compelling Web-based experiences that provide users with a rich interface and architecture for working with data over the Internet or corporate intranets.

The latest release from Macromedia is the Macromedia Flash MX Data Connection Kit, which contains the Macromedia Firefly Components and a developer edition of Macromedia Flash Remoting MX. This article focuses on how you can use the Firefly component architecture to create compelling Web-based Macromedia Flash interfaces that work directly with XML data and provide users with the functionality that they have come to expect from traditional desktop applications. Along the way, we will learn about Firefly's plug-in architecture for working with multiple data sources as well as its built-in shadowing technology for effectively updating data to the server. Keep in mind that Macromedia Flash applications are optimized for the Web with a very small footprint, Macromedia Flash Player. The rich client has ubiquitous deployment across multiple platforms, including all versions of Windows and Macintosh as well as multiple browsers, including Internet Explorer, Navigator, and Safari.

How Does Firefly Work?

Firefly was designed to provide developers with a flexible component-based architecture for retrieving, manipulating, and saving data from multiple data sources within Macromedia Flash MX. The foundation of this architecture consists of a client-side data management engine represented by the Connector, DataSet, and Resolver components. Working together, these components provide developers with a complete workflow for data management on the client. This workflow provides the following major aspects of a sophisticated application framework: Model-View-Controller for user interface management and synchronization, flexibility of data source interaction, and intelligent updates (see Figure 1).

Firefly provides developers with several data-aware visual components, such as the Grid, ComboBox, and EditBox. These components plug in seamlessly to the Firefly DataSet to allow users to view and edit their data. As the data is edited, the DataSet is updated and records the changes. Multiple visual components can point to the same field in the DataSet. As the value of the field is changed or when a new record is selected, all the controls are automatically updated to show the correct value.

Built for the Future

The Firefly plug-in architecture allows developers to work with multiple data sources including XML, Macromedia Flash Remoting MX, and Microsoft SQL Server. Each plug-in includes a specialized Connector and Resolver component. These components are built with an intimate understanding of their respective data source. Many of the Connectors contain additional properties that take advantage of specific functionality that is provided by the data source, such as the ability to retrieve data with a query or a server-side template. The Resolver contains similar functionality and is responsible for converting the changes that are made to the data into a DeltaPacket (XML-based set of instructions) that can be read by the server (e.g., the XML Resolver uses XPath statements to define which nodes are changed in an XML document).

Using this architecture, additional plug-ins can be built to read and write data from other sources such as Oracle, SQL Server, or Xindice, using any HTTP-compliant protocol like SOAP, XML-RPC, and Macromedia Flash Remoting MX. Simply changing the plug-in allows your application to talk to another data source without having to rewrite your code (see Figure 2).

The Shadow Knows

Shadowing is the process of tracking changes made to the DataSet in real time and generating a DeltaPacket that can be

AUTHOR BIO
Paul Gubbay is a director of engineering at Macromedia. Previously, Paul held the role of CEO at CyberSage Software, where he spent several years building the vision and infrastructure of the company. Under Paul's guidance, CyberSage focused on emerging technologies such as XML, Java, and Macromedia Flash to deliver leading edge product offerings.

sent to the server to update the original data source. This process provides developers with the ability to create optimized applications that can update batch changes to multiple records at one time, reducing network traffic and allowing multiple users to work with and update the same data source simultaneously (see Figure 3).

The DataSet uses the Resolver component to create the DeltaPacket when the data is changed. Each Resolver has an intimate understanding of the remote data source and generates a DeltaPacket that is specific to the type of data source (XML, SQL Server, or Flash Remoting) that is being updated.

TimeTrax Application

Now we will build an application in Macromedia Flash MX using Firefly. This example gives a high-level overview of the steps taken to build TimeTrax, a time entry application for a fictitious company called SoftCo. You will learn how easy it is to get started with Firefly and utilize the built-in features that are available to you out of the box. Note that this is not a step-by-step tutorial.

The TimeTrax application allows users to add, edit and delete time entries over the Web using a Macromedia Flash interface built with Firefly (see Figure 4).

On the server, TimeTrax uses two Java servlets. The getTimeData servlet returns XML data. The setTimeData servlet accepts an XML-based DeltaPacket containing a snapshot of the data that was modified by the user. Using the Firefly XML plug-in we can work with any XML data source. The getTimeDataservlet returns the XML packet shown in the following snippet.

```
<datapacket>
 <row act="10" custId="0" date="05/12/2002"
   billable="true"
duration="3" id="1" notes="Here is note(1)"
   projId="1" rate="45.50"
servId="0"/>
 <row act="9" custId="0" date="05/13/2002"
   billable="true"
duration="8.25" id="2" notes="Here is
   note(2)" projId="1"
rate="45.50" servId="1"/>
 <row act="8" custId="0" date="05/14/2002"
   billable="false"
duration="7.75" id="3" notes="Here is
  note(3)" projId="1"
rate="45.50" servId="1"/>
 <row act="7" custId="0" date="05/15/2002"
   billable="false"
duration="6.5" id="4" notes="Here is note(4)"
   projId="1"
rate="45.50" servId="2"/>
 <row act="6" custId="0" date="05/16/2002"
   billable="false"
duration="5.5" id="5" notes="Here is note(5)"
projId="1"
rate="45.50" servId="2"/>
</datapacket>
```

Layout the screen

In Macromedia Flash there is a concept of a stage that represents a screen in an application. Our first step is to drop the Firefly components onto the stage to build the time entry screen. We begin by setting up the Firefly data engine using the XML Connector, DataSet, and XML Resolver components, selected from the component panel. Note that these components will not show up in the final Macromedia Flash movie.

As the components are placed onto the stage, we define their properties using the visual property editor. The XML connector is first. The XML Connector is responsible for retrieving data and converting it into a format that can be used by the DataSet component. Here we specify the URL to the Java servlet providing us with our XML data. We also use XPath to define the node within the XML file that represents a record in our DataSet.

Next we define the DataSet component. The DataSet allows us to define subnodes or attributes as fields to be manipulated within our application. Once the data is mapped into the DataSet we can perform complex manipulations, sorting and filtering. Again we use XPath to define an attribute of our XML file to represent a field within our DataSet. In the example shown in Figure 5, the rate field is being set up as a money field and is mapped to the rate attribute within our XML packet.

Finally, we set up the XML Resolver component. The XML Resolver is responsible for saving data back to the server. As a user makes changes to the data, the XML Resolver generates XPath statements that define the nodes and/or attributes that are modified within the XML packet. In order to do this we must define the nodes and/or attributes that uniquely identify the path to the data being modified within the XML packet.

Now that we have set up the Firefly data engine we can easily build the interface using the Firefly visual components. As we drag and drop the visual components onto the stage we use the visual property editor to attach them to the DataSet. We can also specify additional properties such as row height or horizontal scrolling for the Grid component.

As the user makes changes to the data the DeltaPacket is automatically generated. When the user chooses to save his or her changes, the DeltaPacket is sent up to the Java servlet, where it can be processed and can update the main XML source. The sample DeltaPacket shown in the following snippet shows how the instruction set looks when working with the XML Resolver.

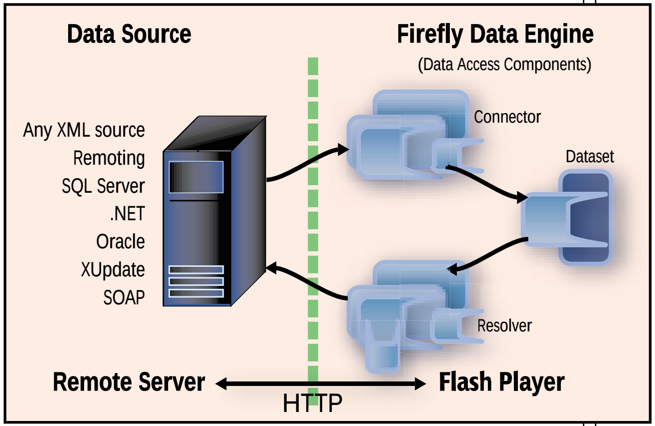


Figure 1 • Firefly data engine

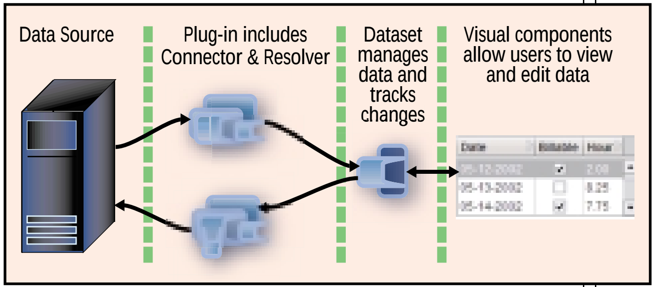

Data Source
Plug-in includes Connector & Resolver
Dataset manages data and tracks changes
Visual components allow users to view and edit data
Date
Billable
Hour
05-12-2002
2.00
05-13-2002
8.25
05-14-2002
7.75

Figure 2 • Firefly plug-in architecture

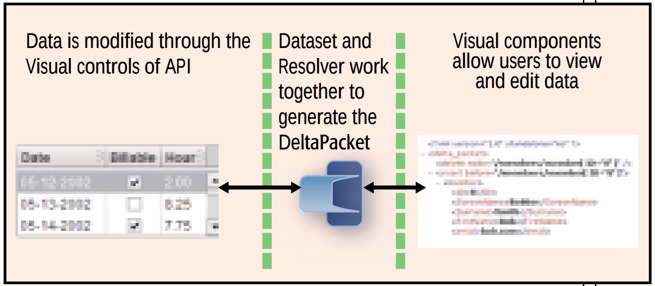
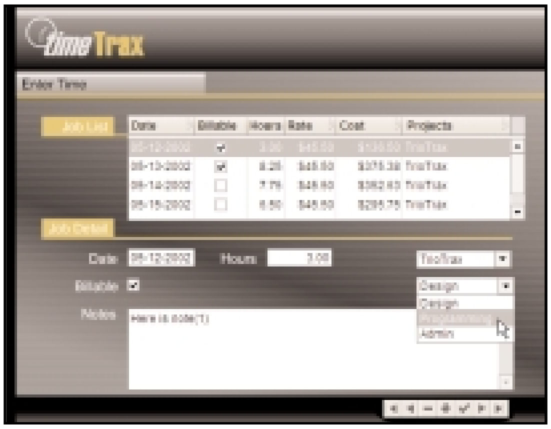

Data is modified through the Visual controls of API
Dataset and Resolver work together to generate the DeltaPacket
Visual components allow users to view and edit data
Date
Billable
Hour
05-12-2002
2.00
05-13-2002
8.25
05-14-2002
7.75

Figure 3 • Shadowing



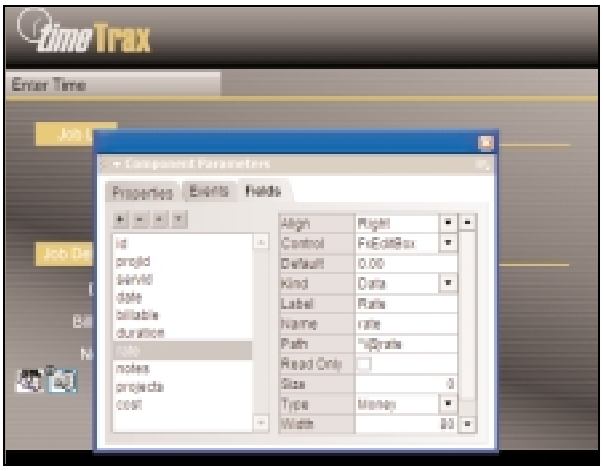
Date	Billable	Hours	Rate	Cost	Projects
05-12-2002	✓	2.00	$45.50	[illegible]	TimeTrax
05-13-2002	✓	8.25	$45.50	$375.38	TimeTrax
05-14-2002	☐	7.75	$45.50	$352.63	TimeTrax
05-15-2002	☐	6.50	$45.50	$295.75	TimeTrax

Date: 05-12-2002 Hours: 2.00

Billable: ✓

Notes: Here is note(1)

Figure 4 • TimeTrax application



id
projId
servId
date
billable
duration
rate
notes
projects
cost

Align	Right
Control	FxEditBox
Default	0.00
Kind	Data
Label	Rate
Name	rate
Path	"/@rate
Read Only	
Size	0
Type	Money
Width	80

Figure 5 • Setting up the rate field

```
<delta_packet>
 <delete node="/datapacket/row[@id='3']" />
 <delete node="/datapacket/row[@id='4']" />
 <insert before="/datapacket/row[@id='6']">
  <row act="10" custId="0" date="05/22/2003"
billable="true"
duration="3" id="7657" notes="Enter some new notes"
  projId="0"
rate="60" servId="0" />
 </insert>
 <modify node="/datapacket/row[@id='6']">
  <update attribute="billable">true</update>
 </modify>
 <modify node="/datapacket/row[@id='9']">
  <update attribute="date">08/15/2001</update>
 </modify>
</delta_packet>
```

The DeltaPacket contains three different types of nodes: delete, modify, and insert. Each of these nodes points to a unique node in your XML data source that represents a row of data in the DataSet. The packet contains only the information you need to update your XML data source.

- ***Node:*** This holds the XPath that uniquely identifies the node to be updated within your XML data source.
- ***Before:*** This attribute is used only for nodes that are going to be inserted. It describes where the new XML node should be inserted within the XML data source.
- ***After:*** This attribute is used only for nodes that are going to be inserted. It describes where the new XML node should be inserted within the XML data source.
- ***Update:*** This subnode is used only in conjunction with the modify node. It is used to identify the subnodes or attributes of the modified node that should be updated.

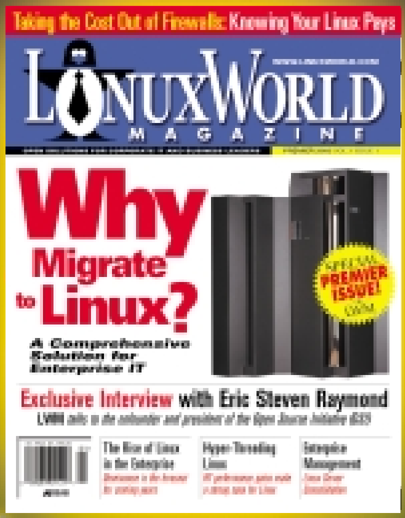
Summary

The Macromedia Flash MX Data Connection Kit and the Macromedia Firefly Components are part of an ongoing effort to make it easier to use Macromedia Flash MX to build rich interfaces for the Web that interact seamlessly with back-end data. The plug-in architecture and built-in shadowing technology provide developers with a powerful metaphor for working with multiple data sources in an intelligent and meaningful way. The built-in feature set of the components such as client-side filtering, sorting, data manipulation, and offline capture of changes to the data save considerable coding time for you as a developer, while allowing you to build Web-based applications with feature sets similar to those you might find in a traditional desktop application.

PGUBBAY@MACROMEDIA.COM

Premiering August 2003

www.LinuxWorld.com



Taking the Cost Out of Firewalls: Knowing Your Linux Pays
LINUXWORLD MAGAZINE
Why Migrate to Linux?
A Comprehensive Solution for Enterprise IT
SPECIAL PREMIER ISSUE!
Exclusive Interview with Eric Steven Raymond
The Rise of Linux in the Enterprise
Hyper-Threading Linux
Enterprise Management

The Leading Magazine for Corporate and IT Managers

LinuxWorld Magazine

There is no escaping the penetration of Linux into the corporate world. Traditional models are being turned on their head as the open-for-everyone Linux bandwagon rolls forward.

Linux is an operating system that is traditionally held in the highest esteem by the hardcore or geek developers of the world. With its roots firmly seeded in the open-source model, Linux is very much born from the "if it's broke, then fix it yourself" attitude.

Major corporations including IBM, Oracle, Sun, and Dell have all committed significant resources and money to ensure their strategy for the future involves Linux. Linux has arrived at the boardroom.

Yet until now, no title has existed that explicitly addresses this new hunger for information from the corporate arena. *LinuxWorld Magazine* is aimed squarely at providing this group with the knowledge and background that will allow them to make decisions to utilize the Linux operating system.

Look for all the strategic information required to better inform the community on how powerful an alternative Linux can be. *LinuxWorld Magazine* will not feature low-level code snippets but will focus instead on the higher logistical level, providing advice on hardware, to software, through to the recruiting of trained personnel required to successfully deploy a Linux-based solution. Each month will see a different focus, allowing a detailed analysis of all the components that make up the greater Linux landscape.

Regular features will include:

- *Advice on Linux Infrastructure*
- *Detailed Software Reviews*
- *Migration Advice*
- *Hardware Advice*
- *CEO Guest Editorials*
- *Recruiting/Certification Advice*
- *Latest News That Matters*
- *Case Studies*

SAVE 30% OFF!

REGULAR ANNUAL COVER PRICE $71.76

YOU PAY ONLY

$49.99

12 ISSUES/YR

**OFFER SUBJECT TO CHANGE WITHOUT NOTICE*

SUBSCRIBE TODAY!

WWW.SYS-CON.COM
OR CALL
1-888-303-5282

FOR ADVERTISING INFORMATION:
CALL 201 802.3020 OR
VISIT WWW.SYS-CON.COM

SYS-CON MEDIA
The World's Leading i-Technology Publisher

Next-Generation Web Presentation

WRITTEN BY
DAVID QUIMBY

XML enables adaptive experience

The future of Web presentation isn't high tech, it's high concept. And it's here today. It's not an information superhighway – it's an adaptive avenue.

Flexible, Web-based presentation engines are creating new options for Web presentation. As an integrated presentation and viewing environment that enables active presentation of Web content, a presentation engine uses XML-based presentation specifications to define the user experience. As a managed service that enables active presentation without reprogramming of content and without installation of software in the client or server environment, a presentation engine uses the Web services model to integrate with an array of host environments.

Web presenters and Web viewers alike are growing impatient with the limited navigation and presentation options of the existing Web environment. Static navigation methods like point-and-click don't hold viewers' attention sufficiently to draw them deep into Web content. And these primitive methods squander advances in digital bandwidth by diverting the "cognitive bandwidth" of viewers from content to navigation, producing "knowledge fatigue." A greater page-per-click ratio is essential to improving the satisfaction and productivity of the Web experience.

Dynamic presentation methods like animation and streaming media are available. But they're not feasible on all sites and whole sites... and they require installation of specialized components in the server and client environments. Web presenters want economical ways to deliver more content in a more dynamic mode. Despite the sophisticated experiences that high-end methods can produce, site producers are frustrated with the expense and inertia of high-end media development. They want the ability to reassemble content quickly and leverage their existing content investments for new and different audiences and situations. The Web awaits more efficient and dynamic presentation methods – implemented on a broad scale – to achieve its next stage of real growth in value and activity.

The landscape of Web presentation capabilities is changing. It offers more options – and more power – in the middle ground between these extremes. Web presenters can provide an interactive experience without imposing the burden of manual navigation. Presenters know site terrain better than viewers, and they can improve satisfaction and productivity by actively guiding the viewer experience. Adaptive presentation methods will be characterized by presentation engines and virtual sites. XML-based presentation models will enable adaptive presentation and expedite its adoption.

Adaptive Presentation: The Middle Ground

Some Web presenters are demonstrating their impatience with the extremes of high-end dynamic presentation and low-end static navigation by migrating slideshow content to the Web. They have the right motivation and they're moving in the right direction. But migrating conventional slideshow content is a transitional (and suboptimal) solution. This approach seems quick and easy, but it has serious limitations. Slideshow content is inherently static and monolithic, and it lacks the flexibility/agility of pages constructed from Web objects. Conventional slideshow applications aren't optimized for Web presentation, and they don't support essential features like auto-advance across multiple objects. The Web environment needs presentation solutions optimized for presentation of Web content in slideshow fashion – not suboptimized versions of desktop applications.

Viewers need more active content and presenters need more flexible content. Those requirements aren't supported by the current model of Web presentation or the current array of presentation environments. They demand a new presentation paradigm and a new set of presentation solutions. The middle ground between high end and low end is a vast wasteland, devoid of practical alternatives. It's where most of the progress will occur in the next development cycle (see Figure 1). Emerging alternatives are beginning to populate the territory. This segment will be characterized by relatively radical, discontinuous innovation and the emergence of disruptive technologies. The slideshow-migration trend described above signals the need for more dynamic and flexible content. The real answer to this need isn't migration of slideshow content to the Web. It's continued design and implementation of content

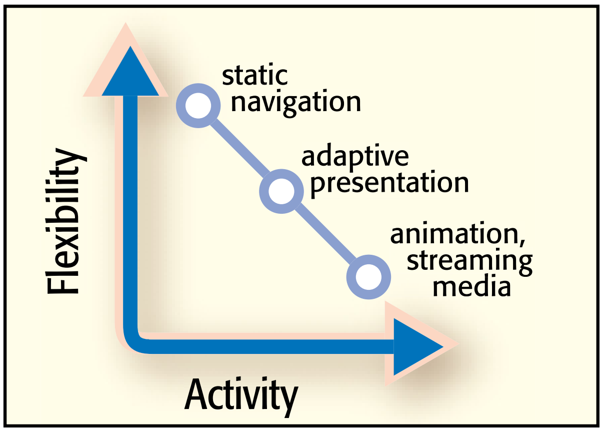


Figure 1 • Relative capabilities of presentation paradigms

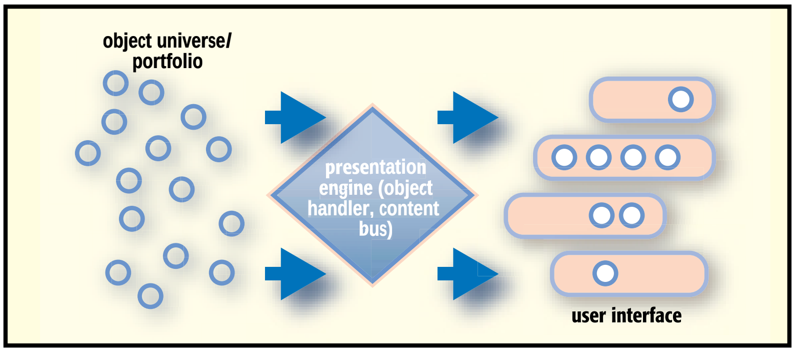
optimized for the Web – acceleration of that trend, including production of more graphically oriented content – and presentation of that content as objects in slideshow format.

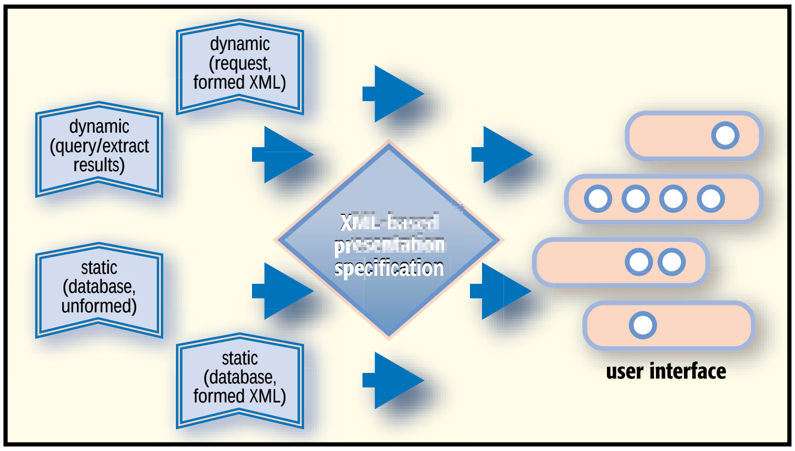
The static, hierarchical Web site as we know it will gradually disappear. Web sites will gradually become "vessels of objects" that can be presented dynamically and flexibly under various circumstances. Instead of being confined to a hierarchical, static identity, Web sites will become virtual "receptacles of potential." Presentation engines will unleash that potential, producing adaptive sites. A presentation engine includes server-side components for static and dynamic assembly of presentation objects and a client-side presentation component for accelerated delivery of objects to the user interface as directed by the assembly components (see Figure 2). Assembly and presentation components interact on the basis of presentation specifications, and XML will be the format of choice for most interactions.

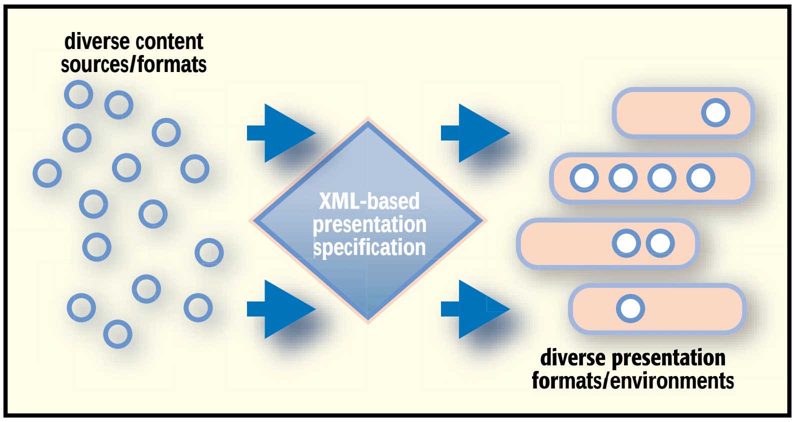
Just as current Web publishing technology produces coherent Web pages from individual objects, presentation engines produce coherent presentations (site "views") from individual pages (pages that are, in turn, constructed dynamically from individual objects). In the adaptive scenario, individual Web pages aren't fixed in location and limited in purpose – they become flexible resources for general use within the object portfolio. The rapid rise of site-based search illustrates this trend. Site-based search has proven to offer a more flexible method of content delivery than hierarchical navigation, and represents the majority of the search opportunity in general.

The combination of Web sites as vessels of objects (and Web pages as objects) plus emerging platforms for composing dynamic presentations from object portfolios defines the next generation of Web presentation. Presentation engines as "object handlers" will emerge in nearly every Web context, transforming inherently static content into dynamic experiences. Choreography will become a dominant motif as Web designers assemble the elements of their object universe to create the desired effect across time. Web content will dance – or walk, or run, or crawl – depending on the context and preferences of the presenter and viewer. Object handlers will serve as a "content bus" that operates on a layer separate from the content itself, nonintrusively organizing and accelerating it.

Content (and content development tools) will improve incrementally to leverage the new opportunities that presentation engines provide. But the presentation engines themselves will provide the most interesting developments. They'll leverage the Web services model to create presentations without reprogramming content, and without installing components in the server or client environments. They'll focus on assembling coherent, television-like viewing experiences from discrete objects. Just as component-based system development is object assembly across a landscape of functionality, next-generation Web presentation is object assembly across a landscape of time. Just as dynamic content frameworks like JSP and ASP produce modular presentations of content from individual objects at a point in time, presentation engines produce modular presentations of content from individual objects across a period of time. Just as HTML governs the presentation of individual Web pages, XML-based presentation specifications govern the presentation of entire page sequences (see Figures 3 and 4). The document-type definition (DTD) for adaptive presentation represents a universally available "presentation protocol" that any presentation service can access. XML-based presentation specifications enable seamless interaction of diverse server-side assembly components with client-side presentation components (see Listing 1).


object universe/ portfolio
presentation engine (object handler, content bus)
user interface
Figure 2 • Presentation engines as content drivers


dynamic (request, formed XML)
dynamic (query/extract results)
XML-based presentation specification
static (database, unformed)
static (database, formed XML)
user interface
Figure 3 • Diversity of server-side assembly components


diverse content sources/formats
XML-based presentation specification
diverse presentation formats/environments
Figure 4 • XML-based interoperability of presentation environments

Broader Implications for XML and Web Services

Besides enabling interaction among the assembly and presentation components of a particular presentation engine, XML-based presentation specifications enable decoupling of content sets from presentation services. Any (XML-based) presentation environment can access any content set that conforms to the adaptive presentation DTD (see Listing 2).

The Web Service Choreography Interface (WSCI), an interface description language for complex, multicycle transaction sequences recently proposed by Sun Microsystems and BEA Systems, recognizes the need for definition of Web services that operate across a period of time instead of at a point in time. The WSCI experience might suggest an analogous Web Service Rendition Interface (WSRI) that governs single-cycle presentation sequences (see Table 1).

```
<?xml version="1.0"?>

<!DOCTYPE adaptive-rendition SYSTEM "adaptive-rendition.dtd">

<adaptive-rendition>

  <presenter-definition>       adaptivestudio                                                          </presenter-definition>
  <presenter-mode>             static-formed                                                           </presenter-mode>
  <viewer-definition>          adaptivestudio                                                          </viewer-definition>

  <rendition-title>            3M Innovation - Technology Portfolio                                    </rendition-title>

  <element-duration>           5                                                                       </element-duration>
  <sequence-rotation>          1                                                                       </sequence-rotation>
  <control-mode>               foreground                                                              </control-mode>
  <process-level>              single                                                                  </process-level>
  <viewing-mode>               full+no control                                                         </viewing-mode>
  <media-level>                http://www.3m.com/audio/3minnovation.asf                                </media-level>
  <launch-mode>                internal                                                                </launch-mode>
  <preview-mode>               automatic                                                               </preview-mode>
  <action-mode>                forward                                                                 </action-mode>
  <pause-mode>                 automatic                                                               </pause-mode>
  <focus-mode>                 automatic                                                               </focus-mode>
  <termination-mode>           close                                                                   </termination-mode>

  <url1>                       http://www.3m.com/tech/lightmanagement/                                 </url1>
  <url2>                       http://www.3m.com/tech/lightmanagement/products_overview.jhtml          </url2>
  <url3>                       http://www.3m.com/tech/lightmanagement/products_tech.jhtml              </url3>
  <url4>                       http://www.3m.com/tech/lightmanagement/products_creative.jhtml          </url4>
  <url5>                       http://www.3m.com/tech/filmsolutions/                                   </url5>
  <url6>                       http://www.3m.com/tech/filmsolutions/strength.jhtml                     </url6>
  <url7>                       http://www.3m.com/tech/filmsolutions/clarity.jhtml                      </url7>
  <url8>                       http://www.3m.com/tech/filmsolutions/stability.jhtml                    </url8>
  <url9>                       http://www.3m.com/tech/fuelcells/                                       </url9>
  <url10>                      http://www.3m.com/tech/fuelcells/our_products.jhtml                     </url10>
  <url11>                      http://www.3m.com/tech/fuelcells/our_expertise.jhtml                    </url11>
  <url12>                      http://www.3m.com/tech/fuelcells/our_availability.jhtml                 </url12>

  </adaptive rendition>
```

Listing 1 • XML-based presentation specification with external DTD

Application Development and Market Acceptance

Existing Web browser and database technology enables a standards-based environment that accommodates either static, manually defined content or dynamic, automatically defined content. Dynamic formats include both query results and extract results. Presentation specifications include an array of presentation objects and an array of presentation attributes that govern presentation behavior. An emerging XML-based presentation model will enable a universe of presentation objects to interact seamlessly with a universe of presentation environments.

This new presentation paradigm will establish entirely new presentation possibilities. Because presentation engines are economical and easy to implement, they extend the potential for dynamic presentation to all sites and whole sites. Presentation engines will become the PC of Web presentation by making dynamic presentation widely available. As dynamic presentation becomes widely available, it will become pervasive. Viewers will gradually experience the Web in a whole new way.

Because presentation engines interface easily with existing capabilities like search functions, e-mail applications, and content management systems, they'll also become the spreadsheet of Web presentation. Presentation engines will find a diverse, almost unimaginable array of applications. Besides presenting site and catalog tours, they'll present query results at both the site and portal levels and they'll bring life to e-mail presentation. Scrolling through long, tedious e-mail messages will give way to lively presentations of individual, Web-based objects. Presentation engines are equally capable in the Internet and the intranet/extranet environments, and they'll serve objects adeptly within commerce hubs, corporate libraries, and document management systems.

A cluster of new providers will emerge with various offerings differentiated by their inherent functionality, features, and architectural configurations. It would be no more logical for individual applications to develop dedicated presentation environments than it would be for those applications to spawn their own Web browsers. They'll leverage third-party presentation services that deliver their

	simple, single-cycle presentation sequences	complex, multicycle transaction sequences
dynamic, time-oriented, governing a period of time	WSRI	WSCI
static, space-oriented, governing a point in time	XSL HTML	

Table 1 • Relative scope/complexity of interface standards

```
<?xml version="1.0"?>

<!ELEMENT adaptive-rendition         (environment-definition, rendition-description, rendition-attributes, rendition-objects)>

<!ELEMENT environment-definition     (presenter-definition, presenter-mode, viewer-definition)>
<!ELEMENT rendition-description      (rendition-title)>
<!ELEMENT rendition-attributes       (element-duration, sequence-rotation, control-mode, process-level, viewing-mode,
                                     media-level, launch-mode, preview-mode, action-mode, pause-mode, focus-mode,
                                     termination-mode)>
<!ELEMENT rendition-objects          (url1, url2, url3, url4, url5, url6, url7, url8, url9, url10, url11, url12)>

<!ELEMENT presenter-definition       (#PCDATA)>
<!ELEMENT presenter-mode             (#PCDATA)>
<!ELEMENT viewer-definition          (#PCDATA)>
<!ELEMENT rendition-title            (#PCDATA)>

<!ELEMENT element-duration           (#PCDATA)>
<!ELEMENT sequence-rotation          (#PCDATA)>
<!ELEMENT control-mode               (#PCDATA)>
<!ELEMENT process-level              (#PCDATA)>
<!ELEMENT viewing-mode               (#PCDATA)>
<!ELEMENT media-level                (#PCDATA)>
<!ELEMENT launch-mode                (#PCDATA)>
<!ELEMENT preview-mode               (#PCDATA)>
<!ELEMENT action-mode                (#PCDATA)>
<!ELEMENT pause-mode                 (#PCDATA)>
<!ELEMENT focus-mode                 (#PCDATA)>
<!ELEMENT termination-mode           (#PCDATA)>

<!ELEMENT url1                       (#PCDATA)>
<!ELEMENT url2                       (#PCDATA)>
<!ELEMENT url3                       (#PCDATA)>
<!ELEMENT url4                       (#PCDATA)>
<!ELEMENT url5                       (#PCDATA)>
<!ELEMENT url6                       (#PCDATA)>
<!ELEMENT url7                       (#PCDATA)>
<!ELEMENT url8                       (#PCDATA)>
<!ELEMENT url9                       (#PCDATA)>
<!ELEMENT url10                      (#PCDATA)>
<!ELEMENT url11                      (#PCDATA)>
<!ELEMENT url12                      (#PCDATA)>
```

Listing 2 • DTD for adaptive presentation

content to the Web browser in standard formats, providing a critical mass of functionality for presenters and a consistent interface for viewers. Nearly every application that currently presents Web content in static mode will integrate with an object handler as a "content driver" to produce a more dynamic viewing experience.

Competition will emerge around various approaches to the new paradigm, but the landscape will be dominated by new entrants with fresh, original thinking that will bring flexible and low-cost alternatives to light. An XML standard for a common presentation specification will enable various "presentation brokers" to process their unique formats interchangeably. Presentation engines also support audio content, and they'll be adept at integrating aural and visual objects into seamless multimedia presentations. They'll also allow for sharing of control between presenter and viewer, and they'll provide rich, accessible annotation features that preserve viewer orientation.

Site viewers (Web viewers, content viewers) will become as common to the Web environment as document viewers (Adobe's Acrobat Reader) and media players (Macromedia's Flash, Real Networks' RealPlayer, Microsoft's Media Player). They'll make the typical experience of Web viewing more continuous – more like viewing television. The most interesting developments for viewers will be viewer-initiated composition and automatic generation of presentations without presenter intervention. Viewers will use simplified versions of presenter-oriented tools to compose dynamic presentations of their favorite Web content. And they'll invoke "automatic composition" functions to explore the depths of their favorite sites, effortlessly. The emergence of automatic composition will instantly unleash storehouses of existing Web content for dynamic viewing. With that development, the Web browser will graduate from its current status as a mere pointer to its ultimate role as a true browser.

These new presentation capabilities will be especially valuable to business users in the disciplines of knowledge management, corporate memory, business intelligence, decision support, project management, and operations management. Anywhere clusters of knowledge accumulate, object handlers will serve as convenient instruments for aggregating the clusters into coherent and reusable "knowledge objects" and "learning objects." The ability to create shared experiences by building and preserving clusters of Web objects – as quickly and easily as using e-mail and word-processing applications – will mitigate information overload and transform the potential for groups to engage in collaborative design and organizational development. An adaptive, distributed approach to interactive learning environments will surpass the current, monolithic approaches that co-locate the integrative principle (library systems) and the productive principle (book publishing). Although adaptive presentation of Web objects will find a beachhead in interactive learning and knowledge management applications, it will enter the mainstream and impact the consumer segment in the realms of interactive journalism and interactive commerce. Adaptive presentation will shift the emphasis in commerce from "selling" to "learning," recognizing the goal-driven nature of consumer behavior and introducing a new era of "commerce as learning."

Despite the ultimate value of manual composition, the compelling experience of automatic composition will be the firestarter for adaptive presentation. It will bring rapid acceptance of the new medium among Web viewers. Viewer acceptance will promote adoption of the new paradigm by Web presenters, and Web viewers will find ever-increasing amounts of precomposed content at their disposal. As the Semantic Web gains momentum and unified classification of Web content becomes more pervasive, it will feed the "object appetites" of presentation engines and further stimulate their adoption.

Author Bio

David Quimby is the founder and CEO of Adaptive Avenue Associates, Inc. Adaptive Avenue is a provider of next-generation Web presentation services and the developer of Adaptive Studio™, an XML-based rendition broker. You can learn more about adaptive presentation in general, Adaptive Avenue in particular at www.adaptiveavenue.com.

DQUIMBY@ADAPTIVEAVENUE.COM

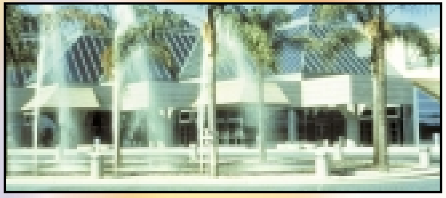
International Web Services Conference & Expo

Web Services Edge WEST 2003

web services EDGE conference&expo

SEPT. 30 - OCT. 2, 2003

Santa Clara, CA



EXTENDING THE ENTERPRISE WITH WEB SERVICES THROUGH JAVA, .NET, WEBSPHERE, MAC OS X AND XML TECHNOLOGIES

Featured technologies and topics will include:

- Focus on Web Services
- Focus on XML
- Focus on Mac OS X
- Focus on Java
- Focus on .NET

XML

WebSphere

Microsoft .net

Mac OS X

COMING IN 2004

web services EDGE conference&expo

BOSTON

FEBRUARY 24-26, 2004

Over 100 participating companies will display and demonstrate over 300 developer products and solutions.

Over 2,000 Systems Integrators, System Architects, Developers, and Project Managers will attend the conference expo.

Over 60 of the latest sessions on training, certifications, seminars, case studies, and panel discussions will deliver REAL World benefits, the industry pulse and proven strategies.

CONTACTS:

Exhibit & Sponsorship	Conference & Education
GRISHA DAVIDA	MICHAEL LYNCH
201 802-3004	201 802-3055
grisha@sys-con.com	mike@sys-con.com

www.sys-con.com

Web Services Edge WEST 2003

WEB SERVICES EDGE CONFERENCE & EDUCATION

web services EDGE conference&expo

SEPT. 30 - OCT. 2, 2003
Santa Clara, CA

WEB SERVICES TECHNOLOGY

Presentations will include discussions of security, interoperability, the role of UDDI, progress of the standards-making bodies, SOAP, and BPM. Case studies cover the design and deployment of Web services in the marketplace.

Sessions will focus on:
- Interoperability
- Enterprise Networks
- Web Services Management
- Web Services Standards
- Web Services Orchestration
- Security (WS-Security, SAML)
- BPEL4WS
- UDDI: Dead or Alive?
- ebXML & Web Services
- EAI & Web Services
- RPC vs. Messaging: Uses and Differences
- User Interfaces for Web Services
- Web Services Best Practices
- Service Oriented Architecture

JAVA TECHNOLOGY

The Java Track features presentations aimed at the beginner, as well as the seasoned Java developer. Sessions will explore the whole spectrum of Java, focusing on J2EE, application architecture, EJB & J2ME. In addition the Track will cover the latest in SWT, Ant, JUnit, open source frameworks, as well as an in-depth look into the vital role that Java is playing in building and deploying Web services.

Sessions will focus on:
- Enterprise Java 1.4
- Ant Applied in "Real World" Web Services
- Developing Application Frameworks w/SWT
- Empowering Java and RSS for Blogging
- JUnit: Testing your Java w/JUnit
- JDK1.5: The Tiger
- Simplifying J2EE Applications
- Using IBM's Emerging Technologies Toolkit (ETTK)
- Apache Axis
- Meeting the Challenges of J2ME Development
- Integrating Java + .NET
- Squeezing Java

XML TECHNOLOGY

Presentations will focus on the various facets of XML technologies as they are applied to solving business computing problems. Sessions will include emerging standards in XML Schemas, XML repositories, industry applications of XML, applying XML for building Web services applications, XML/XSLT/XQuery-based programming using Java/.NET, XML databases, XML tools and servers, XML-based messaging, and the issues related to applying XML in B2B/EAI applications. The XML Track is geared for audiences ranging from beginners to system architects and advanced developers.

Sessions will focus on:
- XML Standards & Vocabularies
- Introduction to XForms
- Securing Your XML and Web Services Infrastructure
- XQuery Fundamentals: Key Ingredient to Enterprise Information Integration
- XML and Enterprise Architecture: Technology Trends
- Standards-Based Enterprise Middleware Using XML/Web Services
- XML and Financial Services
- Canonical Documents for Your Business: Design Strategies
- XPath/XSLT 2.0: What's New?
- XML Schema Best Practices
- XML in EAI, Enterprise Portals, Content Management

MAC OS X

OS X represents a new wave of operating systems. It combines the ease of use of a Mac with the power of Unix. Sessions in this track will highlight the use of the Mac OS X platform in applications and Web services development, deployment and management.

Sessions will focus on:
- Introducing OS X (Panther): What's New?
- Quick Applications using AppleScript
- Enterprise Java and OS X
- Developing Web Services Using WebObjects
- Xserve: Ease of OS X and Power of Unix
- Introducing Quartz: 2D Graphics for Apple
- OS X for the Unix Developer
- Securing OS X Applications
- Java and OS X: A Perfect Marriage
- Programming Rich User Interfaces Using Cocoa

.NET TECHNOLOGY

Presentations will explore the Microsoft .NET platform for Web services. To the average developer, it represents an entirely new approach to creating software for the Microsoft platform. What's more, .NET development products - such as Visual Studio .NET - now bring the power of drag-and-drop, GUI-based programming to such diverse platforms as the Web and mobile devices.

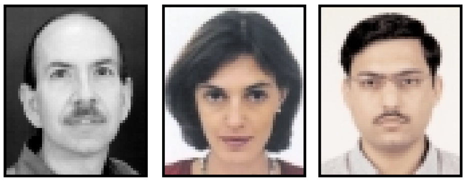
Sessions will focus on:
- ASP.NET
- Security
- VB.NET
- .NET and XML
- Smart Device Extensions for VS.NET
- Best Practices
- Shared Source CLI
- .NET Remoting
- Smart Devices in Health Care Settings
- Mobile Internet Toolkit
- ROTOR
- Portable .NET
- ASP.NET Using Mono
- Using WSE with IBM's WSTK
- GUI applications Using Mono
- Portals – Windows Sharepoint Services/Sharepoint Portal Server
- Windows Server 2003 and IIS 6
- .NET and Java Interoperability
- Distributed .NET for Financial Applications
- Developing C# with Eclipse

For more information visit
www.sys-con.com
or call
201 802-3069

PRODUCED BY SYS-CON EVENTS

MEDIA SPONSORS:

WebServices Journal · JAVA Developer's Journal · WebSphere Developer's Journal · XML Journal · .NET Developer's Journal · WebLogic Developer's Journal · LinuxWorld Magazine · wireless Business&Technology · Linux Business Week · CF Advisor · ColdFusion Developer's Journal · PowerBuilder Developer's Journal · market wire · HSP Street.com · OASIS · SD Times · LinuxWorld.com · SAMS · RealIndustry



WRITTEN BY **MICHAEL CAREY, DANIELA FLORESCU & NITIN MANGTANI**

Integrating Enterprise Information on Demand with XQuery – Part 2

XQuery for EII...the saga continues

In Part I of this article (*XML-J*, Vol. 4, issue 6), we introduced the enterprise information integration (EII) problem and explained how the XML query language XQuery and related technologies – specifically XML, XML Schema, and Web services – are central to enabling this age-old problem to be successfully addressed at last.

We provided a technical overview of the XQuery language and presented a simple "single view of Customer" example to illustrate XQuery's role in the EII domain. The example was based on an electronics retailer that wanted to share customer information across three portals – portals for customer self-service, credit approval, and product service. The information to be integrated resided in a variety of back-end information sources, including two relational database management systems, an SAP system, and a Web service.

In this article, our XQuery/EII saga continues. In this installment, we look at how EII relates to two other technologies designed for integration tasks, namely enterprise application integration (EAI) and extract-transform-load (ETL) tools. We also take a brief look at BEA Liquid Data for WebLogic, an XQuery-based EII offering, and discuss how XQuery and Liquid Data were put to use recently in a telecommunications-related customer project.

What About EAI?

Given the industry buzz around EAI today, a natural question about EII is "so why bother?" That is, why isn't a modern EAI solution alone – for example, a workflow engine with XML-based data transformation capabilities – sufficient to solve the EII problem? The answer is, in principle, that EAI is in fact sufficient to solve the EII problem. A developer could always choose to hand-build a set of workflows, writing one workflow per application-level "query" to deliver the desired information back to the calling applications. In the example from Part I of this article, three hand-tailored workflows could instead be written to provide information retrieval capabilities comparable to our XQuery-based solution. But is that the best approach, in terms of development time and maintenance cost?

The basic question here is when to use a declarative query language (XQuery in the case of modern EII) versus constructing code in a procedural language (a workflow language in the case of EAI). The lessons from the relational database revolution are clear: When applicable, a declarative approach offers significant advantages. Instead of hand-constructing a "query plan" (EAI workflow) to extract the needed data from each of the data sources in some manually predefined order, the EII approach allows a single, smaller, and simpler declarative query to be written.

The resulting benefits should be obvious. First, the user does not need to build each query plan by hand, which could involve a considerable effort. Instead, the user specifies (when defining the core view) what data sources are relevant and what logical conditions relate and characterize the data to be retrieved. Second, queries can be optimized automatically by the EII middleware, resulting in an optimal query execution plan (order of accessing the sources, queries or methods to extract the data, etc.) for each different query. For example, using EAI, one central workflow could be written to retrieve all of the customer information in Part I's example, and then other workflows could be written to first call this workflow and then further filter the results. However, in the EII approach, the query processor will (for each query) prune out irrelevant data sources as well as push SQL selection conditions (such as only retrieving "Open" support cases in Listing 2 of Part I) down to any RDBMS data sources. Third, as the data sources change over time in terms of their schemas, statistics, or performance, the EII user will not be forced to rewrite all of his or her queries. Simply maintaining each base view query and re-optimizing the other queries will adapt their query execution plans to the new situation. In contrast, in the case of EAI, many workflows would have to be rewritten to handle most such changes.

There really isn't an either/or choice to be made between EAI and EII at all. Both technologies have critical roles to play in an overall enterprise integration solution. These technologies are complementary: EII provides ease of data integration, while EAI provides ease of process integration. EII is appropriate for composing integrated views and queries over enterprise data. EAI is the appropriate technology for creating composite applications that orchestrate the functional capabilities of a set of related but independent applications, Web services, etc. Moreover, EII can be used to handily augment EAI in scenarios where workflows need to access integrated data views. For example, if our electronics retailer wanted its order process to offer free shipping to customers who have ordered more than $1,000 of goods during the year and who have accumulated more than 5,000 reward points, the integrated view of customer from Part I could be used to easily access the relevant information from within the order entry workflow.

What About ETL?

Another technology related to EII is ETL. In fact, ETL tools are designed precise-

AUTHOR BIO

Michael Carey is the architect for BEA Liquid Data for WebLogic. His previous jobs include three years at IBM working on various DB2-related projects and 12 years as a professor at the University of Wisconsin-Madison. He is a recognized expert on database system architectures and performance evaluation.

Daniela Florescu is an editor for the W3C XQuery language. She works at BEA Systems on the Liquid Data team and was the architect of the XQuery query processor used for the data transformation engine of WebLogic Integration 8.1. She is widely known for her past work on data integration and query processing.

Nitin Mangtani is currently the technical program manager for BEA Liquid Data for WebLogic. As a founding member of the Liquid Data team, Nitin has been instrumental in defining the overall product vision as well as shipping Liquid Data. Prior to joining BEA, he worked on distributed order management systems at i2 Technologies.

ly for the purpose of integrating data from multiple sources. These tools are therefore another category of software that naturally leads to a "why bother with EII?" question – why isn't ETL technology the answer? As you'll see, the answer is again that both technologies have their place in modern IT architectures.

ETL tools are designed for use in moving data from a variety of sources into a data warehouse for offline analysis and reporting purposes. As the name suggests, ETL tools provide facilities for extracting data from a source; transforming that data into a more suitable form for inclusion in the data warehouse, possibly cleansing it in the process; and then loading the transformed data into the warehouse's database. Typical ETL tools are therefore focused on supporting the design and administration of data migration, cleansing, and transformation processes. These are often batch processes that occur on a daily or weekly basis.

Data warehouses and the ETL tools that feed them are invaluable for enabling businesses to aggregate and analyze historical information. For example, our electronics retailer might very well want to keep track of customer data, sales data, and product issue data over a period of years in order to analyze customer behavior by geographic region over time, improve their credit card risk model, and so on. A data warehouse is the appropriate place to retain such data and run large analytical queries against it, and ETL technology is the right technology today for creating, cleaning, and maintaining the data in the warehouse. However, ETL is not the right technology for building applications that need access to current operational data – it doesn't support the declarative creation of views or real-time access to operational data through queries.

For applications that need to integrate current information, Part I of this article showed how XQuery can be used to declaratively specify reusable views that aggregate data from multiple operational stores and how XQuery can be used to write XML queries over such integrated views. We also explained how standard database query processing techniques, including view expansion, predicate pushdown, and distributed query optimization, can be applied to XQuery, making XQuery-based EII an excellent technological fit for such applications.

Clearly, both ETL and EII technologies have important roles to play in today's enterprise. ETL serves to feed data warehouses, while EII is an enabler for applications that need timely access to current, integrated information from a variety of operational enterprise data sources. As with EAI, there are also cases where the two technologies come together. As one example, an ETL tool could be used to help create and maintain a cross-reference table to relate different notions of "customer id" for use in creating XQuery-based EII views across different back-end systems. As another example, an ETL-fed data warehouse could be used to build a portal for analyzing the historical behavior of a company's top customers, with an EII tool used to allow click-through inspection of the customers' purchases in the past 24 hours.

Putting XQuery-Based EII to Work

For the reasons discussed in this article, XQuery-based EII middleware is an emerging product segment that promises to deliver the tools and technology needed in this important space. One commercially available XQuery-based middleware product is BEA Liquid Data for WebLogic. Liquid Data is capable of accessing data from relational database management systems, Web services, packaged applications (through J2EE CA adapters and application views), XML files, XML messages, and, through a custom function mechanism, most any other data source as well. For illustration purposes, the architecture of Liquid Data is depicted in Figure 1. Liquid Data provides default XML views of all of its data sources and provides an XQuery-based graphical view and query editor for use in integrating and enhancing information drawn from one or more data sources. It includes a distributed query processing engine as well as providing advanced features such as support for query result caching and both data-source-level and stored query–level access control.

As a final example of the applicability of XQuery to enterprise information integration problems, we'll describe an actual customer integration exercise where Liquid Data was put to use. In that project, a large telecommunications vendor wanted to create a single view of order information for one of its business divisions. The goal of the project was to make integrated order information available to the division's customers (other businesses) through a Web portal, enabling their customers to log in and check on the status of their orders, as well as making information available to the division's own customer service representatives.

The division had data distributed across multiple systems, including a relational database containing order summary information and two different order management systems. Order details were kept in one or the other of the two order management systems, depending on the type of order. Functionality-wise, a limited view of order details was provided through the customer order status portal that the division built using Liquid Data, whereas customer service representatives were permitted to see all of the order data through their portal. In both cases, it was possible to search for order information by various combinations of purchase order number, date range, and order.

The use of XQuery-based EII technology enabled the customer to complete their portal project in much less time than they had expected it to take with traditional technologies, and their total cost of ownership was also lower due to the reusability of Liquid Data assets and the low cost of maintenance enabled by EII.

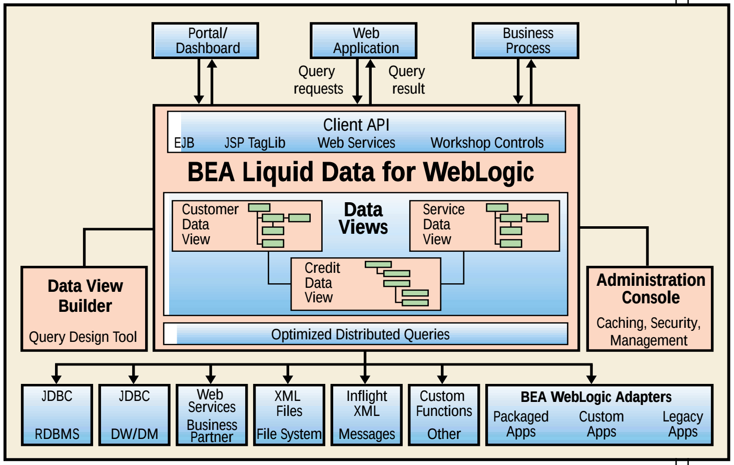


Figure 1 • Liquid Data architecture

Summary

In this article, we have explained how XQuery is beginning to transform the integration world, making it possible to finally tackle the enterprise information integration problem where past attempts have failed. In Part I we provided an overview of XQuery and illustrated how it could be used to integrate the disparate information sources of a hypothetical electronics retailer. In Part II we discussed the relationship of EII to EAI and ETL technologies and then briefly presented BEA's XQuery-based EII product and described one of the customer projects in which it was used.

MCAREY@BEA.COM

DANIELAF@BEA.COM

NITINM@BEA.COM

WRITTEN BY **TROY TOLLE**

Using Multiple URIResolvers for the Same Stylesheet

Added flexibility opens new opportunities

A URIResolver, an interface defined in the javax.xml.transform package, is used to process a URI and create a Source object out of it. All Java developers working with XSLT have to decide which URIResolver they're going to use to resolve the URI.

In small examples and code snippets this is very simple. However, when working with large systems that are styling dynamic content, the decision on which URIResolver to use can be a very important one that will help create a more flexible and scalable application. This article will show how a URIResolver is utilized to resolve a URI and how to create a URIResolver that will allow these references to be resolved differently within the same stylesheet.

When we develop with Java and XSLT, a URIResolver is used to perform this function on the XSLT elements xsl:import and xsl:include as well as the XSL function document(). Commonly, relative URIs are used in these pieces of XSLT to reference other stylesheets or node sets. This will work beautifully if the developer is always aware of the base URI for the context in which he or she is currently developing. Keeping track of this can become complicated as more stylesheets are added to a system in different paths. For example, consider the folder structure shown in Figure 1.

If we're developing on a Windows box, we can import 02.xsl from 01.xsl

```
<xsl:stylesheet version="1.0" xmlns:
xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:import href="c:/root/02.xsl"/>
```

and 03.xsl from 02.xsl.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
<xsl:import href="c:/root/folder01/03.xsl"/>
```

This works great for development, but as soon as we move to the testing or production environment, where the base of the directory structure and platform is not the same, we have to go through and change all of the xsl:import, xsl:include, and document() calls in each of the XSL files. Of course, this is not an acceptable solution either because once the code is approved for production, no code modifications should be necessary to move from the testing to the production environment.

The URIResolver interface helps solve these problems. We can create a URIResolver that will always know where it should find the proper stylesheet. Before we see how this is done, let's make sure that we understand the URIResolver interface.

The URIResolver Interface

The URIResolver interface exists in the javax.xml.transform package and is part of the JAXP API (see Figure 2). It has a single method, resolve, that any implementing class must provide. This method takes two String parameters, href and base, and returns a Source object. The href parameter corresponds to the href in the xsl:import or xsl:include elements. It is also the first parameter in the document() function. For example, in an import from a stylesheet <import href="02.xsl"/> the href parameter passed to the resolver is the string 02.xsl. The base parameter is not as straightforward. If the xsl:import, xsl:include, or document() call is made from within the primary document, then the base is the base URI of that document or the second parameter in the case of document(). However, if the message is sent from an external entity, then the base is the URI of that calling external entity. Let's consider the previous example of stylesheets 01.xsl, 02.xsl, and 03.xsl. The base for the import element in 01.xsl of <import href="c:/root/02.xsl"/> is c:/root/. Because we import 03.xsl from 02.xsl we are importing from an external entity, which makes the base for the import element <import href="c:/root/folder01/03.xsl"/> have a base of c:/root/02.xsl.

Now that we understand the URIResolver interface, we can put it to use in helping us resolve our problem. We would like to build a URIResolver that is smart enough to know the path to all of our stylesheets. Knowing that we will always deploy a WAR file helps us with a quick potential solution. If we can figure out what the path to our application is dynamically on any server, then we can solve the problem (see Listing 1).

With this URIResolver we are able to dynamically determine the path to our stylesheet using the ServletContext. When transforming, we can set this to be our URIResolver with:

```
TransformerFactory factory =
  TransformerFactory.newInstance();
factory.setURIResolver(new
  ServletContextURIResolver
  (getServletContext(), "/style"));
```

Instead of worrying about changing references to the stylesheets, we can simply rely on the URIResolver to appropriately find the stylesheet documents. When the resolver is invoked, the ServletContext is used to find the real path to the directory containing the stylesheet documents. When the resolve

AUTHOR BIO
Troy Tolle is a consultant with CrossLogic Corporation. He has a master's degree in computer science from North Carolina State University and is a Java Sun Certified Programmer. His work with CrossLogic includes architecting and implementing J2EE- and XML-based software solutions.

method is called, the href will be appended to this path and a Source object created from that combined path. With this URIResolver in place the xsl:import, xsl:include, and document() need only to reference the relative path from the stylesheet directory. For example, if our stylesheets exist in the style directory on our application server, we can reference a stylesheet “mystyle.xsl” in the following manner: <import href="/mystyle.xsl"/>. We no longer have to specify the URI from the root of our system. We can start with a directory structure that we know will exist.

Using this solution for the problem of locating different stylesheets works great, but it does limit us to only one URIResolver for a transformation. Being able to reference multiple URIResolvers could be useful for retrieving stylesheets and documents from several different locations and for resolving custom stylesheets on the fly. How can we use multiple URIResolvers within one transformation of a particular stylesheet? There are several possible solutions to this problem, but we’ll investigate one particular design that involves the ability to switch URIResolvers based upon the href and base passed to the installed URIResolver. This allows the stylesheet programmer to place key values in the xsl:import, xsl:include, and document() calls from the stylesheet that can cause different URIResolvers to be used. Consider the diagram in Figure 3.

Instead of using the ServletContext URIResolver, we will use the Dispatching URIResolver. The purpose of this DispatchingURIResolver is to simply dispatch the work to other URIResolvers. It does not do any resolving on its own. To accomplish this task, it makes use of two other classes, the DelimitedKeyParser and the BuilderURIResolverFactory. When the resolve method is called on the DispatchingURIResolver, it will use the DelimitedKeyParser to determine whether a special key has been specified for locating a unique resolver. If a key was found, then the BuilderURIResolverFactory creates a URIResolver that it associates with the key. If a key was not found, then the BuilderURIResolverFactory simply creates the default URIResolver. Assuming that there would be only a few cases in which a unique URIResolver would be needed within an application, it’s important to provide this default case. This allows an application that uses a single URIResolver to use this pattern and keep itself open to future changes. The single URIResolver that is used would be installed as the default URIResolver and created every time. Once the factory is asked to create the appropriate URIResolver, the DispatchingURIResolver simply dispatches the work off to that new URIResolver. The resolve() method of the Dispatching URIResolver is shown in Listing 2.

Now that we know what’s involved in dispatching to the proper URIResolver, let’s take a look at an implementation of the DelimitedKeyParser and BuilderURIResolverFactory.

Let’s assume that we’ll use the ServletContextURIResolver as our default URIResolver. This means that, for the most part, our stylesheet imports will look like <import href="/02.xsl"/>. If our application must also access stylesheets that exist elsewhere on the Internet, we will need to use a different resolver. We’ll call this resolver a RemoteStyleURIResolver. It will return the source of a stylesheet that exists on its server (see Figure 4).

We’ll still need to specify an href to the proper stylesheet, but we must also identify it as a remote stylesheet. If we create an import statement that looks like <import href="remote.style#/style.xsl" />, the DelimitedKeyParser will look for a # character delimiting the key and the href parts. In this example, the key would be remote.style and the href to be passed on to the URIResolver should be /style.xsl. The getKey method assumes that if it does not find the # character, then the key is the entire href. If this is incorrect, then the factory will not find any URIResolver associated with it and the default URIResolver will still be requested. The getHref method returns all of the String after the # character, or the entire String if no # is found. Finally, the getBase simply returns the entire base since we’re not using it to determine any information in this example. If another delimiter is needed for the application instead of the default # character, then the new character delimiter can be set on the DelimitedKeyParser with the setDelimiter() method.

The BuilderURIResolverFactory has the responsibility of creating the appropriate URIResolver for a given key. To allow the concrete implementations to be dynamic if needed, the IURIResolverBuilder interface is used. This allows the creation of each URIResolver to be done in any manner that the application sees fit. The IURIResolverBuilder interface has a single method, build(), that is called from the BuilderURIResolverFactory create() and createDefault() methods. Adding IURIResolverBuilders to the

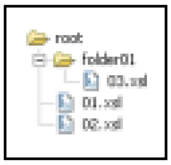


Figure 1 • Folder structure

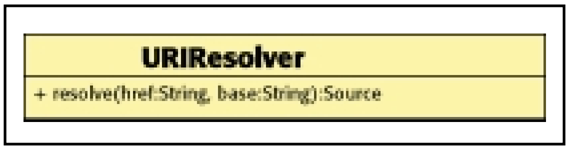
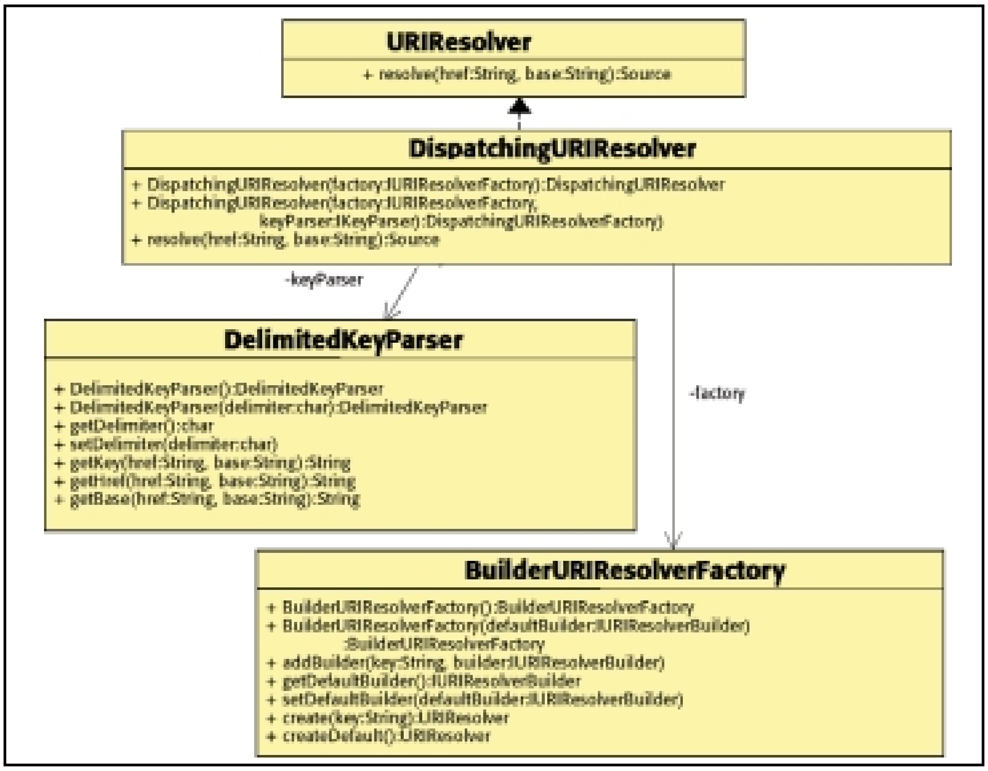


Figure 2 • URIResolver interface


URIResolver
+ resolve(href:String, base:String):Source
DispatchingURIResolver
+ DispatchingURIResolver(factory:IURIResolverFactory):DispatchingURIResolver
+ DispatchingURIResolver(factory:IURIResolverFactory, keyParser:IKeyParser):DispatchingURIResolverFactory)
+ resolve(href:String, base:String):Source
-keyParser
-factory
DelimitedKeyParser
+ DelimitedKeyParser():DelimitedKeyParser
+ DelimitedKeyParser(delimiter:char):DelimitedKeyParser
+ getDelimiter():char
+ setDelimiter(delimiter:char)
+ getKey(href:String, base:String):String
+ getHref(href:String, base:String):String
+ getBase(href:String, base:String):String
BuilderURIResolverFactory
+ BuilderURIResolverFactory():BuilderURIResolverFactory
+ BuilderURIResolverFactory(defaultBuilder:IURIResolverBuilder):BuilderURIResolverFactory
+ addBuilder(key:String, builder:IURIResolverBuilder)
+ getDefaultBuilder():IURIResolverBuilder
+ setDefaultBuilder(defaultBuilder:IURIResolverBuilder)
+ create(key:String):URIResolver
+ createDefault():URIResolver
Figure 3 • Multiple URIResolvers

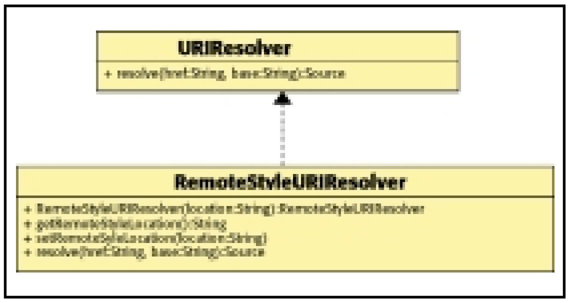

URIResolver
+ resolve(href:String, base:String):Source
RemoteStyleURIResolver
+ RemoteStyleURIResolver(location:String):RemoteStyleURIResolver
+ getRemoteStyleLocation():String
+ setRemoteStyleLocation(location:String)
+ resolve(href:String, base:String):Source

Figure 4 • RemoteStyleURIResolver

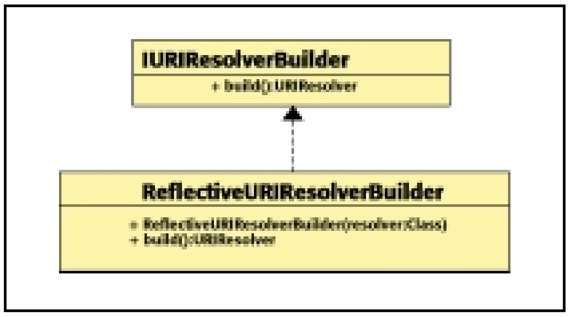

IURIResolverBuilder
+ build():URIResolver
ReflectiveURIResolverBuilder
+ ReflectiveURIResolverBuilder(resolver:Class)
+ build():URIResolver

Figure 5 • URIResolverBuilder

Builder URIResolverFactory is very simple. There are two methods provided on the BuilderURIResolverFactory that allow the addition of an IURIResovlerBuilder implementation. The addBuilder (String key, IURIResolverBuilder builder) method is very straightforward, taking String as the first parameter and a builder as the second. The key identifies the partnering builder. When the factory is requested to create a new URIResolver through the create() method, it is given a key. This key will be used to retrieve the builder that was associated with it through the addBuilder method. The default IURIResolverBuilder can be specified either at the creation time of the factory through the constructor, or with the setDefaultBuilder(IURIResolverBuilder builder) method. This builder will be used when the createDefault method is invoked.

A simple, concrete implementation of the IURIResolverBuilder interface is one that uses the same instance of the given builder for each invocation of the resolve method. The SingleInstanceURIResolverBuilder stores the URIResolver it's given at its creation and returns it every time the build method is called. Special caution should be taken when using this builder to ensure that the URIResolver that is kept is not affected by a multithreaded environment.

Another implementation of a IURIResolverBuilder is the Reflective URIResovlerBuilder, which is thread-safe by nature. It takes a URIResolver as a parameter to its constructor. When its build() method is invoked, a new instance of the URIResolver is created and returned. The URIResolver class that is given to the ReflectiveURIResolver must have a default constructor for the build() method to successfully execute. Using the IURIResolverBuilder interface allows not only the generic creation of URIResolvers (as we see with the ReflectiveURIResolverBuilder), but also the dynamic creation of URIResolvers based upon specific application data (see Figure 5).

We now have the ability to use two different URIResolvers for the same transformation of a single stylesheet. We can make this all work by installing our DispatchingURIResolver as the resolver for the transformations. The following example shows how we can use this DispatchingURIResolver with the ServletContextURIResovler as the default resolver and the RemoteStyleURIResolver used for any key found in the xsl:import, xsl:include, or document() calls using the key "remote:style" (see Listing 3).

This implementation is very simple, straightforward, and ready for immediate

XML-J ADVERTISERINDEX

ADVERTISER	URL	PHONE	PAGE
Altova	http://xmlj.altova.com/code	978-816-1600	36
Ektron	www.ektron.com/xmlj	603-594-0249	35
IBM	ibm.com/websphere/seeit		2
LinuxWorld Conference & Expo	www.linuxworldexpo.com	888-303-5282	6
LinuxWorld Magazine	www.sys-con.com	888-303-5282	15
CTIA	www.ctiashow.com		11
Mindreef	www.mindreef.com	603-465-2204	4
SYS-CON Media	www.sys-con.com/2001/sub.cfm	888-303-5282	27
SYS-CON Reprints	www.sys-con.com	201-802-3026	26
Web Services Edge West 2003	www.sys-con.com	201-802-3069	20

use. If the DelimitedKeyParser or the BuilderURIResolverFactory implementations do not meet the needs of an application, developers may choose to implement their own implementations. This is made possible by the use of the IKeyParser and IURIResolverFactory interfaces within the DispatchingURIResolver. While the provided BuilderURIResolverFactory and DelimiterKeyParser should meet the needs of most situations, the solution is flexible enough to take other forms because of the use of these interfaces (see Figure 6).

The use of the DispatchingURIResolver gives developers creating applications using XSLT the flexibility to use more than one URIResolver for a Transformer. This opens up the door for many possibilities, including pulling stylesheets from different sources and generating dynamic stylesheets.

TTOLLE@CROSSLOGIC.COM

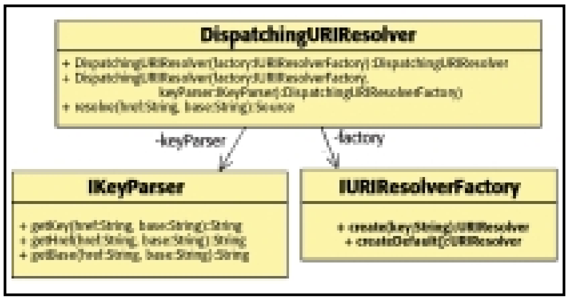

DispatchingURIResolver
+ DispatchingURIResolver(factory:IURIResolverFactory):DispatchingURIResolver
+ DispatchingURIResolver(factory:IURIResolverFactory, keyParser:IKeyParser):DispatchingURIResolverFactory)
+ resolve(href:String, base:String):Source
-keyParser
-factory
IKeyParser
+ getKey(href:String, base:String):String
+ getHref(href:String, base:String):String
+ getBase(href:String, base:String):String
IURIResolverFactory
+ create(key:String):URIResolver
+ createDefault():URIResolver

Figure 6 • DispatchingURIResolver

LISTING 1 •

```
import java.io.*;
import javax.servlet.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

public class ServletContextURIResolver
implements URIResolver {
 private String realPath = "";
 public ServletContextURIResolver(Servlet-
Context context, String virtualPath) {
  this.realPath = context.getRealPath(vir-
tualPath);
 }
 public Source resolve(String href,
String base) throws TransformerException
{
  try {
   return new StreamSource(new
File(this.realPath + href));
  } catch (Throwable t) { /* do nothing
*/ }
  return null;
 }
}
```

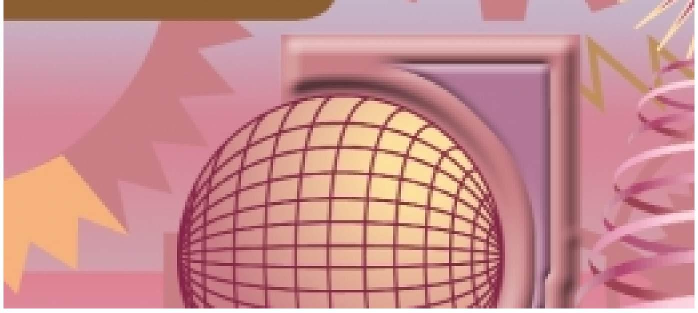
LISTING 2 •

```
public Source resolve(String href, String
base) throws TransformerException {
 IKeyParser parser = getKeyParser();
 URIResolver resolver = null;
 try {
  resolver =
getFactory().create(parser.getKey(href,
base));
 } catch (Throwable t) {
  try {
   resolver = getFactory().createDe-
fault();
  } catch (CreationException ex) {
   throw new TransformerException(ex);
  }
 }
 return
resolver.resolve(parser.getHref(href,
base), parser.getBase(href, base));
}
```

LISTING 3 •

```
TransformerFactory tFactory = Transformer-
Factory.newInstance();
ServletContext sContext = getServletCon-
text();
URIResolver dResolver = new ServletCon-
textURIResovler(sContext);
IURIResolverBuilder dBuilder = new Sin-
gleInstanceURIResolverBuilder(dResolver);
BuilderURIResolverFactory dFactory = new
BuilderURIResolverFactory(dBuilder);
URIResolver rsResolver = new
RemoteStyleURIResolver("http://www.cross-
logic.com");
dFactory.addBuilder("remote:style", new
SingleInstanceURIResolverBuilder(rsRe-
solver));
factory.setURIResolver(new Dispatchin-
gURIResolver(dFactory));
```

Download the Code
www.sys-con.com/xml



Multipass Validation with XSD and Schematron Part 1

WRITTEN BY
ERIC J. SCHWARZENBACH
& DAVID KERSHAW

Two schemas may be better than one

If it is important that your XML documents are correct, catching mistakes early is, of course, much less costly than catching them later. This should not be news to any XML developer.

But "correct" often means more than just a simple validity test at the end of the development process. Today, no one schema language covers all the bases. Different languages offer different possible measures of correctness. In combination, multiple schemas may provide the rich structure an application or organization needs for success.

The most common of the newer schema languages is W3C XML Schema (XSD). Classwell Learning Group, a division of textbook publisher Houghton Mifflin, has put XSD at the center of their content-driven applications since 2001. While XSD is far more powerful than DTD for defining structure and data types, limitations remain. Classwell found that a rules-based schema language, Schematron, could provide additional validation rules to ensure consistency between classes of documents and to enforce conventions in the XSD schemas themselves. It has become particularly evident that, given XSD's notorious complexity, consistent and thought-out design patterns are crucial for the effective use of XSD. Classwell has been working to establish their own set of best practices for XSD development that can be monitored by a combination of validation tools.

As a rules-based system, Schematron takes an approach to validation that's distinctly different from that of XSD. When the two are paired together, Schematron adds significant prescriptive flexibility that complements XSD's stronger expression of structure. Schematron came out of research by Rick Jelliffe at Academia Sinica, but is interesting from a practical perspective because it provides capabilities today that will only be available in XML Schema 1.1 or 2.0 in the future.

This article shows how to build a productive framework for multipass validation within Altova's popular XMLSPY environment. The example we use is drawn from Classwell's use of Schematron rules to enforce best practices in XSD design. In designing a new layer for an already complex development process, we felt the issue of productivity was important. To reduce the chance of error and speed up complex validation, we leveraged XMLSPY's rich scripting capabilities. The XMLSPY Scripting Environment enabled us to add UI elements that controlled a flexible and manageable multilanguage validation process with minimal effort.

Defining Rules for Best Practice

Our goal was to enforce a short list of practices that experience taught us should be required patterns. For this article we have selected a subset of Classwell's XSD rules that demonstrate the concept, while keeping the examples simple.

- Ensure the root element is an extension of a base rootType, guaranteeing a certain minimum tagging of the content. XSD does not define which element is intended to be the document root, but we can assume it will be the first global element defined.
- Schema must be versioned (XSD allows a version attribute for tracking – enforce its use).
- XSD files that define much of the common tagging must be consistently used – for our example here, para.xsd and root.xsd.
- Disallow the use of the xs:string data type (xs:string allows text formatting we do not want within PCDATA – xs:token should be used instead since it rules out new lines, tabs, leading and trailing spaces, or runs of more than one space between words).
- Some elements are crucial to our system – when they are processed the system is particular about their use; however, XSD lets you "shadow" global element definitions. So include a rule disallowing the definition of certain elements locally, for our example, the element "locator".

These rules are really just a start. But in the interest of not overwhelming the reader with information specific to Classwell, we will stop with these five. No doubt every development group has its own similar rules.

Framework Requirements

Our goal for this article is straightforward: we want to demonstrate an easy path to multiple language validation within a common IDE using an example that makes sense in day-to-day work. With that aim in mind we kept our framework's requirements lightweight.

1. A user must be able to easily validate a document using an arbitrary number of validation commands associated with that document.
2. Each validation command needs to execute an arbitrary command-line process that could give feedback on any kind of validation.
3. The results of each validation command must be reported to the user.
4. The GUI must provide the following operations:
   - Set up a validation command
   - Remove a validation command
   - Set up a schema
   - Remove a schema
   - Add a command/schema pair to a document
   - Remove a command/schema pair from a document
   - Validate using the set of all command/schema pairs associated with the document

5. Commands must be able to be written with variables that are swapped for the schema file path and the XML document path before execution.
6. The solution must be reasonably easy to set up.

Neither of us saw a significant need to guarantee the order of validation at this time, so we left that off the list. However, our final implementation could easily be modified for strict adherence to an order of validation commands.

Overview of Tools Used

XMLSPY

Altova's XMLSPY product is in its fifth successful version this year. XMLSPY is the leading XML IDE and for most people doesn't need much further introduction.

The XMLSPY scripting environment, however, is less well known. Although tightly integrated with XMLSPY, the scripting environment is a separate Visual Basic–like tool used for forms-based customizations. Scripting projects have a choice between JavaScript and VB Script – we used JavaScript for this article, but either language offers a rich set of Windows components in addition to the XMLSPY COM API.

XMLSPY's API has three main concerns:
- The application object model (projects, views, documents, etc.)
- XML and file-processing functions (validate, save, generate schema, etc.)
- A DOM-friendly XML object model

We'll get into each of these areas, as well as leverage the scripting environment's forms and event-handling abilities. Also, we used Windows Scripting Host objects to interface with the file system and command-line processes.

Schematron

Schematron is a lightweight language wrapped around XPath. There are several Schematron tools freely available on the Internet. Of these we chose zvonSchematron (available at www.zvon.org).

Simply put, zvonSchematron is an XSLT file. Using zvonSchematron is a two-step process. After you have created your Schematron schema first transform it using the zvonSchematron.xsl file and your favorite XSLT processor – we used Altova's stand-alone XSLT Engine.

Then take the resulting XSLT file and apply it to your XML document. The final outcome is usually an HTML file that itemizes any lack of conformance to your schema. As you will see, within our framework it made more sense to dispense with the HTML and just output single lines of text. To avoid distraction we made that modification to the XSLT output of zvonSchematron, rather than modifying the generator.

Bridging the XMLSPY-Schematron disconnect

Out of the box, XMLSPY does not have special support for Schematron or multipass validation. This lack of support is not a big surprise; most XML tools also do not support either concept. What's good about XMLSPY is that its API makes it simple to add the capability we need using its scripting environment, as we did, or as an XMLSPY plug-in.

We created a framework for executing a set of command-line processes associated with a file and capturing the results. We stored the validation commands associated with an XML document in processing instructions (PI) within that document. Each PI holds a validation command name (mapped to an actual command by the user) and a schema name mapped to the path to a schema document.

Another common approach we did not take is to store Schematron in the appinfo elements of an XSD schema. We talked about taking this approach but decided that it had at least two drawbacks – it required a tighter coupling of XSD to Schematron than we wanted (remember that at Classwell the same Schematron rules validate a class of XSD), and we felt it would be a more difficult implementation to present in a short article.

The XMLSPY Scripts

We needed the following macros:
- ***SetupCommand:*** Sets up a named command in a commands file
- ***RemoveCommand:*** Removes a named command from the commands file
- ***SetupSchema:*** Sets up a named schema in a schemas file
- ***RemoveSchema:*** Removes a named schema from the schemas file
- ***AddValidation:*** Adds a command/schema pair to an instance document as a PI
- ***RemoveValidation:*** Removes a command/schema pair from an instance document
- ***Validate:*** Validates using the set of command/schema pairs found in the validate PI within the current document and reports the validation results

These macros drive forms and global functions. The user will run a macro by selecting its name on XMLSPY's Tools menu. The last one, Validate, will not have its own front-end form, but it will result in one of two simple forms popping up at the end of the validation process to report the outcome. After our project is complete, under a simple standard configuration which we will introduce, XMLSPY will offer each macro as a selection on the Tools menu without users having to do any further setup.

To facilitate our current implementation and future script reuse we located much of the programmatic work in global functions. This approach is a straightforward means of increasing readability. What is less well known is that XMLSPY scripting projects (.prj files, usually kept in the XMLSPY install directory) can be easily shared. So while the code in this article may not be quite ready for the corporate XMLSPY script library, we have provided it in a format that makes it easy for you to generalize it for reuse in your own projects.

Creating the macros

The best way to start is by stubbing out the macros. Open XMLSPY's scripting environment by clicking on the Tools menu and selecting "Switch to Scripting Environment…". In the scripting environment on the Project menu under the Modules folder you should see three items. Right-click on the item named "(XMLSpy Macros)" and select "Add Function".

In the New Function dialog type "Validate" and click "OK". If you again right-click on "(XMLSpy Macros)" and this time select "View Code" you see the code window on the right-hand side of the application. In the top right-hand corner of the code window there is a pull-down menu that shows the names of the available macros. "Validate" should be among those you see listed.

Each macro is simple. In general, they do three things: call a setup function, open a form, and call an error report function. The setup function is responsible for clearing global variables and making sure the macros are visible on the Tools menu. Each macro's form handles display and invokes global functions that do the heavy lifting. The error reporting call pops up a dialog if an error condition is present. In general the code you need to get this functionality is:

```
showMacros();
Application.ShowForm("form_name");
 reportError();
```

However, the Validate macro you just created is the exception in that it does not have its own form, so replace "Application.ShowForm("form_name");" with a call to a global function "validate".

Double-click on "(Global Declarations)" to bring up the code view of the global code space. Any functions or variables declared in this area are available to all macros, forms, and event handlers. For now, just stub out a validate function like this:

```
function validate() {
  //
// validation code goes here.
//
}
```

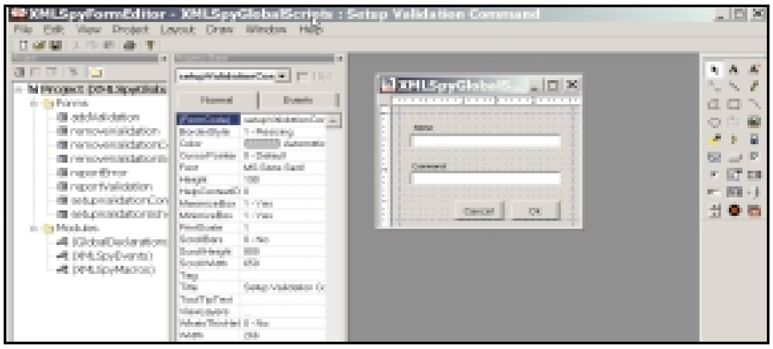
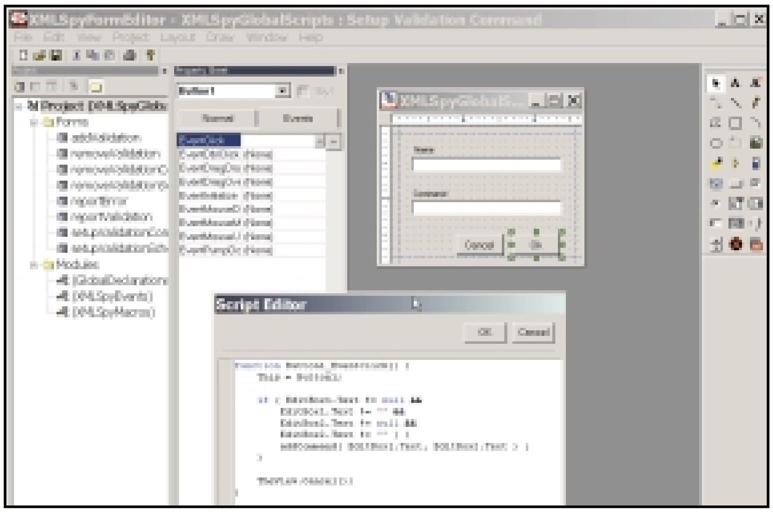


Figure 1 • Laying out the setupValidationCommand form



```
function Button_EventClick() {
    This = [illegible];

    if ( EditBox1.Text != null &&
         EditBox1.Text != "" &&
         EditBox2.Text != null &&
         EditBox2.Text != "" ) {
         addCommand( EditBox1.Text, EditBox2.Text );
    }

    TheView.Cancel();
}
```

Figure 2 • Adding event code to the OK button of the setupValidationCommand form

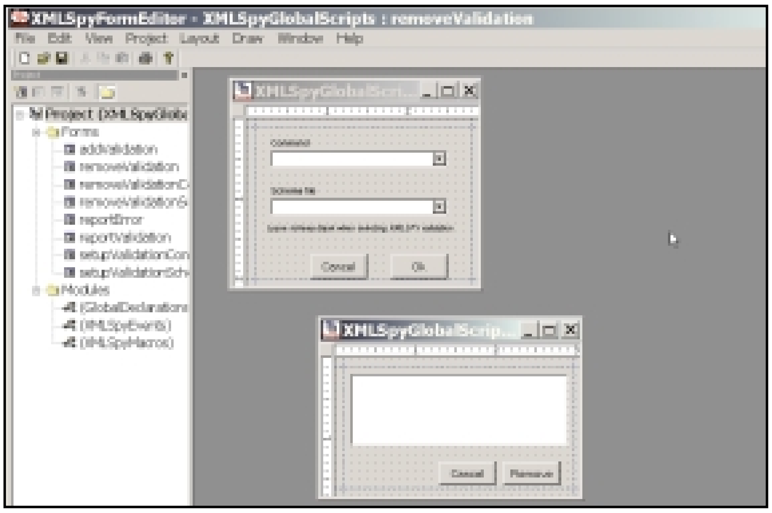


Figure 3 • addValidation and removeValidation forms

At the same time add a stub of the showMacros and reportError functions.

Now add the other macros we need using the same process. For each one add the three lines using the form name given here:

- ***Setup Command:*** setupValidationCommand
- ***Remove Command:*** removeValidationCommand
- ***Setup Schema:*** setupValidationSchema
- ***Remove Schema:*** removeValidationSchema
- ***Add Validation:*** addValidation
- ***Remove Validation:*** removeValidation

Building the forms

Creating the six forms you need is straightforward. Click on the Project menu and select "Add Form". A blank form will appear with a name something like "Form1". If the properties sheet is not open, click on the Layout menu and select "Properties…".

We kept track of validation commands and schemas in a pair of text files. Commands and schemas are managed separately so they can be more easily reused, thereby saving the user from having to retype the same command or file path. Each command or schema is listed on a line as a name-value pair. The name is used instead of the value in the forms and PI for readability.

Each of the forms for validation commands is similar to the analogous one for schemas. We will walk through the command forms together and leave you to create the matching schema forms independently (of course, you can always download and use our code, available at www.sys-con.com/xml/sourcec.cfm). Start with the Setup command. Click on the (Form Code) property of your new form and enter "setupValidationCommand". In the Title property enter "Setup Validation Command".

Now you need some form elements. To make sure you have access to the widgets you need click on the View menu, select "Toolbars" and then "Object Bar". Drag two buttons onto the form. Double-click on their text and change one to "Cancel" and one to "OK". In the property sheet of each set the Button Type property to "Cancel" and "OK" respectively.

Next drag two text boxes on the form. One of these will take the name of a command. The other will take a schema name. At this point the form's look and feel is essentially complete (see Figure 1).

To wire the form's elements together you need to add some code. Click on the Property Sheet's Events tab, and then click on the form's Cancel button. On the Property Sheet, next to "EventClick" where it says "(None)" click the mouse. A code dialog opens giving you a way to associate code with a mouse-click event on that button. The only behavior we need from this button is for it to close the form. Add the following line and click "OK":

```
TheView.Cancel();
```

The form's OK button will do a bit more work. Click on that button and then click on the same place in the Property Sheet's Events tab. The form needs to call an addCommand function with the Text properties of EditBox1 and EditBox2 as the arguments, as shown in Figure 2.

Let's keep things focused on the forms and macros for now. Just as you did for "validate" stub out an addCommand function in the global area. You will return to the global functions later.

Moving right along, create a second form and name it "removeValidationCommand". This form removes a named command from the commands file. Like the add form it needs two buttons, but in addition to those it only needs a pull-down select element. The select is filled from a global function that returns an array of command names. Add this code to the element's EventInitalize area:

```
 var cmds = getCommands();
for ( i = 0; i < cmds.length; i++ ) This.AddItem(
  cmds[i], i );
```

Then in the OK button's EventClick area add this code:

```
removeCommand( ComboBox1.GetText( ComboBox1.Selection ) );
TheView.Cancel();
```

Again stub out the getCommands and removeCommand functions in the global area.

The schema forms are basically the same as these command forms. Create them with the same steps, remembering to give their functions the appropriate names.

Of the four remaining forms, the two reporting forms are also virtually identical. reportError lists any error conditions found during a macro run. reportValidation shows the output, positive or negative, for each validation command. If a document is valid all commands will be listed as passed, otherwise this form will show the issues.

Starting with the reportError form, add an OK button and a list box. The list box shows each section of the validationError string on its own line. We decided to add ";" between each section of the message. In the EventInitalize area of ListBox1 you will break the string on that character with this code:

```
 if ( validationError == null ) {
  return;
 }
 var split = validationError.split( ";" );
 for ( i = 0; i < split.length; i++ ) This.InsertString(
  i, split[i] );
```

Create the reportValidation form in the same way. Then add the two report variables to the global area.

The final two forms you need are not significantly dissimilar from what you have already created. addValidation and removeValidation are responsible for adding and removing PI from the current document. Each PI holds the name of one validation command and one schema. When the validate function is called all of the validate PI in the current document are collected and each command is attempted. The forms look as shown in Figure 3.

The main new features of these forms are the following items. In the list box on removeValidation, the EventInitalize area holds:

```
 var names = getPIValues("validate");
 for ( i = 0; i < names.length; i++ ) {
  This.AddString( names[i] );
 }
```

so you need to stub out a getPIValues function. The OK button's EventClick needs:

```
 var number = ListBox1.Selection;
 var text = ListBox1.GetText( number );
 if ( number != -1 ) {
  removePIByMatch( text );
 }
 TheView.Cancel();
```

Again, stub out the new functions called. Over on the addValidation form, the OK button's EventClick has:

```
 if (  ComboBox1.GetText( ComboBox1.Selection ) ==
  xmlspyValidation ) {
  if (  ComboBox2.GetText( ComboBox2.Selection ) !=
    xmlspyValidation &&
        ComboBox2.GetText( ComboBox2.Selection ) != "" &&
        ComboBox2.GetText( ComboBox2.Selection ) !=
          null ) {
   printError( "addValidation(onClick)", new Error(0,
     ComboBox1.GetText( ComboBox1.Selection ) + " can not
     be matched to: " + ComboBox2.GetText(
     ComboBox2.Selection ) ) );
   TheView.Cancel();
   return;
  }
 }

 writePI( "validate", "command", ComboBox1.GetText(
  ComboBox1.Selection ), "schema", ComboBox2.GetText(
  ComboBox2.Selection ) );
 TheView.Cancel();
```

What is happening is a check to see if the user is attempting to select XMLSPY native validation and a schema. This combination is not permitted because XMLSPY makes that association through a different channel, and changing that behavior is outside the scope of this article (but it is very doable to programatically associate a schema with a document for the XMLSPY validator on-the-fly). But if that misassociation is not the case, the form calls writePI. The PI written will have a name of "validate" and two name-value pairs. The first named value is called "command" and holds the validation command name. The second is "schema" and holds the schema name.

After you finish these forms and stub out all the functions you called you are done with the visual side of things. You should be able to call your macros from the Tools menu of XMLSPY and see your forms appear.

Looking Ahead

In the second part of the article we will give a quick tour of XMLSPY's core XMLData object as you finish up the global functions. Then we'll turn to the XML side of things. You will see how to implement the work group XSD design rules we outlined above as a Schematron schema. More importantly, you will see how the framework you are creating can help you apply those rules as a part of your regular development process.

Author Bios

After working in software development in various industries over the past decade, Eric Schwarzenbach eventually came to specialize in electronic publishing. The issues of document-oriented XML are his focus: document and knowledge modeling, processing, and management frameworks, as well as XML databases and searching. He's written custom document management systems and worked with native XML databases such as Tamino. He currently serves as unofficial Tamino XDBA and content engineering lead at Classwell Learning Group.

David Kershaw is the professional services manager at Altova, Inc., the XMLSPY company. David brings 11 years of software engineering, project and product management to Altova. His previous positions include serving as the director of engineering at Classwell Learning Group and the group director of engineering at Organic, Inc. Mr. Kershaw received his Masters from Harvard University and his Bachelors from the University of Massachusetts.

DAVID.KERSHAW@ALTOVA.COM

WRITTEN BY JOEL AMOUSSOU

XML Certification Quizzer

Some interesting aspects of XML syntax

In this month's "XML Certification Quizzer," we are going to review some interesting aspects of the syntax of XML and related technologies. You can learn more about the syntax of the language by running the sample code in the questions with an XML parser or an XSLT processor. The skills you'll learn here will prepare you for IBM Test 141 on XML and related technologies.

Question 1 (Information Modeling)

Which of the following XML documents is well-formed?

A.
```
<!DOCTYPE a [
<!ELEMENT a (b | c)>
<!ENTITY % ent "<!ELEMENT b EMPTY>">
%ent;
<!ELEMENT c EMPTY>
]>
<a><b/></a>
```

B.
```
<!DOCTYPE a [
<!ENTITY % ent "(b|c)">
<!ELEMENT a %ent;>
<!ELEMENT b EMPTY>
<!ELEMENT c EMPTY>
]>
<a><b/></a>
```

C.
```
<!DOCTYPE a [
<!ENTITY % ent "<b/>">
<!ELEMENT a (b|c)>
<!ELEMENT b EMPTY>
<!ELEMENT c (#PCDATA)>
]>
<a>%ent;</a>
```

Select one answer.

Explanation: Choice A is the correct answer. The parameter entity reference %ent; is replaced with the element declaration for element b.

Choice B is not correct because in the internal subset of the document type declaration (the part delimited by [and]>), parameter entity references can occur only between markup declarations, not within markup declarations. Note that this well-formedness constraint is not applicable to parameter entity references in the external subset.

Choice C is not correct because parameter entity references can be referenced only in the DTD, not in the XML instance. The parser will treat %ent; as just a string, not an entity reference.

Question 2 (XML Processing)

Consider the following XML document:

```
<?doc myDoc?>
<a xmlns:d='http://xcert.org'>
 <b>Hello</b>
 <c>World!</c>
</a>
```

Which of these SAX events will be reported by a nonvalidating parser in the order specified?

A.
```
startDocument()
processingInstruction()
startPrefixMapping()
startElement()
characters()
startElement()
characters()
endElement()
characters()
startElement()
characters()
endElement()
characters()
endElement()
endPrefixMapping()
endDocument()
```

B.
```
startDocument()
processingInstruction()
startPrefixMapping()
startElement()
attributes()
ignorableWhiteSpace()
startElement()
characters()
endElement()
ignorableWhiteSpace()
startElement()
characters()
endElement()
ignorableWhiteSpace()
endElement()
endPrefixMapping()
endDocument()
```

C.
```
startDocument()
processingInstruction()
startPrefixMapping()
startElement()
startElement()
characters()
endElement()
startElement()
characters()
endElement()
endElement()
endPrefixMapping()
endDocument()
```

Select one answer.

Explanation: Choice A is the correct answer. XML documents often contain white space to make them more readable. However, XML parsers must report all characters that are not markup to the application. A nonvalidating parser will report the white space and the line ends to the application with the characters() method.

Validating parsers report white space

AUTHOR BIO

Joel Amoussou is founder and chief learning architect of XMLMentor.Net, where he develops blended learning solutions for building and assessing XML skills. Joel is the author of an XML exam simulator and teaches live e-learning courses on XML certification.

in element content using the ignorableWhiteSpace() method rather than the characters() method. Assume that the XML document has the following DTD:

```
<!ELEMENT a (b,c)>
<!ATTLIST a

xmlns:d  CDATA #FIXED
'http://xdocs.org'>
<!ELEMENT b (#PCDATA)>
<!ELEMENT c (#PCDATA)>
```

Element a can only contain child elements b and c. A character inside element a (including white space characters) would make the document invalid. The white space before elements b and c would be called ignorable white space. If namespace processing is enabled by the parser, startPrefixMapping() and endPrefixMapping are used to report the start of the scope of a prefix-URI Namespace mapping and the end of the scope of a prefix-URI mapping, respectively.

Question 3 (Architecture)

The XML Encryption and Syntax Specification defines an EncryptedData element. Which of the following elements is a mandatory child element of EncryptedData?

A. EncryptionMethod
B. Ds:keyInfo
C. CipherData
D. EncryptionProperties

Select one answer.
Explanation: Choice C is the correct answer. EncryptedData is derived from the EncryptedType abstract type. The following is the complex type definition of the EncryptedType type:

```
<complexType name='EncryptedType'
  abstract='true'>
<sequence>
<element name='EncryptionMethod'
type='xenc:EncryptionMethodType'
            minOccurs='0'/>
  <element ref='ds:KeyInfo'
    minOccurs='0'/>
  <element ref='xenc:CipherData'/>
  <element ref='xenc:Encryption
    Properties' minOccurs='0'/>
 </sequence>
 <attribute name='Id' type='ID'
   use='optional'/>
 <attribute name='Type' type='anyURI'
   use='optional'/>
 <attribute name='MimeType'
   type='string' use='optional'/>
 <attribute name='Encoding'
   type='anyURI' use='optional'/>
</complexType>
```

The EncryptionMethod child element is optional and describes the encryption algorithm applied to the cipher data. The ds:KeyInfo child element is optional and contains information about the key used to encrypt the data. The CipherData child element is a mandatory element that contains the CipherValue or CipherReference. The CipherValue element contains the encrypted octet sequence as base64 encoded text. The CipherReference element provides a reference to an external location containing the encrypted octet. The EncryptionProperties child element is optional and contains additional information concerning the generation of the EncryptedData element.

The following is an example of an encrypted medical record in which the content of the IllnessInfo element is encrypted:

```
<?xml version='1.0'?>
<MedicalRecord
xmlns='http://medrec.org/'>
 <PatientName>James Lambert<Patient
  Name/>
 <IllnessInfo>
  <EncryptedData   xmlns='http://
  www.w3.org/2001/04/xmlenc#'
   Type='http://www.w3.org/2001/04/
   xmlenc#Content'>
   <CipherData>
    <CipherValue>A23B45C56</
      CipherValue>
   </CipherData>
 </EncryptedData>
 </IllnessInfo>
</MedicalRecord>
```

Question 4 (Testing and Tuning)

A complex type definition is specified in an external schema that has no target namespace. Which of the following XML Schema elements would you use to import the complex type into a schema, and extend the complex type by adding an attribute to its declaration?

A. import
B. redefine
C. include
D. redeclare

Select one answer.
Explanation: Choice B is the correct answer. The redefine element can be useful for schema authors who need a declarative mechanism for managing the evolution of their schema. Modifications to the schema can be described with the redefine element.

The redefined schema document must have the same targetNamespace as the redefining schema document, or no targetNamespace. If the redefined schema document has no targetNamespace, then it is converted to the redefining schema document's targetNamespace.

Within the redefine element, the base types in type definitions must be the types themselves. Attribute groups and model groups are defined as supersets or subsets of their definitions in the redefined schema.

Question 5 (XML Rendering)

Consider the following XML document:

```
<?xml version="1.0"?>
<catalog>
  <item>
    <manufacturer>aero corp</
      manufacturer>
    <partnumber>FC-7612</partnumber>
    <price>34</price>
  </item>
  <item>
    <manufacturer>parts inc</
      manufacturer>
    <partnumber>B765-10</partnumber>
    <price>75</price>
  </item>
  <item>
    <manufacturer>cat inc</
      manufacturer>
    <partnumber>JK766-6</partnumber>
    <price>56</price>
  </item>
</catalog>
```

Using XSL formatting objects, which XSLT stylesheet would you use to transform the XML document for printed output?

A.
```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/
  XSL/Transform" version="1.0">

<xsl:template match="catalog">

<fo:table table-layout="fixed">

   <fo:table-body>

     <xsl:for-each
select="catalog/item">
      <xsl:sort select="price" sort-
order="ascending"
data-type="number"/>
      <fo:table-row>
          <fo:table-cell>
```

```
<fo:block>
<xsl:value-of select="manufacturer"/>
</fo:block>
</fo:table-cell>
        <fo:table-cell>
<fo:block>
<xsl:value-of select="partnumber"/>
</fo:block>
</fo:table-cell>
        <fo:table-cell>
<fo:block>
<xsl:value-of select="price"/>
</fo:block></fo:table-cell>
     </fo:table-row>
    </xsl:for-each>
   </fo:table-body>
 </fo:table>

</xsl:template>

</xsl:stylesheet>
```

```
B. <xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/
  XSL/Transform" version="1.0"

xmlns:fo="http://www.w3.org/1999/XSL/
  Format">

<xsl:template match="catalog">

<fo:table table-layout="fixed">
   <fo:table-column
     column-width="50mm"/>
   <fo:table-column column-
width="50mm"/>
   <fo:table-column column-
     width="50mm"/>

   <fo:table-body>

    <xsl:for-each
      select="catalog/item">
     <xsl:sort select="price"
       order="ascending"
data-type="number"/>
     <fo:table-row>
        <fo:table-cell>
<fo:block>
<xsl:value-of select="manufacturer"/>
</fo:block>
</fo:table-cell>
        <fo:table-cell>
<fo:block>
<xsl:value-of select="partnumber"/>
</fo:block>
</fo:table-cell>
        <fo:table-cell>
<fo:block>
<xsl:value-of select="price"/>
</fo:block>
</fo:table-cell>
     </fo:table-row>
    </xsl:for-each>
   </fo:table-body>
 </fo:table>

</xsl:template>

</xsl:stylesheet>
```

```
C. <xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/
  XSL/Transform" version="1.0"
     xmlns:fo="http://www.w3.org/
       1999/XSL/Format">

<xsl:template match="catalog">

<fo:table table-layout="fixed">
   <fo:table-column column-
width="50mm"/>
   <fo:table-column column-
     width="50mm"/>
   <fo:table-column column-
     width="50mm"/>


    <xsl:for-each
      select="catalog/item">
     <xsl:sort select="price"
       order="ascending"
data-type="number"/>
     <fo:tr>
        <fo:td>
<fo:block>
<xsl:value-of select="manufacturer"/>
</fo:block>
</fo:td>
        <fo:td>
<fo:block>
<xsl:value-of select="partnumber"/>
</fo:block>
</fo:td>
        <fo:td>
<fo:block>
<xsl:value-of select="price"/>
</fo:block>
</fo:td>
     </fo:tr>
    </xsl:for-each>
 </fo:table>

</xsl:template>

</xsl:stylesheet>
```

Select one answer.

Explanation: Choice B is the correct answer.

Choice A is not correct because the XSL formatting object namespace is not declared. The <xsl:sort> element must have an "order" attribute, not a "sort-order" attribute.

Choice C is not correct because element names tr and td are not defined by the XSL specification.

Conclusion

To get the hands-on coding experience that you need in the real world, you should download some XML tools and try to run a few samples. It is also helpful to create small applications using real documents or data. You can obtain free open source XML software from http://xml.apache.org.

JOEL@XMLMENTOR.NET

~continues from page 5~

pler than HTML and Java. Application changes and maintaining the information cost less, too, since XML uses templates instead of custom coding.

But the true leverage comes when you can give the same application different looks simply by changing a stylesheet or using a new one. This gives you flexibility that wouldn't otherwise have been affordable.

In addition, once customers are hooked on stylesheets (which they are very likely to be if you help them along on the first go-around) they will be taking on ongoing development work you might otherwise have had to do. Logic would also dictate that with your customer service measurably improved, your sales should also rise and outpace your competition.

Better customer service, greater flexibility, lower costs, and potentially higher sales: that's style with some real substance. It might be time to start asking customers to send you their stylesheets.

Author Bio

Tim Matthews is the founder of Ipedo, an XML infrastructure software company in Redwood City, CA.

TIM@IPEDO.COM

Ektron

www.ektron.com/xmlj

Altova

http://xmlj.altova.com/code