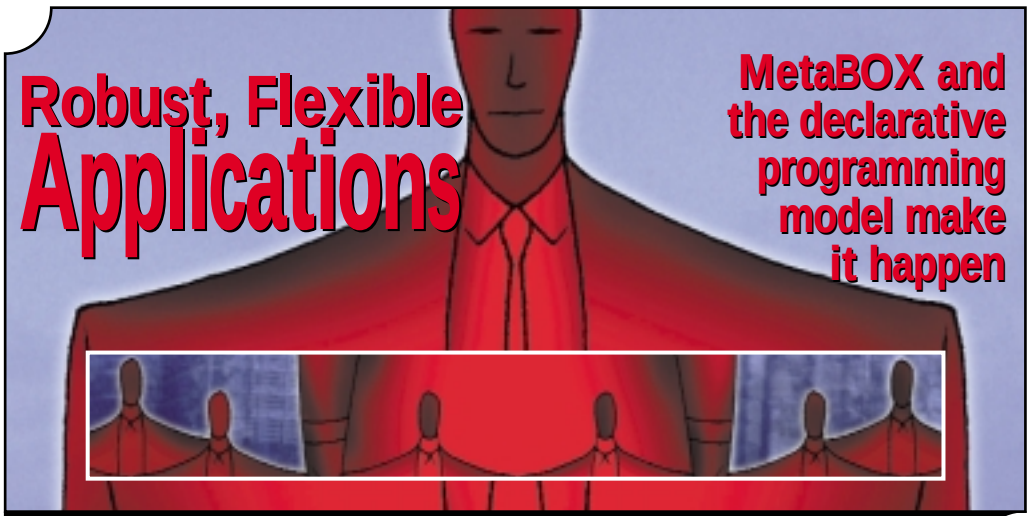
Enterprise Solutions • Data Management • XML Labs

XML JOURNAL

THE ULTIMATE XML ENTERPRISE RESOURCE

December 2002 | Volume: 3 Issue: 12



Marcelo F. Ochoa and Roel Franken 22

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 sim@polarisventures.com

EDITORIAL
Editor-in-Chief
John Evdemon jevdemon@sys-con.com
Editorial Director
Jeremy Geelan jeremy@sys-con.com
Managing Editor
Jennifer Stilley jennifer@sys-con.com
Editor
Nancy Valentine nancy@sys-con.com
Associate Editors
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
Neil Bradley, Chris Brandin, Roel Franken, Jeremy Geelan, Sawat Hansirisawat, Casey Hibbard, Ivan Imana, Ari Kermaier, Edward Kierklo, Joe Morgan, Marcelo F. Ochoa, Patrick Rasche, Michael Rowell, Hitesh Seth

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.

SYS-CON MEDIA

Is XML Complete?

WRITTEN BY **JEREMY GEELAN**

The XML-DEV discussion list – the open, unmoderated list supporting XML implementation and development and managed by OASIS – recently considered the burning question of whether XML is complete...or is still missing something.

Renowned XML expert Simon St. Laurent, who initiated the discussion, commented in his opening remarks that, looking at the three original aspects of XML – which he defines as syntax, linking, and styling – it looked to him like they may be close to done. "There is ongoing work on XML," he wrote, "but I think it's fair to characterize both XML 1.1 and Namespaces in XML 1.1 primarily as efforts to clean up outstanding issues (Unicode tracking, excess namespace declarations) while possibly sliding something new in (NEL, IRIs) rather than any kind of new structure or significant modification."

As for XML Schema, St. Laurent, who is author of books like *XML Elements of Style*, *XML: A Primer*, and *Inside XML DTDs*, noted that – so far as he could tell, anyway – the topic had exhausted its participants and he hadn't heard much talk on that front. "Critics seem to have moved elsewhere," St. Laurent opined, "and while supporters occasionally talk about trimming it, there's not much activity there."

What about XLink and XPointer? "XLink is finished, though hardly anyone uses it," continued St. Laurent, "and XPointer is apparently in a dash to the finish before the Working Group (WG) expires on 31 December. Whether XPointer will find more use than XLink thus far remains to be seen."

That left XSLT. "XSLT is certainly growing in its 2.0 development process," was St. Laurent's view, "but it's far from clear that there's strong community support for more than a few pieces of it. XSLT could use a cleanup/minor addition process, but that doesn't seem particularly likely to happen, at least given current World Wide Web Consortium (W3C) activity."

St. Laurent concluded with a couple of open questions: "Are there more issues like Character Entities waiting to surface? Or (apart from that issue) can XML declare victory and call itself complete?"

Not surprisingly, given that XML-DEV has been the premier XML discussion list in the world since its creation in January 1997 by two UK-based professors, Peter Murray-Rust of the University of Nottingham and Henry Rzepa of Imperial College, there were plenty of dissenters once St. Laurent had expressed his viewpoints.

Andrew Watt, for example, joined the discussion to say that, so far as he could see, "We are moving to an XML world with approximately two data models: 1) DOM in the browser for SVG, XHTML, etc., and 2) XQuery/XPath/XSLT/Infoset/AABSABI for the rest." He wondered if that was how others saw things. Watt also noted that, as he put it, "There is a ton of work ongoing on XQuery (and therefore also on XSLT 2.0 and XPath 2.0). And, if my guess is correct, they will pretty much go straight on to add update functionality to XQuery."

St. Laurent was not impressed. "XQuery is a toolkit for working with XML, not part of XML itself," he replied, adding: "The same can be said of XSLT, though XSLT was pretty clearly part of the original style project."

Naturally enough, given the nature of the question ("Is XML Complete?"), the issue of whether XML 2.0 is on its way was raised. Richard Tobin chimed in to say that he wasn't aware of there having been any serious study on this question yet.

"I have heard a vast range of views on what XML 2.0 should be if there is one," Tobin wrote, "from a unification of XML 1.x + Namespaces + Infoset + no substantive changes, to a completely redesigned language."

Douglas Husemann offered his thoughts on XML 2.0 as follows: "Yes an implementation that takes Namespaces into account from the very beginning would be a good thing...but perhaps standardization on specific standards for infosets would be a good thing also," he added. "It would be easier for generated languages to work together in a future implementation."

Let us give the last word to Simon St. Laurent himself, since it was he who started the whole discussion: "I expect XML applications will go on for another decade or more regardless of whether there's any change in the XML core," he declared. Or, in other words: yes, XML is – give or take – complete.

Activities like the recent XML Security specs are largely, in St. Laurent's view, just an adaptation of existing security specs to XML structures. That's to say, it's just an XML application rather than a change in the XML core.

"It's (often) good stuff, and it's interesting," St. Laurent concluded, "but it's about using the infrastructure XML provides rather than building the XML infrastructure."

AUTHOR BIO

Jeremy Geelan, editorial director of SYS-CON Media, speaks, writes, and broadcasts about the future of Internet technology and the business strategies appropriate to the convergence of business, i-technology, and the future.

JEREMY@SYS-CON.COM

SYS-CON MEDIA
135 Chestnut Ridge Rd., Montvale, NJ 07645
TELEPHONE: 201 802-3000 **FAX:** 201 782-9637

PRESIDENT and CEO
Fuat A. Kircaali fuat@sys-con.com

BUSINESS DEVELOPMENT
VP, Business Development
Grisha Davida grisha@sys-con.com

COO/CFO
Mark Harabedian mark@sys-con.com

ADVERTISING
Senior VP, Sales & Marketing
Carmen Gonzalez carmen@sys-con.com
VP, Sales & Marketing
Miles Silverman miles@sys-con.com
Advertising Director
Robyn Forma robyn@sys-con.com
Advertising Account Manager
Megan Ring-Mussa megan@sys-con.com
Associate Sales Managers
Carrie Gebert carrieg@sys-con.com
Kristin Kuhnle kristin@sys-con.com
Alisa Catalano alisa@sys-con.com
Leah Hittman leah@sys-con.com

SYS-CON EVENTS
Conference Manager
Michael Lynch mike@sys-con.com
Regional Sales Managers, Exhibits
Michael Pesick michael@sys-con.com
Richard Anderson richarda@sys-con.com

CUSTOMER RELATIONS
Customer Service Representative
Margie Downs margie@sys-con.com

JDJ STORE
Manager
Rachel McGouran rachel@sys-con.com

WEB SERVICES
VP, Information Systems
Robert Diamond robert@sys-con.com
Web Designers
Stephen Kilmurray stephen@sys-con.com
Christopher Croce chris@sys-con.com
Online Editor
Lin Goetz lin@sys-con.com

ACCOUNTING
Accounts Receivable
Kerri Von Achen kerri@sys-con.com
Financial Analyst
Joan LaRose joan@sys-con.com
Accounts Payable
Betty White betty@sys-con.com

SUBSCRIPTIONS
SUBSCRIBE@SYS-CON.COM
1 888 303-5282
For subscriptions and requests for bulk orders, please send your letters to Subscription Department
Cover Price: $6.99/issue
Domestic: $77.99/yr (12 issues)
Canada/Mexico: $99.99/yr
all other countries $129.99/yr
(U.S. Banks or Money Orders)
Back issues: $12 U.S. $15 all others

The Real Success of XML

WRITTEN BY **HITESH SETH**

XML is a relatively simple development. Yet it has probably been the most powerful development to date in the world of i-Technology.

From being the launchpad for information delivery on the Web (XHTML, SMIL, SVG) to electronic business communication (Chem standards, ebXML, RosettaNet, OAGIS, EDI/XML), security (SAML), Web services (SOAP, WSDL, UDDI, BPEL4WS), data transformation (XSLT/XSL), and speech recognition and telephony integration (CCXML, SALT, VoiceXML), XML has quickly grown as the universal markup language to define practically anything. It has been the vehicle for practically anything that flows through the Web and has been influential in converting the Web into a ubiquitous platform for application delivery and information exchange.

An example comes to mind. In September 1998, I was participating in SAP's TechEd technology conference in Los Angeles, in a session focused on how an interactive speech recognition system can be integrated with the R/3 ERP system. It was pointed out that as part of the speech recognition process, a developer is required to build a grammar for the system that specifies a set of utterances that a user may speak, and a particular grammar format tuned for the speech recognition platform was presented. I wondered if that grammar format could be described in XML; sure enough, earlier this year, W3C Voice Browser Activity announced a candidate recommendation of the XML-based Speech Recognition Grammar Specification.

This is really the success of XML.

However, it has not always been a bed of roses for XML, which has had – and probably always will have – its share of issues.

For instance, soon after the XML specification was released, the XML community realized that DTDs weren't going to be sufficient for the definition of the rich, strongly typed vocabularies that we'd like to define using XML. This led to the development of a large list of data-definition focused markup languages (MLs). Overall, I think it took us way too long to get the XML Schema specification out the door.

There is such growth around the various industry-specific vocabularies that it can sometimes beget confusion in the IT community. In the world of financial services, for example, there are many MLs out there, some of them overlapping one another significantly. Of late, we saw a similar fragmentation happening again, with the emergence of a number of initiatives focused around the whole concept of Web services orchestration and business process management – BPML, XLANG, WSFL, BPEL4WS, and so on.

I would not dispute the value of these efforts. Some of these initiatives have undoubtedly enjoyed an excellent level of participation by expert technology and business players. However, a key lesson that the XML revolution has taught us is that simplicity is best. It is important that we keep that message in mind – and stop the clutter. Furthermore, in these tough economic times, we need to concentrate our efforts only on those standards that can impact real-world business by solving real-world problems.

Who should really receive the credit for the ever-rising success of XML? My own favorites are the W3C and a number of large software development organizations, particularly Microsoft. And it looks like I am not alone in thinking this. Initial results from the online poll that we have initiated at www.sys-con.com/xml show that W3C and Microsoft Corporation are our readers' favorites as well.

As the medium for the collective evolution of the core set of standards around XML, the W3C has played a key role. Meantime, Microsoft has been instrumental in supporting XML in practically everything that it develops, from supporting XML/XSL in IE to the new .NET Framework, to including XML in the .NET Speech platform (using SALT) – and even in the newly announced Office 11. Microsoft has also been a key visionary around Web services and has initiated some remarkable collaborative work with IBM, another pioneer in this space.

Special credit should go to those technology-savvy CIOs of the Fortune 1000 organizations who have laid down a clear vision of IT in their organization as "XML friendly and compliant." No technology is ever greater than its business applications: the initiative and funding of real-world business organizations have really made XML what it is today, and it's no coincidence that the focus of ***XML-J***, in addition to being a technical journal, is on real-world business scenarios, issues, and implementations of XML.

With all this success, one might be forgiven for thinking that XML had pretty much achieved all its goals. On the contrary, I think what we have seen so far is really just the beginning of the success of XML.

AUTHOR BIO

Hitesh Seth is the chief technology officer of ikigo, Inc., a provider of XML and Web services monitoring and management software.

HKS@HITESHSETH.COM

Reader Feedback

Don't Define What the XML Means

Re: "*Don't Define a Data Model*"

How about "Don't define what the XML means" instead? Somewhere inside the XML will be a tag called <address>, and you need to know whose address it is. If you don't get agreement on that, the application will fall apart. Defining "Whose address is this?" is just the sort of thing data models do.

You are going to need something like a data model, information model, or object model. Okay, so "rigid data models" have done a disservice in the past, but we can't just do without some model of the information. Just having "an extensible information container that allows the data within it to change throughout the life of the business application" is a recipe for confusion – unless someone can tell you what the data means.

Robert Worden
via e-mail

Essential Solutions

Re: "*XML and FOP for Industrial Needs*"

This article is quite interesting because it discusses a common nontechnical human-interface problem. Solutions to such problems are essential for the ongoing success of e-commerce from a business standpoint.

U.N. Umesh
via e-mail

This article gives very useful information for displaying output in PDF and other formats instead of HTML. It's very enlightening.

Sumanth
via e-mail

Letters may be edited for grammar and clarity as well as length. Please e-mail any comments to John Evedemon (jevdemon@sys-con.com)

Co-Inventor of XML Says Office 11 Is 'A Huge Step Forward for Microsoft'

XML-J NEWS DESK

The story that sparked the hottest XML discussion of the year

Now that the newly XML-enabled version of Microsoft Office, code-named "Office 11," is in its first official beta release, ***XML-J Industry Newsletter*** went straight to Tim Bray, co-inventor of eXtensible Markup Language, and asked for his exclusive views on this improvement in what Microsoft routinely – if immodestly – characterizes as "the world's leading suite of productivity software."

Asked if he'd been involved at all in the XML-enabling of Office 11, Bray replies that he hadn't: "No, not in the slightest," he assures us. However, he did receive extended hands-on demos of the alpha and beta software, he says, which gave him the opportunity to test-drive and evaluate the suite.

Word Files Are Now Also XML Files

When asked how XML-enabling will make a difference in MS Office, Bray quickly zeroes in on what in his view is the key differentiator in an XML-enabled Office suite versus the current one. "The important thing," he explains, "is that Word and Excel (and of course the new XDocs thing) can export their data as XML without information loss. It seems Word can also edit arbitrary XML languages under the control of an XML Schema, but I'm actually more excited by the notion of Word files also being XML files."

So it's a breakthrough? Bray has no doubts whatsoever: "The XML-enabling of Office was obviously a major investment and is a major achievement," he declares without hesitation.

"Built around an open, internationalized file format," he continues, warming to his theme, "Office 11 is going to be a huge step forward for management, independent software developers, and Microsoft."

What is the precise significance of the internationalized file format? Bray, who is also CTO and founder of Antarctica Systems Inc, clarifies as follows: "When I say 'open and internationalized,' I'm just saying that these are the two most important benefits that occur when you make information available in XML." In other words, he is saying XML enables the exchange of any form of data across heterogeneous systems, platforms, and applications.

"So it seems to me," he concludes, in delightfully prophetic mode, "that when the huge universe of MS Office documents becomes available for processing by any programmer with a Perl script and a bit of intelligence, all sorts of wonderful new things can be invented that you and I can't imagine."

That's praise indeed, from the man behind XML itself!

Office 11 is expected to ship in mid-2003 after user testing. Geared toward enterprise users, it will contain components compatible with the .NET initiative. The XML technology incorporated in this beta version supposedly allows data to be more easily exchanged and shared between different programs.

If it all works according to plan, this would obviously increase productivity and interconnectivity. How so? Well, organizations of all sizes often store their data without a common format and in a variety of places (for example, CRM databases and accounting systems). As a result, information workers within those organizations have difficulty accessing the data they need, or, if they can locate it, they find that the data is in an unsuitable format.

"Office 11 is going to be a huge step forward for management, independent software developers, and Microsoft"

Microsoft's aim with Office 11 is to make connecting and using data simpler, and its path to success in this respect is a function of its broad support for standards-based XML.

XML-J Industry Newsletter applauds the initiative and will continue to follow this development from a variety of angles in future issues, including bringing you insights from within the heart of the Microsoft XML design team itself in Redmond, WA.

XML-JOURNAL@SYS-CON.COM

OAGIS: A 'Canonical' Business Language for Web Services

The missing piece

WRITTEN BY
MICHAEL ROWELL

With all of the capability of Web services and the hype surrounding them, there's still a missing piece. Most people don't talk about this missing piece, despite the fact that it's what end users care the most about.

Integration isn't possible without an understanding of the payload of the message. While protocols are "cool," the customers' "end-in-mind" is business integration. For this to happen, Web services must be able to communicate the information of the business transactions being performed. At the end of the day, my message must reach you and your business application such that your business application understands the intent of the message and its impact on the business process in which the two of us are currently involved.

Web services promises integration of business applications over the Internet by exchanging data, sharing tasks, and automating business processes – all while lowering the cost of integration, regardless of operating system, language, and location. However, most Web services today are informational in nature, not transactional. For example, many financial companies offer Web services that give stock quotes, and Amazon has Web services that allow you to check the availability of a book and its price. But no one currently offers a Web service that enables a stock trade or book purchase. It's possible to perform these transactions in a business-to-consumer scenario in which a customer buys a book via a Web page, but not in a business-to-business scenario in which the business applications on each side make the transaction happen. The promise of Web services is just this – the ability to automate these types of tasks so that business applications can be performed with minimal human intervention.

> Web services have so far been employed mainly for internal integration efforts. But the first few projects involving Web services for B2B are making their debut, and experts say mainstream use of Web services for external integration should get under way next year.
>
> It is expected that over the next several years, Web services will mark a shift in distributed computing toward loosely coupled, standards-based service-oriented architectures.

To do this, Web services need to be able to communicate the information needed for business transactions to occur. This simply isn't provided by the agreed-to set of Web services standards, nor is it provided through the extended set of Web services standards.

The Open Applications Group Integration Specification (OAGIS) provides for both vertical and horizontal requirements for integration. This is the case whether an integration is implemented via Web services or any other technology. The Open Applications Group (OAGI) made a conscious decision to craft OAGIS to be technology sensitive but not technology specific. Because of this decision, OAGIS has been implemented over a variety of different technologies: Web services, ebXML, RosettaNet Informational Framework (RNIF), BizTalk, and Message-Oriented-Middleware to name a few. (OAGI provides white papers on their Web site, www.openapplications.org, that describe using OAGIS with several of these frameworks.)

In this article, we'll look at the Web services standards and how to use OAGIS with Web services.

Web Services

"Web services" is a glamorous term for some very old and established ideas. Web services is actually a stack of standards that complement each other. These standards fall into one of four layers; Table 1 shows the standards typically associated with Web services and identifies the layer in which each belongs.

It's generally agreed that the standards shown in Table 1 are the base Web services standards. There are also other protocol standards that help make Web services secure and provide quality of service and routing, among other things. These additional standards aren't listed here because they differ depending upon the Web services camp. There are other resources that identify the standards of each side – it's always interesting to see the major players aligning themselves differently around competing standards.

Suffice it to say that as with any technology, it will take time for the vendors to align themselves around a single set of standards. The good news is that the core set is defined, and all of the major players agree with the set.

As Web services moves forward, for at least the near term, there will be different camps that back the different competing standards for providing quality of service and security. Of these additional specifications, the following seem to be gaining the most traction in both camps: WS-Security, WS-Transaction, WS-Coordination, and BPEL4WS.

Regardless of how these additional protocol standards fall out, Web services will be able to ensure security and quality of service when communicating business transactions, and to indicate the business process that is being performed and what should occur next. A lot of smart people are working on these important protocol standards.

But what about the business transactions and messages

Discovery	**UDDI** (Universal Description, Discovery and Integration) – An XML-based framework to enable businesses to discover each other, and define how they must share information in a global registry.
Description	**WSDL** (Web Services Description Language) – An XML format for describing Web services in a common way. WSDL describes: • Interface information describing all publicly available functions • Data type information for all message requests and message responses • Binding information about the transport protocol to be used • Address information for locating the specified service
Packaging	**SOAP** (Simple Object Access Protocol) – A lightweight XML protocol for exchange of information in a distributed environment.
Transport	**HTTP, SMTP, FTP (any other transport mechanism can be used)** Pretty much any transport mechanism can be used underneath Web Services to carry the data from point A to point B. These are typically used when communicating over the Internet for obvious reasons.

TABLE 1 Commonly agreed-to stack of Web services and common standards in each layer

needed to perform those business transactions? Nowhere in the Web services stack are the business messages addressed. WSDL provides the XML description of how the API is to act, but providing the data type definition does *not* provide the definition of the content of the messages to be exchanged. In the case of XML messages, WSDL indicates the parameter name and the type of string, and that's it.

With Web services we now have the ability to communicate my XML message from my application to you and your application. Web services allow information to flow much like utility services such as electricity and water.

The Need for a Canonical Business Language

Combining XML content with the "cool" XMLized protocol standards of Web services is a common way of moving XML content for integration. This sounds good!

But wait, we still have a problem: anyone can define their XML content in any way they want. So each trading partner with whom a company does business (using Web services or any other means) must perform a data transformation. At the same time, each application a company has internally has its own XML dialect. All that's been accomplished is the moving of the point-to-point transformations of EDI and proprietary formats to point-to-point transformations of XML to XML mappings. While this is better than where we were, it's still not the "end-in-mind." Organizations must deal with the point-to-point mappings shown in Figure 1.

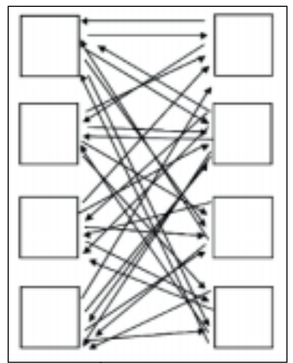


FIGURE 1 The mess created by point-to-point communication and mapping

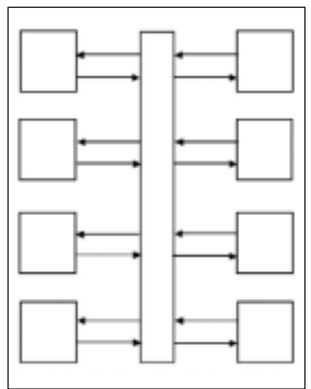


FIGURE 2 Using a common "canonical" business language for integration simplifies integration and cuts costs

Consider this simple real-world example: if I use SAP R/3 and you use PeopleSoft, and we choose to do business together at some point, my SAP message must be translated into your PeopleSoft XML message in a point-to-point translation. Now add three more applications to the mix for one organization, making a total of five applications for the one organization to integrate. Figure 1 shows the mess that this point-to-point approach creates. Now add a small supply chain of ten companies, each having five internal applications that must be integrated. Do you really want to do this "simple" integration in a point-to-point model? This certainly doesn't provide the ROI that the end user expects of Web services.

If we look at the math of point-to-point integration, a company must maintain n (n – 1) possible connections, where n is the number of integration points (or endpoints). What's needed is a common base that information can be mapped to and from. This would allow an organization to cut the number of integration points that must be maintained to $n * 2$ (see Figure 2).

The resources needed to support each of these integration models are shown in Figure 3. Which would you rather support?

Needless to say, integrating business applications using a canonical business language not only saves time and money during implementation, it also saves time and money in the maintenance of these integrations. Consider adding another application to the scenario shown in Figure 1. Each application that must communicate with the new application must be touched again. In the case of Figure 2, the existing applications are insulated from the new addition by the canonical business language; as a result, they aren't affected by the addition. Likewise, the new application doesn't have to translate its data formats to each of the applications it must interface, and vice versa.

OAGIS in Web Services

Expanding on the analogy used earlier, using OAGIS with Web services resembles how we receive electricity through a standard set of plugs, outlets, and a current. (Granted, we don't all use the same plugs and outlets, and currents differ from country to country). Hopefully, we can avoid these differences in Web services by extending the de facto set of standards for Web services to use OAGIS as the common current. The Web service defines the shape of the plugs (in the form of SOAP), and WSDL describes the shape of the outlet. OAGIS defines the current (the canonical business language) that travels over the wire.

As discussed in "OAGIS: A 'Canonical' Business Language" (***XML-J***, Vol. 3, issue 9), OAGIS has a proven track record for integrating business-to-business and application-to-application transactions.

OAGIS 8.0, along with the collaboration of industry vertical groups and the OAGIS overlay capabilities, makes it possible to use a single specification for integration between horizontal and vertical implementations, using the horizontal elements, types, and structures as appropriate within a vertical implementation.

Using OAGIS as the canonical business language allows other industry vertical XML standards to be plugged in as OAGIS's overlays. OAGI is working with many different industry groups to create more of these overlays. Because of this,

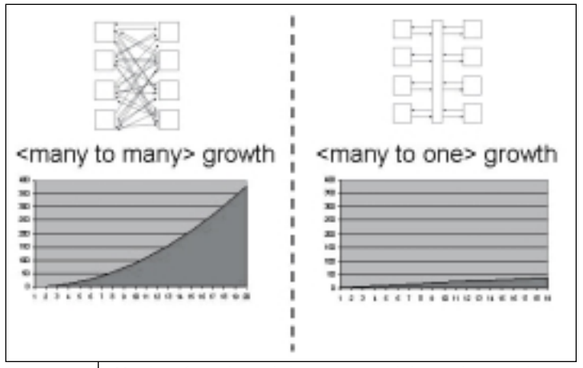


FIGURE 3 Number of people needed to support each integration model as the number of interfaces grows

using OAGIS with Web services makes it possible to achieve the end-in-mind of everywhere-to-everywhere integration *…today*.

Enabling OAGIS Web Services

OAGIS provides a set of business scenarios that are intended to be used as examples to help implementers find a scenario that closely matches the integration they need to perform. (No matter how smart the designers of a specification are, they'll never come up with all of the possible ways that businesses conduct business.) Once a scenario has been selected, the next step is to identify the messages or Business Object Documents (BODs) that will be used to pass the information between the business applications or businesses being integrated. This is done by using the scenario as a guide and identifying the BODs needed to carry the information; in this step, any additional BODs that may be needed are also identified.

Next, it's simply a matter of enabling the passing of the information in a BOD via Web services. Depending upon whether the back-end application has support for OAGIS, you may have to transform the data into and out of a BOD in order to enable communication.

There are two sides to every Web service: a client or requester, and server or provider. A typical provider development plan for Web services looks like this:

1. Develop the functionality of your service by performing the function intended by connecting to back-end applications. In the case of OAGIS Web services, the API must be able to provide and consume BODs.
2. Develop a service wrapper for the functionality developed in Step 1. This can be either an XML-RPC or a SOAP service wrapper.
3. Next, provide a service description for SOAP in WSDL; for XML-RPC, in some human-readable instructions.

A Little History

It's been said that in order not to repeat the mistakes of the past, it is important to remember our history as we move forward. Otherwise, we are destined to repeat it.

Business has been done throughout history by passing transactional documents from customer to supplier and back again. Yes, sometimes a broker may be in the middle of some of these transactions, but the messages passed are still transactional in nature (i.e., PurchaseOrders, Invoices, Shipment, etc.). This has happened even before the advent of computers. It's not likely that it's going to change much in the next five to ten years.

Since the advent of computers, we humans have attempted to automate these transactions. We have EDI, which was/is good for communicating information between large organizations. However, with EDI, EDIFACT defined the transactions and each vertical industry took the transactions and wrote guidelines around how that industry would use the EDI standards, often using fields for purposes for which they weren't originally intended. Because of this, each industry's EDI standard is unique today. Add to this the fact that EDI is difficult and expensive to implement. In general, EDI has only been implemented in industry by large organizations. These large organizations typically specify how their tier-one and possibly tier-two suppliers do business in their version of EDI. This falls apart quickly when you go beyond tier-one suppliers to these large organizations. This is because often these tier-two, tier-three, and tier-four suppliers are smaller organizations that work with many different industries. In the EDI model, these small organizations are asked to implement the different versions of EDI. They simply can't afford to do this.

The promise of the Internet and XML is lower cost of integration. Now, instead of having to buy in to an expensive VAN, smaller organizations can send information via the Internet. XML provides a machine-readable format that is used to pass information by defining a descriptive tag to indicate the meaning of the information enclosed.

Of course, there must be protocols that make these transmissions secure, provide the sequencing of the messages and timeouts, etc. These capabilities are being addressed by Web services and will be provided by competing Web services vendors.

It's important to note that EDI is considered by many to be the first generation of business-to-business communication. EDI is still how a majority of business-to-business transactions are made. As a general rule, companies don't replace systems that work, especially in this economy. OAGIS/XML complements EDI. OAGIS/XML enables performance of business transactions in request/reply, publish/subscribe, push, and if necessary, batch exchange models. EDI typically operates in a pushed batch model in which information is sent between trading partners at given increments of time and processed (granted, this is not always the case). Because EDI is installed and working, it's not going to be replaced by OAGIS/XML. Rather, the two can complement each other nicely when companies need to address the additional exchange models where OAGIS/XML offer advantages.

"WSDL provides the XML description of how the API is to act, but providing the data type definition does not provide the definition of the content of the messages to be exchanged"

Componentization of Web Services

Yes, the componentization of business applications and/or Web services allows, for example, an interface that handles addresses to be used by the larger components of the application/service. This allows us to have a common service that handles the address information that can be shared across the business applications that can take advantage of it.

However, in order to conduct business, a large-business component is needed to give the address contextual meaning. For example, one address may be the shipping address of the ShipToParty on a PurchaseOrder; another may be the contact address of the broker agency that brokered the purchase for the end customer. Without the context of the purchase order and the further context of the given parties, it wouldn't be possible to determine what the address service was to do with the address being provided.

This example shows that there can be multiple layers of Web services; it's a matter of which granularity a business chooses for exposing these services. Today, business is done by passing transactional information between the larger-grained business services (applications) within businesses.

Layering the Web services such that the fine-grained services are available to the large-grained services within a given portal or enterprise allows us to ensure that the context and legal bindings associated with these transactional documents (purchase orders, invoices, etc.) are maintained. Without this context, the address information is meaningless.

Only within the context of larger-grained services does it make sense to provide finer-grained functionality Web services that provide the needed services for common components.

OAGIS 8.0 was designed with this in mind. While OAGIS 8.0 provides the example scenarios and BODs, these BODs are crafted by assembling lower-level common components, compounds, and fields. In this manner, the address component used in a purchase order and the address used in an invoice are consistent. Furthermore, they make use of the same definition for address, which comes from the OAGIS Resources/Components XML Schema file.

This is extended to all OAGIS components. It's important to note that noun-specific components are provided in the corresponding Noun XML Schema file.

4. Deploy the service. This can mean installing and running the service on a stand-alone server or integrating with an existing Web server.
5. Publish the existence and specifications of your new service. This is typically done using a UDDI directory, either the public UDDI directory or a private company UDDI directory.

A typical client development plan for Web services looks like this:

1. Identify and discover the services that are relevant to the application you're creating. This usually involves searching the UDDI Business Directory for partners and services.
2. After identifying a service you want to use, locate a service description. If it is a SOAP service, you'll typically find a WSDL document that describes the methods interface. In the case of OAGIS Web services, these interfaces will be able to consume BODs provided by your client. The WSDL should indicate the BODs expected.
3. Create a client application. If the service has a WSDL file, you can automatically create the client code via a WSDL invocation tool.
4. Finally, run your client application to invoke the Web service.

Show Me the Code

To expand on this a bit further, what do the steps above require an application to do? Note that the Web services standards provide wrappers for codifying XML-based messaging. SOAP provides an envelope for the message payload or body, just as an envelope does for postal mail. The SOAP message example shown in Listing 1 shows how an OAGIS BOD ProcessPurchaseOrder is carried within a SOAP message that uses the PurchaseOrder_Operation described in the WSDL enable method on the server.

Notice that the OAGIS ProcessPurchaseOrder is simply wrapped in the SOAP envelope as the SOAP body for the example message; the content of the message is not altered, it is simply wrapped for transport.

The corresponding WSDL simply describes the Web services method that will receive the ProcessPurchaseOrder message. It also describes the response message, in this case, the AcknowledgePurchaseOrder (see Listing 2).

There's a variety of Web services tools on the market that help in creating the WSDL description from the exposed methods. Many can even generate executable code from the WSDL descriptions. Likewise, the Web services engines can automatically wrapper content inside the SOAP envelope.

Summary

Don't forget the business language! While the business language doesn't have the "cool" factor of the protocol standards, it is still essential for integration.

Let's not repeat the mistakes of the past. Hopefully, we've learned something through the years. By using a canonical business language for integration, we don't limit ourselves to point-to-point, one-off integrations.

OAGIS is the only canonical business language available today! OAGI enables this by working with industry groups to ensure that OAGIS meets the specific needs of these industry groups, while maintaining the horizontal base that enables communication across domains. OAGI will continue to expand upon the capabilities of OAGIS and the industries that are supported in this manner.

By providing the missing piece needed for integration OAGIS ensures that Web services can provide the ROI promised. Without using OAGIS, Web services implementations will use proprietary XML data formats that are incompatible with one another, which will lead to the mess shown in Figure 1 and serve only to add to the Babel!

Using OAGIS with Web services enables integration with back-end applications and across company boundaries within and outside of a given vertical industry. This enables small, medium, and large organizations to work on a level playing field.

For more information about OAGI or OAGIS 8.0, please see the Open Applications Group's Web site at www.openapplications.org.

AUTHOR BIO

Michael Rowell, chief architect for OAGI, is responsible for technical and content development for the group's specifications. He led the OAGIS 8.0 architecture team in adapting OAGIS to XML Schema, resulting in OAGIS 8.0. Michael has been involved with XML since the beginning, in his current position as well as previously with IBM, in both solution provider and tool provider roles.

MROWELL@OPENAPPLICATIONS.ORG

LISTING 1

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/">

 <SOAP-ENV:Body>
  <oaws:PurchaseOrder_Operation xmlns:oaws="http://www.ope-
napplications.org/webservices">
   <oa:ProcessPurchaseOrder xmlns:oa="http://www.openapplica-
tions.org/oagis"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openapplications.org/oagis
../BODs/ProcessPurchaseOrder.xsd" revision="8.0" environ-
ment="Test" lang="en-US">
 <oa:ApplicationArea>
  ....
 </oa:ApplicationArea>
 <oa:DataArea>
  <oa:Process/>
  <oa:PurchaseOrder>
   ....
  </oa:PurchaseOrder>
 </oa:DataArea>
</oa:ProcessPurchaseOrder>
  </oaws:PurchaseOrder_Operation>
 </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

LISTING 2

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:oa="http://www.openapplications.org/oagis/"
targetNamespace="http://www.openapplications.org/webser-
vices/PurchaseOrder.wsdl">

 <message name="ProcessPurchaseOrder">
  <part name="ProcessPurchaseOrder" type="xs:string"/>
 </message>
 <message name="AcknowledgePurchaseOrder">
  <part name="AcknowledgePurchaseOrder" type="xs:string"/>
 </message>

 <portType name="PurchaseOrder_PortType">
  <operation name="PurchaseOrder_Operation">
   <input name="ProcessPO" message="ProcessPurchaseOrder"/>
   <output name="AcknowledgePO"
message="AcknowledgePurchaseOrder"/>
  </operation>
 </portType>

 <binding name="PurchaseOrder_Binding"
type="PurchaseOrder_PortType">
  <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="PurchaseOrder_Operation">
   <input>
    <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.openapplications.org/oagis"/>
   </input>
   <output>
    <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.openapplications.org/oagis"/>
   </output>
  </operation>
 </binding>

 <service name="ProcessPurchaseOrders">
  <port name="ProcessPurchaseOrder_Port"
binding="PurchaseOrder_Binding">
   <soap:address
location="http://localhost:8080/soap/servlet/soaprouter"/>
  </port>
 </service>
</definitions>
```

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

WRITTEN BY **CHRIS BRANDIN**

Maximizing the Usefulness of XML

Good grammar and style are key

Is XML documents or data? Should XML be managed with a database or a document management system? Should XML be managed at all, or is it simply a data interchange standard? These are among the most common questions people ask about XML when they're trying to get a grip on what XML is.

These questions get at what we do with XML, not what it is. Another question: Is a computer a word processor or a video game? Neither? Both? Actually, a computer is more. Word processors are computers, as are video games; but it doesn't work the other way around. We cannot define a computer in terms of a single application, or even a multitude of applications for that matter. Similarly, we cannot restrict our definition of XML in terms of its uses.

XML is a standard for expressing information in a complete form. It contains not only data, but also context and attributes. XML is, in other words, informationally complete. That's why it's useful for so many different things. XML plays an important role in unifying information from disparate applications. It can likewise play a role in unifying disparate functions within applications. Does this mean that we can use XML for every aspect of what an application does – gather, present, store, manage, transport, transform, and process information? In a word – *yes*. There are a number of important advantages in doing this, among them:

- ***XML is intuitive:*** It's easy to understand for technical and nontechnical users alike.
- ***XML is simple and extensible:*** We can build applications faster, and change them much faster.
- ***XML is standardized and ubiquitous:*** Applications can be built to be compatible with almost everything.
- ***XML is flexible and heterogeneous:*** We can eliminate the rather excruciating exercise of trying to fit information in predefined containers.
- ***XML expresses complete information:*** We can build unified and universal information models behind applications, instead of a series of redundant but different data models serving individual functions.
- ***XML encompasses standards:*** (such as XML Schema) for providing robust and flexible controls for maintaining information integrity.

As long as XML was used as a container for data it was sufficient to consider only syntax when building documents. To do more, we must consider grammar and style as well. Obviously, proper syntax is necessary for XML to be usable at all. Good grammar ensures that once XML information has been created, it can be subsequently interpreted without an inordinate need for specific (and redundant) domain knowledge on the part of application program components. Good style ensures good application performance, particularly when it comes to storing, retrieving, and managing information.

Most application programs encompass the same basic functions – input, presentation, communication, processing, and information management. Although the same information underlies each of these functions, different data models have traditionally been employed to accommodate application components that accomplish these tasks. Even with XML, remodeling the same information in different ways for each component of an application is inefficient and yields programs that are buggy and difficult to maintain and change. A better way is to create a central, unified model for information that can adequately accommodate all functions of an application (see Figure 1).

To create XML information models that can do more than accommodate single functions, we have to look at XML in a different way – as an information domain rather than simply a transport standard. For simple data transport applications we can use XML any way we like, as long as it is syntactically correct. To do more, complete, provable, and unambiguous XML models must be built. Creating robust universal information models in XML is easier than it may seem at first. The first step is to understand the basic patterns inherent to information expressed in XML.

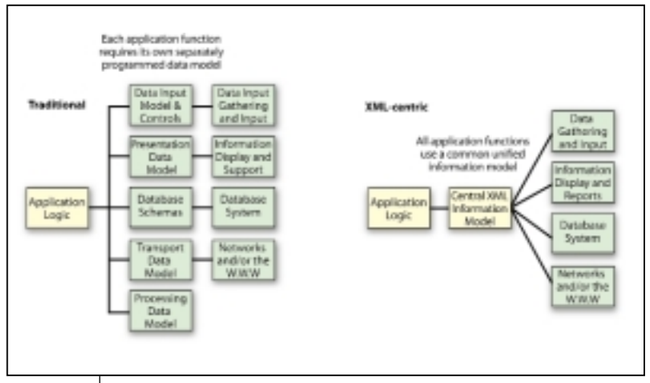


FIGURE 1 Application environments

AUTHOR BIO

Chris Brandin is the inventor and chief architect of NeoCore's technology and products, and a pioneer in parallel processing computing. Before NeoCore, he was the CEO of Business Operating Systems, Inc., where he developed computer systems used by the New York Stock Exchange and other exchanges. He also designed control integrated circuits for PC I/O bus management, music synthesizer hardware and software, and other products marketed by Symantec, NEC, Alcatel and others. Brandin has consulted on patent issues for Motorola, AT&T, Bell Labs, Belcore, Wang, IBM, Intel and others.

XML consists primarily of tags, attributes, and data elements. Tags provide context; in other words, tags describe what data elements in their scope are. Attributes provide information about or indicate how to interpret data elements in their scope. Data elements represent data in the traditional sense. The structure of XML also provides information – about hierarchy, groupings, relationships, etc. It is possible to create meaningless XML. For example, you could create perfectly correct XML by taking the entire text from a telephone book and simply putting "<Telephone_ Book>" at the beginning and "</Telephone_ Book>" at the end. It would be perfectly correct XML, but not useful XML. In this case XML is being used solely as a container for a block of data, and provides no context for the information contained therein. At the other extreme we have XML where all information is expressed in a semantically meaningful way. For example, consider the following XML fragment:

```
<Telephone_Book_Listing>
 <Name>
  <Last> Smith </Last>
  <First> Tracy </First>
 </Name>
 <Telephone>
   <Area_Code> 719 </Area_Code>
   <Number> 555-1234 </Number>
 </Telephone>
</Telephone_Book_Listing>
```

The explicit patterns in the example above are:

- This is a telephone phone book listing.
- The listing's last name is "Smith".
- The listing's first name is "Tracy".
- The telephone number's area code is "719".
- The telephone number is "555-1234".

Patterns implicit in the example are:

- "Smith" and "Tracy" belong to the instance group "Name" and are the last and first name, respectively.
- "719" and "555-1234" belong to the instance group "Telephone" and are the area code and number respectively.
- Everything belongs to the instance group "Telephone_Book_Listing".

Explicit patterns are typically converted to database columns with some or all of them being available as query terms. At least one XML information management system (NeoCore XMS) automatically organizes itself around these natural XML patterns and indexes them all without the need for any database design. Implicit patterns are used to determine groupings, relationships, and sometimes convergence points for query set intersections.

Much like a Web browser can determine how to display HTML information based on presentation metadata embedded in the HTML, application components can determine how to treat and interpret XML information based on semantic metadata embedded in it. XML can contain any kind of metadata. This is where XML differs from HTML. Where HTML was targeted to a single function – the presentation of information – XML fulfills a more universal purpose – the complete characterization of information.

Key to creating useful XML is to create semantically meaningful XML first. The easiest way to do this is to simply create XML representations that are easy for humans to read and understand, which is what we did in the sample XML fragment. If you were to create a manual entry form for the XML fragment it would probably look something like Figure 2.

Note that the XML fragment is a direct and obvious analog of how you would represent the information on a manual entry form. All we have done is represent the preprinted parts as tags and the filled-in parts as data elements. The hierarchy we created is likewise obvious. This method of creating grammatically valid XML may seem so simple as to hardly be worth mentioning, but the fact is that grammatically valid XML is relatively rare. To illustrate why, we will examine some common mistakes programmers make when creating XML.

These examples will use an application dealing with colorimeter readings. A colorimeter is a device that measures color using tristimulus readings utilizing a number of color models. For example, a colorimeter can be used to measure computer monitor colors using red-green-blue components of light (this being the "RGB" color model). This particular colorimeter can provide readings in multiple resolutions. A manual form for transcribing colorimeter readings might look something like Figure 3 after being filled in.

A typical way to characterize this information in XML follows:

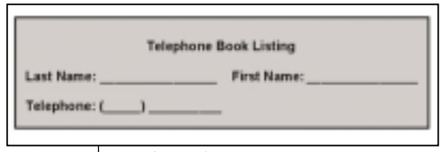


FIGURE 2 Manual entry form

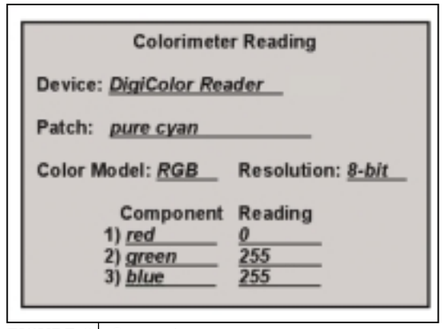


FIGURE 3 Colorimeter reading

```
<colorimeter_reading>
 <device> DigiColor Reader </device>
 <patch> pure cyan </patch>
 <RGB resolution=8 red=0
  green=25blue=255 />
</colorimeter_reading>
```

Here, the information modeler has made a couple of optimizations in the interest of saving space. First, the "Color Model: RGB" field has been collapsed into the single tag, "<RGB>". This is a reasonable thing to do as the color model information arguably (and unambiguously) indicates what the readings are for, and therefore qualifies as true context. Second, the readings themselves have been collapsed into three attributes: "red=0", "green=255", and "blue=255". Expressing data elements as attributes is a common practice. Although syntactically correct, this approach brings a number of problems:

- Attributes provide information about or information on how to interpret data elements in their scope. The readings are not attributes; they represent data from the colorimeter.
- The first attribute "resolution=8" is needed to correctly interpret the readings. Attributes apply to items in their scope, not each other. We have a mixture of attributes and data expressed as attributes with no indication of how they relate.
- All four attributes apply to nothing because there is nothing in their scope.

Another common way this form might be modeled is shown in Listing 1. Here, the

entire contents of the form have been flattened into one level of hierarchy. There are several problems with this model:

- Two of the data elements, "RGB" and "8", provide context that is necessary to interpret the readings correctly. They are not cast in a way that puts the readings in their scope.
- Which readings values belong to which bands is ambiguous. Even though you could argue that the ordering implies grouping, a database may not know the difference and return false query results. For example, suppose you wanted to return any colorimeter reading with a band of "green" and a value of "0". According to this model, there is no unambiguous indication of which band belongs to which reading, and this XML fragment contains both terms; so a database could falsely return this fragment even though it actually has a green band reading of 255.
- This model is not extensible. If we wanted to include readings in another color model (CIE xyY, for example), there would not be a practical way to do so.

Listing 2 shows the most literal XML representation of the colorimeter readings form.

Although this listing is grammatically correct, it is not optimal – nor is it particularly good style. This model represents another common practice: expressing context as data elements in name/value pairs rather than as tags. The weaknesses of this model include:

- Two of the information items, "color_model" and "resolution", do not mean anything by themselves. They actually represent metadata necessary to correctly interpret and understand the meaning of the readings. Moreover, they are stand-alone (they have no scope), so their applicability is unspecified. The "patch" information item, on the other hand, is correctly cast as a separate item because it is not needed to determine how to interpret the readings; rather it specifies what the readings are for.
- The "component" information items are not actually data. They represent the color bands for the readings, so they are really context. The major drawback to breaking information down into name/value pairs is that it can adversely affect performance because simple queries against the information item result in join operations. This is often done to emulate the ability to support heterogeneous information in relational database management systems. Information modelers who find themselves doing a lot of this sort of thing may want to consider using an XML information management system that natively supports heterogeneity – performance and scalability will be much better.

There's no one "perfect" XML information model for this application. By applying a few simple techniques, however, a model can be built that is unambiguous, performs well, and will serve all components of the application program. First we examine each information item in order to determine what it really is – data, context, or attribute. Going down the list of items in the original colorimeter reading form, the following interpretations and actions would certainly be reasonable:

- "Colorimeter Reading" is the name of the form, so using this item as the root name is appropriate.
- "Device: DigiColor Reader" represents a stand-alone information item about the specific equipment used to gather the readings, so it will be represented as a stand-alone tag/data element pair.
- "Patch: pure cyan" represents a stand-alone information item about the entire dataset, so it will be represented as a stand-alone tag/data element pair.
- "Color Model: RGB" specifies a context for all the readings. It could be cast as an attribute or as a tag. Because it specifies what the readings are in the aggregate, rather than how to interpret the readings individually, it will be included as a tag at the root of the readings.
- "Resolution: 8-bit" directly applies to the interpretation of the readings below it, so it will be cast as an attribute encompassing all three readings.
- The "Component" and "Reading" items are arguably redundant. You could cast these as tags, but in the interest of efficiency we will eliminate them.
- The readings each consist of two parts: a color band component and the reading. We will make the color band components tags, as they are context for the readings. The readings will be cast as data elements.

Converting all this into an XML fragment yields the following:

```
<colorimeter_reading>
 <device> DigiColor Reader </device>
 <patch> pure cyan </patch>
 <color_model_RGB resolution=8>
  <red> 0 </red>
  <green> 255 </green>
  <blue> 255 </blue>
 </color_model_RGB>
</colorimeter_reading>
```

This model fulfills all the requirements of good grammar and good style, yet remains relatively terse:

- All tags truly represent context and have the proper scope.
- There are no meaningless or redundant tags.
- All data elements truly represent data. No data elements are necessary for the proper interpretation of other data elements.
- Attributes are cast such that they directly apply to the interpretation of all data elements in their scope and no others.
- There are no tag/data element pairs masqueraded as attributes.
- The model is extensible. If we wanted to add readings in the CIE xyY model, for example, we could do so by simply adding a group called "<color_model_CIE_xyY>" to the XML fragment.

So far, we have discussed creating semantically valid XML and how it applies to how application components interpret information. The next step involves how applications can be architected to leverage XML as a central information model in an unambiguous and provable environment. To accomplish this, XML must be used in a somewhat more stringent way than many application developers are accustomed to. From the discussions above we can establish a series of rules for modeling information in XML:

- Data elements should be used to express only data, not metadata.
- Tags should indicate what the data elements in their scope are, not how to interpret them.
- Tags should apply directly to all data elements in their scope.
- Tag hierarchies should clearly group information items belonging together.
- Tag hierarchies should clearly separate information items not directly related to each other.
- Attributes should indicate how to interpret or provide information about information items in their scope.
- Attributes should not be critical to interpreting what the information items in their scope are.
- Attributes should apply directly to all information items in their scope, and no others.
- Attributes should be understood by all application components that will encounter them.

The last rule listed is very important and bears some explanation. XML is very flexible, in fact, too flexible the way it is often used. The one practice most responsible for making XML information models difficult to prove and control is the abuse of attributes. Developers regularly use tag/data element pairs and attributes interchangeably in the interest of avoiding data-bloat (an unnecessary practice because even trivial compression techniques can eliminate that problem). This brings an unfortunate ambiguity to information expressed in XML. Attributes are intended to provide information about or how to interpret items in their scope. If an application program encounters a tag that it does not understand, it will ignore everything in its scope, including all attributes. If an application program recognizes all the tags leading up to a data element, for example, it knows what the data element is; but if an attribute is encountered that is not understood, the application program may or may not know how to interpret the data element, and this situation is ambiguous. In order to enforce information integrity controls we need a provable, unambiguous mechanism. Attributes and XML Schema definitions provide suitable mechanisms to do this, as long as an exception is thrown whenever an attribute that is not understood is encountered. To control XML information we need to use a combination of attribute interpretation and schema validation. This gives us an arbitrary and fine-grained degree of control unachievable with traditional databases. This should be done once, with a centrally controlled mechanism. To accomplish this we need to architect enterprise applications in a new way. Traditionally, application programs serve as user interfaces and database systems manage and control data. In the XML world, a three-component architecture should be employed: the user-facing application, the XML information management system, and an information integrity enforcer (schema validator and attribute interpreter) (see Figure 4).

The information integrity enforcer can be implemented in a number of ways: as a server-side extension, as a standard application program component, or as a Web service. The trick is to have a common information integrity enforcer for each category of application. In order to have applications interact with XML information in a consistent and provable manner, the following model can be adopted:

- If all tags leading up to a data element are understood, the application can assume it can correctly determine what that data element represents.
- If a tag leading up to a data element is not understood, the application must assume it does not have a complete context for that data element.
- An application component may ignore information containing a tag that is not understood.
- If all attributes having a data element in their scope are understood, the application component can assume it can correctly interpret that data element.
- An application must not assume it can correctly interpret any data elements that are in the scope of an attribute that is not understood, or violate it. It is up to the information integrity enforcer module (whether a separate module or an integral part of the application) to indicate that such an attribute has been encountered and to which information items it applies. Such exceptions should be treated as catastrophic in so far as the information items in question are concerned.
- An application must be able to interpret attributes and their contents unless there is a specific rule stipulating that an attribute can be ignored (a font attribute, for example, may be ignored by application components not responsible for presentation).
- Schema validation violations should be treated as potentially catastrophic information integrity failures.

Following these XML information modeling guidelines is not substantially more difficult than building semantically weak XML, and the payback can be considerable. Not only can XML be used to integrate disparate data sources, but it can be used as a unified means to express the information underlying functional components of application programs as well. This results in better application programs that can be built faster and maintained with less effort. One thing is inevitable: as information technology progresses into things like the semantic Web and more and more computer systems need to be integrated, information expressed in XML will have to become increasingly semantically valid in order for us to be able to keep up. XML already contains all the elements necessary to do this – we just have to be a little more thoughtful about the ways we use it.

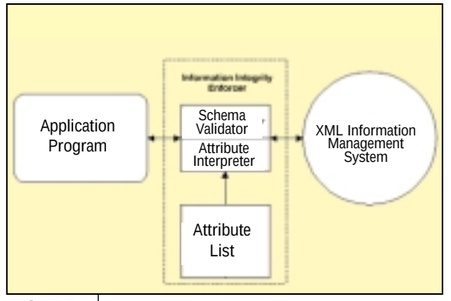


FIGURE 4 Information integrity enforcer

CBRANDIN@NEOCORE.COM

LISTING 1

```
<colorimeter_reading>
<device> DigiColor Reader
   </device>
<patch> pure cyan </patch>
<color_model> RGB
   </color_model>
<resolution> 8 </resolution>
<band> red </band>
<value> 0 </value>
<band> green </band>
<value> 255 </value>
<band> blue </band>
<value> 255 </value>
</colorimeter_reading>
```

LISTING 2

```
<colorimeter_reading>
 <device> DigiColor Reader
   </device>
 <patch> pure cyan </patch>
 <color_model> RGB
   </color_model>
 <resolution> 8-bit </resolu
   tion>
 <reading_1>
  <component> red </component>
  <reading> 0 </reading>
 </reading_1>
 <reading_2>
  <component> green </compo
   nent>
  <reading> 255 </reading>
 </reading_2>
 <reading_3>
  <component> blue </component>
  <reading> 255 </reading>
 </reading_3>
</colorimeter_reading>
```

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

WRITTEN BY **IVAN IMANA**

The Travel Industry's First Data Integration Web Service

Adelman Travel partners with Oreceipt

Adelman Travel Group, a leader in online booking and a top-30 travel management company, has partnered with Oreceipt, a company that specializes in providing XML-based Web services, to enable Adelman to integrate data, applications, and processes with its clients.

As a result of this partnership, Adelman is the first travel agency to enable clients to access travel invoices from the Web and print and/or e-mail them to appropriate parties within their own organizations. In addition, the XML-based travel receipts, which contain actual data, enable Adelman's clients to integrate travel data into their back-end systems for expense reporting and accounting.

Oreceipt, a San Jose–based company specializing in XML- and Web services–based solutions, offers a unique interbusiness data integration solution that can help companies lower costs by integrating data and processes with customers and suppliers. Oreceipt focuses specifically on post-transaction data, such as invoices, receipts, and warranties, and the processes that utilize this data. The case study presented here is based on travel sector business but is applicable to a variety of other businesses.

At Adelman Travel Group, one of the earliest adopters of Web services and XML in the travel industry, we saw the benefits of integrating data and processes with customers and realized that XML and Web services are the best means of achieving integration goals.

Like any other business, the travel business generates a lot of data before and after sales, and this data needs to be shared between buyers and sellers. Automation in sharing and processing this data can increase efficiency and lower costs for everyone involved in the transaction.

Billing and invoicing is one area in which we saw a good opportunity to offer XML- and Web services–based data integration capabilities. We needed a solution that offered:

- Capability of delivering XML data using several different protocols: HTTP, e-mail, and FTP.
- Ability to customize data to meet individual needs. Some customers require that specific or additional content be added along with the invoicing information, while other customers want the XML to be transformed to a different grammar before delivery.
- Provision of a Document Management Service for customers who don't require XML-based invoices on a regular basis, but need the ability to search and find specific invoices as required. These customers also desire the ability to forward the XML or HTML versions of the selected invoices by e-mail.
- Good invoice presentation capabilities. One requirement, for example, is that whenever the XML data is presented to travel and accounting managers, the invoices should look very similar to the paper version of our invoices.
- Tracking of travel and billing information on a per department or business unit basis – the solution had to be capable of providing this data to customers at the individual department level as well as provide consolidated billing information.
- A good framework that enables easy extension and customization as new requirements surface.
- Good management capabilities that require very little attention from the system administrators (when attention is required, the system must be very easy and intuitive to use).
- Good professional services to help us customize solutions in a quick and reliable manner.

Oreceipt provides a Web services–based solution that meets all these requirements. It comprises several loosely coupled Web services components that come together to provide a flexible, extensible solution (see Figure 1).

XML Generator

The XML Generator is responsible for accessing the data source and generating XML for documents needed by customers, such as invoices, receipts, and warranties. This service can be automatically generated using a GUI-based configuration tool that allows users to describe the data source and the conversion rules. This method is significantly faster than traditional development methods.

After the XML data is generated, the XML Generator provides the data that the XML delivery Web service will deliver to the appropriate recipients; this can include the Document Management Service as well.

XML Delivery

XML Delivery is a Web service that receives incoming XML documents,

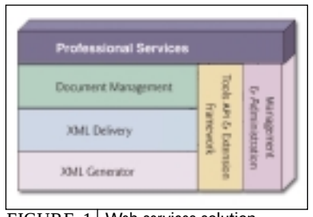


FIGURE 1 | Web services solution

AUTHOR BIO

Ivan Imana is the technology solutions manager for Adelman Travel Group. He is responsible for evaluationg all of Adelman's automated systems, from networking to telecommunications. Ivan received a BSBA from San Diego State University.

determines the recipients, and based on their specific needs, delivers the data using HTTP, e-mail, or FTP. This service is also responsible for applying transformation rules to convert the data to a format that's desired by individual customers.

Document Management Service

The Document Management Service is a Web-based document management solution that allows customers to track their invoices online. They can search for these documents based on many criteria, view and print these documents, forward them to other people by e-mail, and so on.

The Document Management Service is very extensible. It allows our IT department to write custom triggers for document arrival and removal to offer business-specific services. For example, we use this feature to extract travel industry–specific information, such as the PNR locator and traveler name, from incoming documents to provide additional search capabilities that other industries don't require.

API and Extension Framework

This is a set of Java classes and libraries available with the solution that allows our IT department to write custom software around the XML documents or develop extensions and customizations for the document management solution. We utilize this to provide customized XML documents that meet our customers' needs as well as non-standard document delivery requirements imposed by some customers.

Management and Administration Tools

These are Web-based tools that allow our system administrators to configure the system. These tools allow them to:

- Create new customer accounts for document managers
- Modify or delete customer accounts
- Set up relationships between individual and consolidated accounts for customers
- Specify XML delivery options for customers
- Specify XML translation options for customers
- Specify server information about e-mail servers and XML delivery Web services

Professional Services

Professional services offered by Oreceipt helped us expedite deployment of the solution. The following services are provided (see Figure 2):

- Design and development of XSLT to convert XML documents into presentable HTML documents that look similar to the paper version of the corresponding documents
- Extension and customization of the solution to meet our business-specific needs

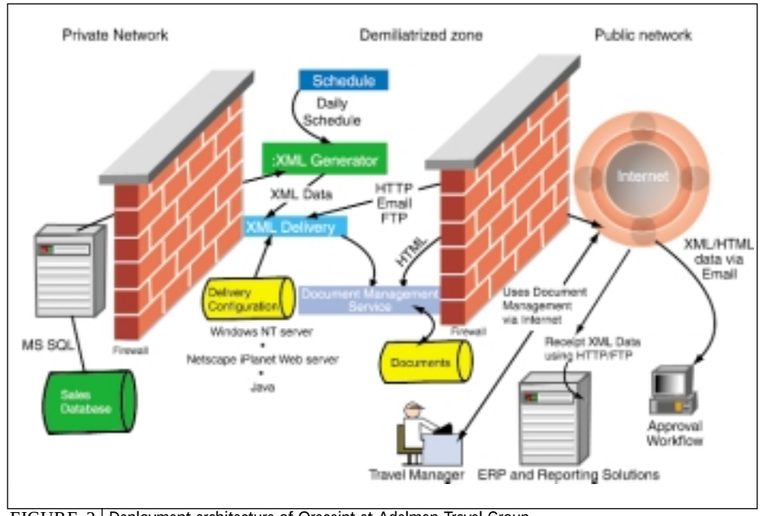


FIGURE 2 | Deployment architecture of Oreceipt at Adelman Travel Group

How We Satisfy Customer Requirements Using Oreceipt

Authorization purposes

Pacific Life is one of Adelman Travel Group's largest customers. Like many other companies, Pacific Life requires travel to be authorized by the appropriate department heads and managers, and that invoices for all bookings be sent for approval before the start of the next business day. They also require that the invoices be presented in HTML, with Pacific Life–specific information in the invoice. The invoice had to be delivered via e-mail every morning to allow distribution to appropriate managers for authorization before the actual travel.

The Oreceipt solution schedules XML to be generated every morning (before business hours) for all the invoices from the previous day's bookings. These XML documents are immediately forwarded to the XML delivery Web service, which transforms the XML based on Pacific Life–specific requirements and e-mails them to the appropriate person(s) at Pacific Life for approval and authorization.

Integration with expense reporting

Another customer required that invoices be provided in XML format for integration with their reporting and analysis system. They desired that the data be delivered to them via FTP. They also required that the contents of the data be transformed to an XML grammar specified by their application software and delivered on a weekly rather than a daily basis. Oreceipt's FTP and custom data translation capabilities meet this customer's needs.

Search and print invoices as needed

Yet another major customer requires facilities to search and find specific invoices during their expense reconciliation process. They require a Web-based service that can be used for this purpose. This customer also has several accounts and needed a facility to search for invoices across these accounts with a single query. Document Management Service, with its account consolidation service, enabled us to meet this customer's needs.

As more of our customers sign on to receive this service, we'll provide many additional services as required by our customers, and Oreceipt's customizability and professional services will certainly be useful in this regard.

For more information about Oreceipt, visit www.oreceipt.com. For more information about Adelman Travel, visit www.adelmantravel.com.

IIMANA@ADELMANMAIL.COM

Build Applications with Less Code

WRITTEN BY
MARCELO F. OCHOA
& ROEL FRANKEN

MetaBOX and the declarative programming model

This article shows how to build database Web applications using Oracle, two open-source frameworks, and MetaBOX software. Building database Web applications is commonly a repetitive task; many such applications are based on insert/delete/update statements of tables in a form of simple rows, multiple rows, or master/detail combinations.

To make this kind of application, many products offer different solutions – we can divide them into procedural or declarative approaches. These products provide wizards or code generators to help developers in the developing stage.

Depending on which technologies these generators use – pure HTML or XML – developers need to work more or less to get the desired output and maintain their applications. The procedural approach requires developers to know what they need and how to obtain it. Code generators for this approach generally use some kind of metadata from the applications such as tables, views, primary keys, foreign keys, and so on, and generate templates of the code for the application. An example of this approach is JSP generator of Oracle9*i* JDeveloper.

Developers who work with the declarative approach handle more abstract concepts such as table usage, permissions on these tables, primary keys, foreign keys, LOVs, and so on.

Both approaches have benefits and drawbacks. For example, the declarative approach has good productivity, portability, quality, and performance behaviors, but it's less flexible, and the performance improvement is for a certain application's domain.

On the other hand, the procedural approach is good for performance and flexibility, but the environment is less productive and less portable, and the quality of the code depends on the developer's skill. This article will explore a product developed on top of two open-source projects, named MetaBOX, that provides declarative development for Oracle Web applications. It uses XML as the key technology to decouple presentation from content, because many of a Web application's changes involve presentation concerns.

MetaBOX uses the DBPrism Servlet Engine framework as a connector for the Apache Cocoon framework and generates the application's XML code inside an Oracle database running a stored procedure. This means that you use the database engine not only to execute SQL statements, but also to directly return a complex XML representation of the application data stored inside as well.

What Is DBPrism?

DBPrism is an open-source framework based on servlet technology to generate XML from a database. It began as servlet replacement for the Oracle Web Server PL/SQL Cartridge; then it was extended to generate XML into the database side, transforming it in an active database.

Other technologies, such as Apache XSP or Oracle XSQL Servlet, generate XML into the Web server side and use the database only to return SQL. They can use stored procedures, but 99% of these applications put the logic into the Web server side.

How does it work?

DBPrism is a middleware component that interprets a request from different sources and executes a database stored procedure using the information coded into the request.

DBPrism automatically transforms, for example, a standard HTTP request into a stored procedure call using the arguments extracted from the URL. After this execution DBPrism collects the generated XML, or HTML, and returns it to the requester.

Working as stand-alone servlet, DBPrism is fully compatible with the Oracle mod_plsql component, providing seamless integration with applications developed with Oracle WebDB or Oracle Designer Web Server Generator.

DBPrism also includes a DBPrismGenerator for the Apache Cocoon framework, which provides the framework with a new functionality and gets the XML from the database side, executing a stored procedure. This kind of XML generation moves application logic closest to the data and leaves presentation concerns to the Web server layer (see Figure 1).

DBPrism components

- ***Wrappers:*** DBPrism accepts requests from different sources, then it needs to adapt them to the internal representation using components called wrappers, which are also responsible for returning the generated XML or HTML.
- ***Engine:*** A component that implements the core functionality of DBPrism framework, it uses generic components in an internal format and doesn't deal with database-dependent information such as types of arguments supported by stored procedures, representations of generated pages, and so on. The engine uses the concept of the DAD to get the necessary information about username and password, as well as the type of database of a specific connection; this DAD is defined by the site administrator in the prism.properties file. This file also includes transactions defined by URL demarcation, connection pooling information, and other DBPrism parameters.
- ***Adapters:*** Components that talk with database-dependent components. DBPrism includes one adapter for every database supported; by default it includes support for Oracle 7.x, 8*i*, 9*i*, and Oracle Lite, but the design permits future add-on support for other databases (see Figure 2).

Note about DADs: DBPrism uses the concept of DAD extracted from the URL information. This means, for example, for a URL like http://server:port/xmlj/samples/DEMOj.startup, that xmlj will be used as the DAD name; then in the prism.properties file there will be an entry identified by the key "xmlj", with the information to connect to the database. This information includes not only username and password, but also the type of database, for example, 8*i* or 9*i*; default page if no stored procedure is given in the URL information; and other database-specific information.

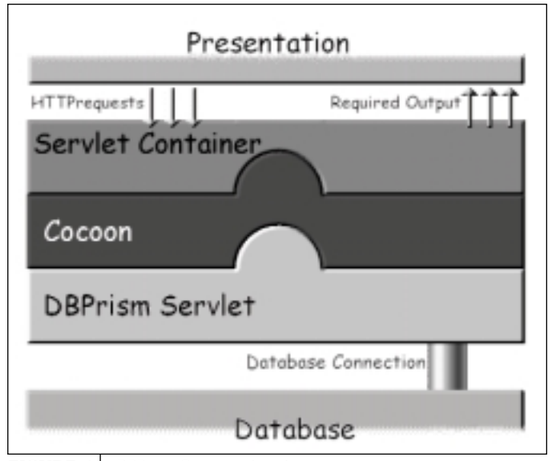


FIGURE 1 DBPrism Cocoon integration layer

What Is Cocoon?

Apache Cocoon is an XML publishing framework from Apache Software Foundation. It uses XML and XSLT transformations as basic technologies and includes sophisticated implementation for caching, content aggregation (portal aggregation), and so on.

Cocoon is designed for scalability and performance because it uses a pipeline SAX-based approach, which means that Cocoon doesn't make in-memory, DOM-based representations of XML documents. Consequently, it doesn't require a great deal of memory to process large XML documents or multiple concurrent requests.

The Cocoon framework design provides multiple points of interaction to extend or customize components. As a basic mechanism for getting the XML source, Cocoon provides generators that interact with most data sources, including file systems, RDBMS, LDAP, native XML databases, and network-based datasources. DBPrism provides a new high-performance generator for Cocoon that includes support for parallel content aggregation and Edge Side Included Invalidation Protocol (www.w3.org/TR/esi-invp).

Cocoon's endpoints are serializers; they adapt content delivery to the capabilities of different devices like HTML, WML, PDF, SVG, RTF, and so on. You can run Cocoon as servlet (our approach) as well as through a powerful, command-line interface.

Separation of concerns (SoC)

Many products, such as Oracle XSQL Servlet or Sun's JSTL XML Utility tags for JSP, provide server-side transformation of XML using XSLT, but the most important innovation that Cocoon adds is the SoC-based design, which divides the development stage of a Web application into four separated areas joined by clear "contracts" that define their operability and concerns.

This concept allows separated people with common skills in different working groups to increase their productivity, while reducing management costs.

The pyramid of contracts defines the four areas of concern and the five contracts between them (see Figure 3).

The Cocoon pyramid model of contracts

Removing the contract between style and logic solves many of the problems of Web site development, because graphics designers and programmers have different skills. Developers concentrate their efforts on the application logic and content without taking into account presentation concerns. Graphic designers don't work with application logic; they concentrate their efforts on the look and feel of the application. MetaBOX software addresses two of these concerns, logic and content, using the declarative programming model.

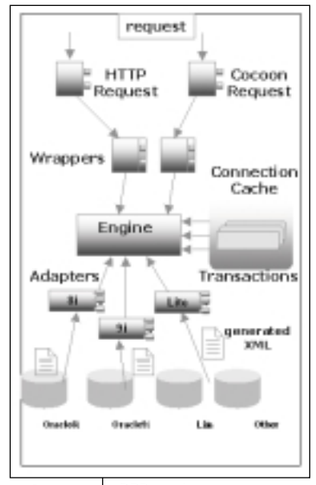


FIGURE 2 DBPrism framework components

Figure 4 shows the execution flow into the Cocoon system for every request. The request starts evaluating a regular expression matcher and defines which execution plan is used to generate the desired output. A particular execution plan could involve the following components:

- ***Matchers:*** Attempt to match a URI with a specified pattern for dispatching the request to a specific processing pipeline. Matchers help specify a pipeline processing for a group of URIs.
- ***Generators:*** Used to create an XML structure from an input source (file, directory, stream, DBPrism, etc.). DBPrism generates the dynamic XML inside the database by executing a stored procedure.
- ***Transformers:*** Used to map an input XML structure into another XML structure. For example, XSLT Transformer, Log Transformer, SQL Transformer, and I18N Transformer.
- ***Aggregators:*** Content aggregation means that an XML document could be composed (aggregated) by many sub documents.
- ***Serializers:*** Used to render an input XML structure into some other format (not necessarily XML), for example, HML Serializer, FOP Serializer, Text Serializer, and XML Serializer.

A part of sitemap.xmap file is shown below. Sitemap.xmap file is a configuration file used by Cocoon to define its component and portal URL matching configuration.

```
<map:match pattern="/index.html">
  <map:generatesrc="static/index.xml"/>
  <map:transformsrc="stylesheets/stat
ic-xml2html.xsl"/>
     <map:serialize type="html"/>
     </map:match>
```

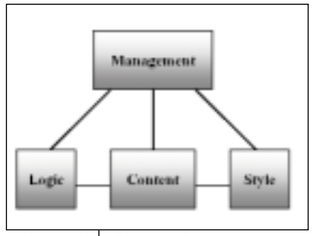


FIGURE 3 Pyramid of contracts

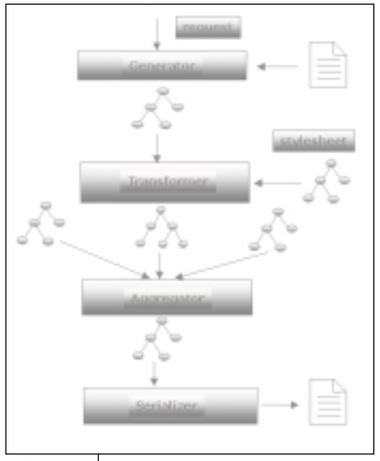


FIGURE 4 Execution flow

The pipeline example shown in the preceding code defines a matching expression for the /index.html page of a particular Web site. This home page will be generated by loading a static page "index.xml" located in the "static" directory, then it will be converted using a stylesheet "static-xml2html.xsl" located in the "stylesheet" directory, and finally it will be serialized to HTML format.

MetaBOX

MetaBOX is a software product based on the concept of self-describing systems. It enables software developers to build Web applications for the Oracle platform – applications that can be tailored to the personal preferences of individual end users.

As content and presentation are separated from each other in the MetaBOX concept (inherited from the pyramid model of Cocoon), applications are more manageable. This separation makes it possible to change the look and feel of an application completely, without having to change its code. All the output produced by MetaBOX is in XML format, and the look and feel is defined in XSL stylesheets (and Cascading StyleSheets when HTML is generated). This makes MetaBOX compliant with W3C-endorsed and supported standards. MetaBOX is developed using standard Oracle8*i*/9*i* components, allowing seamless integration with existing Oracle applications.

MetaBOX has all the benefits provided by declarative programming, plus personification, support for different media types, and multilingual support.

Architecture

MetaBOX is based on a three-tier architecture. This means that client, Web server, and database tiers could all reside in different hardware.

The MetaBOX repository could manage different applications (client applications) with one instance of MetaBOX at runtime. This means that different applications will use the same MetaBOX engine, and this engine will generate the correct output for each application based on the user's input (request) and the application's metadata stored in the MetaBOX repository.

The Web server sends the client's instructions to the appropriate application. The application processes a user's request by calling one or more PL/SQL procedures. PL/SQL is Oracle's procedural programming language, which runs entirely in the database. PL/SQL is the fastest way to process data into the Oracle platform. The MetaBOX itself is built entirely with PL/SQL and consists of a number of stored procedures (packages).

The result of any command sent to MetaBOX is always an XML document, and it is returned to the Web server tier for rendering to the target output by Cocoon, which uses the appropriate XSL stylesheet. Users see the result in a Web browser or in another device such as a PDA or a PDF viewer.

Building Web Forms Using Metadata

Two types of metadata

Within the MetaBOX concept, two kinds of metadata can be stored for every individual application:

- ***Database metadata:*** The "lower level" of the metadata repository: tables, keys, columns, key columns, etc. This kind of metadata can be almost completely reverse engineered from the database dictionary.
- ***Presentation metadata:*** Screen and reports definitions: the presentation metadata is created by the application developer when he or she defines the application. For this, a separate application was built (using MetaBOX, of course…).

Types of screens MetaBOX can generate

Building a metadata-based environment means you have to think about what functionality your environment will support. MetaBOX supports the following screens, which are typical for data-oriented applications:

- ***Multirow screens:*** Database records are shown one below the other.
- ***Single-row screens:*** A single database record is shown.
- ***Input screens:*** A collection of fields that have no relationship with a database record is shown; this is used for collecting user input, e.g., when running a wizard.
- ***DML screens:*** A record can be deleted, updated, or inserted.
- ***Search screens:*** For entering criteria used by Oracle Text and then cached in an internal cache.
- ***Result screens:*** For browsing through the search results in cache (thus avoiding the need to search again when the Next Page button is pressed).
- ***List of Values (LOV) screens:*** Multirow screens that are shown in a separate window when the user wants to find one or more values and put them in a field on the main screen.
- ***Menus***

All these screens can be created in a declarative way: the application developer inserts records into the metadata repository, for example, when creating a new screen a record is added to the SCREENS/FORMS table.

Building screens with MetaBOX

Using MetaBOX to build a screen involves the following preliminary steps:

- ***System tasks:***
  - -Setting up Apache, which includes specifying virtual paths to image-files, CSS-files, etc.
  - -Setting up the application-specific XSL stylesheets at the specified locations; for a typical application these can be taken from the Console application.

FIGURE 5 Console application

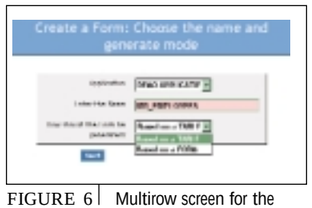


FIGURE 6 Multirow screen for the EMP table

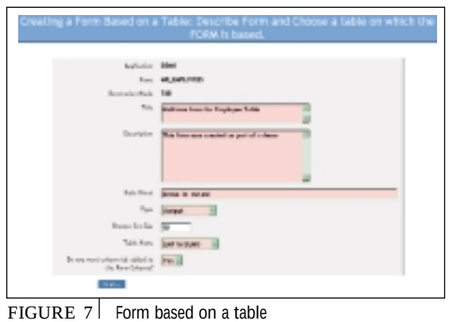


FIGURE 7 Form based on a table

-Setting up the DBPrism configuration: setting up a DAD.
-Setting up the Cocoon configuration: defining one or more pipelines.

- ***MetaBOX Tasks:***
  -Creating the application.
  -Reverse engineering the lower-level metadata.

Then the browser-based Console application can be used to create the first form (see Figure 5).

In this example, a DEMO application was created, after which the EMP(loyee) and DEPT (department) tables (from the Oracle Scott demo schema) were reverse engineered. An EMP record can reference a DEPT record with a foreign key.

In the screen capture, the application developer is about to start the form wizard, by which forms (screens) are created. We will create a multirow screen for the EMP table (see Figure 6).

In Figure 7, the Name is used for calling the form from the browser. The Title will be the title of the HTML page when the form is shown in the browser. The Description is for reference only. The Stylesheet can be used within the Console framework to identify the virtual path. When the form is called from a menu, the stylesheet will transform the data into the HTML Multirow screen. The TYPE=OUTPUT identifies the form as a screen that will show data from the database. The Browse Set Size determines how many records are shown on each page. Table Name names the table from which the data will be retrieved. (Note: this could very well be a view, in which records are sorted.) The last property of a form determines whether a reference to the database column is added to each Form Column (field). This must be Yes for any form that interacts with the database.

After clicking Finish on the last confirmation screen, the form is created. The form appears in the outliner menu of the Console application (see Figure 8).

Now the form is ready to be run on DBPrism/Cocoon. So in our browser we request: http://server:port/dbprism/webdemo/demo_ie_mr/scott.wrapper.webdem?p_context_name=FORM_NAME&p_context_value=MR_EMPLOYEES.

- ***Server:*** The name of the server running the Web server
- ***Port:*** The port on which the servlet engine (OC4J, Orion, Tomcat) listens
- ***dbprism:*** The name of the DBPrism application deployed on the servlet engine
- ***Webdemo:*** The DAD to use
- ***demo_ie_mr:*** The name of the stylesheet (DEMO_IE_MR.XSL). This is an input argument to the Cocoon Pipeline, defined as:

```
<map:match pattern="*/**">
  <map:generate  src="/webdemo/{2}"/>
   <map:transform src="stylesheets/{1}.xsl"/>
   <map:serialize type="html"/>
 </map:match>
```

So Cocoon uses the first argument (DEMO_IE_MR), after appending the .XSL extension to indicate the stylesheet to use to transform the source (XML) coming from: /webdemo/scott.wrapper.webdem?p_context_name=FORM_NAME&p_context_value=MR_EMPLOYEES.

The part starting with "scott." is the call to the application's wrapper stored procedure that will generate the XML.

As you can see, a fully functional screen is generated, with the ability to browse (next/previous page), filter, and sort. Of course the individual fields could be hidden, removed, reformatted, or given another label (see Figure 9).

Features

MetaBOX builds stateless applications, thereby allowing for simple deployment on the Internet. Some of its features are:

- Domains (either enumerated or based on a query)
- Oracle Text queries and caching of retrieved records
- Generating menus based on database roles
- Default values
- Exporting/importing applications' metadata
- Wizards determining a sequence of screens, based on user input and database data
- Assigning (multilingual) error messages to database constraints

How to take care of complexity

While creating simple applications is faster, easier, and cheaper when using metadata systems, this does not mean that using metadata makes it impossible to create complex applications. Within the MetaBOX concept, complexity is handled in three ways:

- ***Use of PL/SQL:*** With the MetaBOX concept "wrapper-stored procedures" are used. This means that every client application encapsulates the MetaBOX logic with an application-specific PL/SQL procedure: the wrapper. In this wrapper-stored procedure various tasks can be performed, such as the initializing of the application and adding application-specific logic. This existence of a wrapper-stored procedure allows you to use the robustness of PL/SQL programming language to tackle complex calculations and data operations. Object-oriented programming, creating complex data types, autonomous transactions…all this can be done with PL/SQL. Also, PL/SQL can wrap Java code stored in the database.

FIGURE 8 Form appears in the outliner menu of Console application

- ***Logic in the database:*** Another way to manage complex applications is the use of views together with triggers instead of triggers alone. This feature of the Oracle RDBMS allows a data modeler to develop a complete set of tables, views, and triggers separated from the MetaBOX application. To store as much business logic as possible in the database with triggers and views is always a good thing. Anyone who has migrated an application from a client/server platform to an Internet-based platform knows that any logic stored in the screens needs rebuilding (and retesting). But logic stored in the database can be reused without a problem. In addition, putting the logic in the database instead of the user interface (screen) also reduces duplication of code when other applications operate on this data (e.g., batch procedures).

  The rule is simple: think data. Put your code as close as possible to your data. The Oracle platform has many features (sometimes unknown to most servlet programmers) that can help you do this.

- ***Use of XSLT:*** The last part of managing complexity within the MetaBOX concept is the use of XSLT. Where Cascading Stylesheets (CSS) can give your screen another color or font, XSLT can create new buttons, assign new logic to these buttons, etc. Within the MetaBOX concept a set of general-purpose stylesheets is used by different applications. These general-purpose stylesheets perform functions such as determining the type of a field, retrieving a domain, printing the label on screen, and so on. In short, they perform the basic things that no application programmer wants to do time after time. To make sure the application developer can add his own application-specific code, the XSL:IMPORT element is used in conjunction with a "placeholder." Placeholders are present in the imported stylesheet and they can be overruled by the importing stylesheet – thereby allowing for application-specific output. This concept is shown in Figure 10.

The general-purpose stylesheets have these placeholder templates in every part of the HTML, so the controlling application can steer the content of the HTML, while many difficult tasks (like drawing a field with a select box – a domain) are done automatically in the imported general-purpose XSLT logic.

Of course, the imported XSLT logic in the general-purpose stylesheets is quite complicated. However, in this concept, solving a problem once allows you to reuse your solution in a constructive way.

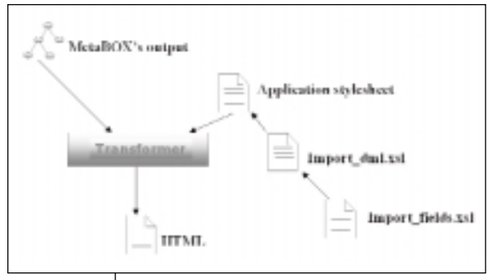


FIGURE 10 Application-specific output

Conclusion

Using XML/XSLT in conjunction with metadata is a fine combination. The techniques are similar in the way they allow reuse of general-purpose code. Using these techniques together with the XML publishing framework Cocoon leads to robust, flexible, and database-centered applications.

Resources

- www.dbprism.com.ar/dbprism/doc/Home.html
- http://xml.apche.org/cocoon
- www.importalis.com
- http://otn.oracle.com

Author Bio

Marcelo Ochoa works at the System Laboratory of Facultad de Ciencias Exactas of the Universidad Nacional del Centro de la Provincia de Buenos Aires and as an external consultant and trainer for Oracle Argentina. He divides his time between University jobs and external projects related to Oracle Web technologies. He has worked in several Oracle-related projects like translation of Oracle manuals and multimedia CBTs. His background is in database, network, Web, and Java technologies. In the XML world he is known as the developer of the DB Generator for the Apache Cocoon project, the framework that permits generation of XML from the database side.

Roel Franken is consultant at Importalis, a small firm specializing in creating data-centered XML applications for Oracle databases. He has worked wth Oracle technology for the past 12 years, mostly for governmental institutions and the insurance industry. Roel was actively involved in creating the Importalis metadata Web generator, the MetaBOX.

MOCHOA@IEEE.ORG
RFRANKEN@IMPORTALIS.COM

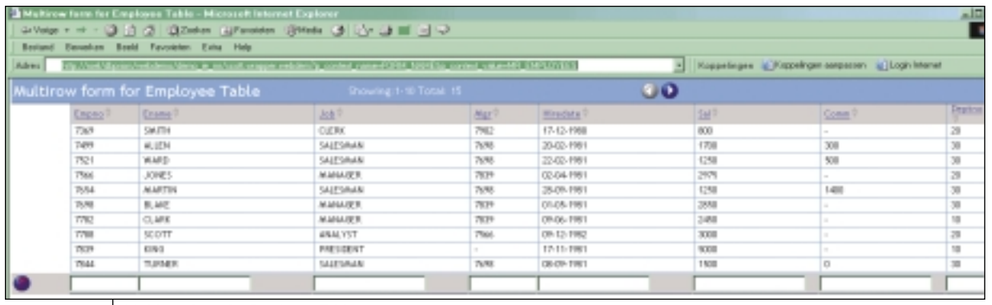


FIGURE 9 Multirow form for Employee Table

WRITTEN BY **SAWAT HANSIRISAWAT**

Building a Flexible Presentation Framework

An XSLT approach

Information interchange has become key to survival in an increasingly wired world. It is in this context that industries have adopted XML as the enabling mechanism to achieve application integration. XML is ideally suited for designing a Web application that has multiple pages, some of which display common business information.

If XML can be used to represent the application's business information, then XSL can be used to transform the XML for each of these pages. While Web designers can design the pages for the Web application, they can't be expected to design XSLT stylesheets. Stylesheet designers, on the other hand, are good at embedding transformation logic but are not conversant with writing HTML code or presentation tricks. This makes it hard to write XSLT stylesheets that transform the XML to HTML specific to each page. If Web designers can add special "tags" as placeholders for content, then those tags can be transformed at runtime using stylesheet rules.

This article will address performance, flexibility, and ease of maintenance in the context of a framework that combines the capabilities of Java servlets, XML, and XSLT to present business information as Web pages to an end user.

The Basics

XML is a subset of SGML. It was developed to provide features that are related to Web publishing environments.

XHTML is well-formed HTML. The loosely defined HTML specification can introduce interpretation problems to the browser and make it harder for HTML writers to validate the correctness of the HTML pages. Most browsers today attempt to interpret and render incorrect syntax in HTML pages, but sometimes the interpretation can be completely incorrect. This causes the rendered pages to be meaningless to the user. XHTML helps solve this problem since it requires HTML to be well formed. This means XHTML writers could validate their documents before publishing them. XHTML browsers can render XHTML documents correctly, eliminating the guesswork. Therefore, users can rely on the correctness of the document that is rendered by the browser.

XSLT is a high-level declarative language, intended for transforming XML documents into other XML documents. XSLT expressions are used to describe rules for transforming XML documents.

The Problem

Developing a Web application is a very challenging task that requires individuals of different skill levels. For example, a Web designer has organizational and artistic skills and uses design tools to design the layout for Web pages. But a Web designer doesn't necessarily have any technical skill, and may have little or no knowledge of HTML, which is the final product of the design. The Web designer may also lack the programming skills to develop an application that retrieves relevant data from data sources for use in Web pages. On the other hand, a Web application developer has programming skills. He or she knows how to write Web applications to retrieve relevant data from data sources.

Both the Web designer and the Web application developer have to coordinate through the entire development of a Web application, modifying the same set of Web pages, typically as HTML or JSP pages. Changes to the Web pages by one person usually affect the other. For example, if a Web page designer changes the layout of a Web page, he or she might accidentally delete the work done by a Web developer. Or, changes to a Web page by a Web developer might also accidentally cause changes to the layout. If these people don't work for the same company, sharing the same set of Web pages during development can pose a very difficult problem. Also, maintaining HTML Web pages is challenging, since HTML pages aren't easy to validate because well-formed HTML syntax is not guaranteed. Changes to HTML pages could potentially render them unusable or cause them to display differently on different browsers.

The Architecture

To solve these problems, we need to minimize the dependency between Web designers and Web application developers. We could allow the Web page designer to work on Web (XHTML) pages, and let the Web application developer work only on the application, with little or no dependency on the actual Web pages. This would tremendously increase the efficiency and productivity of both. Changes by either would have no impact on the other throughout the development of the Web application.

Requiring that the page be produced in XHTML instead of HTML can solve the maintenance problem of HTML pages. If Web pages are produced in XHTML, it can validate syntax correctness easily using various tools. This will guarantee that changes to Web pages do not render them unusable or cause them to be rendered differently by different browsers.

By introducing special tags to XHTML pages, the Web page designer can concentrate on design, without worrying about sharing Web pages with the Web application developer. These special tags are invented as a contract between Web page designer and Web application developer. At the beginning of and throughout the development of a Web application project, both the Web page designer and the Web application developer agree upon a set of special tags that the Web page designer needs for designing Web pages and the Web application developer needs to supply to the Web page designer. The Web page designer inserts these special tags into XHTML pages, where data supplied by the application program will be populated at runtime. The Web application developer references these special tags in the application code. A common XSLT stylesheet is developed to transform these special tags in the final process.

After Web page designs have been finalized, the Web designer and the Web application developer together define what data needs to be used and in which Web pages. Then, they define a set of data needed for each Web page. Along with the data for each Web page, names of all data are uniquely defined. The Web page designer uses special tag "fields" as placeholders for each type of data needed in a Web page. An attribute "name" of each "field" is used to uniquely define each data element in each Web page. (See bold lines in Listing 1, XHTML Web Page.)

The Web application developer develops a corresponding XML template for each XHTML Web page. (See bold lines in the example of a corresponding XML template, below.)

```
<template>
<field name="user.username" />
<field name="user.password" />
</template>
```

The Web application developer writes application code to retrieve data and references these field elements in the corresponding XML template. At runtime, requests from the browser to the Web application result in invocation of the corresponding piece of application code, which in turn retrieves the needed data and inserts it into the corresponding XML template. A simple XSLT rule is used to insert this XML template into the corresponding XHTML document. A final transformation using a common XSLT is used to transform this XHTML into an XHTML document that is understood by the browser. This final XHTML document is then returned to the browser.

Discussion

Maintenance and isolation of the development of Web applications can be done quite easily. But XSLT is an interpreted process – every request results in interpreting an XML document, which can be a slow process. Having to interpret and transform XML documents for every request can incur high overhead to the Web application. Optimization is needed to minimize these overhead costs. Caching the XML template, XHTML document, and common XSLT stylesheet will reduce the overhead by eliminating the need for reconstruction for every request.

For easy and rapid Web application development, a tool can also be developed for the XML template and can generate the application code using information from the generated XML template to insert the data element into the XML template. This will reduce the requirement that the application developer has to know XML and XSLT and reduce development time because the application developer writes less code and, of course, there are fewer bugs to find and fix.

Conclusion

Of the many approaches that exist to develop a Web application, an approach to using standard XHTML and XML templates with a set of common XLST can reduce Web application development time and provide ease of maintenance for the Web application.

Resources

- Ladd, Eric; O'Donnell, Jim; Morgan, Mike; and Watt, Andrew H. (2000). *Using XHTML, XML and Java 2 Platinum Edition*. Que.
- Goldfarb, Charles F., and Prescod, Paul. (2001) *Charles F. Goldfarb's XML Handbook, 4th Edition*. Prentice Hall.
- XSL Transformations (XSLT) Version 1.0. W3C Recommendation. November 16, 1999.
- Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation. November 6, 2000.
- XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), A Reformulation of HTML 4 in XML 1.0. W3C Recommendation. January 26, 2000, revised August 1, 2002.

SAWAT@CYSIVE.COM

AUTHOR BIO

Sawat Hansirisawat is a senior software architect at Cysive, Inc., based in Reston, VA. Sawat has worked in the software industry for over 20 years and currently works in the product development group on the Cysive Cymbio Interaction Server. Sawat earned an MS degree in computer science from American University and holds a BS degree from Chulalongkorn University in Thailand.

LISTING 1 Sample of XHTML Web page

```
</td></tr>
<tr valign="top"><td colspan="2">
    <img src="/images/topb.gif" width="771"
height="62" alt="topb" border="0"/></td>
</tr>
<tr valign="top"><td bgcolor="White"
width="750">
<table cellpadding="2" cellspacing="4" bor-
der="0" width="771">
<tr><td>
    <form action="DispatcherServlet"
method="post">
    <input type="hidden" name="page"
value="RegisterUser" />
    <table cellpadding="2" cellspacing="4"
border="0">
        <tr><td valign="top" align="right">
            <table>
                <tr><td>
                    <font face="arial,hel-
vetica" size="2">User Name</font>
                </td></tr>
            </table>
          </td>
          <td>
            <field name="user.username" />
        </td></tr>
        <tr>
            <td valign="top" align="right">
            <table>
                <tr><td bgcolor="White">
                    <img
src="/images/spacer.gif" width="5" height="1"
alt="spacer" border="0"/></td>
                <td>
                    <font face="arial,hel-
vetica" size="2">Password</font>
                </td></tr>
            </table>
            </td>
            <td>
              <field name="user.password" />
            </td></tr>
    </table>
    <br/>

    <table>
        <tr><td>
            <input type="image"
src="/images/RegisterUser.gif" name="add"
    border="0"/>
        </td></tr>
    </table>

    <br/>
    </form>
    </td></tr>
    </table>
    </td></tr>
</table>
</body>
</html>
```

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

SUBSCRIBE TODAY TO MULTIPLE MAGAZINES ONLINE

Go To www.sys-con.com

and receive your **FREE** CD Gift Package VIA Priority Mail

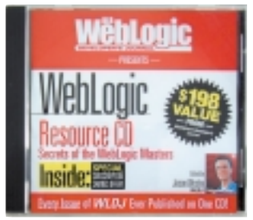
Each CD is an invaluable developer resource packed with important articles and useful source code!

More than 1,400 Web services and Java articles on one CD! Edited by well-known editors-in-chief Sean Rhody and Alan Williamson, these articles are organized into more than 50 chapters on UDDI, distributed computing, e-business, applets, SOAP, and many other topics. Plus, editorials, interviews, tips and techniques!

LIST PRICE $198

The most complete library of exclusive *JDJ* articles compiled on one CD! Assembled by *JDJ* Editor-in-Chief Alan Williamson, more than 1,400 exclusive articles are organized into over 50 chapters, including fundamentals, applets, advanced Java topics, Swing, security, wireless Java,... and much more!

LIST PRICE $198



The most complete library of exclusive *WLDJ* articles ever assembled! More than 200 articles provide invaluable information on "everything WebLogic", including WebLogic Server, WebLogic Portal, WebLogic Platform, WebLogic Workshop, Web services, security, migration, integration, performance, training...

LIST PRICE $198

The most complete library of exclusive *CFDJ* articles on one CD! This CD, edited by *CFDJ* editor-in-chief Robert Diamond, is organized into more than 30 chapters with more than 400 exclusive articles on CF applications, custom tags, database, e-commerce, Spectra, enterprise CF, error handling, WDDX... and more!

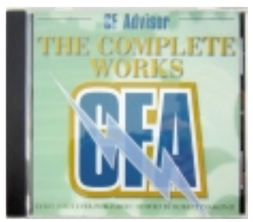
LIST PRICE $198

The largest and most complete library of exclusive *XML-Journal* articles compiled on one CD! Edited by well-known editors-in-chief Ajit Sagar and John Evdemon, these articles are organized into more than 30 chapters containing more than 1,150 articles on Java & XML, XML & XSLT, <e-BizML>, data transition... and more!

LIST PRICE $198

The most up-to-date collection of exclusive *WSDJ* articles! More than 200 articles offer insights into all areas of WebSphere, including Portal, components, integration, tools, hardware, management, sites, wireless programming, best practices, migration...

LIST PRICE $198



The complete works of *CF Advisor* are now on one CD! Check out over 200 exclusive articles on topics such as e-commerce, custom tags, Fusebox, databases, object-oriented CF, WDS, upgrading CF, wireless, Verity, and more. Plus, find interviews, editorials, and source code!

LIST PRICE $198

Pick the CDs to go with your Multi-Pack order

- Pick one CD with your 3-Pack order
- Pick two CDs with your 6-Pack order
- Pick three CDs with your 9-Pack order

- ☐ Web Services Resource CD
- ☐ Java Resource CD
- ☐ WebLogic Resource CD
- ☐ ColdFusion Resource CD
- ☐ XML Resource CD
- ☐ WebSphere Resource CD
- ☐ CF Advisor Complete Works CD

Your order will be processed the same day!

AND SAVE UP TO $400 AND RECEIVE UP TO 3 FREE CDs!

Pick a 3-Pack, a 6-Pack or a 9-Pack ☑

TO ORDER: Choose the Multi-Pack you want to order by checking next to it below. Check the number of years you want to order. Indicate your location by checking either U.S., Canada/Mexico or International. Then choose which magazines you want to include with your Multi-Pack order.

☐ 3-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.	
☐ 6-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.	
☐ 9-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.	

☐ **Java Developer's Journal**

U.S. - Two Years (24) Cover: $144	You Pay: $89 /	Save: $55 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $49.99 /	Save: $22
Can/Mex - Two Years (24) $168	You Pay: $119.99 /	Save: $48 + FREE $198 CD
Can/Mex - One Year (12) $84	You Pay: $79.99 /	Save: $4
Int'l - Two Years (24) $216	You Pay: $176 /	Save: $40 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **Web Services Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **.NET Developer's Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **XML-Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **WebLogic Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

☐ **ColdFusion Developer's Journal**

U.S. - Two Years (24) Cover: $216	You Pay: $129 /	Save: $87 + FREE $198 CD
U.S. - One Year (12) Cover: $108	You Pay: $89.99 /	Save: $18
Can/Mex - Two Years (24) $240	You Pay: $159.99 /	Save: $80 + FREE $198 CD
Can/Mex - One Year (12) $120	You Pay: $99.99 /	Save: $20
Int'l - Two Years (24) $264	You Pay: $189 /	Save: $75 + FREE $198 CD
Int'l - One Year (12) $132	You Pay: $129.99 /	Save: $2

☐ **Wireless Business & Technology**

U.S. - Two Years (24) Cover: $144	You Pay: $89 /	Save: $55 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $49.99 /	Save: $22
Can/Mex - Two Years (24) $192	You Pay: $139 /	Save: $53 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $79.99 /	Save: $16
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **WebSphere Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

☐ **PowerBuilder Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

RECEIVE YOUR DIGITAL EDITION ACCESS CODE INSTANTLY WITH YOUR PAID SUBSCRIPTIONS

3-Pack

Pick any 3 of our magazines and save up to $275.00

Pay only $175 for a 1 year subscription plus a **FREE CD**

- 2 Year – $299.00
- Can/Mex – $245.00
- International – $315.00

6-Pack

Pick any 6 of our magazines and save up to $350.00

Pay only $395 for a 1 year subscription plus **2 FREE CDs**

- 2 Year – $669.00
- Can/Mex – $555.00
- International – $710.00

9-Pack

Pick all 9 of our magazines and save up to $400.00

Pay only $495 for a 1 year subscription plus **3 FREE CDs**

- 2 Year – $839.00
- Can/Mex – $695.00
- International – $890.00

WRITTEN BY **ARI KERMAIER, JOE MORGAN**

Digital Signatures and Encryption

Ensuring the integrity and confidentiality of your data

While the flexibility, extensibility, and simplicity of XML have made life easier for a lot of business application developers, the core XML specification doesn't address security in any way. The openness of XML doesn't make security any easier, and developers who leverage the benefits of XML are still left with the daunting task of finding a way to secure their data. While data security can have various meanings in different situations, the fundamental ideas that concern us here are the integrity and confidentiality of the data.

Ensuring data integrity means making sure that the data cannot be tampered with without detection. Digital signature algorithms can provide this type of security. When a signature is combined with trusted authentication of the signer, it can also provide legal evidence that a particular person signed the data, making it difficult to repudiate such a signature later.

Data confidentiality is required when the data must be concealed from everyone except for its intended recipients. Strong symmetric key encryption algorithms are typically used to accomplish this task.

An E-Business Example

Imagine the following scenario: Alice's Retail Goods frequently orders products from Bob's Wholesale Goods. Every time Alice orders another shipment of products from Bob, Bob creates a purchase order. The purchase order is an XML document that contains information about the order, Alice's billing information, and Bob's bank account information (see Listing 1).

After Bob generates the purchase order, he sends it to his bank. The bank charges Alice's credit card for the total price of the order and transfers that amount to Bob's bank account. The system is very simple and straightforward.

All goes well for a time, but eventually a series of disputes over submitted purchase orders prompts the bank to institute new security measures. The bank now requires that each purchase order be digitally signed before it is submitted to the bank.

What Is a Digital Signature?

Digital signatures are an important element in electronic security because they can be used to ensure the integrity, authenticity, and nonrepudiability of data. Simply defined, a digital signature is an encrypted hash of an electronic document. Commonly used digital signature schemes such as RSA and DSA employ a pair of mathematically related keys. Each person wishing to create digital signatures must have a unique key pair – one key of the pair is kept private and used for signing and the other is made public and used for verifying its owner's signatures.

Signing a document with the RSA algorithm is accomplished by first computing a hash of the document data using a secure digest algorithm such as SHA-1 or MD5. Then the signer's private key is used to encrypt this hash. The result is the digital signature. Verifying a signature consists of using the signer's public key to decrypt the signature hash, and then comparing the result to a freshly generated hash of the signed document. If the two hashes are identical, then the recipient has successfully verified the signature.

Returning to our example, if a purchase order were to be modified after Alice signed it, then the hash that Bob computed over the modified document would not match the hash that Alice computed over the original document. Thus, if Bob can verify the signature, he is assured of the integrity of the document. Alternatively, if a private key other than Alice's had been used to encrypt the hash, then Bob would not be able to correctly decrypt this signature using Alice's public key and verification would fail. If Bob can verify the signature, he is assured that Alice's private key, which only Alice should have, was used to generate the signature and that the document is authentic. In addition, if a trusted certificate authority has asserted that the key Bob believes to be Alice's key does in fact belong to Alice, this gives Bob the ability to demonstrate to others (in court, for example) that Alice signed the document. This allows the signature to offer an assurance of nonrepudiability of the signed document.

Digital Signatures and XML

The World Wide Web Consortium and the Internet Engineering Task Force have jointly created a standard that describes processing rules and syntax for the use of digital signatures in XML documents. The specification defines a schema and corresponding DTD that specify how XML signatures can be represented.

An XML signature contains the basic encrypted hash of the signed document, along with information that tells the recipient of the document what data was signed and which algorithms were used. An XML signature can be included inside of the document to which the signature applies, or it can exist in a separate document. An XML signature can be applied to multiple documents or document subsets, or even non-XML data.

The Phaos XML Toolkit fully implements the XML signature standard, using intuitive APIs that will be familiar to Java developers accustomed to Sun's JAXP interface. Next we will demonstrate how the Phaos toolkit can be used to write code to generate and verify XML signatures.

Signing the Purchase Order

Suppose that the bank's policy requires both the vendor (Bob) and the buyer (Alice) to sign the purchase order. Bob generates the document, attaches his signature, and sends it to Alice. Alice verifies Bob's signature, attaches her own signature, and returns the purchase order to Bob. Bob verifies Alice's signature and submits the order to the bank. Finally, the bank verifies both signatures and performs the funds transfer.

An XML signature can be computed over an entire document or over one or more subsections of a document. In our scenario, Bob signs the item list to attest to the price quotes and Alice signs the same data to acknowledge her agreement to pay those prices.

Listing 2 illustrates Java code that generates a signature computed over the <ItemList> element of the purchase order from Listing 1. After reading in and parsing the purchase order document, and obtaining the signer's private key, we are ready to construct the XML signature:

1. First we create an instance of the com.phaos.xml.dsig.XSSignature class and append it to the purchase order document. The result is a <Signature> element in the XML Signature namespace, "http://www.w3.org/2000/09/xmldsig#", containing an ID attribute with the value "VendorSignature".
2. The <Signature> element needs a <SignedInfo> child element that will contain the XML data to be signed. This is done by creating a com.phaos.xml.dsig.XSSignedInfo object and appending it to the XSSignature object. The <SignedInfo> element is also given child elements specifying the signature algorithm, RSA with SHA-1, and the XML canonicalization algorithm.
3. The <SignedInfo> element must have a <Reference> child element for each document or section to be signed. In this case we create a single com.phaos.xml.dsig.XSReference object, and use the URI fragment "#ItemList" to identify the element to sign. The algorithm to use for hashing the data, SHA-1, is also specified here.
4. Finally, the vendor's private key is used to sign the data. The XSSignature.sign() method hashes the <ItemList> element's XML and appends the resulting digest value to the <Reference> element. Then it computes an RSA signature value over the <SignedInfo> element and appends the resulting value to the <Signature> element.

The resulting signed purchase order document is shown in Listing 3. A second <Signature> element will be appended when the buyer subsequently signs the purchase order as well.

After Bob and Alice have each signed the purchase order and verified each other's signatures, Bob sends the purchase order to the bank. The bank can then verify both Alice's and Bob's signatures.

Listing 4 contains Java code to verify Bob's signature using his public key certificate. After reading in and parsing the purchase order document, and obtaining the signer's public key, we can verify the signature:

1. The first step is to locate the <Signature> element in the document. This can be done in any manner you wish. In our example we've chosen to use the convenience method getElementById() provided by the Phaos XML toolkit's XMLUtils class, which uses the value of the <Signature> element's ID attribute. Alternatively, we could have used the JAXP 1.1 APIs to search for any <Signature> elements in the XML Signature namespace.
2. Once the <Signature> element is found, we instantiate an XSSignature object to process it.
3. Now we can use the vendor's public key to verify the signature by invoking the XSSignature.verify() method. XML signature verification proceeds in two phases. First, the integrity of the signed data is validated by hashing the <ItemList> and comparing the result to the digest value contained in the <Reference> element. Then the signature over the <SignedInfo> element is validated using the signer's public key.

What About Securing Confidential Data?

Vendors like Bob have an objection to the bank's proposed system. When a purchase order is sent to the buyer to be signed, the buyer can see the vendor's bank account information. Bob prefers to keep this information confidential.

The W3C's XML Encryption specification addresses this requirement. The standard allows selected parts of a document to be encrypted, while the document as a whole remains valid XML. The data is encrypted using a strong symmetric encryption algorithm such as AES.

Listing 5 illustrates how the Phaos XML toolkit can be used to encrypt the vendor's <AccountInfo> element in the purchase order document:

1. Locate the vendor's <AccountInfo> element. In this example we use the getElementsByTagName() method of the org.w3c.dom.Document interface provided by the JAXP 1.1 API.
2. Instantiate a com.phaos.xml.enc.XEEncryptedData object. This results in the creation of an <EncryptedData> element in the XML-Encryption namespace, "http://www.w3.org/2001/04/xmlenc#", containing an ID attribute with the value "EncryptedVendorInfo". Additionally, the <EncryptedData> element is given a Type attribute that indicates that only the content of the <AccountInfo> element will be encrypted, excluding the <AccountInfo> tag.
3. Next, the instantiation of a com.phaos.xml.enc.XEEncryptionMethod results in creation of the <EncryptionMethod> element that will identify the algorithm to be used for encrypting the account data – in this case AES-128 (the U.S. government's Advanced Encryption Standard).
4. Finally, the <AccountInfo> element is encrypted using a secret AES key that the vendor shares with the bank. The resulting cipher text is enclosed in the <EncryptedData> element, which then replaces the <AccountInfo> element in the purchase order document.

Listing 6 shows what the purchase order looks like after Bob has signed it and encrypted his account information. When the bank receives the purchase order, it must use the secret AES key to decrypt the vendor's account information before the funds transfer can be performed.

Conclusion

Our purchase order example provides a basic introduction to the capa-

AUTHOR BIOS

Ari Kermaier is a senior software engineer with Phaos Technology Corp., specializing in Java development toolkits for cryptography, PKI, and XML security. Ari holds an MS degree in computer science from New York University and an MS in operations research from Columbia University.

Joe Morgan is a senior software engineer with Phaos Technology Corp., specializing in Java development toolkits for cryptography, PKI, and XML security. Joe has extensive experience working with Java and XML technologies.

bilities of XML signatures and encryption to secure XML communication. The specifications' richness and flexibility allow them to be adapted for virtually any XML security need, from database encryption to secure workflow applications to the exchange of authorization credentials.

The XML signature and encryption specifications have become the foundation of a number of other valuable XML security protocols, including:

- ***XKMS:*** The W3C's XML Key Management Specification provides an XML Web services framework for relieving client software of the complexities of traditional PKI.
- ***SAML:*** The OASIS consortium's Security Assertion Markup Language uses XML signatures and encryption to protect the exchange of security credentials for single sign-on and other applications.
- ***SOAP-SEC:*** The SOAP Security Extensions specification details how XML signatures can be used to secure SOAP messages.

Securing your data can be a complex undertaking. The example we have provided here is intentionally simplified, but it demonstrates the potential of the XML signature and encryption standards, and the ease with which these standards can be utilized with the Phaos XML Security Suite. The toolkit offers much more than is presented here, and gives powerful, easy-to-use APIs to developers who need to integrate data security into their XML solutions.

References

- *W3C XML Signature specification:* www.w3.org/TR/xmldsig-core
- *W3C XML Encryption specification:* www.w3.org/TR/xmlenc-core
- *W3C XML Key Management specification:* www.w3.org/TR/xkms2
- *OASIS Security Assertion Markup Language:* www.oasis-open.org/committees/security
- *Phaos Technology Corp.:* www.phaos.com

ARIK@PHAOS.COM

JMORGAN@PHAOS.COM

LISTING 1

```
<PurchaseOrder OrderNumber="0123456789">
  <ItemList Id="ItemList">
    <Item>
      <Name>Pogo Sticks</Name>
      <UnitPrice>$10.99</UnitPrice>
      <Quantity>50</Quantity>
      <Total>$549.50</Total>
    </Item>
    <Item>
      <Name>Frisbees</Name>
      <UnitPrice>$2.99</UnitPrice>
      <Quantity>100</Quantity>
      <SubTotal>$299.00</SubTotal>
    </Item>
    <SubTotal>848.50</SubTotal>
    <SalesTax>59.40</SalesTax>
    <Total>907.90</Total>
  </ItemList>
  <BillingInfo Id="BuyerInfo">
    <Name>Alice's Retail Goods</Name>
    <CreditCard Type="AMEX">
      <Number>1234 567890 12345</Number>
      <Expires Month="1" Year="2005" />
    </CreditCard>
    <PostalAddress>
      <Street>123 East 45th</Street>
      <City>New York</City>
      <State>NY</State>
      <Zip>67890</Zip>
    </PostalAddress>
    <Telephone>(555)555-5555</Telephone>
  </BillingInfo>
  <AccountInfo Id="VendorInfo">
    <Name>Bob's Wholesale Goods</Name>
    <AccountNumber>123-1234567-12</AccountNumber>
    <PostalAddress>
      <Street>321 West 54th</Street>
      <City>New York</City>
      <State>NY</State>
      <Zip>67980</Zip>
    </PostalAddress>
    <Telephone>(555)444-4444</Telephone>
  </AccountInfo>
</PurchaseOrder>
```

LISTING 2

```
// Signs the Items element of the purchase order
// using the vendor's private key.

import com.phaos.crypto.PrivateKey;
import com.phaos.xml.dsig.*;
import com.phaos.xml.utils.XMLURI;
import org.w3c.dom.Document;

// Input and parse the purchase order document.
Document doc = ...;

// Get the vendor's private signing key.
PrivateKey signingKey = ...;

// Create the Signature element and append it
// to the root element of the document.
XSSignature sig = XSSignature.newInstance(doc,
"VendorSignature");
sig.appendTo(doc.getDocumentElement());

// Create the SignedInfo element.
XSSignedInfo sigInfo = sig.createSignedInfo();
sigInfo.setSignatureMethod(XMLURI.alg_rsaWithSHA1);  //
Signature algorithm
sigInfo.setC14NMethod(XMLURI.alg_c14n);  // XML Canonicalization
algorithm
sig.setSignedInfo(sigInfo);

// Add a Reference element that targets
// the the ItemList element.
XSReference ref = sig.createReference();
ref.setURI("#ItemList");
ref.setDigestMethod(XMLURI.alg_sha1);  // Hash algorithm
sigInfo.addReference(ref);

// Compute the signature value.
sig.sign(signingKey, null);
```

LISTING 3

```
<PurchaseOrder OrderNumber="0123456789">
  <ItemList Id="ItemList">
    <Item>
      <Name>Pogo Sticks</Name>
      <UnitPrice>$10.99</UnitPrice>
      <Quantity>50</Quantity>
      <Total>$549.50</Total>
    </Item>
    <Item>
      <Name>Frisbees</Name>
      <UnitPrice>$2.99</UnitPrice>
      <Quantity>100</Quantity>
      <SubTotal>$299.00</SubTotal>
    </Item>
    <SubTotal>848.50</SubTotal>
    <SalesTax>59.40</SalesTax>
    <Total>907.90</Total>
  </ItemList>
  <BillingInfo Id="BuyerInfo">
    <Name>Alice's Retail Goods</Name>
    <CreditCard Type="AMEX">
      <Number>1234 567890 12345</Number>
      <Expires Month="1" Year="2005" />
    </CreditCard>
    <PostalAddress>
      <Street>123 East 45th</Street>
      <City>New York</City>
      <State>NY</State>
      <Zip>67890</Zip>
    </PostalAddress>
    <Telephone>(555)555-5555</Telephone>
  </BillingInfo>
  <AccountInfo Id="VendorInfo">
    <Name>Bob's Wholesale Goods</Name>
    <AccountNumber>123-1234567-12</AccountNumber>
    <PostalAddress>
      <Street>321 West 54th</Street>
      <City>New York</City>
      <State>NY</State>
      <Zip>67980</Zip>
    </PostalAddress>
    <Telephone>(555)444-4444</Telephone>
  </AccountInfo>
  <Signature Id="VendorSignature"
      xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
      <SignatureMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
      <Reference URI="#ItemList">
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmld-
sig#sha1"/>
        <DigestValue>5+IG6+QCz7ZN/f3na3c/WE/xTK8=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>

GA4yjyM6Qn6CEQGnqjnXffUc0BBHsSkmbzj2CaAUetPh4HmartvkMmlkPxtQ

tPvaUKBGbD40ko77bqvWm2bKIN/jBBy5xQFxfMerqLQcbrHLicGfLrNk9PbL
      3tvLori3SwdAi3WGq9bLvo7ubdknindu8NC0ysrhbBXv3AgVrnM=
    </SignatureValue>
  </Signature>
</PurchaseOrder>
```

LISTING 4

```
// Verifies the vendor's signature in
// the XML purchase order document.

import com.phaos.cert.X509;
import com.phaos.crypto.PublicKey;
import com.phaos.xml.utils.XMLUtils;
import com.phaos.xml.dsig.XSSignature;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

// Input and parse the purchase order document.
Document doc = ...;

// Get the vendor's public key.
X509 cert = ...;  // Vendor's certificate.
PublicKey pubKey = cert.getPublicKey();

// Find the vendor's Signature element.
Element e = XMLUtils.getElementById(doc, "VendorSignature");
XSSignature sig = new XSSignature(e);

// Verify the signature using the vendor's public key.
if (sig.verify(pubKey))
    System.out.println("Vendor's signature is valid.");
else
    System.out.println("Vendor's signature is invalid.");
}
```

LISTING 5

```
// Encrypts the AccountInfo element using
// the AES-128 algorithm.

import com.phaos.crypto.SymmetricKey;
import com.phaos.xml.utils.XMLURI;
import com.phaos.xml.enc.XEEncryptedData;
import com.phaos.xml.enc.XEEncryptionMethod;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

// Input and parse the purchase order document.
Document doc = ...;

// Get the secret AES-128 key shared with the bank.
SymmetricKey aesKey = ...;

// Find the vendor's AccountInfo element.
Element acctInfo =
doc.getElementsByTagName("AccountInfo").item(0);

// Encrypt the AccountInfo element's content using
// AES-128, and replace it with an EncryptedData
// element with Id="EncryptedVendorInfo".
XEEncryptedData encData = XEEncryptedData.newInstance(
    doc, "EncryptedVendorInfo", XMLURI.obj_Content);
XEEncryptionMethod encMethod =
    encData.createEncryptionMethod(XMLURI.alg_aes128_CBC);
encData.setEncryptionMethod(encMethod);
XEEncryptedData.encryptAndReplace(acctInfo, aesKey, encData);
```

LISTING 6

```
<PurchaseOrder OrderNumber="0123456789">
  <ItemList Id="ItemList">
    <Item>
      <Name>Pogo Sticks</Name>
      <UnitPrice>$10.99</UnitPrice>
      <Quantity>50</Quantity>
      <Total>$549.50</Total>
    </Item>
    <Item>
      <Name>Frisbees</Name>
      <UnitPrice>$2.99</UnitPrice>
      <Quantity>100</Quantity>
      <SubTotal>$299.00</SubTotal>
    </Item>
    <SubTotal>848.50</SubTotal>
    <SalesTax>59.40</SalesTax>
    <Total>907.90</Total>
  </ItemList>
  <BillingInfo Id="BuyerInfo">
    <Name>Alice's Retail Goods</Name>
    <CreditCard Type="AMEX">
      <Number>1234 567890 12345</Number>
      <Expires Month="1" Year="2005" />
    </CreditCard>
    <PostalAddress>
      <Street>123 East 45th</Street>
      <City>New York</City>
      <State>NY</State>
      <Zip>67890</Zip>
    </PostalAddress>
    <Telephone>(555)555-5555</Telephone>
  </BillingInfo>
  <AccountInfo Id="VendorInfo">
    <EncryptedData Id="EncryptedVendorInfo"
        Type="http://www.w3.org/2001/04/xmlenc#Content"
        xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
cbc"/>
      <CipherData>
        <CipherValue>

66dzvkvWxFcO4Vn0Q+8RzTZ08365qmumKdO+MHtcm4REerVH5ONEqqCIt5Tu
OUaaIok51vjjZoVdoihm8NFrVfTrEwG7Hu2L3emO4DFR42qqcRPBc/vtrF+m
9FdVLT3J3ZAWxXMxsP+xPLavu3Jm+l3QWW5r+cknGIiimkE8QifP6gLWQZWz
9lu6nhIjCB2dt8MwFU1PDlMPII3nW4Lqt7uLCq3ZRhxj9qVyPYDyhAPmzXBR
WkPCnL8ZKqluuW7ykTO29ksTzH+JLl0Ho4Mox5FS1/n1UtvaGNaVKGb9bs1B
eT7TDJp24V+ZAaBDPzpCS3T5w/aKV7VO9fOtWBoaGTdWw0mZmMPSE3dHrUHG
WPi+n/xXTzH4gWS/pmpbS2DSTNZ1NqGLxAAu1/wHIdNoT3sDY+aeGgAISZCd
MbjOxppdylfPO5A/kvts+LflY3kLOX7JBLdGuR0o/SeP5ctwFRtz8qnMF1Cj
          ZrMoGknpO6Q=
        </CipherValue>
      </CipherData>
    </EncryptedData>
  </AccountInfo>
  <Signature Id="VendorSignature"
      xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
      <SignatureMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
      <Reference URI="#ItemList">
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmld-
sig#sha1"/>
        <DigestValue>5+IG6+QCz7ZN/f3na3c/WE/xTK8=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>

GA4yjyM6Qn6CEQGnqjnXffUc0BBHsSkmbzj2CaAUetPh4HmartvkMmlkPxtQ

tPvaUKBGbD40ko77bqvWm2bKIN/jBBy5xQFxfMerqLQcbrHLicGfLrNk9PbL
      3tvLori3SwdAi3WGq9bLvo7ubdknindu8NC0ysrhbBXv3AgVrnM=
    </SignatureValue>
  </Signature>
</PurchaseOrder>
```

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

XML Authoring and Editing Tools

Characteristics and limitations

WRITTEN BY
NEIL BRADLEY

In this article, you'll find an in-depth exploration of six distinct categories of XML document editors. With an appreciation of the characteristics and limitations of the tools available, you'll be able to choose those that best fit your needs.

Categories

I'll explore the following categories of XML editors:

- Form-based editors
- Word processors
- Text editors
- Word processors with style mappings
- Word processors with XML enhancements
- XML word processors

In assessing the relative merits of each category I consider efficiency, ease of use, cost of ownership, and ability to access document model rules (defined in a DTD or schema definition) in order to control or guide the authoring process. I also cover an issue that I call "the hidden-tags problem," which is explained fully at the end of this article. For now, it's sufficient to know that it involves the confusion that can arise when the context of the current cursor position is ambiguous because element tag delimiters are not visible. A closely related question concerning the merits of hiding XML concepts from authors is also discussed.

I'm aware that there are products on the market that defy categorization, and acknowledge that a product that appears on the surface to fit into one category might nevertheless include features that overcome the limitations that I ascribe to that category. I offer my apologies in advance to the vendors of such products, and encourage them to emphasize these features in their promotional material.

Form-Based Approach

The first category is radically different from the others, and the reason why goes to the heart of the distinction between data-centric documents (such as SOAP messages) and narrative documents (such as this article). This distinction emerged when data-centric applications were established as the primary new use of markup technologies shortly after the release of the XML standard. An avalanche of free and low-cost products quickly arrived that dispensed with the free-text editing environment in favor of a form-based approach.

Often Java-based, these products typically show a structured view of the entire document structure in one panel, from which an element can be selected, and a small set of text field boxes in another panel for editing the content and attributes of the currently selected element (see Figure 1).

However, some products manage to combine the navigation and editing aspects to make editing a little more intuitive and efficient (see Figure 2). These tools are ideal for creating small, data-centric documents, and for making minor edits to existing documents of any kind. Tight control over the authoring or editing process is usually maintained, using rules extracted from a document model, so the author is prevented from creating an element in the wrong location.

Free-Text Approach

Form-based editors tend to handle mixed content (text mixed with elements) inelegantly, because each pseudo-element (chunk of text between elements) has to be created and edited separately. Authors of narrative documents rarely use this type of editor because they're accustomed to using sophisticated free-text editing products that include WYSIWYG interfaces that visually distinguish between block-level elements (such as headings and paragraphs) and inline elements (such as emphasized or superscript phrases, words, and individual characters).

The remaining categories are all based on the free-text editing approach, and are introduced in an order that suggests increased specialization, sophistication, efficiency, and cost.

Standard word processors

Standard word processors can be used to create XML documents (we'll explore word processors that are enhanced to handle XML later). The tags are visible, and the author may manually style the content of specific elements (e.g., the header is in a larger font in Figure 3).

Clearly, this approach is not ideal when authors have to key the XML tags, which is a slow and unreliable process. However, most word processors include a macro language that could be used to store the keystroke sequences needed to create specific element tags. Perhaps more seriously, there's no possibility of controlling or validating the document being created. Not only is the document model ignored, but even critical, well-formed markup constraints aren't checked, so a document could be saved with errors so serious that XML applications wouldn't even be able to read it.

This category isn't a serious proposition, except for use in an emergency. I included it mainly as a reminder that one of XML's great strengths is that XML documents can be created using any tool that can produce a text file. All word processors have a "save as text" or "export ASCII" option, and these applications are so ubiquitous that almost anybody with a computer can create an XML document.

Even the need to import XML into the native word processor format and export back to XML format after editing (a "round-tripping" process) is itself a cause of concern. Each time the file is exported, it has to be named again. The author has to enter the original file name, make no mistakes in doing so, and respond to a "this file already exists" warning. This can be tedious and error prone, though an application may have a built-in scripting language that could be used to automate this activity.

Text editors

The first practical approach for the authoring and editing of narrative documents replaces the word processor with a text-editing application.

One immediate advantage of text-editing applications over word processors is that their native storage format is simple text, so there's no need to import and export XML documents; they can be opened, saved, and closed like any other text document.

Although these products don't have the ability to format and style segments of the document in different ways (as word processors can), we all know that we should be focusing on content rather than appearance, so this isn't necessarily a bad thing. Yet most authors would appreciate the ability to quickly identify elements visually, by the style of their content.

In their most basic form, these products have the same weaknesses as word processors (with the exception of avoiding the need for importing and exporting). But there are two levels of XML sensitivity that may be supported by text editing tools.

First, the presence of XML tags among the text is distracting to authors. Sophisticated text editors can be taught to recognize language-specific keywords and tag delimiters. In this case, XML tags can be recognized and color-coded, and elements can sometimes be indented to show hierarchical context, thus helping the author to avoid making mistakes and marginally improving the legibility of the document. Also, end tags may be generated automatically as soon as the start tag is completed.

Second, many text editors have features that support the development of software source code, and include the ability to save the file, activate a compiler, catch any errors or warning messages, identify a line number in such a message, and then allow the user to scroll to a line containing an error simply by selecting the relevant error message. This feature can be exploited to run command-line–based XML parsers in the same way.

Specialized Tools

Form-based editors, word processors, and text editors rely upon products that the author might already have, or might be able to acquire for little or no money. The next three categories require a more serious level of expenditure.

The following categories remove XML markup from the editable text in order to improve the user interface and to remove the possibility of tags being incorrectly keyed or accidentally corrupted. Start tags and end tags are either removed entirely or replaced by icons.

Style mapping

The fourth category relies upon the mapping of XML elements to paragraph and character-level styles, usually defined in a stylesheet within a word processor or DTP (desktop publishing) application (see Figure 4).

The mappings are performed either by a separate software application or by a plug-in conversion filter for importing from XML or exporting to XML. When both kinds of filter exist, round-tripping becomes possible, meaning that XML data can be flowed-in to the package for publishing purposes, but edited before publishing (possibly to fit the text to a limited space), and then flowed-out to XML in order to keep the XML version synchronized with the published document. This procedure is the real target of these products, rather than supporting an editing solution. Although it's possible to use them to enable a standard word processor or DTD package to be used as an XML editor, this approach is more expensive than the previous two categories, yet still offers no validation or control over editing, and fails to overcome the import/export difficulties discussed earlier. In addition, it raises the hidden-tags problem. To be fair, the vendors of these products rarely if ever make the claim to be creating an editing tool.

Word-processor extensions

The fifth category marries a popular word processor to the specific needs of XML editing by tightly integrating an XML-sensitive software layer that effectively sits between the word processor and the user. This is one step up from the previous category because it adds the editing control that's also found in the first category. In fact, these products are usually quite good for authoring, but sometimes struggle, due to the hidden-tags problem, when employed to edit an existing document (though it isn't unknown for such products to generate some kind of tag location marker in order to avoid this problem).

XML word processors

The final category is the most sophisticated and most expensive option. XML-based word processors are built from the ground up to specialize in the editing of XML documents. They generally have an interface similar to that of a word processor, but in this respect they divide into two sub-categories.

Some XML word processors attempt to hide the complexity of XML from the author by hiding all XML markup, and presenting a formatted rendition of the document. The stated goal of these products is to simplify the authoring process to the point that the author is either unaware of the fact that an XML document is

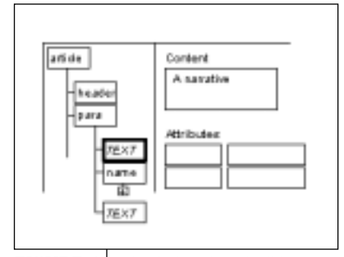


FIGURE 1 Simple form-based editing

FIGURE 2 Integrated form-based editing

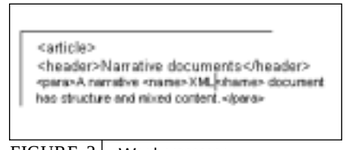


FIGURE 3 Word processor

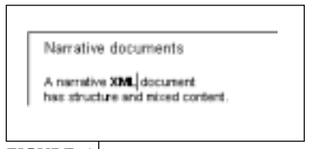


FIGURE 4 Style mapping

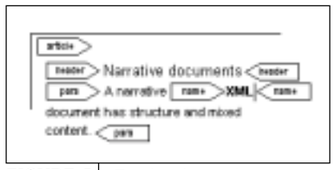


FIGURE 5 Tags location markers

being created, or is at least not distracted by the fact. But the similarity of this approach to the word-processor extension option should be noted because it shares the same potential problem of hidden tags.

Other XML word processors have a number of display modes, including the full formatted, or "tags-off," approach just discussed, but also including at least one kind of "tags-on" mode. Typically, icons that include the element names replace the start tags and end tags (see Figure 5).

However, my own favorite mode (supported directly by a few products, and indirectly using stylesheets in others) reduces element tags to much smaller icons that look like a single character, such as "[" for the start tag and "]" for the end tag. For example, "this is a [bold] word" indicates that the word "bold" is within an element. The identity of the element is established by styling the content, or by showing the name elsewhere (perhaps both).

> **"XML isn't really about markup, but about the concept of unambiguous identification of units of information that form meaningful sequential and hierarchical relationships"**

Hidden-Tags Problem

When tags are shown, either in full text form or as icons of some kind, the cursor (shown as "|" in the examples below) can be placed in an explicit contextual location. For example, the cursor is inside the embedded element in "a [bold|] word," but outside of it in "a [bold]| word." But when tag locations are not shown, the location of the cursor can become ambiguous, as in "a bold| word." This is the hidden-tags problem.

Note that this problem arises when the hiding of tags is adopted out of necessity (some of the earlier categories of editor), and also when it's done out of a genuine attempt to hide the complexity of XML from authors (some XML word processors).

Context issues can arise even when authoring new text. Consider a document model for an article that includes lists that consist of a number of list items, and allows for the possibility of more than one paragraph within a list item. The current cursor position appears to be at the end of the document, though in fact it actually precedes a series of end tags ("...|</para></item></list></article>"). Now, what should happen when the author presses the "return" key? Certainly, the current paragraph should be ended, but maybe a new paragraph within the same item should be started, or maybe the current item should be ended and a new one started, or maybe the entire list should be ended and a new paragraph started after the list. Whichever action is performed, it may not be obvious to the document author what has happened, and it may be even less obvious to the author how to override the default action.

Hiding the tags may well simplify the main text display, but it inevitably adds complexity elsewhere. Typically, in an attempt to work around the hidden-tags problem, an editor will include numerous buttons to perform such actions as "insert element below," "insert element within," and "select parent." It's also not easy to present the author with a reliable list of allowed actions when the context is unclear. For example, if the author can either end a list and start a new one, or create an embedded list, then the presence of the element name "list" in a menu of options is hardly helpful.

Is tag hiding desirable?

XML isn't really about markup, but about the concept of unambiguous identification of units of information that form meaningful sequential and hierarchical relationships. With this in mind, we should ask if it really is a good idea, in principle, to try to hide XML from authors.

My own view is that authors need to be aware that they're creating an XML document and understand both the benefits and responsibilities that this approach implies. They need to think about structure in order to produce accurate, useful documents.

A hidden-tags solution can be considered when large numbers of authors are involved and the document model is very simple (keeping costs low and training simple), but a professional XML word processor with at least one "tags-on" view should be used in other cases. We should be promoting XML, not trying to hide it from authors.

Resources

There are many XML editors, far too many to mention here. For some lengthy lists of XML editors see www.xmlsoftware.com/editors.html and www.xml.com/pub/pt/3.

AUTHOR BIO

Neil Bradley is an XML consultant working for Rubus, a UK-based interactive systems integrator. Neil has specialized in publishing applications of SGML since 1986 and XML since its release, as a developer, analyst, and consultant. He is a regular speaker at related conferences and has written a number of related books: The Concise SGML Companion, The XML Companion, *and* The XSL Companion.

NEIL.BRADLEY@RUBUS.COM

WRITTEN BY **CASEY HIBBARD**

EPA Simplifies Multistate Data Exchange with XML

A pilot project

A massive government agency isn't the first place you'd think of to find leading-edge use of XML. But then that's just the type of setting where the emerging standard for cross-platform data exchange is needed most.

Picture this: in each of the 50 states, there is at least one environmental agency responsible for enforcing state and U.S. Environmental Protection Agency regulations. From the corner gas station to the local manufacturing plant, thousands of businesses and organizations nationwide are responsible for meeting strict environmental guidelines and reporting their compliance regularly to these state agencies. On a monthly, bimonthly, or quarterly basis, they send data in hard-copy reports with details such as their air emissions, pollutants discharged into bodies of water, hazardous waste disposal – any activities that could affect environmental integrity.

In turn, states must pass that data on to the EPA in Washington, DC. Traditionally, data exchange between states and the EPA, and vice versa, has been a painstaking, resource-intensive manual process. That's because the EPA and state environmental agencies all have different legacy database systems, from Oracle to SQL to DB2.

Because the systems can't easily communicate with each other, environmental organizations nationwide have resorted to rekeying information or translating data into another format to send to the national office. Forget exchanging data among states. That too demands manually translating information into another format that the requesting state can understand.

But an ambitious movement within the EPA and state environmental agencies has led to a surprisingly simple, yet innovative, solution to this complex data problem.

States and the EPA Join Forces for Pilot Project

In 2000, a group of information technology managers from a handful of states put their heads together. At the time, each state was translating environmental data on thousands of facilities into batch cards to send to the EPA. A considerable amount of staff time in each of the 50 states and at the EPA was devoted to the exchange of data.

"We'd put time and resources into getting the data into our system, then have to spend time managing a translator to format data to send to the EPA," explained Dennis Burling, information technology manager at the Nebraska Department of Environment Quality.

A number of states and the EPA began envisioning a process whereby information from diverse platforms could be easily transferred through a network in a common XML format. States or the EPA could simply enter a query or make a request over the network and immediately retrieve their desired information, a process that would eliminate the need for staff to manually rekey or translate data into other formats.

The primary environmental agencies for Nebraska, Utah, Delaware, and New Hampshire; the EPA; and an EPA contractor began exploring ways to simplify their data exchange. The group knew from studying industry trends that other organizations and the private sector were employing XML to accomplish cross-system data exchange.

"XML kept floating to the top as the best way to exchange information. Others outside the environmental arena, in the private sector, had already demonstrated that you could exchange information with it," Burling said.

The four states and the EPA decided to launch a pilot project to prove the feasibility of using XML to accomplish their data exchange. Among the test states, Delaware was using a SQL server on Windows 2000, Nebraska was using DB2 on IBM/AS400, New Hampshire was using two Oracle databases on Windows NT, and Utah was using an Oracle database running on Novell. The diverse mix of systems made this a perfect group for the pilot.

The goal would be to exchange basic facility information, such as names, addresses, contact people, and other general information. A server at each state would be mapped to this back-end data. The next step was finding a developer experienced in writing these types of applications.

EPA Finds an Out-of-the-Box Solution

Colorado Springs-based XAware answered the call. Instead of a developer, XAware offered ready-made, out-of-the-box data integration middleware. The XAware XA-Suite, a development environment and virtual XML server, provided an easy way to create bidirectional data exchange among diverse systems. Essentially, the technology aggregates data from multiple back-end systems into a single XML view that all systems can read.

"With a virtual XML database, they could retain the data in its existing format with no changes to their existing infrastructure or additions to staff," said Rohit Mital, vice president of engineering and cofounder of XAware.

The EPA Network Node Project, as it was named, enlisted XAware during the first phase. Using the virtual XML database, the pilot project required absolutely no software development at the state agencies. The solution was essentially deployed at the click of a button. A single XAware developer simply created a business process to communicate with each legacy database. This configuration, the only real customization needed, took less than one day for each database. In only three days, the XML database was ready to go to each state. As an additional service, XAware created the Web services–based user interface that would allow the states to search each other's databases.

Additionally, because the data remained in its existing format, environmental agencies didn't have to worry about errors creeping into data when it was translated into another format, as was the case previously.

Since this was a pilot project, state IT departments had few internal resources to devote to the initiative, and limited funds to spend on consultants. For Nebraska, the project involved just 40–60 hours of staff time.

The virtual XML database performed just as expected. When it was queried to search the three databases, the search results came back in a single XML file. It was one solution for all three states – at a fraction of the cost and time expected.

"We were able to successfully show that we could generate an XML file and push it over the Internet to another location and display it through a Java test application," Burling said. "Once in place, this will save us a lot of time because we will enter information into our system only once and have a fully automated exchange of information to the EPA."

On the heels of that success, representatives from the four initial states got together with other states at a meeting in New Mexico. From there, in August of 2001, they began a second phase that included the EPA and six states: Nebraska, Delaware, Florida, New Hampshire, New Mexico, and Utah. Once again, Nebraska, along with New Mexico, counted on the XAware XA-Suite. And again, the six states proved their ability to query each other's systems and return information.

A Cookbook for Other States

Currently, Mississippi, Maine, and Virginia have joined in, forming an eight-state project, called Node 1.0, in which the states hope to actively exchange facility information and air emission data by early 2003. At that time, these trailblazing states and the EPA expect to have a completed template that the rest of the environmental agencies nationwide can adopt. Burling calls it "a cookbook for other states to go by to build their nodes." Each state will have a node, or location, on the network.

"Our goal is to create a model that other states' environmental organizations can easily adopt and implement quickly," Burling said.

In the meantime, states are using funds from EPA "network readiness" grants to prepare their data and systems for this network, now called the National Environmental Information Exchange Network (NEIEN).

Going forward, Burling said that Nebraska plans to engage XAware's XA-Suite and Web services as it exchanges more data across the network and looks to change the reporting process, which is currently in hard-copy format.

As state budgets shrink, and states and the EPA increasingly need quick access to data, particularly on hazardous materials, rolling out XML in all states becomes more urgent.

"In these times, state governments are having shortfalls, so this process, when implemented, will enable states to put those resources to use in other areas and do more important environmental work," Burling said. "Ensuring the quality of the environment is really what our job is."

CASEYHIBBARD2002@YAHOO.COM

AUTHOR BIO

Casey Hibbard is an independent writer based in Santa Fe, New Mexico. For more than five years, she has written news, features, and customer case studies on emerging technologies and companies. Additionally, she formerly served as the technology columnist for the Colorado Springs Business Journal.

WRITTEN BY **EDWARD KIERKLO**

Converting X12 EDI to XML

An economical model

The investment in traditional ANSI EDI is significant, involving established business and technical processes that have proven value and are integrated in everyday production across a spectrum of industries and organizations.

The limitations to this existing technology are that it is expensive to implement, generally limited in scope, somewhat behind the times, and the realm of only a small number of technicians, owing to the dense nature of the data. Information in an ANSI X12 EDI file is rich but indirect. To "read" an EDI file of this kind requires an understanding of the particular X12 standards.

This metadata consists of published and predefined values to which trading partners adhere. The business logic and rules are embedded in the data itself. Commercially available software both maps to this format and deconstructs information according to the semantics and syntax of the X12 Standards for Electronic Data Interchange (see sidebar: What Is ANSI EDI?).

Listing 1 illustrates a minimal purchase order (Listings 2–4 can be found at www.sys-con.com/xml/sourcec.cfm).

E-commerce has evolved considerably from the early days of computer-to-computer transactions. In the advent of numerous technological advances, other more robust platforms have since emerged. The commerce eXtensible Markup Language, for example, states flatly that "cXML is better for communicating purchase orders than other formats (such as ANSI X12 EDI 850), because it is flexible, inexpensive to implement, and it supports the widest array of data and attachments."

While others remain contenders, however, EDI is established and accepted. Admittedly, while EDI is significantly represented in business, it is not ubiquitous. However, many dominant retailers, manufacturers, distributors, and others have incorporated it into their supply-chain processes and mandate its use by vendors and suppliers. And even now the government is prescribing EDI for electronic health care exchange by legal decree.

The value proposition is how to leverage current, demonstrated systems while invoking the benefits of newer approaches.

Listing 2 is a schema that transposes EDI into an XML format that is explicit and self-defining using the structural components of the X12 standards. The conversion is designed to be "as is" – the output represents only what exists without undue elaboration as to the nature of EDI. This information is then self-describing. The rendered document adds no external logic and uses only those minimal components of EDI.

Listing 3 takes the original EDI example referenced earlier and displays it in XML (www.redixml.info).

The XML file can now be evaluated by means of XSLT or other methods for interrogation, stored as supplementary transactional records, or "repurposed" into an internal or external organization standard for input into other practices.

Listing 4 is an XSL stylesheet that repackages the original EDI file. Given the terse syntax of EDI, the nodes are repetitive. The primary mechanism for discovery is the value of the "id" attribute of the "element" element.

The purchase order found in Figure 1 compactly encapsulates aspects of the purchase order in Listing 1, with accompanying terminology embedded in X12.

An ongoing debate continues regarding how to harness decades of legacy EDI in collaboration with eXtensible Markup Language. There is a plethora of XML dialects that seek to leverage existing EDI vocabularies. Among these are Value Chain Markup Language (www.vcml.net), ebXML (www.ebxml. org), and cXML (www.cxml.org), as well as the X12 organization (www.x12. org).

This particular "literal" translation methodology keeps the original data integrity intact. It is not necessarily meant to make a non-EDI audience more EDI literate. The benefit of this kind of rendering is that it can be a low-impact way of entertaining interoperability and linking to other systems.

EDWARDK@REDIXML.INFO

What Is ANSI EDI?

The American National Standards Institute (www.ansi.org) is a federation that promotes standards by consensus. The Accredited Standards Committee for X12 (www.x12.org) oversees these initiatives. ASC X12 develops, maintains, interprets, publishes, and promotes the proper use of American National and UN/EDIFACT International Electronic Data Interchange Standards. X12 also consists of numerous subcommittees representative of industry or business function.

The Data Interchange Standards Association (www.disa.org) is the secretariat for the ASC X12 standards development process. DISA publishes and is the only official source for X12 standards.

A "release" is a complete set of X12 standards. A "subrelease" is an affiliated modification to a previous release. Releases are not compatible. Notation refers to version, release, and subrelease. For example "004010" is version 4, release 1, and subrelease 0 or abbreviated as "4010", which was made available in 1997.

The release is organized by sections: Control Standards, Transaction Set Tables, Segment Directory, Data Element Dictionary, Code Sources, and Interactive Standards. An EDI transmission could be categorized as a nested entity with repeated sections.

A Transaction Set ID represents a specific context and purpose. For instance Transaction Set Identifier "100" has a Transaction Set Title of "Insurance Plan Description".

Semantically, "Segments" are functional descriptors and "Elements" are particular ascribed values. Segments may be likened to records within a file while elements are fields.

Logically an EDI batch submission is a one record variable–length file.

The adoption rate by American industry is significant, but it is not ubiquitous. It has permeated the supply chain methodologies of major manufacturers, retailers, distributors, and others who mandate its adoption by vendors and suppliers.

AUTHOR BIO

Edward Kierklo is a software developer residing on the fringe of Silicon Valley. He has been working with Electronic Data Exchange and implementing e-business solutions for the past eight years. His systems and business background includes implementation of transactional solutions in retail, finance, and healthcare.

Purchase Order

Transaction Set Identifier Code 850

Purchase Order Number SLKPPO01

Date 19981002

Free-Form Message Text UPS CONSIGNEE

Name SARA LEE KNIT PRODUCTS

Address Information
1000 E. HANES MILL ROAD

City Name WINSTON-SALEM,

State or Province Code NC

Postal Code 27105

Number of Line Items 2

Hash Total 2

FIGURE 1 Purchase order with terminology embedded in X12

```
ISA~00~          ~00~
~14~SLKP COMM-ID   ~12~YOUR COMM-ID
~981103~1017~U~00401~000006240~0~P~>*
GS~PO~SLKP COMM-ID~YOUR-COMM-
ID~19981103~1017~4227~X~004010*
ST~850~000014679*
BEG~00~SA~SLKPPO01~~19981002*
N9~ZZ~001*
MSG~UPS CONSIGNEE*
N1~ST~SARA LEE KNIT PRODUCTS~92~05*
N3~1000 E. HANES MILL ROAD*
N4~WINSTON-SALEM,~NC~27105*
PO1~001~1~EA~1~PE~~~IN~PLP436M*
PID~F~08~~~PRODUCT DESCRIPTION 1*
DTM~002~19981101*
PO1~002~1~EA~1~PE~~~IN~PLP437M*
PID~F~08~~~PRODUCT DESCRIPTION 2*
DTM~002~19981101*
CTT~2~2*
SE~14~000014679*
GE~1~4227*
IEA~1~000006240*
```

LISTING 1 Minimal purchase order

Patrick Rasche is a senior systems analyst with BearingPoint (formerly KPMG Consulting) in McLean, Virginia. He has more than seven years of experience in application and product development using J2EE and Microsoft technologies.

REVIEWED BY **PATRICK RASCHE**

eWebEditPro+XML 2.5

by Ektron Inc.

Ektron, Inc.

5 Northern Blvd. Bldg. 6
Amherst, NH 03031 USA

Phone: +1 (603) 594-0249

E-Mail: +1(603) 594-0258

Web: www.ektron.com

Test Platform

OS: Dell PWS340
Processor: 2.40GHz Intel Pentium IV processor, 60GB disk
Memory: 512MB of memory, Windows XP

Specifications

Platforms: Browsers: Netscape and IE Integration: ASP, ASP.NET, ColdFusion, Lotus, JSP, Perl, and PHP
Pricing: $599 for 10 named user license, $120 for annual maintenance

As a client/server application developer, I've often had to evaluate third-party user interface tools and weigh the value their functionality provides versus their cost and the inherent problems of integrating them within the application I was building. In recent years, as more and more applications are moving to the Web (or being built from scratch for the Web) I've found that developers have far fewer choices of user interface tools than we had in the days of desktop apps.

For me, it always seems to be a question of browser compatibility. For example, there are lots of cool tools out there that take advantage of ActiveX, but they only work in Microsoft browsers. I've never worked for an organization that is 100% dedicated to Microsoft browsers, so I've never had the luxury of using these tools. Recently, an application I was working on had a requirement for an XML/HTML editor that could be placed in a Web page. I faced the task of either building it myself or finding a third-party product that met my needs. I wasn't surprised to find that there were very few choices out there. But I was surprised that one of them, eWebEditPro+XML from Ektron, met my requirements and supported Netscape and IE.

eWebEditPro+XML

eWebEditPro+XML is a full-featured WYSIWYG HTML editor, but it's so much more. eWebEditPro+XML is also an ActiveX component that can be embedded within a Web page, with support for both Internet Explorer (versions 4.01 and above) and Netscape (versions 4.6 and above). eWebEditPro+XML provides an API so that developers can integrate the product with their Web applications, and the user interface is completely configurable. The "+XML" version adds powerful XML support. eWebEditPro+ XML is easy to install and use and the support is excellent.

Installation

The product is available for download from the Ektron Web site and is activated by a license file that Ektron e-mails you once you purchase the product. Evaluation versions function for 30 days. Server installations require you to unzip the download to your Web server. Client installations are simple: if you're using Internet Explorer 5.0 or above, an auto install will kick in when you view an eWebEditPro+XML-enabled Web page for the first time (other users must manually download the client install program and run it). The auto install runs over the Web, so your Web server administrators may need to provide proper access to the client install EXE on the Web server. The Ektron knowledge base and FAQ address most of the installation issues I came up with.

One note about licenses: they are issued on a named-user cost structure (as opposed to concurrent users) and are valid for a single domain. License costs can escalate for applications distributed to a large user base. If you don't need the features offered by the +XML version of the product, eWebEditPro provides all the other benefits and costs half as much. In addition to the production domain license, Ektron provides two additional licenses for staging and development with the purchase of annual maintenance (e.g., www.mydomain.com, staging.mydomain.com, development.mydomain.com). The product will function in Web pages residing in any subdirectory of the licensed domains but not in subdomains.

Using eWebEditPro+XML

If you know how to use a word processor, you can use eWebEditPro+ XML. Its editing interface and functionality are similar to those of Word or WordPerfect. Users can jump right in with little or no training.

Adding eWebEditPro+XML to a Web page requires only a few lines of HTML and JavaScript. Experienced Web developers can take advantage of the product's API to create powerful text-editing applications and content management applications, or to generate the Web pages themselves with the product. eWebEditPro+XML comes with integration support for ASP, ASP.NET, ColdFusion, Lotus, JSP, Perl, or PHP and has full documentation and sample applications.

Technically speaking, eWebEditPro+ XML is an ActiveX component developed with Microsoft Visual Basic. eWebEdit Pro+XML supports Netscape 4.6 and above and IE 4.01 and above. When deciding between developing the solution myself and purchasing a third-party tool, I found that I could build a fairly feature-rich editor myself using DHTML and leveraging MSXHTML. DLL, minus a few perks (like spell check), in a fairly short amount of time. But I'd be locked into Internet Explorer and some of our users still prefer Netscape (not to mention that spell check was one of our requirements). eWebEdit Pro+XML solved my browser-dependency problem and gave me spell check to boot. This is in addition to the simple configuration, technical support, and XML capabilities. It was easy to justify the cost.

Developers can quickly and easily configure the eWebEditPro+XML interface by editing a single configuration file. This includes showing or hiding toolbars, menus, and buttons, and adding custom buttons, menus, and JavaScript functionality. These configurations can be made on the fly at runtime as well as via JavaScript, if you so desire. eWebEditPro+XML also features international language support.

The feature that sets eWebEditPro+ XML apart from other HTML editors is the XML support. Developers can insert custom XML tags to create data entry templates or data placeholders (see Figure 1). Our solution uses XML tags as data placeholders combined with user-entered content, allowing our users to produce correspondence templates that, when merged with data, produce form letters with content customized for each recipient. With most HTML editors, inserting XML tags would result in an error or the editor would simply ignore the tags. eWebEditPro+XML handles undefined XML tags out of the box – no integration or customization is necessary. eWebEditPro+XML allows you to configure your custom tags to control the way they are displayed and the way they accept and handle data entry by the user.

Figure 1 shows the product in use. As you can see from looking at the toolbar, most of the popular functions are there: cut, copy, paste, find, print, undo, redo, spell check, numbering, bullets, indent, outdent, left align, center, right align, justify, font face, font size, font color, background color, bold, italic, underline, superscript, subscript, and many more. eWeb EditPro+XML supports the insertion of tables and images as well. The product is integrated with Tidy, an HTML formatting utility that strips extraneous characters from MS Word and MS Excel files converted to HTML.

Summary

Ektron has a winning combination in eWebEditPro+XML – great product, great documentation, and great support. Although this is my first experience with Ektron, this was without a doubt the easiest third-party tool integration effort I have experienced. There were very few surprises and my questions were answered the same day on the company's development forum. I feel I have just scratched the surface of the product's capabilities. If you are looking for a tool to implement text or content editing in a Web application, I suggest you download an evaluation version and give it a test drive.

Product Snapshot

Level Beginner to advanced programmers

Pros

- Full-featured WYSIWYG HTML Editor
- Easy to Install, integrate, and use
- Excellent support, documentation, and examples
- Supports XML tags

Cons

- Cost

PRASCHE@BEARINGPOINT.NET

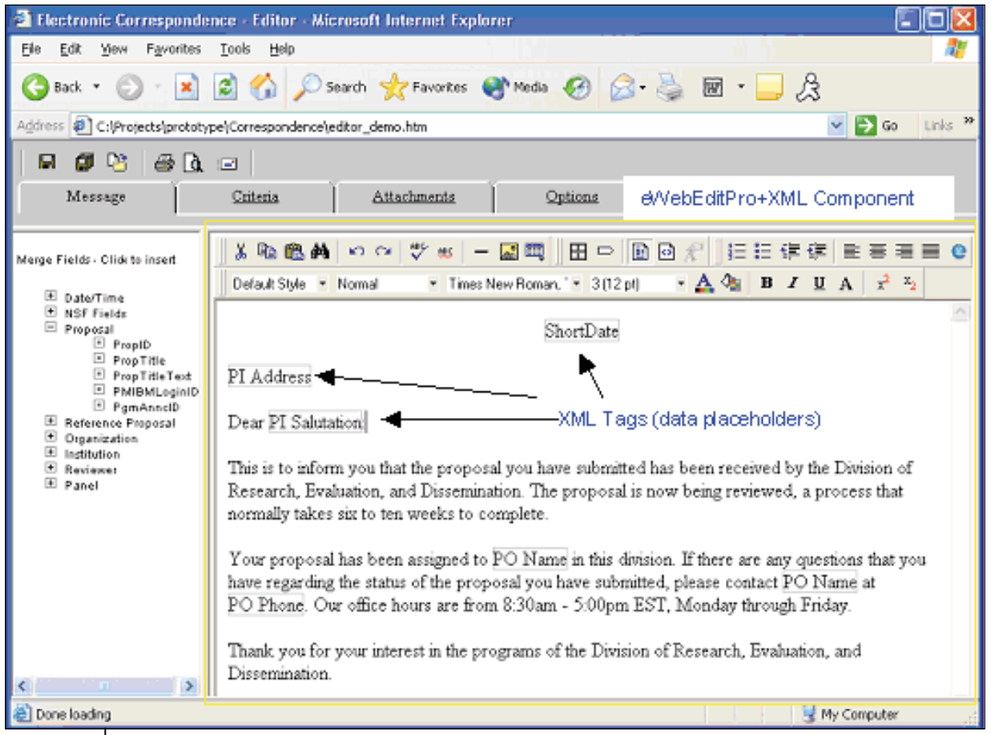


FIGURE 1 | eWebEditPro+XML Interface

WRITTEN BY **HITESH SETH**

Introduction to SALT

Unleashing the potential

Speech Application Language Tags (SALT) is a set of XML-based tags that can be added to existing Web-based applications, enhancing the user interface through interactive speech recognition. In addition, SALT can be used to extend Web-based applications to the telephony world, thereby providing an opportunity to unleash the potential of a huge user community, users of normal touch-tone telephones.

SALTforum, an organization founded by Microsoft, Cisco, SpeechWorks, Philips, Comverse, and Intel, has spearheaded development of the SALT specification, now in its 1.0 release.

Multimodality: Beyond Standalone Web and Speech Applications

On one hand, we have the ubiquitous World Wide Web, which provides the core standards-based networking infrastructure and connects a large number of users to consumer and business applications. On the other hand, Interactive Voice Response (IVR) and touch-tone systems–based telephony applications have also been around for some time, providing basic connectivity (typically using the touch-tone and prerecorded speech interfaces) to telephony users. In both scenarios what we're really using is a single modality. In the case of interactive speech recognition applications, it is touch-tone or speech input and speech output, while in the Web application scenario, we're using a basic graphical user interface as the application modality.

Multimodality means that we can utilize more than one mode of user interface with the application, something like our normal human communications with each other. For instance, consider an application that allows us to get driving directions – while it's typically easier to speak the start and destination addresses (or even better, shortcuts like "my home," "my office," "my doctor's office," based on my previously established profile), the turn-by-turn and overall directions are typically best viewed through a map and turn-by-turn directions as well, something similar to what we're used to seeing on MapQuest. In essence, a multimodal application, when executed on a desktop device, would be an application very similar to MapQuest but would allow the user to talk/listen to the system for parts of the application's input/output as well – for example, the starting and destination addresses. That's multimodal.

Imagine the same application using the same interface on a wirelessly connected PDA. Now we're talking true mobile/multimodal application. If we let our imaginations go a little bit wilder, we can easily extend the same application to the dashboard of our car or any other device we can imagine working with. That's really the vision, which, given the current state of technology, isn't far away. Another modality that can be added to the example application would be a pointing device that would zoom the map, focusing on a particular location.

So how does SALT fit in with all this? SALT has been built on the technology required to allow applications built using SALT to be deployed in a telephony and/or multimodal context.

SALT Application Model

With the evolution of the SALT specification and the platforms and tools around it, the exact architecture of SALT-based interactive, speech-driven applications is yet to surface.

Figure 1 presents an abstract representation of application architecture for deploying and using SALT-based applications. As expected, it's very similar to that of a Web application, with two major differences. In this case the application is also capable of delivering dynamic speech interactions (if the appropriate browser is capable of handling SALT, e.g., through an add-in or natively), and a stack is present that represents a set of technologies broadly representing the integration of speech recognition/synthesis and telephony platforms.

A note of caution: this diagram is really a conceptual representation. Where exactly the SALT browser/interpreter and speech recognition/synthesis components fit in depends on the capabilities of the end-user device/browser – actual implementation of the SALT stack may vary based on vendor implementations.

The Advanced Speech Recognition (ASR) component focuses on recognizing spoken user utterances based on speech grammars, whereas the Text-to-Speech synthesis component is focused on dynamically converting text messages into voice output.

When SALT applications are used in the telephony world, the telephony integration component connects the speech platform with the world of telephones, the Public Switched Telephony Network (PSTN). This is typically achieved by integrating telephony cards with the analog/digital telephony lines of your telephony provider (your phone company).

When SALT applications are used to enhance the interactions of Web-based applications by adding multimodality to the application, a typical Web application delivery framework (based on TCP/IP/HTTP/HTML/JavaScript etc.) is used for delivering the Web application, and the speech/telephony platform is used for the "speech/voice" aspect of the whole interaction, depending on the nature of the connection and the location of the speech recognition/synthesis components. Both interactions can happen together seamlessly, as part of the same user session, on the user's choice.

SALT & VoiceXML

It's clear that SALT and VoiceXML have utilized the Web application delivery model as an open platform for delivering telephony applications. However, VoiceXML and SALT have different technical goals – whereas VoiceXML tends to focus on development of telephony-based applications (applications used through a phone), SALT focuses on adding speech interaction and telephony integration to existing Web-based applications and enable true multimodality. In this case, I would also like to highlight the development of another upcoming standardization initiative called X+V (which stands for XHTML+Voice), an effort to combine VoiceXML with XHTML to develop multimodal applications.

Another difference between SALT and VoiceXML is the overall approach that has been utilized to develop applications. Whereas VoiceXML is pretty much declarative in nature, utilizing its extensive set of tags, SALT is very procedural and script oriented, having a very small set of core tags.

Also, it's important to understand that SALT actually utilizes key components of the standardization effort carried at the W3C Voice Browser Activity, including the XML-based Grammar Specification and the XML-based Speech Synthesis Markup Language. Both these specifications have been used by the VoiceXML 2.0 specification as well.

Hello SALT

The best way to introduce a language is to show what it actually looks like. Following the tradition of the famous "Hello World" program, let's see how a SALT application says "Hello World."

```
<html xmlns:salt="http://www.salt
  forum.org/2002/SALT">
 <body onload="hello.Start()">
   <salt:prompt id="hello">Hello
    World</salt:prompt>
 </body>
</html>
```

As you can see , SALT tags (<prompt>) have been added to an existing XHTML document and are initiated through methods – Start(). For instance, when the above document is loaded in a SALT 1.0-compatible browser (it could be a telephony browser or a multimodal desktop/handheld browser) it would say "Hello World" using a Text-to-Speech (TTS) engine.

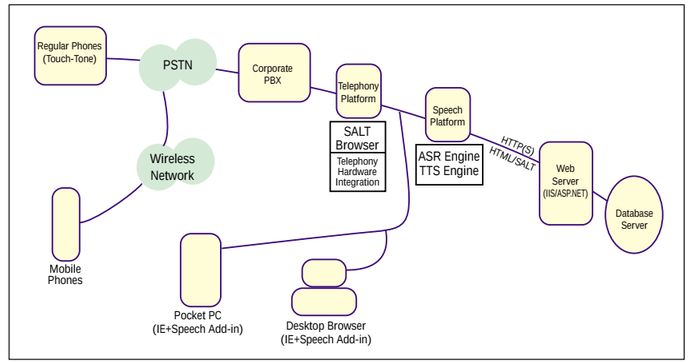


FIGURE 1 SALT application architecture

Going further with our exploration of SALT, let's look at how SALT applications provide speech recognition capability. The code shown in Listing 1 can be used as the basic template for an interactive order information system.

To understand the various components of this multimodal application, let's look at a snapshot of the sequence of actions performed when this application is launched within a SALT-compatible browser.

- When the application is started, the RunApp() JavaScript function initiates the prompt "Welcome" to be played.
- It is followed by the playing of the prompt "AskOrderNumber".
- Speech recognition is then initiated for the form parameter "number" using the external XML grammar OrderInfo.xml‰.
- Once the recognition is over, the function processOrderStatus() submits the value ("number") to the server-side script "GetOrderStatus.aspx", which would ultimately look up the sales application and provide order information.

Elements of SALT

Since SALT uses the underlying XHTML (or similar) markup and JavaScript as the core application delivery model, the core language represented by the SALT 1.0 specification is really a small collection of tags – prompt, listen, grammar, record, dtmf, and smex. Table 1 shows technical highlights and example usage of elements of SALT language.

Microsoft .NET Speech SDK

Support for SALT-based application development is available from Microsoft through the .NET Speech SDK. A Beta 2 of the SDK is available from Microsoft's Web site. The SDK has three key components:

1. An add-in for Microsoft Internet Explorer that recognizes SALT tags and allows users to interact with the application using the desktop's microphone and speakers/headphones
2. A set of ASP.NET-based speech controls, which allow developers using Microsoft Visual Studio .NET to create multimodal/telephony applications and/or add speech interactivity to existing Web applications developed using Microsoft .NET and ASP.NET frameworks.
3. Development and debugging tools, including grammar builder, prompt builder, and speech debugger tools to aid in the development of grammars and prompts.

It's important to understand that a SALT-based application can be delivered using a non-ASP.NET Web application framework (e.g., Perl or JavaServer Pages). What the .NET Speech SDK provides is ease of development in adding speech to your existing Web applications or creating new applications.

AUTHOR BIO

Hitesh Seth is the chief technology officer of ikigo, Inc., a provider of XML and Web services monitoring and management software. A freelance author and well-known speaker, he regularly writes on application development using J2EE and Microsoft .NET, EAI/B2B integration, Web services, and speech and wireless applications.

Element	Key Functions and Highlights	Usage Scenarios
<prompt>	• Main element for speech-output • Play prompts and use TTS • TTS is delivered through W3C SSML • Prompts are queued and played back by using the Queue and Start methods	<prompt id="RecordedPrompt"> <content href="welcome.wav"/> </prompt> <prompt id="DynamicPrompt"> Did you say <value targetelement="txtOption" targetattribute="value"/>? </prompt>
<grammar>	• Used within <listen> elements for specifying input grammars • Specifies a set of utterances that a user may speak to perform an action • Support XML form of W3C SRGS (Speech Recognition Grammar Specification)	**Inline Grammar** <grammar xmlns="http://www.w3.org/2001/06/grammar"> --- line grammar in SRGS format --- </grammar> **External Grammar** <grammar src="Employees.grxml" type="application/srgs+xml"/>
<record>	• Used within <listen> elements or speech recording • One <listen> can have only <record> element.	<listen id="recordMessage" onreco="processMessage"> <record beep="true"/> </listen>
<listen>	• Used for speech recognition and/or audio recording • Once the results have been obtained, a <bind> element can be used to process the results • Provides Start(), Stop() and Activate methods to initiate speech recognition/recording	<listen id="listenEmployeeName"> <grammar src="MyGrammar.grxml"/> <bind targetelement="txtName" value="//employee_name"/> </listen>
<bind>	• Used within <listen> elements for processing results from speech recording or speech recognition	
<dtmf>	• Used in telephony applications for recognizing DTMF based touch-tone inputs • Similar to <listen> element	<dtmf id="dtmfPIN"> <grammar src="PIN_Grammar.grxml"/> <bind value="//pin" targetelement="txtPIN"/> </dtmf>
<smex>	• smex = Simple Message Extension • For platform specific functionality • Example logging/telephony call control, etc.	

TABLE 1 | SALT elements

Conclusion

With the emergence of SALT, it is pretty clear that there has been some confusion regarding a single standard for development of open standards-based telephony applications. Even with different key technology objectives and implementation models, both VoiceXML and SALT can be utilized to develop telephony applications. However, I believe that the SALT 1.0 specification and Microsoft's participation in the speech application world is very positive. In this article we took a quick look at the technical highlights and capabilities that the SALT 1.0 specification can provide to existing Web-based applications, extending them to provide multimodal capability and/or extend existing applications to the telephony world. I'll continue my exploration of SALT in another article, in which I'll explain how to develop SALT–based multimodal and telephony applications using Microsoft .NET Speech SDK.

References

- *SALT Forum:* www.saltforum.org
- *SALT 1.0 Specification:* www.saltforum.org/downloads/SALT1.0.pdf
- *Microsoft .NET Speech SDK:* www.microsoft.com/speech

HKS@HITESHSETH.COM

LISTING 1

```
<html
xmlns:salt="http://www.saltforum.org/2002/SALT">
 <body onload="RunApp()">
 <form id="OrderForm" method="post"
   action="GetOrderStatus.aspx">
  <input name="number" type="text"/>
 </form>
 <salt:prompt id="Welcome">
  Welcome to ABC Ordering System.
 </salt:prompt>
 <salt:prompt id="AskOrderNumber">
  What is the order number?
 </salt:prompt>
 <salt:listen id="recoOrder"
onreco="processOrderStatus()">
  <salt:grammar src="OrderInfo.xml"/>
 </salt:listen>
 <script>
  function RunApp() {
   if (OrderForm.number.value=="") {
    Welcome.Start();
    AskOrderNumber.Start();
    recoOrder.Start();
   }
  }
  function processOrderStatus() {
   OrderForm.number = recoOrder.text;
   OrderForm.submit();
  }
 </script>
 </body>
</html>
```

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

XML NEWS

World Wide Web Consortium Publishes XForms 1.0 as a W3C Candidate Recommendation

The World Wide Web Consortium (W3C) has announced the release of the XForms 1.0 Candidate Recommendation. XForms 1.0, the foundation for next-generation Web-based forms, combines the ability to separate purpose, presentation, and results with the eXtensible Markup Language (XML).

Advancement of this document to Candidate Recommendation is a statement that the specification is stable. The W3C XForms Working Group invites the Web development community at large to implement the specification and demonstrate interoperability.
www.w3.org

NeoCore Broadens XML Connectivity Through Data Junction Alliance

(*Austin, TX, and Colorado Springs, CO*) – NeoCore Inc., developer of the self-describing NeoCore XML Information Management System (XMS) database that reduces the cost and time-to-market for business applications, and Data Junction, setting the industry standard for data transformation tools, have formed a technology alliance to deliver best-of-class product integration and services.

NeoCore leverages Data Junction integration technology to provide the connectivity that allows users to import data from a variety of sources and generates XML content from the aggregation of different databases. In this capacity, Data Junction supports NeoCore customers by eliminating the cumbersome problems associated with sharing data among applications, thus managing customers' XMS integration needs.
www.neocore.com
www.datajunction.com

Is COMDEX Saying Goodbye?

(*Las Vegas*) In the world of computing, it's well known that everything is supposed to get smaller…it's just that no one ever thought it would one day happen so dramatically to COMDEX, once one of the world's largest technology shows.

This year COMDEX suffered the largest drop in numbers from one year to the next of any previous show. It was so catastrophic that no one would publicly admit it: the talk of the town, accordingly, was the discrepancy between the numbers being reported (125,000) and the numbers in attendance (more like 75,000).

Compare that to the 225,000 claimed just two years ago for COMDEX 2000 and you begin to see why Los Angeles–based Key3Media Group, Inc., the show's current producers, were openly contemplating filing for Chapter 11 bankruptcy protection immediately after this year's show was over, a scenario that may well have proven real by the time you read this report.
www.comdex.com

OracleWorld Offers In-Depth Knowledge on Building Information Management Systems

(*San Francisco*) An estimated 23,000 attendees took part in this year's OracleWorld San Francisco conference, November 11–14 at Moscone Center in San Francisco. Attendees at this year's event gained in-depth knowledge on how to achieve the highest levels of reliability and performance from Oracle's information management systems, at the lowest total cost of ownership.

Oracle addressed its use of Linux database clusters to provide reliable, fast data on demand at a fraction of the cost of traditional IT systems. Oracle highlighted significant customer successes and partner momentum with Dell, Intel, Red Hat, and other Linux vendors.

Attendees experienced more than 400 IT strategy and technology sessions in nine focus areas, along with a world-class keynote series that included technology industry luminaries from Dell Corporation, EMC Corporation, HP, Intel Corporation, and Oracle.
www.oracle.com/oracleworld

Altova Extends Market Leadership in Web-Based XML Content Editing Market

(*Beverly, MA*) – Altova, Inc., producer of XMLSPY, a leading XML tool with over 650,000 registered users and 90% of the market share, has announced the release of AUTHENTIC 5, immediately available as a placeholder control for the newly released Microsoft Content Management Server 2002 (CMS 2002). AUTHENTIC 5 is a standards-based, browser-enabled document editor that allows business users to seamlessly capture thoughts and ideas directly in XML format through a word processor-like interface; the content can then be saved to CMS 2002 for subsequent retrieval and transformation, unlocking corporate knowledge. Microsoft CMS 2002 is the latest addition to a family of Microsoft .NET servers, which reduces the time, cost, and complexity associated with building, deploying, and maintaining mission-critical, content-rich Web sites.
www.altova.com

WESTBRIDGE XML SOAP MONITOR DETECTS AND MONITORS SOAP TRAFFIC

(*Mountain View, CA*) – Westbridge Technology, Inc., a provider of security and monitoring solutions for XML Web services, has announced general availability of the Westbridge XML SOAP Monitor. The Westbridge XML SOAP Monitor enables enterprises to easily and effectively monitor their networks for all XML Web services traffic without requiring changes to the network.

Many IT departments are using XML Web services from both top-down initiatives and grass roots development efforts. Because XML Web services are so easy to use, much of the SOAP traffic across a corporate network is likely unknown and therefore unsanctioned traffic. "Because Web Services pass through traditional firewalls, many enterprises are unaware of the sensitive data that may be flowing on their networks. As Web services become more prevalent on the desktop, the problem of unauthorized or sensitive SOAP traffic on the network will only get worse," said Jason Bloomberg, senior analyst at ZapThink.

www.westbridgetech.com