Data Management • Enterprise Solutions • XML Labs

XML JOURNAL

THE ULTIMATE XML ENTERPRISE RESOURCE

October 2003 Volume:4 Issue:10

xml-journal.com

SYS-CON MEDIA

XML JOURNAL

FOUNDING EDITOR
Ajit Sagar ajit@sys-con.com

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

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

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

CONTRIBUTORS TO THIS ISSUE
Andy Astor, Steve Bailey, Dan Foody,
Paul Fretter, Allan Jones, Kunal Mittal,
Dare Obasanjo, Srinivas Pandrangi, Hitesh Seth,
Anupam Singh, Phoebe Yi, Sophie Zhong

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

FOR LIST RENTAL INFORMATION:
Kevin Collopy: 845 731-2684,
kevin.collopy@edithroman.com
Frank Cipolla: 845 731-3832,
frank.cipolla@epostdirect.com

WSRP Reignites Interest in Portals

WRITTEN BY **HITESH SETH**

Web Services for Remote Portlets, or WSRP, was recently approved as a standard by OASIS. Although a number of Web services standards are being worked on by different OASIS technical committees (TCs) – around Web services orchestration, management, security, reliable messaging, and ebXML – WSRP is particularly interesting as it brings out the benefits of open standards–based Web standards to the world of enterprise portals.

To understand what WSRP is all about, it's important to understand portals. Consider a financial services company, for example your bank. You typically have a savings and/or checking account. Once you've developed a decent relationship with your bank, you usually end up using additional services such as credit cards, bill payment, mortgage/student/auto loans, and so on. With the advent of e-business computing, a number of these services are available online through your bank's Web site. Typically, since these applications serve an individual purpose, they stand alone and have their own user interfaces.

But wouldn't it be nice to have all the applications available in a single application as a bunch of modules, under a common look and feel, using the same user profile information (including user IDs and passwords)? Also, if the various applications were available as "pluggable modules," it would be really nice to be able to personalize your view of the bank. The three Ps – presentation, profile, and personalization – are where portal technology began.

A portal is a framework that allows a company to create a unified presentation mechanism that is used to deliver key applications in a personalized manner to its users. The users can be customers, employees, business partners, or investors. The mini-applications launched from a portal are specialized views of the main application, providing key information that would be of immediate interest to the user within a unified portal. These mini-applications, or views, are called portlets. (It is important to point out that some vendors call them Web parts, others gadgets.)

Following in the footsteps of the popular public customized/personalized portals such as My Yahoo, a set of "pure-play portal vendors," including Plumtree and Epicentric, quickly brought the portal to the enterprise. The term Enterprise Information Portal (EIP) was coined and an integrated set of technologies including the three Ps was integrated into a "portal server." Going forward, portals were further broadened to include other capabilities such as customization, content management, and collaboration. Quickly the platform vendors realized that portals were key aspects of their platform growth strategy. Through internal organic growth or through acquisitions, they established portal server strategies tightly integrated with their own applications and/or database servers. Today, the portal technology landscape is quite broad and includes platform, application, and pure-play vendors such as BEA, IBM, Microsoft, Oracle, PeopleSoft, Plumtree, SAP, Sun, and Vignette.

So how does WSRP fit into all this? Until now vendors didn't really have a standard architecture for developing portlets. Based on your vendor's technology, you ended up developing portlets using their proprietary SDK, and if for some reason you needed to support another portal architecture, you would end up rewriting it. Also, until now there wasn't a consistent architecture/methodology followed by the vendors to loosely separate the portlets from the portal server. Typically, enterprises would develop portlets to their own applications and install them right on top of the portal server. This was okay for initial deployment, but when you scale to the 10s (and even 100s) of applications typically available in a large corporation, the local portlet model doesn't really scale. Some of the vendors designed/implemented their own "remote portlet" architectures, but again these were very vendor-specific. How will WSRP benefit us? WSRP provides us with a standards-based architecture, where a portal can truly be implemented in a "shared services" mode and a common IT infrastructure/architecture group can initiate the portal-based application deployment strategy and other internal groups can rapidly build their own WSRP-compliant portlets and get "hooked-on" to the portal. The model will also be very interesting for scenarios in which cross-enterprise portlets are used.

You can get more information on the WSRP standard development, including the specification, at the OASIS Web site, www.oasis-open.org. Also, I plan to closely follow the standard development, tracking the support and implementations, portlets, etc., at http://wsrp.info.

AUTHOR BIO

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

HITESH@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 Kircaali fuat@sys-con.com

VP, Business Development
Grisha Davida grisha@sys-con.com
Group Publisher
Jeremy Geelan jeremy@sys-con.com
Technical Director
Alan Williamson alan@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
Director of Sales & Marketing
Megan Mussa megan@sys-con.com
Advertising Sales Manager
Alisa Catalano alisa@sys-con.com
Associate Sales Managers
Carrie Gebert carrieg@sys-con.com
Kristin Kuhnle kristin@sys-con.com

SYS-CON EVENTS
President
Grisha Davida grisha@sys-con.com
Conference Manager
Michael Lynch mike@sys-con.com
National Sales Manager
Sean Raman raman@sys-con.com

CUSTOMER RELATIONS
Circulation Service Coordinators
Niki Panagopoulos niki@sys-con.com
Shelia Dickerson shelia@sys-con.com
Edna Earle Russell edna@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: $69.99/yr (12 issues)
Canada/Mexico: $89.99/yr
all other countries $99.99/yr
(U.S. Banks or Money Orders)
Back issues: $12 U.S. $15 all others

Sarbanes-Oxley Will Change Your Life

WRITTEN BY **ANDY ASTOR**

This column may require a little patience on your part, but I think it will be worth it in the end. Let's start with a simple premise: within a year, nearly everyone reading these words will be deeply impacted by Sarbanes-Oxley, yet many have never heard of it. The purpose of this note is to offer you a preview of what's to come. In other words, a wake-up call.

First of all, who or what is Sarbanes-Oxley? Simply put, the Sarbanes-Oxley Act (SOA) is the federal law that was put in place last year in response to the scandals at Enron, MCI, and other large public corporations. The law contains a wide variety of provisions around improving corporate ethical behavior, including assurances that companies' financial statements accurately reflect the state of their business. And it puts teeth into those provisions with heavy fines and prison for senior executives if their companies do not comply.

Why, then, is this month's guest editorial in ***XML-Journal*** about federal legislation? After all, this is a technology magazine. Why isn't this a technical editorial about the Semantic Web, RSS, or InfoPath? Well, sometimes it's better to come out from under the technology, and remember why we build this stuff. Usually, the reason is to serve a business, quite often an American public company. If you work for such a company, then you need to understand the implications of Sarbanes-Oxley, because its provisions apply to every publicly traded American company.

Remember Y2K, and the pervasive impact it had on all of our lives just a few years ago? Sarbanes-Oxley is every bit as pervasive as Y2K, but it has no end. Every system that impacts your company's financial statements – even indirectly – will be impacted by Sarbanes-Oxley, presumably forever.

This brings us to the burning question: "How does Sarbanes-Oxley affect me?" After all, this magazine targets XML technologists, and Sarbanes-Oxley is just about reporting accurate financial numbers, right? Actually, no. And it's here that the subject gets interesting.

What is XML all about? Integration of course. XML provides a platform-independent mechanism to integrate disparate, heterogeneous computer systems. What is Sarbanes-Oxley compliance all about? Well, it's about the integration of information across large, heterogeneous organizations in a highly controlled manner. Sound familiar?

Consider this. A key provision of the Sarbanes-Oxley Act (a provision known as Section 404) is that companies must (a) document every business process that impacts their financial reports, and (b) put systematic controls into place that ensure that these business processes produce accurate data. Sounds simple, right? All you have to do is document every single process in your global, multibillion dollar company, and then put controls in place to make sure that the processes execute flawlessly. And if you get it wrong, you can go to jail. As you can imagine, companies are scrambling right now. And matters are only made worse by the deadline for compliance, which for most companies is 2004.

So how will these controls be implemented? I thought you'd never ask.

Many business process controls are being implemented today with manual solutions, such as spreadsheet-based data consolidation, checklist procedures, and other similar solutions. Given the time pressures of Sarbanes-Oxley, and the enormity of the task, there is simply no alternative to these manual, one-off answers. But how long will these manual controls be allowed to exist? After all, like nearly all manual systems, they will be expensive to implement and monitor, prone to failure, and – most important – they simply will not scale well. Remember that these are controls that must cross organizational boundaries, diverse geographies, and heterogeneous systems.

I believe it is self-evident that the CFOs and corporate auditors who today are driving the implementation of mostly manual Sarbanes-Oxley controls will soon turn to the CIOs, seeking cost reduction, better automation and quality control, and scalable solutions. To whom will the CIOs turn? To you. They will look to the integrators; the architects and technologists with the ability to leverage a platform-independent mechanism that can implement enterprise-wide business processes and controls. They will look to you to be the agent of change. Are you ready?

Author Bio

Andy Astor is vice president for Enterprise Web Services at webMethods. In this role, he is responsible for driving the company's Web services strategy and execution, and for evangelizing webMethods' leading position in the Web services marketplace. Andy was recently elected to the board of directors of WS-I, serves on the International Advisory Board of Web Services Journal, *and was IT Strategy track chair for the 2002 Web Services Edge conference in New York City. He received his BA in mathematics from Clark University and his MBA from the Wharton School at the University of Pennsylvania.*

AASTOR@WEBMETHODS.COM

HOME
Enterprise Solutions
Content Management
Data Management
XML Labs

HOME
Enterprise Solutions
Content Management
Data Management
XML Labs

WRITTEN BY **KUNAL MITTAL**

Standards in the Real Estate Industry

An overview of what's new

The fever for new XML specifications for almost anything imaginable has hit the real estate industry. Companies that are actively pursuing some niche in this industry have realized the need to create and adopt standards for communication. As in other industries, however, competing XML standards are emerging, keeping any one standard from reaching a critical mass of adoption and fruition.

This article explores the Real Estate Transaction Standard (RETS) for the real estate industry. I'll talk about how some of these standards can leverage Web services technologies such as SOAP and WSDL. I'll begin by taking a look at the landscape and various XML standards in the real estate industry.

Industry Landscape

Adoption of XML standards will provide a common vocabulary and increase the interoperability of the Multiple Listing System (MLS) data and the different users that access this data on a daily basis, such as realtors, title companies, home insurance companies, and end users looking to buy or sell a house. This will also foster an increase in competition among software vendors seeking to serve the real estate industry. The emerging XML standards are intended to be used over the Internet and can leverage various Web services standards such as SOAP and WSDL for increased interoperability. A critical aspect of these standards is safeguards to protect intellectual property (IP) and the privacy of all involved in a real estate process.

The National Association of Realtors (NAR) is the largest professional association for people involved in all aspects of the real estate industry, with more than 730,000 current members. In 1999, together with several technology leaders, the NAR began an effort to make access to MLS information easier. The Real Estate Transaction Standard (RETS) is an open standard for accessing and exchanging real estate information between all the players involved in the real estate process.

There are several other standards in this space. The RELML specification was first announced in the summer of 1998. This version of the draft is currently implemented in OpenMLS's Real Estate Listing Management system.

The Data Consortium is an open-membership group of more than 50 companies and associations whose goal is an open source standard namespace for the commercial real estate industry. They have posted a namespace with 13 key elements to support XML streams and Create, Read, Update, Delete (CRUD) operations to a relational database. This includes an explanatory guide and dictionary.

The Alliance for Advanced Real Estate Transaction Technology (AARTT) announced an initiative to create open standards for data exchange within the real estate industry in order to streamline the residential real estate industry; this initiative is called CRTML (Comprehensive Real Estate Transaction Markup Language).

The increasing number of standards leads to the lack of adoption of a single standard. XML is a great technology that provides the opportunity to develop several standards. However, if the industry does not adopt *one* standard, XML can't be used to its full potential.

Real Estate Transaction Standard (RETS)

RETS is the standard gaining the most momentum and it appears to be the most comprehensive standard that can influence the real estate industry. RETS is a collection of several documents and tools.

- ***Real Estate Transaction Standard Protocol Specification, version 1.5 (draft 2):*** This document describes the RETS protocol and transaction set. RETS defines a series of computer interactions called "transactions" and serves as the definitive reference for developers who implement them. It establishes the parameters for each specified transaction, both upload and download, as well as the expected behaviors and results that compliant hosts and clients must be able to manage, including error codes. The specification also includes developer notes to explain technical details and a change process document to define how the standard will be administered.
- ***Real Estate Transaction Standard XML DTD (RETML):*** Version 1.5 of the XML DTD for working with the Real Estate Transaction Specification.
- ***Real Estate Transaction Standard Metadata DTD:*** The XML DTD for exchanging metadata between systems or between clients and servers.
- ***RETS Reference Implementation:*** A complete implementation of a client and server written in Java.
- ***RETS IDX Client:*** An implementation written in Visual Basic.
- ***RETS Compliance Tool:*** A scriptable compliance testing tool in Java. The tool accepts scripts written in XML that test the interaction of a client or server against predefined responses.

RETS Dissected

Let's take a closer look at RETS. RETS defines two major components – a RETS server and a RETS client. A user accesses a RETS client to access MLS data. This

AUTHOR BIO
Kunal Mittal is a solutions engineer at Wakesoft, Inc., and a consultant for implementation and strategy for Web services and service-oriented architectures. He has coauthored and contributed to several books on Java, WebLogic, and Web services.

client could be a Web site that is accessed through HTTP, a thick VB client, or another client or server system that uses Web services as the RETS client. The RETS client initiates the appropriate request to the RETS server, usually over the Internet. The RETS server responds with data formatted to the RETS specification. There are open source implementations of RETS clients and servers available. Figure 1 shows the architecture of a typical RETS request-response cycle.

I recommend downloading and installing the reference implementation of RETS from the RETS Web site (see references). The Java version is relatively easy to use and very well documented. It runs on top of Apache Tomcat and uses MySQL as the database. However, you could adapt this to your environment and run it on top of an application server such as JBoss, BEA WebLogic, or WebSphere. Once you get this implementation up and running and go through the steps in the install guide that comes with the download, you will see the Property search screen (see Figure 2).

After clicking on the search button, you will see a search results page as shown in Figure 3.

Underneath these simple JSP screens the reference implementation is using the RETS standard to exchange data. Listing 1 (available at www.sys-con.com/xml/sourcec.cfm) shows an example of an XML listing that is returned through the search function.

RETS and Web Services

RETS can be used in the context of Web services. Wrapping a RETS request or response in a SOAP message is trivial. Exposing the services in WSDL that some niche company in the real estate space provides is also a straightforward task. What's more interesting is what the concept of Web services brings to the real estate industry in addition to the standardization of the data exchange format.

Web services provide the communication endpoints to a service. They enable a transaction to be run asynchronously and a service to automatically discover and interact with new services without prior negotiation. Okay, there you have the buzzwords and phrases for Web services.

The adoption of Web services allows realtors to automatically find and understand how to interact with several of the MLSs in their area. They can use Web services to upload their listings in a simple and consistent way to all the MLSs that have exposed their interfaces using WSDL, and support the SOAP and RETS standards.

Complete Web services implementation has been targeted for RETS 2.0.

Summary

I've discussed the current state of XML technologies and standards for the real estate industry, including a deeper look into the real estate transaction standard, and talked about the reference implementation provided with the RETS. I've discussed at a conceptual level the application of Web services with the RETS specification. There is tremendous potential and need for standardization of communications and data exchange formats to enable greater interoperability of different real estate systems.

References

- Mittal, K. "Web Services and the Real Estate Industry," *Web Services Architect:* www.webservicesarchitect.com/content/articles/mittal01.asp
- *Real Estate Transaction Protocol:* www.rets.org/

KUNAL@KUNALMITTAL.COM

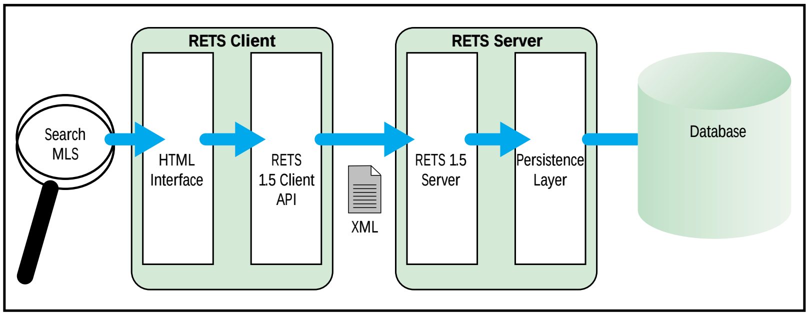
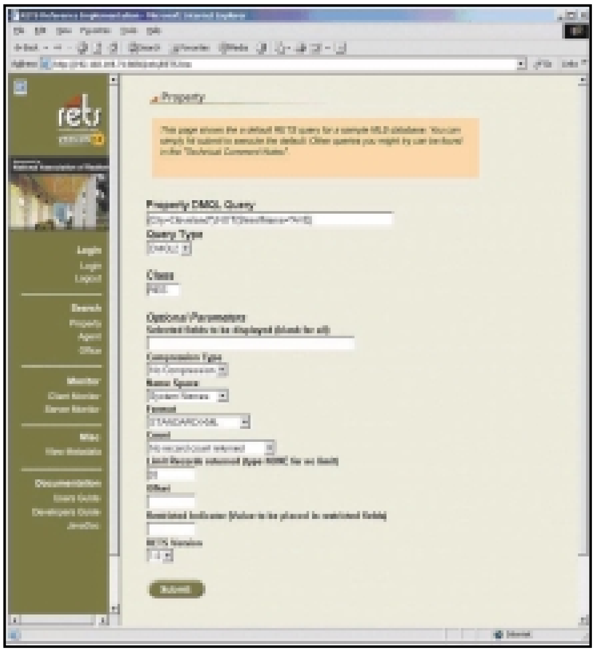


Figure 1 • RETS architecture



This page shows the a default RETS query for a sample MLS database. You can simply hit submit to execute the default. Other queries you might try can be found in the "Technical Comment Notes".

Property DMQL Query
(City=Cleveland*),(ST$SaleStatus=*A*)

Query Type
DMQL2

Class
RES

Optional Parameters
Selected fields to be displayed (blank for all)

Compression Type
No Compression

Name Space
System Names

Format
STANDARD-XML

Count
No record-count returned

Limit Records returned (type NONE for no limit)
20

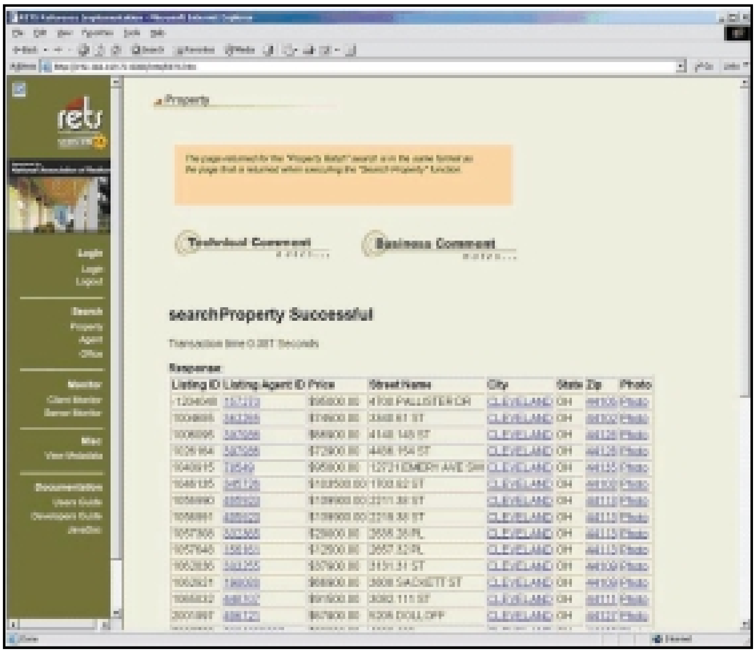
Offset

Restricted Indicator (Value to be placed in restricted fields)

RETS Version
1.5

Submit

Figure 2 • Property search screen



The page returned for the "Property Batch" search is in the same format as the page that is returned when executing the "Search Property" function.

searchProperty Successful

Transaction time 0.287 Seconds

Response:

Listing ID	Listing Agent ID	Price	Street Name	City	State	Zip	Photo
1204048	157133	$95800.00	4708 PALLISTER DR	CLEVELAND	OH	44105	Photo
1004895	342266	$74900.00	3448 61 ST	CLEVELAND	OH	44102	Photo
1006095	587588	$66900.00	4148 145 ST	CLEVELAND	OH	44128	Photo
1026164	587588	$72900.00	4406 154 ST	CLEVELAND	OH	44128	Photo
1040915	73549	$95800.00	12721 EMERY AVE SW	CLEVELAND	OH	44135	Photo
1046135	345126	$103500.00	1900 62 ST	CLEVELAND	OH	44102	Photo
1056990	485923	$109900.00	2211 38 ST	CLEVELAND	OH	44113	Photo
1056991	485923	$109900.00	2218 38 ST	CLEVELAND	OH	44113	Photo
1057868	302368	$29800.00	2658 28 PL	CLEVELAND	OH	44115	Photo
1057646	156161	$12500.00	2657 32 PL	CLEVELAND	OH	44115	Photo
1062836	583255	$37900.00	3151 31 ST	CLEVELAND	OH	44109	Photo
1062821	196003	$68800.00	3808 SACKETT ST	CLEVELAND	OH	44109	Photo
1066832	486102	$91500.00	3082 111 ST	CLEVELAND	OH	44111	Photo
2001897	486120	$67800.00	9208 DOLL OFF	CLEVELAND	OH	44127	Photo

Figure 3 • Search results screen

WRITTEN BY **DAN FOODY**

Building Manageable Web Services from the Ground Up

The management criteria developers must consider

Life is full of compromises, and application development is no exception to the rule. So, when that project deadline is looming (and it always is) you are faced with three options:

1. **Finish the functional application**
2. **Ensure that the application is manageable**
3. **Ask your manager for more time so you can accomplish both**

Since number #3 might be career limiting, it's usually a choice between #1 and #2. And no one gets extra points for manageability if the application doesn't function as advertised. Clearly, creating a fully functional application wins every time – When timeframes are tight, management functionality is often the first thing cut.

Good tools are available to automate the management of traditional applications built on application servers – you can find them in systems management suites like HP OpenView, Tivoli, and Unicenter. Great news if you are building a traditional monolithic application, but what if you're building an application using Web services? How do you build management capabilities into an application that is not stand-alone, but rather composed of a collection of services from multiple sources?

While Web services applications are more flexible, quicker to build, and quicker to adapt than their monolithic counterparts, they are inherently more complex because there are many more moving parts and myriad interdependencies. A problem with a single Web service has the potential to bring down the whole network of Web services. This complexity makes effective management even more critical for Web services applications than for traditional applications. Unfortunately, traditional management systems were never designed to manage applications composed of multiple separate yet interdependent parts.

Web Services Management

This article explores four management criteria that developers must consider when building applications based on Web services: security, monitoring, service location, and versioning. It also examines the specific challenges associated with each criterion.

Security

With traditional applications, managing access control typically means "check IDs at the front door." With Web services applications, there are two additional considerations:

- Web pages are typically protected by URLs; each URL can be assigned different access rights. A complete Web service, however, is represented by a single URL. For example, a Customer Web service may have operations for querying, creating, deleting, and updating customer records, yet it would have only one URL for all four operations. As a result, it's difficult to control access to read-only query operation separately from the operations that change data.
- The "front door" approach to security does not work with an application built from multiple services. You need to consider security at each service, and how the security settings for the individual services relate to the application as a whole. For example, let's assume the Customer Web service depends on two other Web services: Billing History and Order History. To allow each of the four Customer Web service operations (query, create, update, and delete), what security settings do we need on Billing History and Order History to make everything work? When the Customer Web service calls Order History, who should the Order History Web service assume is making the call: the Customer Web service itself, or the caller of the Customer Web service?

Monitoring

The challenges for monitoring mirror those of security. Most management systems and application platforms deal with monitoring at the URL level at best. This means that statistics for all Web service operations on a service are lumped together. Since many Web services include read-only operations (e.g., query) as well as transactional operations (e.g., create/update/delete) – and since the characteristics of read-only operations differ greatly from those of transactional operations – grouping them together generally results in gibberish. To properly manage a Web service, you must have operation-level granularity of statistics.

Beyond monitoring at the operation level, you may also want to further organize the statistics. For example, are you meeting service level agreements for Gold customers using the Customer Web service? Or, what is the average network latency of requests from the Customer Web service to the Order History Web service?

The second challenge for monitoring a Web service application is related to the fact that many distinct parts make up the whole. Knowing how these parts interact is important to knowing how the whole application will work. For example, if some of your customer record updates are failing, does the problem lie with the Customer Web service itself? Or Order History? Or Billing History?

Service location

Since a Web services application is not

AUTHOR BIO

As chief technology officer at Actional, Dan Foody leverages his extensive hands-on experience in enterprise systems integration software toward easing integration through Web services. He is an active participant in the Web services standards including WS-I and OASIS, where he |spearheads Actional's contributions on the OASIS Management Protocol Technical Committee and its efforts to deliver XML-based Web services management standards. He is the author of various application integration standards, and contributed significantly to the OMG standard for COM/CORBA interworking

one piece of software deployed on one system (or on a cluster of identically configured systems), how do the different parts of the application know where the others are? In the example, how does the Customer Web service know where Billing History and Order History reside?

This problem is somewhat easier to solve if a single IT group has oversight of all the separate parts (the customer systems, the order history systems, and the billing history systems). But once ownership for these systems doesn't all rest in the hands of one group, or you need to call Web services from systems outside your own firewall, keeping track of locations becomes a formidable challenge.

Versioning

In cases where different services are owned by different groups or consumed by multiple applications, managing service versioning warrants careful consideration. If the service is relocated to add capacity, how is this done without disrupting consumers? How is an upgrade for the billing system rolled out to the Customer Web service and other consumers? With monolithic systems, all parts of an application are versioned and rolled out at once, so problems of this nature don't arise. But they are critical considerations in the world of Web services.

Addressing Challenges

As with anything in software, all of these challenges can be addressed by simply adding code to your application, which will be explored in this section. However, it's important to weigh the consequences of hard coding. The byproducts of hard coding management functionality into service-based applications include inflexibility and brittleness – side effects that can undermine the benefits of a service-oriented architecture. There are alternatives that help minimize these consequences, which will also be discussed.

Security

Most application server platforms provide two methods of authorization: declarative and programmatic.

Declarative authorization allows you to separate the authorization from the code itself. It can be updated and viewed apart from the application code, meaning you don't need to recompile the application to update the security requirements. The following shows an example of declarative security in Java, which limits the access of the Customer Web service to customer service representatives:

```
<security-constraint>
  <web-resource-collection>
   <web-resource-name>CustomerSvc</
     web-resource-name>
   <url-pattern>/services/
    CustomerSvc</url-pattern>
   …
  </web-resource-collection>
  <auth-constraint>
   <role-name>CustomerSvcRep</
    role-name>
  </auth-constraint>
</security-constraint>
```

Declarative authorization is generally available only at the URL level (unless your Web service application is written using EJBs, which allow deployment-time per-operation authorization). If the ability to delete customer records must be limited to customer service managers, it requires programmatic authorization:

```
void DeleteCust(String custNo)
{
 if(!Context.isUserInRole
  ("CustomerSvcMgr"))
 {
  throw new SecurityException(
   "Please contact your manager");
 }
 …
}
```

Infusing the application with hand-coded security logic will result in brittleness, so it's best to use declarative security as much as possible (to define the minimum requirements for access to any operation) and only layer in programmatic security when absolutely necessary.

Beyond authorization at the Customer Web service, the next challenge is how to perform similar authorization at the Web services used by the Customer Web service. In many cases, these services need to know who the original caller is so they can properly perform their security checks. Doing this by hand is difficult and can open significant security holes. However, there are emerging solutions that leverage the SAML standard (security assertion markup language), which allows you to safely and securely transmit a caller's identity through a chain of Web services.

Monitoring

Beyond the statistics and information that can be gathered directly from your application server, tracking and monitoring per-operation or business-level statistics generally requires adding code into your application. For example, assume you want to track the number of customer records deleted each day, using the account closure rate as a measure of customer satisfaction. This requires the creation of a separate service dedicated to management. In this case, the service might have an operation called AccountClosureRate. The customer delete operation must be updated to compute the statistic:

```
void DeleteCust(String custNo)
{
 if(!Context.isUserInRole
  ("CustomerSvcMgr"))
 {
  throw new SecurityException(
   "Please contact your manager");
 }

 if(Statistics.isFirstDeleteToday())
  Statistics.deleteCount = 0;

 Statistics.deleteCount++;
 …
}
```

Using this statistic the new management Web service would look like this:

```
long AccountClosureRate()
{
 return Statistics.deleteCount;
}
```

You have the choice of making this new management service available as a standard Web service or making it a specialized management service built using a framework such as Java's JMX MBeans. In either case, the primary downside (apart from all the additional coding) is the necessity of invasively modifying your application code to gather additional statistics about the service.

Service location

As discussed, the Customer Web service needs to locate the services it consumes. As the Customer Web service moves through the development-test-production cycle, the location of Order History and Billing History will change (for example, the test versions of these services will be in a different location than the production versions). Hard coding the URLs of these service providers into our Customer Web service won't work.

Most SOAP tools offer some help here. With Visual Studio .NET, for example, you can set the URL behavior property of a Web service to dynamic. This stores the endpoint of the service in a configuration file that can be changed without recompiling the application. Other SOAP tools follow a similar approach of storing the endpoint URLs in configuration files.

While this approach is a step in the

right direction, the downside is that the location of each service provider is stored locally at each consumer. If Order History needs to be relocated, we would need to find and update the configuration files of all Customer Web service installations, along with those of other Order History consumers.

The solution to this problem is a directory service that enables consumers of services to look up the locations of their service providers in the directory – in this case the Customer Web service would look up both Order History and Billing History. Because the directory is shared by all the consumers, you need only change the one directory entry and all consumers see the change automatically.

While you could use any directory service, UDDI is purpose-built for Web services (in fact, a UDDI directory is accessed as a Web service). Each service instance is identified by a unique key (specifically, a binding template key). During development, you find the service you want to use and store the key. You then look up the service endpoint by key at runtime using the UDDI API.

```
BindingDetail      detail;
Vector             templates;
BindingTemplate    template;

detail = uddi.get_bindingDetail
  (orderHistoryKey);
templates = detail.getBinding
  TemplateVector();
template = (BindingTemplate)
  templates.get(0);

OrderHistoryURL = template.
  getAccessPoint().getText();
```

The example above (written using UDDI4J) shows a simple lookup of the endpoint for the Order History Web service, assuming there is only one instance of the server (though UDDI allows multiples for simple load balancing and failover). In a production application you also need to be concerned with caching of UDDI entries. The general policy is to cache the UDDI entry for as long as desired, but if the endpoint is unreachable, reload the entry from UDDI (in the event it's been updated) and retry the request.

Versioning

The ability to change service endpoints with UDDI addresses some versioning issues as it allows you to bring a new version of the service online, update the UDDI entry to point to the new version, and then take the old version offline. If consumers follow the caching behavior described above, all consumers will automatically update themselves to point to the new version of the service.

However, versioning is not always so simple. The strategy outlined above has two limitations: (1) the new version's API must be completely upward-compatible with the older version and (2) all consumers must be cut over to the new service at the same time.

Adequately addressing versioning challenges requires a different architectural approach. For example, assume Billing History is being versioned. The most effective action is to insert an intermediary, i.e., a proxy, in front of the Billing History service. The intermediary can handle the logic necessary to adapt between the old and new APIs, gradually transferring the load to the new version of the service (perhaps one customer at a time, or prioritized by Gold customers vs. Silver customers). After the load transfer, the intermediary continues to adapt between the old and new versions of the Billing History API for all requests. Consumers of Billing History can upgrade to the new API at their leisure, after which they can bypass the intermediary. Once all consumers have upgraded to the new API, the intermediary can be removed.

While writing these types of intermediaries is possible, it is extremely tedious to do so by hand and without a careful design can lead to performance and reliability problems.

Factoring Out Management

Many of the techniques in the previous section require hard coding management behavior into the services themselves – thus locking in the management behavior until a new version of the service is rolled out. Breaking this tight tie between service and management functionality is important if you need your IT environment to respond quickly to change. Management functionality must be factored out of the services themselves so that it doesn't impede change.

There are two ways to factor out management code. The first is to take advantage of the intermediary approach: build the management logic into a proxy that sits in front of the actual service. As consumers talk to the proxy, it performs its management tasks and then forwards requests on to the service provider. This approach clearly separates the management logic from the application, allowing you to update them independently. But, as discussed previously, poorly designed proxies can lead to significant performance and reliability problems.

The alternate approach is to build the management code as "protocol handlers" (SOAP extensions in .NET and JAX-RPC handlers in Java) that plug in to the application platform itself (not into the application code). This approach separates the code from the application and addresses some of the performance and reliability concerns. However, what it gains in performance and reliability it loses in flexibility. Management policy can't be updated without deploying new software on the application servers, reconfiguring the applications themselves, and restarting the applications.

Toward an Automated Solution

In the final evaluation, it's fair to say that hand coding management functionality is not only a hassle, but also ineffective in a Web services environment. The good news is that solutions are now available that automate a lot of what was discussed in this article, and much more.

This new solution category is called Web services management. Web services management solutions can be deployed either as proxies or agents (protocol handlers); the best solutions support both models as there are valuable use cases for both. Web services management solutions provide easy-to-use, flexible environments to build sophisticated dynamically versionable management rules and policy with just a few clicks of the mouse – avoiding all the tedious and time-consuming (not to mention brittle) hand coding you would otherwise have to do.

The ultimate success of a Web services application project includes the ability to effectively manage the Web services throughout their entire life cycle. Web services management solutions provide developers with the tools they need to simplify the process of building in manageability from the ground up.

References

- *The Apache Foundation:* http://xml.apache.org
- *SourceForge:* www.sourceforge.net
- *OASIS:* www.oasis-open.org
- *W3C:* www.w3c.org
- *Sun Microsystems:* www.sun.com
- *OpenOffice:* www.openoffice.org

DAN.FOODY@ACTIONAL.COM

HOME
Enterprise Solutions
Content Management
Data Management
XML Labs

WRITTEN BY **SRINIVAS PANDRANGI**

A Technical Overview of XBRL

How reporting should be done

This article provides a technical overview of the XBRL standard, with emphasis on the architecture of the standard and the potential applications surrounding the business reporting functionality it supports.

Business reporting has been the focus of discussion over the last few years in the wake of increased government oversight. The need for compliance and the need to standardize reporting processes are driving the creation of new applications and standards. XBRL is one such standard that will provide an extensible vocabulary framework within which business reporting can be done. This standard is particularly important due to the broad industry participation in the standardization effort (over 170 members from the financial services, accounting, and technology sectors worldwide) and the increasing evidence of its adoption in practice. For example, in the U.S. the FDIC recommends that all financial institutions post their bank call reports in the XBRL format. The SEC is moving toward requiring that all corporate public financial statements, like 10K filings, be in XBRL format.

XBRL defines a data model for information to be included in business reports, and the basic vocabulary for representing that information in XML. This standard intends to serve a constituency that is spread across a number of industry verticals and across the globe. That brings on an interesting set of challenges. Multiple languages need to be supported for labeling and describing the same business concepts. The same business concepts can be defined and reported in several different ways in different locales. The vocabulary framework also needs to be flexible enough to allow for extensibility with the provision for defining new business concepts and relating them to known standardized concepts. The XBRL specification makes ingenious use of W3C XML Schemas along with XLink to accomplish this task. This article includes a description of how XBRL achieves this flexibility by creating an information model that utilizes XML Schema and XLink to define syntactic constraints for the reporting language, and semantics in terms of relationships between business concepts and associated metadata. The content of the article is based on the latest public draft of the XBRL 2.1 specification.

The XBRL Information Model

The XBRL information model presents us with two components that we can consider: an XBRL instance and its associated taxonomy set. While the XBRL instance contains the concrete facts (e.g., the value of the business concept "sales_per_share") being reported, the taxonomy contains descriptions of the business concepts (e.g., the description of the business concept "sales_per_share", what its syntax is, how it is calculated, etc.) that are being reported. Figure 1 is an illustration of the information model, and a description of this model follows.

The XBRL instance

An XBRL instance encapsulates business facts through a flexible information model. The most basic components of an XBRL instance are facts, which are pieces of information being reported through the instance. An example of a fact could be "sales in the most recent quarter." A simple fact, which is a piece of information with no structural complexity (for instance, the retail price of a product, a numeric value), is referred to as an item. Related facts that need to be considered together are grouped together in more complex structures as tuples. A tuple can contain items and/or other tuples. Each item can be associated with a business concept, and these business concepts are described and defined in detail through a taxonomy set. The items also reference the context in which they are to be interpreted. The context describes such aspects as the time period to which the fact pertains and the applicable reporting scenario ("actual," "budgeted," "pro-forma," etc.) among other things.

The XBRL taxonomy

An XBRL taxonomy describes a vocabulary by defining the syntax for the terms of the vocabulary using XML Schema, and metadata such as descriptions of the terms and relationships between them using XLink. Each XBRL instance references the schema(s) for its taxonomy, and each schema can reference other schemas. The collection of all the schemas that are required to validate the syntax of an XBRL instance becomes part of that XBRL instance's Discoverable Taxonomy Set (DTS). The XBRL instance can also, optionally, reference XLink linkbases that provide additional information about the business concepts being reported in the instance. XLink is a standard for specifying hyperlinks that link resources. XBRL uses XLink as the mechanism for defining relationships between business concepts and additional metadata (for more information about the XLink standard, visit the W3C XLink page at www.w3.org/XML/Linking). In addition

AUTHOR BIO

Srinivas Pandrangi is a lead architect for Ipedo, Inc., a leading vendor of XML information management solutions. He has extensive experience in standards development, having participated in W3C XQuery and Web Services Architecture Working Groups and in the IETF in the development of XML- and Internet security–related standards. At Ipedo, Srinivas leads the design and development of Ipedo's XQuery Query and Integration Engine.

to the XBRL instance, the schemas in the DTS can also contain references to linkbases. Together, all these schemas and linkbases, which describe all the business concepts being reported, form the DTS.

Taxonomy schemas

The taxonomy schemas define the syntactic constraints with which all XBRL instances for that taxonomy should comply. When creating a new taxonomy, you will either start from scratch, or, more likely, you will start from an applicable industry-standard taxonomy and extend it to suit your particular requirements. In either case, this involves importing some schemas and providing your own schema definitions in addition to arrive at the taxonomy schema appropriate for your organization's reporting needs.

The basic XBRL instance schema defines the element types for the XBRL container (the XBRL element), abstract elements for representing items and tuples, and the elements representing contexts. To create a new vertical- or organization-specific taxonomy from scratch, a schema with element definitions for items and tuples related to business concepts in that vertical is created. These element definitions are then placed in substitution groups with the abstract item type and tuple types as their heads. This schema can also include references to appropriate linkbases using W3C XML Schema's annotation facility.

The more likely scenario is that a taxonomy already exists for your industry but you need to perform a little bit of customization, such as adding a few more business concepts to relate them back to the standard concepts. In this case, you will create your taxonomy by importing the industry taxonomy schema and adding new schema components to reflect your custom business concepts. Some industry- or application-specific taxonomies are available at www.xbrl.org/resourcecenter/taxonomies.asp?sid=22.

Taxonomy linkbases

The XBRL specification describes five types of links in the taxonomy linkbases. These links can be organized broadly into three categories:

- Label links (labelLink)
- Reference links (referenceLink)
- Relation links (calculationLink, definitionLink, presentationLink)

Label and reference links relate business concepts to metadata. For instance, label links can associate concepts with text strings that can be used to label (or otherwise document) the concept in a report (e.g., the label "Revenues in most recent quarter" for the item revenueMRQ defined in the taxonomy). Multiple labels can be defined for a single business concept in different languages. The XBRL instance author can decide which labels to use for that particular instance.

Figure 2 illustrates a label link that defines a standard label for a business concept (using the label element), a locator for the business concept (using the loc element), and an arc linking the business concept to the label (using the labelArc element).

Similarly, reference links can associate references to authoritative literature in the business domain. The mechanism used is similar to the label links in that you define a reference link with a locator for the business concept, one or more references to documentation, and a referenceArc defining the association between the locator and the reference(s).

In contrast to label and reference links that relate business concepts to metadata, relation links relate business concepts to other business concepts. For example, calculation links define how a given concept figures in the calculation of another business concept. For example, the concept "profitAfterTax" is calculated from the concepts "profitBeforeTax" and "taxPaid" by subtracting one from the other.

```
profitAfterTax = weight(1) * profitBeforeTax + weight(-1)*taxPaid
```

The relationship between these three business concepts is captured in the calculationLink in Figure 3.

Definition links describe several types of relationships among business concepts, such as generalization-specialization relationships (e.g., "postalCode" is a generalization of "zipCode") among others. Presentation links, as the name implies, define the relationships between concepts from a presentation perspective (e.g., in the presentation of the report, a parent/child relationship should be shown between "sales" and "printerSales").

XBRL Processors

XBRL processors that are fully conformant not only need to perform syntactic validation of the instance documents according to the taxonomy schemas, they also have to perform semantic validation based on the taxonomy linkbases. When processing an instance document, they should make available to applications the metadata gleaned by applying the taxonomy to the instance. Think of the process as being similar to XML Schema validation generating a PSVI. XBRL validation should verify the instance against the taxonomy and make the metadata gained from such a process (for instance, labels detected for an item reported in the instance) available to the application using the processor. In addition to validation, the functionality included in XBRL tools can also include taxonomy editors, XBRL instance creation, XBRL repositories, query and reporting engines, and so on.

XBRL Applications

XBRL provides a business reporting framework. Business reports span a wide spectrum from regulatory compliance to business intelligence. XBRL applications can provide value at many different points in the reporting chain. By standardizing the way business information flows in this chain, you can open up a number of possibilities for using canned applications for composition, analysis, presentation, and any other application of that information. Figure 4

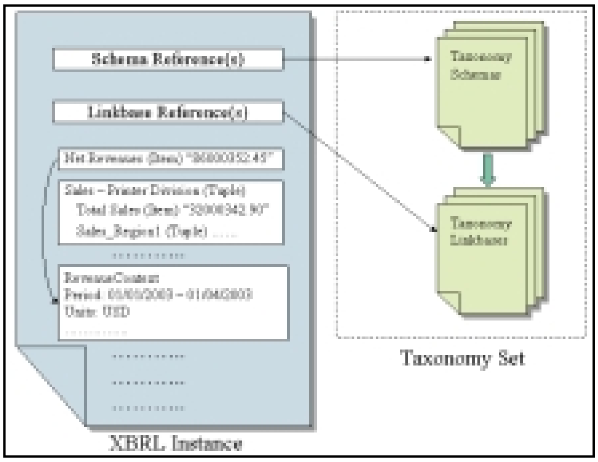
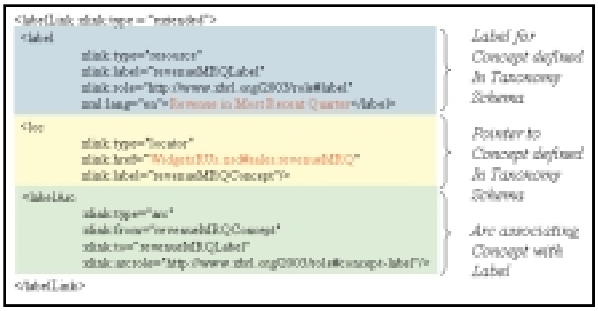

Schema Reference(s)
Linkbase Reference(s)
Net Revenues (Item) "80000352.45"
Sales – Printer Division (Tuple)
Total Sales (Item) "32000342.50"
Sales_Region1 (Tuple) .....
RevenueContext
Period: 01/01/2003 – 01/04/2003
Units: USD
Taxonomy Schemas
Taxonomy Linkbases
Taxonomy Set
XBRL Instance

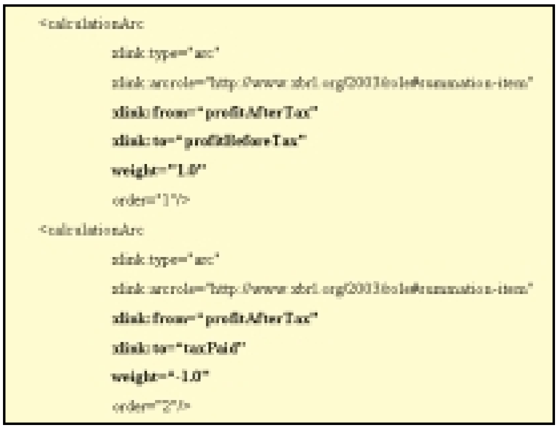
Figure 1 • The XBRL reporting model



```
<labelLink xlink:type = "extended">
<label
        xlink:type="resource"
        xlink:label="revenueMRQLabel"
        xlink:role="http://www.xbrl.org/2003/role#label"
        xml:lang="en">Revenue in Most Recent Quarter</label>
<loc
        xlink:type="locator"
        xlink:href="WidgetsRUs.xsd#sales.revenueMRQ"
        xlink:label="revenueMRQConcept"/>
<labelArc
        xlink:type="arc"
        xlink:from="revenueMRQConcept"
        xlink:to="revenueMRQLabel"
        xlink:arcrole="http://www.xbrl.org/2003/role#concept-label"/>
</labelLink>
```

Label for Concept defined In Taxonomy Schema
Pointer to Concept defined In Taxonomy Schema
Arc associating Concept with Label

Figure 2 • Label link associating a business concept with a label



```
<calculationArc
        xlink:type="arc"
        xlink:arcrole="http://www.xbrl.org/2003/role#summation-item"
        xlink:from="profitAfterTax"
        xlink:to="profitBeforeTax"
        weight="1.0"
        order="1"/>
<calculationArc
        xlink:type="arc"
        xlink:arcrole="http://www.xbrl.org/2003/role#summation-item"
        xlink:from="profitAfterTax"
        xlink:to="taxPaid"
        weight="-1.0"
        order="2"/>
```

Figure 3 • Relationships between concepts "profitBeforeTax," "taxPaid," and "profitAfterTax

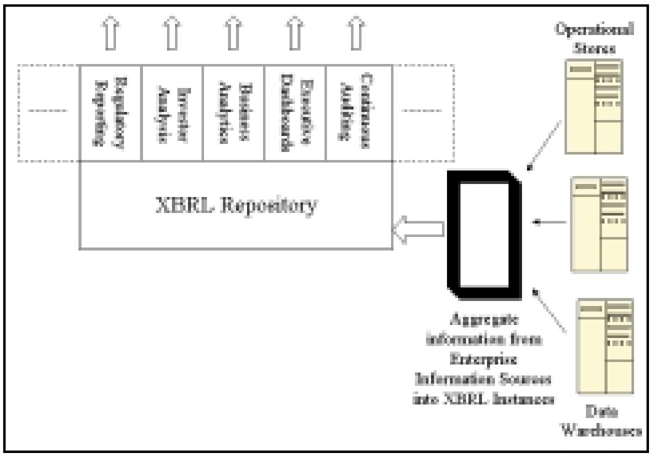


Figure 4 • Sample XBRL application

shows one possible point of view of the value XBRL can bring to business information reporting.

Most enterprises have their business information spread over multiple information systems segmented by product lines, geographical organization, profit centers, and so on. The sample application scenario shown in Figure 4 would involve aggregating information from these disparate sources into the XBRL repository, which can then serve many reporting applications, including regulatory reporting like SEC filings or tax filings, reporting for investors and analysts, reporting to management through executive dashboards, etc.

Wrap-Up

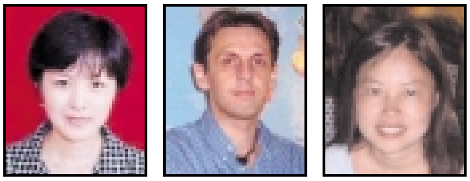
XBRL is a standard that benefits from wide industry participation and is beginning to gain acceptance as the way business reporting will be done. Using XML, it provides a truly extensible framework that enterprises can use to report critical business information. This framework provides advantages at each juncture in the business reporting chain from information preparation, through analysis, through consumption. After two revisions, public review and implementer feedback, this specification has acquired a degree of stability and is ready for prime time.

SRINIVAS@IPEDO.COM

XML-J ADVERTISERINDEX

ADVERTISER	URL	PHONE	PAGE
Altova	www.altova.com	978-816-1600	36
BEA dev2dev Days 2003	bea.com/dev2devdays2003	925-287-5156	13
Comdex	www.comdex.com		9
Ektron	www.ektron.com/xmlj	603-594-0249	35
IBM/Rational	ibm.com/rational/seamless		2
Mindreef	www.mindreef.com	603-465-2204	4
Edith Roman	www.edithroman.com	800-223-2194	17
SYS-CON Media	www.sys-con.com	888-303-5282	29
The 2003 Borland Conference	connect.borland.com/borcon03		7
Web Services Edge East 2003	www.sys-con.com	201-802-3069	27
WowGao, Inc.	www.wowgao.com	416-292-2364	21

General Conditions: The Publisher reserves the right to refuse any advertising not meeting the standards that are set to protect the high editorial quality of ***XML-Journal***. All advertising is subject to approval by the Publisher. The Publisher assumes no liability for any costs or damages incurred if for any reason the Publisher fails to publish an advertisement. In no event shall the Publisher be liable for any costs or damages in excess of the cost of the advertisement as a result of a mistake in the advertisement or for any other reason. The Advertiser is fully responsible for all financial liability and terms of the contract executed by the agents or agencies who are acting on behalf of the Advertiser. Conditions set in this document (except the rates) are subject to change by the Publisher without notice. No conditions other than those set forth in this "General Conditions Document" shall be binding upon the Publisher. Advertisers (and their agencies) are fully responsible for the content of their advertisements printed in ***XML-Journal***. Advertisements are to be printed at the discretion of the Publisher. This discretion includes the positioning of the advertisement, except for "preferred positions" described in the rate table. Cancellations and changes to advertisements must be made in writing before the closing date. "Publisher" in this "General Conditions Document" refers to SYS-CON Publications, Inc. This index is provided as an additional service to our readers. The publisher does not assume any liability for errors or omissions.



WRITTEN BY PHOEBE YI, PAUL FRETTER & SOPHIE ZHONG

Benefits of XML in Business

The preferred choice for data exchange and more

XML has become a hot topic over recent years due to its core qualities: extensibility, platform independence, and self-describing nature to name a few. Considered by many to solve a wide variety of e-problems, it has enjoyed an optimistic following that believes it to be the next IT revolution, comparable to HTML.

Despite the hype, the use of XML in the real world is still mainly limited to data transfer and system integration within large companies. In smaller business, and in the Internet application area (normally the domain of a more traditional RDB, ASP, HTML model), we have seen little use of XML.

This article discusses XML's strengths; why XML is a valuable tool, particularly with respect to data exchange; and also why perhaps XML is not commonly found within small businesses.

Should XML Be Used for Web Front-End Development?

HTML was indeed a revolution. It took us from awkward FTP nerds jotting down IP addresses and passwords to information seekers intuitively clicking from link to link. This took Internet demographics from a narrow subset of IT proficients and academics to a much more varied population, resulting in a corresponding variety of content.

HTML was successful because a problem existed that needed a solution (the Web was difficult to use), and the solution was simple, fairly robust, and cheap.

What issues does XML resolve? When should we abandon HTML in favor of XML and XSL for Web development?

XML cannot be directly compared with HTML – XML describes data, HTML presentation. Yet this differentiation often forms one of the most prevalent arguments in this area.

XML separates content from the presentation layer

XML is designed purely to represent data. It relies on partnership technologies such as XSL to format that data and present it to the user. This concept is valuable, allowing a clear division of responsibility between the two.

That said, separating content and layout is an idea that has been in practice for years. In a loose sense, most commercial Web sites have these two components separated already, with the layout represented in HTML code and content stored in back-end databases. ASP is typically used to merge the content with the presentation, passing the result to the user as HTML.

Some implementations of XML separate this structure further by storing content in XML, business logic in the ASP, and the layout in an XSL file for example. This approach might be beneficial for a large company where these divisions of responsibility are represented by their internal structure. For many smaller companies, it is an unnecessary step.

It remains possible to shield the front-end designer from much of the complexity of the data's origin (perhaps several tables joined together in some convoluted manner). This can be (and usually is) achieved by wrapping the query into a single stored procedure. Even the DB access and connection strings would typically be wrapped up in a simple include file.

In balance then, the notion of separation is valid but not an argument for the use of XML in its own right.

Given that XML is predominantly about the data, what advantages does it have over other Internet storage means? When is it the best choice?

A much-overlooked benefit of XML is that it easily represents hierarchical data within a single entity. An XML Schema can easily be designed to describe a car, for example, each car having one or more doors, each door having several subcomponents, and so on. This could also be represented in a relational database, but it requires several related tables to achieve the same thing, hampering any chance of portability.

Portability is a strong argument for the use of XML when the ability to share the core data is important. XML files can be shuttled between heterogeneous systems with relative ease due to the broad support XML now has.

Another often cited feature is XML's self-describing nature. Before this is considered an advantage to your project, you need to ask why it is important for a user to open the raw XML file and view its contents directly. Certainly end users would prefer a more palatable view. Developers would be ill-advised to view an XML file and assume it represents the full schema. Instead they are likely to consult the official DTD; even the server tasked with parsing this data finds the mechanism more cumbersome.

XML already enjoys widespread support. Unlike ASP, a Microsoft proprietary technology, XML is supported by nearly all platforms and systems, from Windows to different flavored Unix systems, Mac OS, and others such as WAP, Palm, and so on. It also supports Unicode, which can be used for most languages. Therefore, if we want a Web site that can

AUTHOR BIO

Phoebe Yi, PhD, MSc., is an IT consultant. She has worked in the IT industry for more than eight years, specializing in XML, e-commerce, and content management.

Paul Fretter is an IT manager in Midland, UK, with a software devlopment background.

Sophie Zhong has an MSc. in software techniques for image processing. She currently works as a software engineer in a firm specializing in warehouse management and control systems.

easily be switched between different platforms and published in different languages, XML might be a better option.

To summarize, an XML/XSL approach should be used instead of HTML/RDB for Web front-end development when:

- The Web site is really content rich and the content is not suitable for a relational database.
- The site needs to be independent of the operating system.
- The site needs to support multiple languages.
- The core data needs to be transferred from place to place, maybe even across platforms.
- The data is predominantly hierarchical in nature.

XML should perhaps not be considered when:

- Large amounts of data are involved.
- The underlying data is updated in real time (either via the Web or via other back-end systems).
- There's no real need for XML's portability.

Should XML Be Used as the Back-End Database?

An XML document is a collection of data; it has more in common with file formats such as fixed length and CSV than it does with relational databases. Just like any other file format, it is highly portable, but it also supports Unicode and is hierarchical. When, then, is XML suitable to be used as the back-end database?

XML is primarily considered useful for transient data storage, i.e., data that needs to get from A to B, but may soon cease to exist, being superceded by an updated file.

For heavy-duty storage and data manipulation, the power of a relational database is hard to beat. Relational databases rationalize the data into tables that are self-contained and serve multiple systems. Forcing the data together into one entity is not often desirable.

Only under special circumstances can you store data as an XML document without running into problems – small amounts of data, few users, and modest performance requirements. Certainly these conditions do not fit most production circumstances, including an e-commerce environment.

Due to their hierarchical nature, XML files and databases are more suitable to store data that is also hierarchical, i.e., each element has one parent element, but could have many child elements. Invoices and articles can both be described in this way, and they are also the sorts of data you might want to pass around.

Native XML databases, such as Tamino, have been introduced in recent years. They actually come from the historical hierarchical database arena. They're suitable for storing hierarchical data, but not for strict relational data.

To summarize, XML files and databases should be used instead of relational databases to store data *only* if:

- The data is document-centric rather than data-centric.
- The data model is more hierarchical than relational.
- There are no high requirements on security, indexing, multiple accessing, transaction, etc.
- The XML structure contains all the necessary data for this and any other system that might make use of it.

Should XML Be Used for Data Interchange?

XML, as a file format, is highly recommended for electronic information exchange between disparate systems, either within a company (system/application integration) or between companies (B2B integration).

Of course, Electronic Data Interchange between different systems (EDI) existed long before XML. Does XML bring any revolutionary aspect, or at least some improvement, to data exchange in the B2B world?

XML is not the silver bullet

Standardization is the key and should be an easy sell. Take any group of business units that exchange data that is intrinsically the same but export that data in their own native format, and the writing and maintaining of those systems is costly. Standardizing the format allows for fewer errors and lower cost throughout the process and opens the opportunity for "shrink-wrapped" solutions.

But is this benefit really brought to us by XML, or actually by the agreed-upon standard? The answer is probably the latter. The challenge is getting the relevant parties to agree; the actual mechanism used to represent that standard is almost irrelevant, and in most cases could just as easily be achieved by use of a CSV, or fixed length record file.

So, XML does not enable this to take place. It's an option after the important revolutionary bit has happened – the agreeing.

To many though, XML feels like a more intuitive solution. Data elements being encapsulated by their descriptors leaves no one in any doubt as to what they are, but is this a convincing argument? Should developers really be looking at the data without referencing an accompanying schema or file layout? Should they make assumptions about the data based on what they see in one extract? Probably not, but if XML is implemented correctly, the accompanying DTD can be a very powerful argument.

XML helps B2B by data integrity checking

With a good XML Schema it's possible for an XML file to be validated. This characteristic helps a great deal in the B2B data interchange. With traditional layouts this integrity checking is not possible (at least not to anywhere near the degree to which an XML file can). Companies exchanging data with flat files are spending huge amount of time to tidy up the data they receive and transmit.

Both XML Schemas and DTDs are flexible and powerful ways to describe what you expect the data to be like. You can specify that some elements are optional, some are mandatory, some need at least one element, but could have multiple. They describe the structure and the data types.

XML makes EDI accessible to a broader set of users

Electronic Data Interchange (EDI) has existed in e-commerce for more than 20 years. It is designed to exchange structured business documents, such as invoices and purchase orders, electronically between business partners. Although it has great potential to save money for each party, it has not been widely adopted until now. Forrester Research estimates that only about 5% of the companies that could profit from its use actually use EDI. The reasons are that it is quite expensive to set up, and more important, due to the lack of an industry standard, users have to have their own interpreting software to make it work with their proprietary systems. These considerations cause many companies, especially small and medium-sized enterprises, to avoid EDI solutions. So far, the use of EDI has been mainly reserved for larger companies.

With the arrival of XML, people are hoping this will change the current costly, proprietary situation of EDI, and make B2B data exchange easier, and thus reach its potential.

As a good data-transmitting format, XML could be used to replace the old EDI mechanism. XML supporters are working very hard to set up a standard for XML business documents schemas. One good example is ebXML, which is providing schemas for the common commercial transaction document types such as invoices. This standard will have great positive impact on EDI. Once every company in the industry uses the same invoice standard, no special interpreting software is needed, and adopting EDI will become much more straightforward. Most developers cannot build an EDI transaction, but they will be able to create an XML file easily.

XML is better for data exchange with unknown systems

If you know everything about two data-exchanging systems, it might be easier to use some other technology rather than XML for the data exchange. However, when the system is deployed over the Internet, you have less control over it, or when you know nothing about the recipient's system, XML could be a much better solution because use of XML technology is widespread.

The whole idea of B2B and Web services is to enable communication with other vendors, including known and unknown systems. So XML is the preferred choice for data exchange.

XML suits hierarchical data the best

Most of the data exchanged between systems, such as invoices, receipts, and orders, is hierarchical in form. An invoice has an address, which has address lines; it has one or more detail lines, each of which has subcomponents of description, quantity, value, VAT, and so on. XML is most suitable for representation of this kind of data. More than that, XML is extensible, which makes the schema flexible and easily maintained and extended. This is another reason why XML is preferred for e-commerce data interchange.

Other factors

There are certainly many other reasons why XML is becoming a de facto data exchange format for e-business: XML supports Unicode, XML is platform independent, XML is the target of active research and development, XML is an industry-agreed standard, and so on.

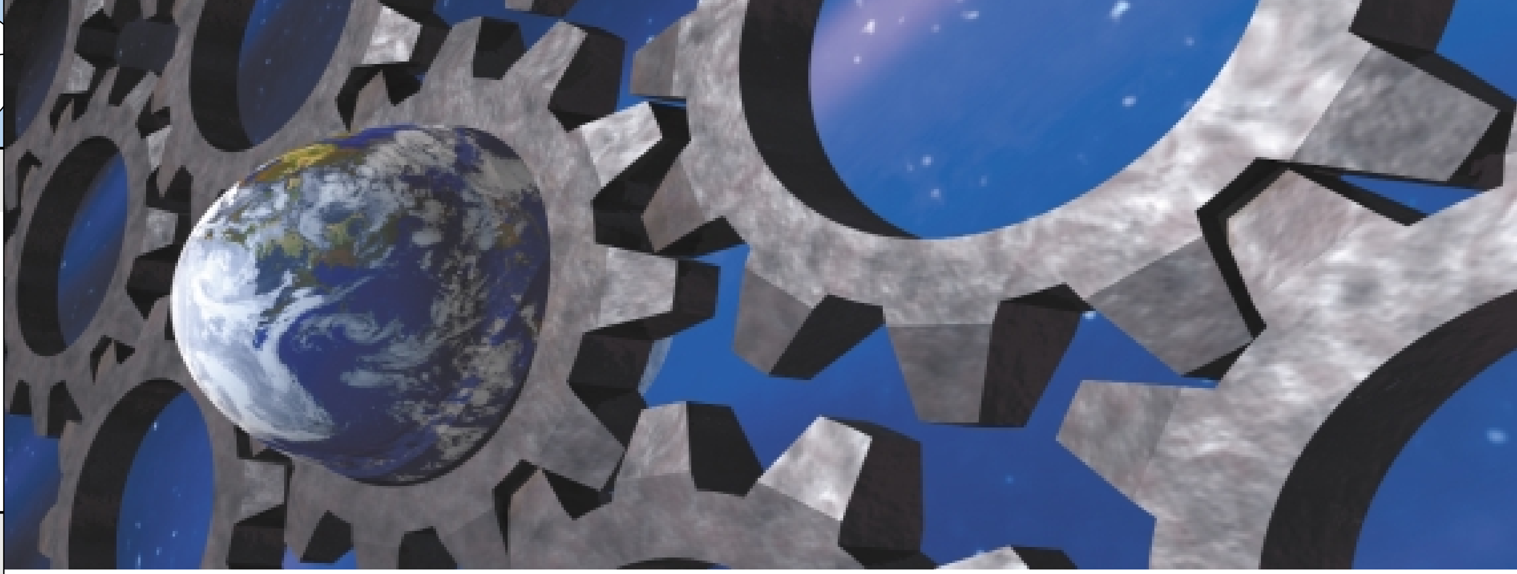
Summary

XML is used as a preferred data exchange format mainly because:

1. XML supports and encourages industry standards.
2. XML has strong validation functionality through DTDs and schemas. This functionality reduces the possibility of exchanging invalid data.
3. XML supports hierarchical data structures and infinitely repeated elements. Many e-commerce transactions, such as invoices and orders, are in that form.
4. XML is supported across platforms.
5. XML supports Unicode.

Perhaps one negative consideration is the inefficiency of the format; with each element being encapsulated by its descriptor for every record, the resulting file can be bloated by a significant factor. In an age where bandwidth is on the increase, this can often be overlooked in favor of its other advantages.

PHOEBEYI@HOTMAIL.COM
FRETTS4@HOTMAIL.COM
SZHONGUK@YAHOO.COM



Leveraging IT Assets

Integrating XML and relational data

WRITTEN BY
ANUPAM SINGH

XML is establishing itself as the standard for exchange of information across enterprises. However, the technology that allows enterprise-class applications to deal with XML processing is still not clearly formulated. This causes most enterprise customers to implement their own architecture. Additionally, their software implementations try to deal with the same set of basic XML processing questions in different layers of the enterprise, rather than as a whole.

Consider a fictitious company that has been using plain text documents or spreadsheets to report its financials. In the late 1990s, the company is asked by regulatory authorities to report more detailed financial information in a new format. The new data format is XML and the standard is XBRL (Extensible Business Reporting Language), an emerging XML-based open source specification for exchanging and processing financial information.

Despite reluctance due to the high-end costs of technology adoption, the CEO agrees to adopt the standard throughout the enterprise. An IT project is begun to foster the adoption of this particular format as the lingua franca for exchange of financial information between different parts of the company.

An IT team rolls up their sleeves and gets to work. Before the team can create the necessary IT infrastructure that allows existing applications to communicate with each other using the new data format, it stumbles upon the following questions:

- Should the IT department create a new application that talks to all other applications?
- Will the IT department purchase new and expensive software to make the new format pervasive?
- What happens to legacy systems – older and reliable systems still widely used in key business functions like billing?
- Given that financial reporting is such a key part of the enterprise, how does the company ensure that their software solutions based upon the new format are reliable?

The IT department decides to design an application to generate one XBRL balance sheet for the entire company. As it sets out to develop this application, it needs to consider how to utilize the various databases throughout the enterprise that house important financial data. The obvious candidate step is the storage of XML as a Binary Large Object (BLOB). However, the IT department must examine additional steps and explore the possibility of pushing most, if not all, work into a relational database.

The IT department decides to generate XBRL balance sheets from "raw" data for every division of the company. Raw data could be paper-based data, data in spreadsheets, or data in relational databases.

The XBRL balance sheets generated by each division are further consolidated, analyzed, and verified to form one single XBRL balance sheet for the company. It is then issued to regulatory authorities as a set of XBRL documents.

As any application development team would do, the IT department tries to identify the core requirements of this application. The main requirements are identified as XML consumption from many sources, XML storage in an efficient and reliable manner, searching XML to perform analysis and verification, and transforming XML to generate other XML formats (see Figure 1).

Using the XBRL balance sheet example, IT developers decide to prototype an XML processing engine inside the application as a simple Java program that uses existing (and free) software components. The prototype uses simple Java programs to map specific tables to specific XML formats; the file system as a simple XML storage mechanism; and the XSLT processor to run transformations over the XML. The architecture looks something like Figure 2.

Almost immediately after putting together the prototype, the IT development team realizes the basic inefficiencies in it. The mapping of SQL to XML seems to create a two-step processing of SQL result sets to XML. Additionally, there is the cost of developing transformation programs that create main memory representations of XML documents. And last but not least, the notion of the file system as a database is extremely antiquated.

The key lesson learned from the prototype is that XML processing cannot be solved as a single problem. Instead, there are different pieces to the XML processing puzzle (see Figure 3).

- ***Shredding/mapping of incoming XBRL:*** When the information is published in XML from the divisions, the application might want to map it back to relational data for some legacy application.
- ***Storage of incoming XBRL:*** If the application does *not* need to map it to its relational format, the application developer might

choose to store the whole XML document into the database.

- ***Indexing of incoming XBRL:*** Though not an immediate and obvious requirement of our application, it is important to create the right indexes to make transformation and searching of the XML as painless as possible.
- ***Query-stored XML:*** To run analysis queries, the application developer will need to run queries across the XBRL documents. For example, "Show me all of the divisions whose net assets are less than the net assets from the same quarter last year."
- ***Combine XML streams:*** In many cases, the application might have the requirement of "pulling" in some other information from a Web service, file system, or spreadsheet. In this case, the application needs to add logic to pull data from these sources and combine this information from "local" XML.
- ***Transform XML:*** Even though we are focused on XBRL as a sample scenario, the same application might need to support another set of XML formats. Instead of writing a new application for each new XML format, the application might need a declarative interface to transform one form of XML to another.

After identifying the above requirements, the IT department has a choice to build the whole application, buy separate software to fulfill these needs, or evaluate existing pieces of software that provide these functionalities. The IT department knows that the solution has to be robust in terms of availability, concurrency, recovery, and transactions. These characteristics bring the relational database to mind.

However, a natural question arises, "Why do we want to work the XML processing functionality into a relational database?" The answer is simple. Enterprise IT departments have invested heavily in relational databases because of scalability, high availability, reliability, and manageability. These requirements are the soul of any IT project. Special XML storage and customized XML processing engines could not have automatically guaranteed the above features without the years of research and development that have already gone into relational databases.

The relational database performs well for storage, indexing, and querying, but mostly with structured data. For semi-structured data like XML, what are the features or solutions that the IT department needs from the database? How does one gain the functionality without loosing the robustness?

The challenge for database vendors is to deal with these new requirements without sacrificing the primary characteristics of a database. Luckily, each XML requirement described above translates to a specific feature in the database. For now, let's continue to solve the problem from the point of view of an IT developer.

In the next few sections, the IT developers classify the solutions into "immediate" and "desired" solutions in the context of a relational database. Immediate solutions can be seen in many commercial databases in one form or the other. (It should be noted that we might use nonstandard SQL extensions for illustration purposes.) The desired solutions are longer-term solutions, which try to create interoperable solutions between W3C XML standards and the familiar SQL-oriented interface in relational databases.

The W3C has issued initial drafts of an XML Query Language called XQuery. Without going into the details of the language, XQuery provides excellent syntactical support for query and transformation of XML data. We will refer to this standard throughout our detailed analysis of database requirements.

Shred

Immediate solution

Shredding or mapping of an incoming XML document is the stage of mapping a hierarchical, tree-structured XML document into a relational table. The simplest mapping tool would be an embedded XPath processor in SQL. A set of XPath queries will identify the nodes that are to be mapped into a table. Furthermore, the XPath queries will use XML Schema information to map from an XML Schema type to the table.

The advantage of this solution is that the data mapping is taking place in the database and the IT developer does not maintain a set of XPath programs outside of the database. Of course, this assumes that the relational database has XPath extensions to the SQL language that look something like the following:

```
 insert into assets_table
select xmlextract("/group/ci:statements.balanceSheet/
  ci:Assets.currentAssets[@numericContext=’C0101’
  ]/text()", balancesheets)
from xmltable
```

It should be noted that the above example is not standard ANSI-SQL but an illustration of extending the SQL language with XPath queries. In the statement above, the SQL language is used to extract the currentAssets XML element from all company balance sheets stored in the balancesheets table and insert it into a relational column in assets_table.

Desired solution

A more declarative solution could be an XQuery-based engine that maps incoming XML documents into SQL tables. The most standard mapping available between XML and SQL is SQLX.

Using XQuery queries, the incoming document could be translated to one or more SQLX documents. These documents can be automatically mapped into the database. Of course, this assumes that the SQL interface in the relational database has been augmented by an XQuery interface and allows "automatic" insertion from SQLX documents into SQL tables.

Store

Immediate solution

Sometimes shredding might not be the best solution because the IT department is required to store the whole XML document to maintain the "original" copy of transactions. The best examples are financial data like mortgage applications, trades, analyst reports, and balance sheets.

Storage of the complete XML document is closely tied with the extent of support for a native XML data type in an RDBMS type. In the simplest solution, the XML document can be stored as a BLOB. But, the BLOB storage does not help the

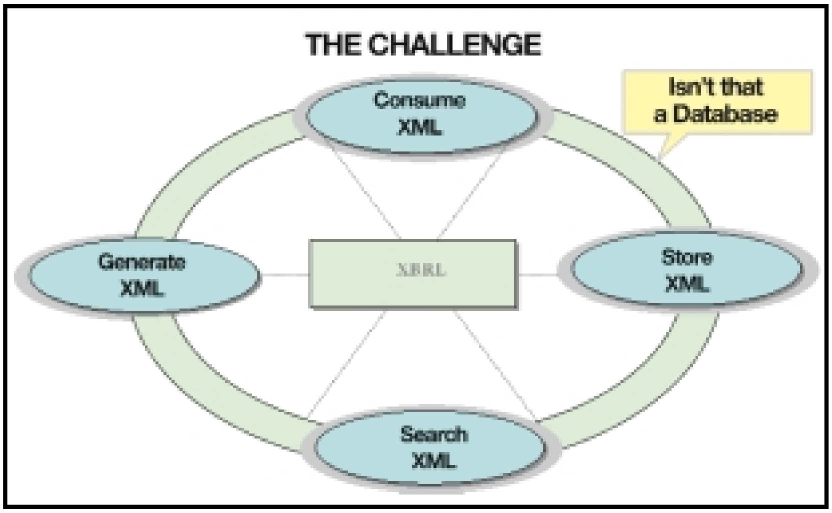


Figure 1 • The challenge

application developer. The storage needs to be complemented with an indexing scheme that allows searches on these documents to run faster. Some of these indexing schemes are described in greater detail in the indexing section.

Desired solution

The RDBMS should identify XML documents as separate, distinct types within the SQL system. As in any other type, the application developer should be able to validate the incoming data against user-defined constraints. In the case of XML, the XML should be parsed for validity against the user-defined XML Schema. For documents without well-defined XML Schema, the document should be at least valid XML. Of course, the insertion, deletion, and updates of XML documents should be transactionally consistent.

Another aspect is updates of parts of the XML documents. There should be a standard language (a la SQL) that allows the user to express updates to an XML document. In time, XQuery is anticipated to have updates added as part of the standard syntax.

Index

Immediate solution

The short-term solution for indexing has been extraction of the data into a base SQL-type column. Conventional relational indexes like B-Trees are then constructed on this base. Whenever XML Query is run, the index is used to qualify the XML document.

There are many problems with this short-term solution. Instead of qualifying a particular element within the document, the whole XML document is qualified. For example, in an XBRL document, if you wanted the total assets only when the total liabilities were greater than $1 million, this solution would only qualify the document without giving the exact part of the XML document that the application needed.

Another case of "non-native" indexing is when all information inside XML documents is shredded and indexed. Even though all values are now indexed using powerful SQL indexes, the fidelity of the XML document is lost, the contextual nature of information is all but destroyed, and the re-creation of the original XML document becomes impossible.

Desired solution

As can be seen, all immediate/short-term solutions are focused on reusing relational indexes that will require a very sophisticated two-way application-level mapping between XML and relational data. This mapping defeats the whole purpose of using the relational engine to reduce the code needed in the application layer.

Therefore, the desired solution is for the database to have native XML indexes. There are many types of native XML indexes:

- ***Path-based indexes:*** These are indexes that allow paths to be resolved without going through the whole document. Thus, the context-based queries can be immediately resolved using the path indexes.
- ***Value indexes:*** These are indexes that allow fast retrieval of XML fragments based on the value within the elements. Thus, all predicated queries can be immediately resolved using the value indexes.
- ***Link indexes:*** These are indexes that allow for faster navigation between fragments of XML data. In many cases, link indexes are useful when re-creating the XML fragment that the application needs.

When all three indexes are available, entire XPath queries can be resolved using the indexes. This avoids the cost of parsing or traversing the entire XML document for each and every XPath query. An example is shown below.

```
"/transaction[transactionData/tradeId >=10 and
  transactionData/tradeId <=20]//trade/tradeHeader"
```

Consider the above XPath query to illustrate the usage of native XML indexes. The initial path (in blue) "/transaction" is resolved using a path index. The predicates (in green) "transactionData/tradeId >=10" and "transactionData/tradeId >=10" are resolved using value indexes. The output fragment (in red) "trade//tradeHeader" is recreated using a link index.

Query

Immediate solution

Where there is data, there are queries. When storage of XML documents has been added to the database, the query language for the database should be modified for searching within these XML documents. Currently, many, if not all, databases support XPath queries embedded inside SQL queries.

As the IT developer knows, there are many XPath implementations available, so why would one choose the XPath processor inside the database? Apart from performance, is there any other gain in performing XPath queries inside the database?

The answer lies in SQL-XML interoperability. In many cases, the application developer needs to qualify XML data based on SQL data or vice versa. For example, consider an application that qualifies an employee's XML entry based on some relational information. We will go back to using our XPath extensions to SQL

```
select manager, ssn
from employeetab, t1
where
"/hr/employee[manager='John
Cox'][contacts/phone/mobile][name/@ssn = " + t1.ssn +
"]/name/@ssn" xmltest xmlcol
```

In the above query written using nonstandard XPath extensions to SQL, we use information stored in the relational database to qualify an XML document. There are many semantic problems with the above example – most of them caused by mixing of the relational data model with the hierarchical XML data model. That brings us to the desired solution.

Desired solution

Even if the XPath queries are embedded in SQL, they are not fully interoperable. Therefore, a fully functional solution to XML querying inside relational databases would:

- Provide complete duality between the SQL as a query language and XML as relational data, i.e., SQL column values can be used to qualify XML data and vice versa.
- Provide XQuery support for duality between XML Query Languages and relational/XML data stored in the RDBMS.
- Provide a high-performance engine that uses age-old, relational optimization techniques to use the indexes described above.

Combine

Immediate solution

In many cases, the basic assumption of data processing in a database implies local storage. But, in the XBRL example, the application developer has to fetch data from different divisions. This data could be transported to the databases as XML documents returned from a Web service, a plain text XML file in the file system, data available at a Web site, or plain relational data sent as a comma-separated file.

The short-term solution for the application developer is to import all the external data inside the database. But this could create multiple copies of the data as well as create an issue of synchronization between the data sources. That brings us to the desired solution.

Desired Solution

Even though we used the colloquial term of "combine" to express data from heterogeneous sources, the better technical term is "federation." Most relational databases have architectures to federate sources of relational data. This infrastructure needs to be extended to streams of XML information available through wrappers, Web services, and Web sites. On the API side, given that these sources are hierarchical in nature, queries across these sources are best expressed in a language like XQuery rather than SQL.

Transform

Immediate solution

Currently there are very few solutions for transformation of XML from one format to another. For example, what if an analyst wanted to use the XBRL document to create a report in RIML formats for a trader? What if the trader wanted to use this information to create an FpML (Financial Products Markup Language)–based trade? The current solutions lean very much toward additional clauses in SQL that let you generate different types of XML. However, these syntactical extensions to SQL are very restrictive as they provide a limited range of formats, are not declarative, and do not let users specify a series of nested XML transformations from one to another and so on.

```
select * from balancesheets for xml
```

In the above example, we transform relational data stored in the table balancesheets into XML data using a simple syntactical construct called "for xml." The results of this query are now generated in XML rather than the base types that application developers are accustomed to.

Desired solution

XQuery provides excellent syntactical support for generation of XML documents. A series of nested XQuery functions and queries could provide the infrastructure for transforming XML documents from one format to another.

Apart from the ability to query XML data (a la SQL), the return clause in XQuery can be used to create XML documents of different formats. The next logical step would be to give complete interoperability between XQuery across relational and XML data.

Using this detailed analysis, the IT department decides to use some immediate solutions around the database features described above. The focus of the IT project shifts from developing an XML infrastructure to using the robust XML infrastructure provided in the database. In some cases, the application developer might have to create a solution through another software piece or through homegrown solutions. But it's better to evaluate and use the infrastructure provided by existing relational database technology that has the credibility earned after years of research, development, and real-life customer usage.

Conclusion

As can be seen from each of the examples, there are many parts to the XML processing puzzle. Many of these parts fit quite elegantly with the roles that traditional databases have played in enterprise-class applications for many decades. For other parts of the puzzle, the relational databases will have to add new features and the relational database users will have a learning curve for these new features. Even though there are many obstacles, for most real-life XML usage, interoperability between relational and XML data cannot be ignored. Traditional characteristics like robustness and reliability cannot be sacrificed. In time, most relational databases will have features that will seamlessly bridge the gap across these different forms of data.

Author Bios

Anupam Singh is a staff software engineer with Sybase, Inc. His work has focused on querying and indexing semi-structured data within an RDBMS engine. Mr. Singh is also a founding member and project lead of Sybase's XML Query Engine Development Team, which works within the Database Server Technology Group at Sybase. Because of the Query Engine Team's work in XML storage, indexing and querying, Sybase Adaptive Server Enterprise is the prominent selection of companies looking for an efficient system to incorporate XML into their IT systems.

ANUPAM.SINGH@SYBASE.COM

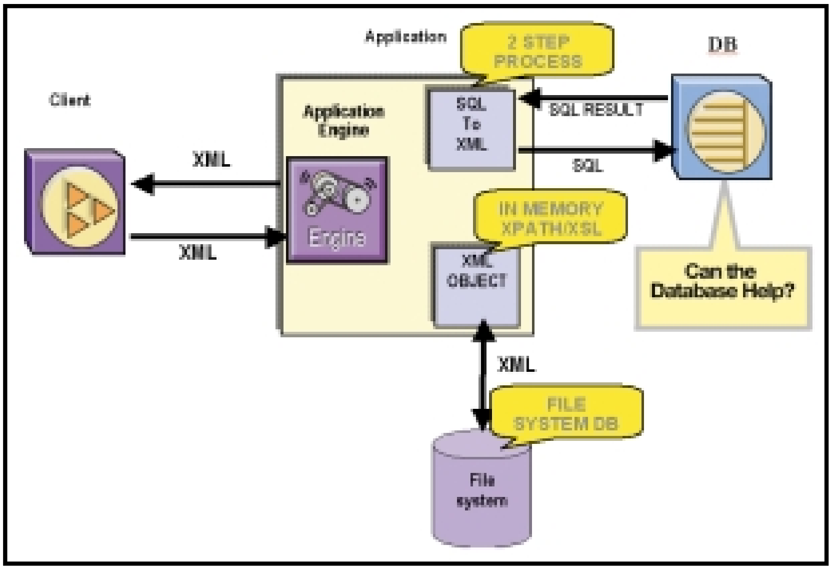


Figure 2 • Architecture

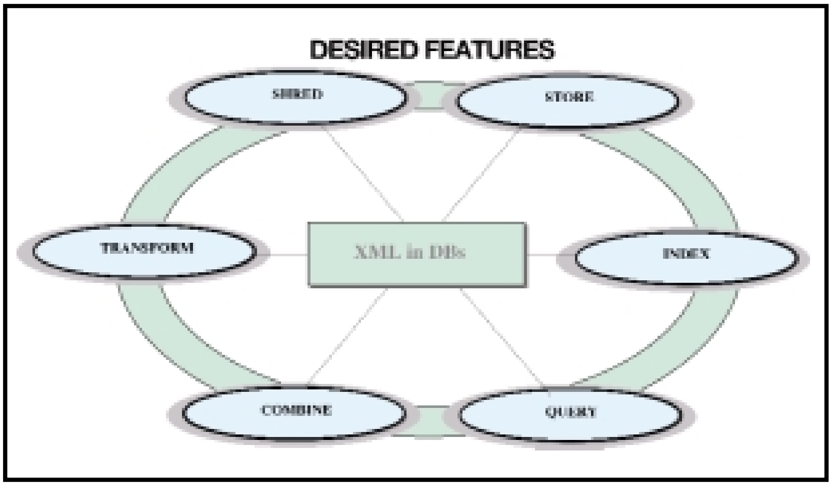
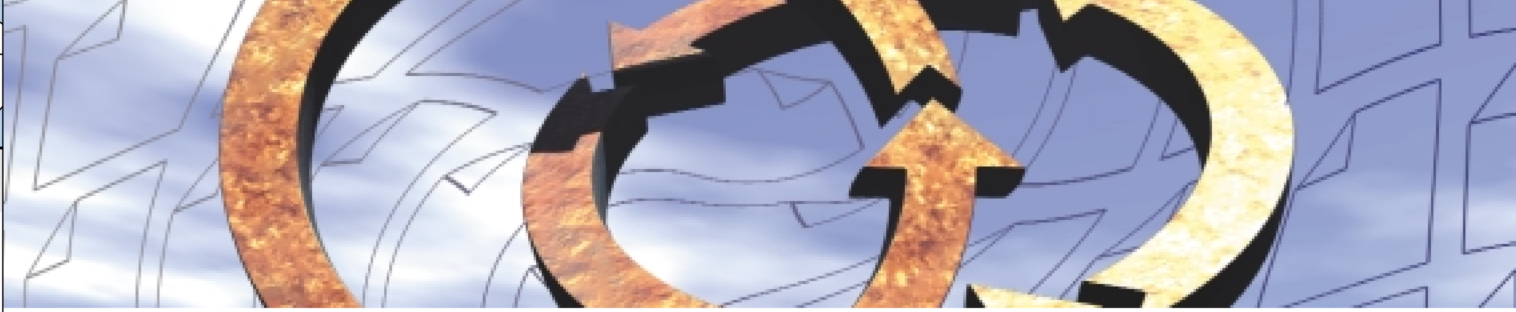


Figure 3 • Desired features



Document-Centric Software Development

WRITTEN BY
**STEVE BAILEY &
ALLAN JONES**

Is it time to evolve in the XML age?

At the heart of any software program lies data, and in the case of Web services and service-oriented solutions this is presented to the underlying software as XML documents. The representation and handling of these documents within the software is a major challenge with traditional development approaches, and often leads to complicated collections of software programs interfering with the clear flow of business information through an application. This article will show that a new approach is essential in order to simplify the complexity of today's software solutions while significantly reducing the cost of inevitable future change.

Changing Environments

Presentation, exchange of data (both in terms of messaging and logic), and the storage of that data are the core requirements for most applications. For most people, XML is first used in the presentation layer because this means that an application can produce many different types of output from the same incoming data using transformations. Web browsers, mobile devices, and PDF file readers can all have output generated from one incoming document, and this means that many applications now produce their output in XML format.

Over time, most people's use of XML spreads throughout the application's layers. It may be used to pass messages around between applications, technologies, or businesses. EAI projects use XML to pass data as messages between systems and middleware technologies. RSS feeds mean that XML data is used to syndicate content across multiple systems. Open standards for business transactions such as Origo, ebXML, RosettaNet, and so on mean that people are moving away from fixed EDI–type data into this new, extensible format for data exchange (e.g., The Accredited Standards Committee [ASCI] X12), which can halve costs.

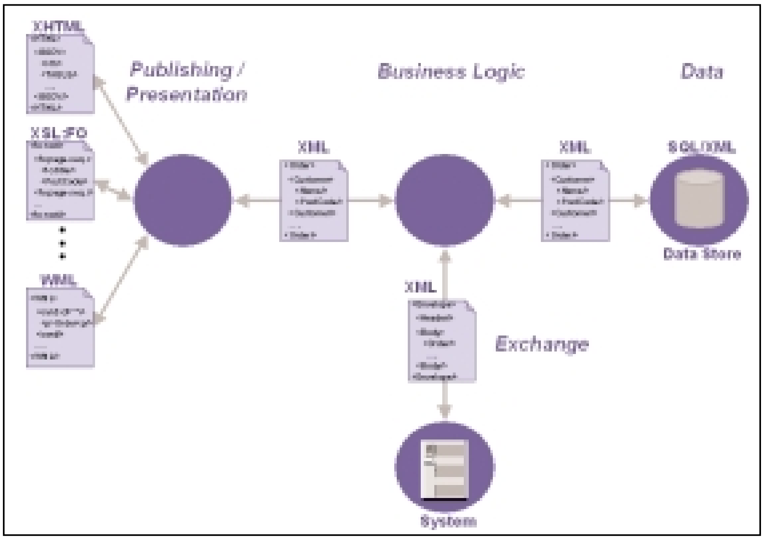


Figure 1 • The evolution of XML adoption throughout the application layers

This has, for some, ultimately led to XML being used as a storage medium for information. This may be as configuration files to aid in control of the underlying business logic, or persistent storage within native XML databases, or message tracking through messaging middleware. In some cases, the business logic of the application itself is contained within XML files (see Figure 1).

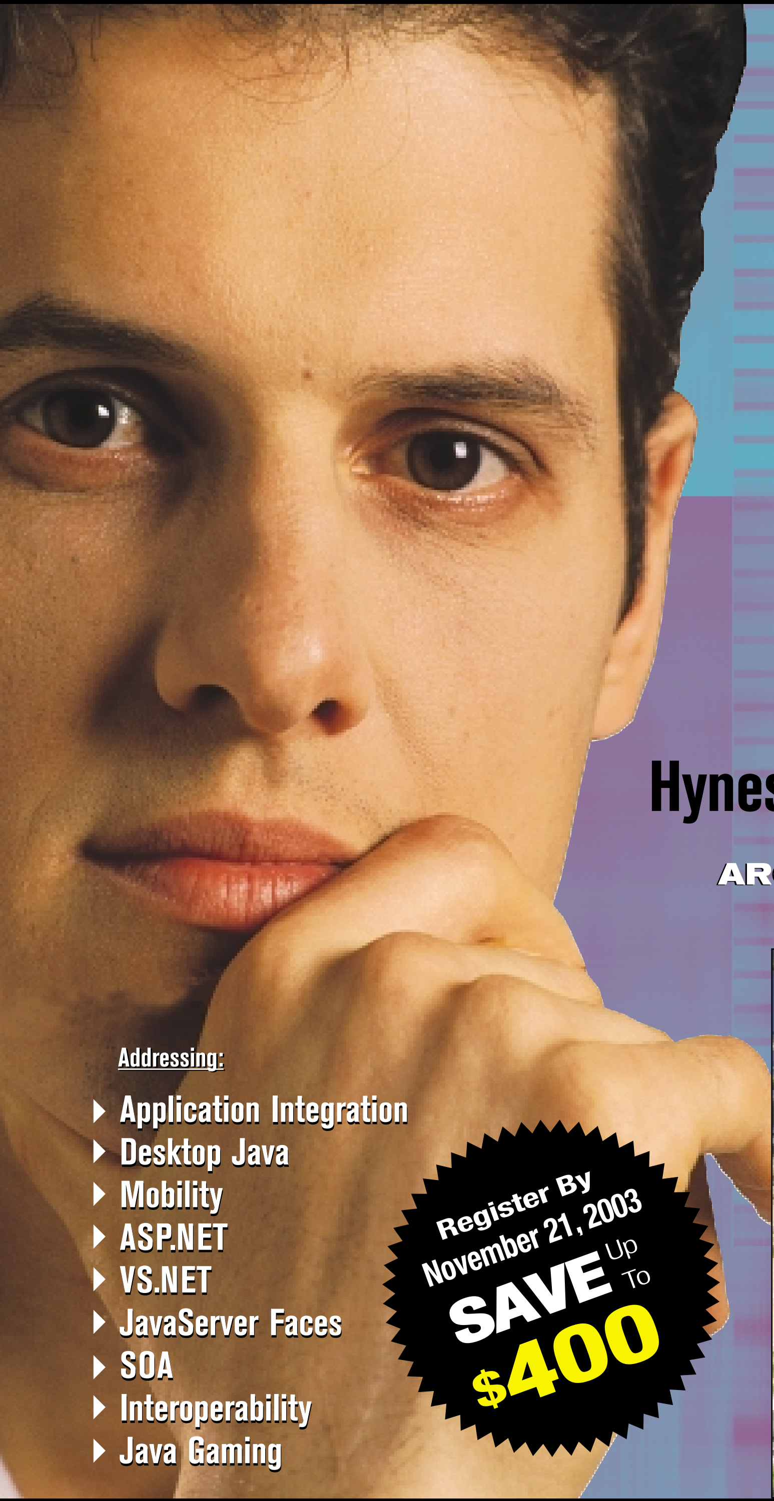
The inherent strengths of XML have ensured that the data is easier to understand because it can now contain the semantic information that describes it, and the extensibility also allows applications to respond more effectively to the changing needs of the business environment. XML has permeated through all of the application's layers, but because the growth of XML usage was evolutionary, many existing systems do not directly process XML data.

Challenges of Processing XML Documents

Traditionally, the main challenge when working with XML is that, at some point, the document must be converted into structures in code, but the business processes themselves are described declaratively in terms of the business document. The business analysts and architects work at the business document and business rules level, but the programmers need to work with the data within the software code in a different manner because the underlying code base does not understand the documents. This separation between the document and the underlying representation in the code means that there is often a need to provide a mapping of some sort between the two layers. This creates the classic disconnect between the business needs and the programmer's coded implementation (see Figure 2).

To make use of an incoming XML document in software code, we have to convert this document into code objects so that we can interrogate it. In the case of an XML document, there are two options available: represent the document as a document (using a document object model such as DOM, for example), or represent the data contained in the document as business objects in the code. This article will discuss the differences between these two approaches and will discuss how a

International Conference & Expo



Edge 2004 EAST

Development Technologies Exchange

February 24-26, 2004

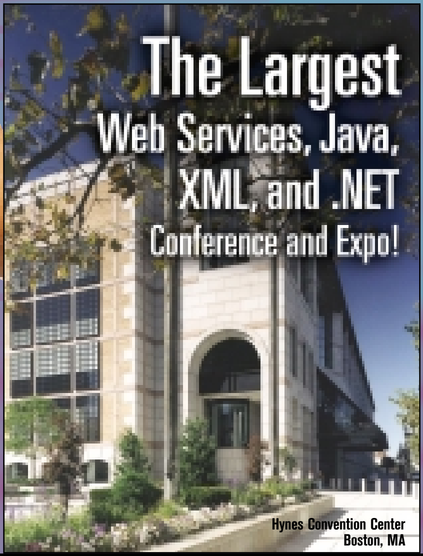
Hynes Convention Center, Boston, MA

ARCHITECTING JAVA, .NET, WEB SERVICES, OPEN SOURCE, AND XML

Addressing:

- Application Integration
- Desktop Java
- Mobility
- ASP.NET
- VS.NET
- JavaServer Faces
- SOA
- Interoperability
- Java Gaming

Register By November 21, 2003 SAVE Up To $400



The Largest Web Services, Java, XML, and .NET Conference and Expo!

Hynes Convention Center
Boston, MA

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

Over 200 participating companies will display and demonstrate over 500 developer products and solutions.

Over 3,000 Systems Integrators, System Architects, Developers, and Project Managers will attend the conference expo.

Over 100 of the latest sessions on training, certifications, seminars, case-studies, and panel discussions promise to deliver real world benefits, the industry pulse and proven strategies.

Full Day Tutorials Targeting
• Java • .NET • XML
• Web Services • Open Source

Conference program available online!
www.sys-con.com/edgeeast2004

Contact information: 201 802-3069 • events@sys-con.com

OWNED BY SYS-CON MEDIA

PRODUCED BY SYS-CON EVENTS

new document-oriented approach can help to deliver far greater flexibility and agility.

Mapping Data into Code Structures

Many automated binding solutions take an XML Schema and generate classes from this structure. This has the unfortunate side effect of tying the code tightly to the incoming XML structure – should the structure of this document or the organization of data within this document change in any way, the code would now not be able to manipulate the new version without some potentially considerable alterations. If the incoming data were invalid (in terms of structure or organization) the code may not be able to deal with this at all.

The common solution to this problem lies in the form of an intermediate binding layer. These bindings sit between the incoming data and the underlying representation of this data in the code and act as a translation layer between the two. This abstracts the incoming data away from the underlying code, gaining an obvious advantage in that the underlying processing is now no longer directly tied to the incoming data.

While this approach is a useful one, it does mean that we're now tied to the underlying data representation in the code. Should the data change (perhaps because the application needs to accept a different XML standard or structure), the underlying software model would need to be changed to deal with this. This might mean that processing is now duplicated across several classes because the data they're processing is different. Even if the data can be mapped, however, we still need to code a new binding layer for this new incoming data structure. As more partners start to use this service, the underlying code will become more and more difficult to maintain.

Reuse of code also becomes an issue in such cases. When the underlying logic is spread and duplicated across the application, it becomes very difficult to see areas where the code can be refactored, or where logic can be reused. This leads to problems in future development and maintenance of the application. This is particularly apparent in highly complex, time-critical applications, because a considerable amount of knowledge is required to understand where the business logic is located, what is actually happening within a given business process, and how that impacts the business documents flowing around.

The binding approach raises other problems because of the syntax of XML. Dealing with namespaces and complex, recursive XML structures can be difficult, and some binding solutions have difficulties in dealing with these. Binding solutions may also lose information contained within the structure of the incoming XML, but even if they don't there may still be problems when marshalling data types into the underlying code, or unmarshalling them back out again afterwards. Consider the differences between the sizes and formats of primitives (such as integers, decimals, arrays, and so on) across different languages and platforms, and mapping these to one another via XML structures may not be possible at all, or might require compromises with far-reaching consequences.

Separation of Logic from Data

Another major disadvantage of converting the document into code structures is that it also separates the logic from the incoming business document context. The program will not act on the data directly – instead, it will be converted into another representation, and at that point it becomes harder to see what has happened to it because it is no longer a human-readable document, but is instead represented as binary data within code. It is now also much harder to debug, because the information must be extracted from the code – it is no longer clearly visible.

As we can see, binding data into code can present a number of challenges. What is needed is a new approach in which evolution and interoperability are simple and easy. This new approach must combine visibility with agility, and must be able to deal with multiple versions of incoming data without the need to reimplement much of the existing business logic (see Figure 3).

Operating Directly on XML Documents

In the real world, documents are used to pass data around – forms, notes, memoranda, and e-mails are all used to convey information and data to the recipients. As these documents are passed around, they are often altered, manipulated, copied, or changed into another format. This "paper trail" of processing is easily visible, and the process can be adapted to adjust to new data and new information quickly and easily.

Bearing that in mind, it appears that keeping our data in the form of documents (or a DOM document object in the code itself) will solve many of our problems. First of all, abstracting the document into a different code structure isn't actually necessary – in fact, it may well be an unnecessary conversion stage. Instead, we can work directly on the document itself by using the open W3C XPath syntax to query the document so that we can identify or query content within that document,

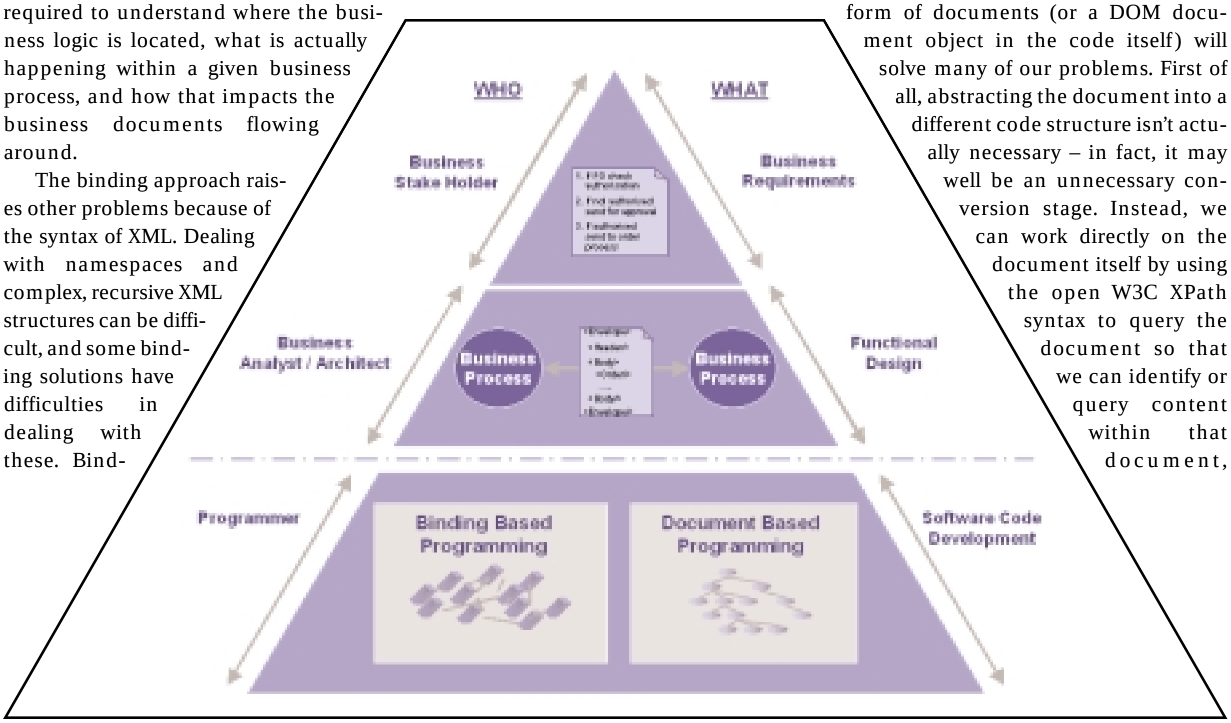
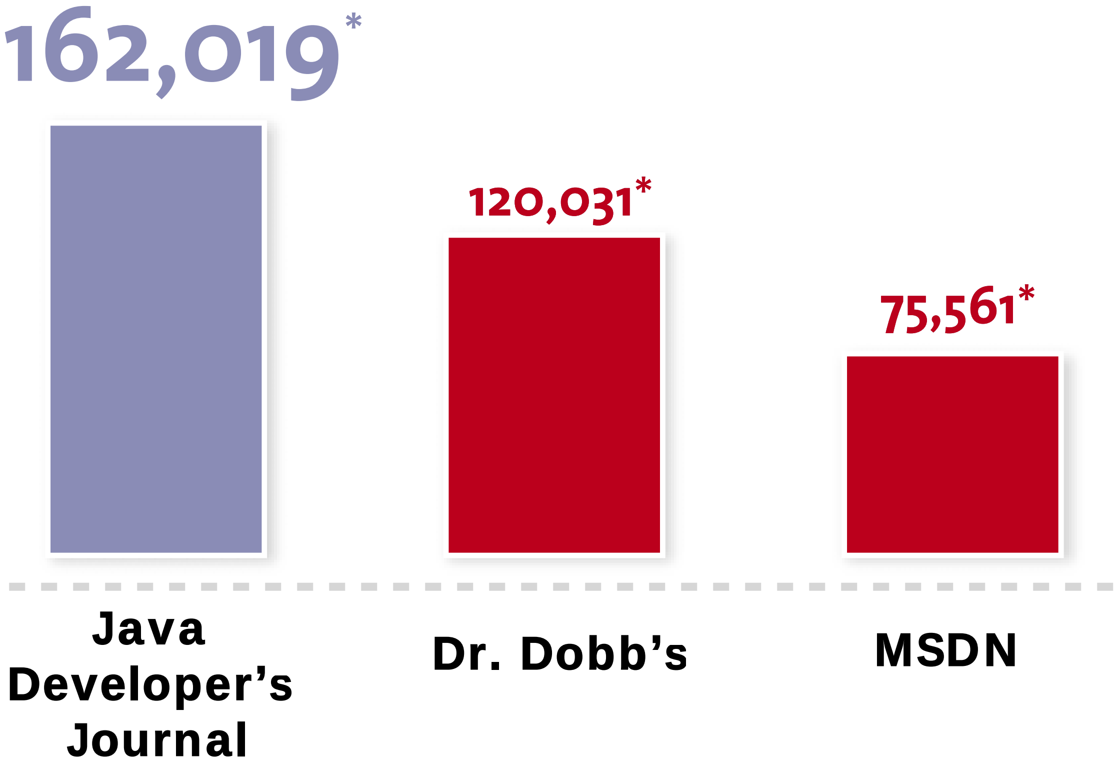


Figure 2 • The classic disconnect between the business needs and the programmer's coded implementation

#1 Circulation in the World!

JDJ is the highest circulation i-Technology magazine in the world!



*JUNE 2003 BPA AUDIT STATEMENTS *All circulation numbers are publishers' most current own data, six-month average circulation through June 2003.

SYS-CON MEDIA

Miles Silverman
VP Marketing

Carmen Gonzalez
Senior VP Marketing

Contact | Carmen Gonzalez: carmen@sys-con.com or 201 802.3021

JAVA DEVELOPER'S JOURNAL

HOME
Enterprise Solutions
Content Management
Data Management
XML Labs

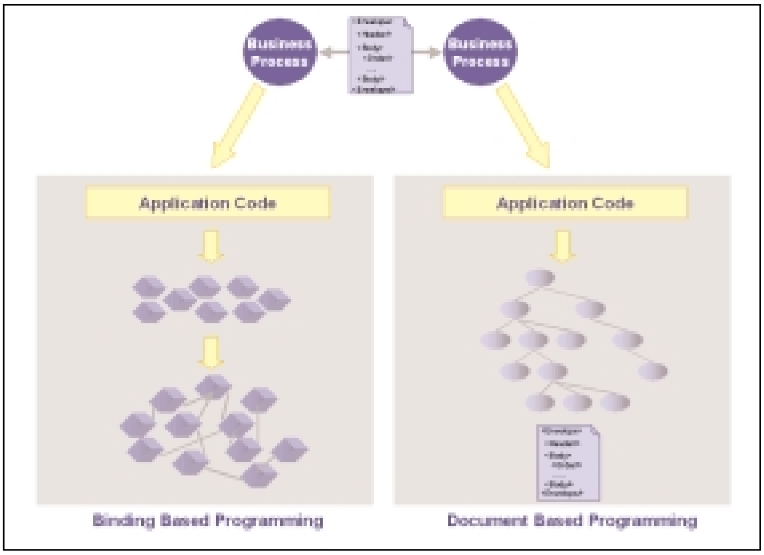


Figure 3 • Is it necessary to turn business documents into coded objects?

and we can then add, remove, or change elements of the document based on the results of these queries using our software language of choice. The main advantage of this approach is that we always have a document available for inspection, making debugging the Web service or application much easier. It also allows for a clear audit trail, which may well be necessary in certain situations.

Another advantage is that it parallels the real-world model of the process in such a way that the business processes are more visible to business stakeholders. More readily understood representations of business processes ensure that future maintenance and development are simplified because the underlying processes are easily visible. The logic acts directly upon the document data, and because there is no mapping (or, at worst, a one-to-one mapping in the case of a DOM object) of the data into collections of code structures, the underlying logic is much easier to see.

This approach also ensures true standards compliance. XML and XPath are open standards, freely available and widely used. Operating directly on XML documents ensures compliance at every stage of development. This approach ensures portability because the application's logic is no longer dependent on the software code language. Passing XML documents around also ensures that interoperability is no longer an issue – if the services we use can accept XML, there is no interoperability problem. All we need to do is to send the document to a different location.

Working on the document directly also ensures that, at every stage, we have access to the information. This mirrors the real-life situation in which a document is passed from person to person. This ensures that debugging the application will be far easier than trying to extract information contained in data models in the underlying code. Each process does something to the document, so by looking at the documents before and after each step we can see exactly what has or has not been done to that document.

Document-Driven Processing

In terms of code, this approach suggests that at each point we need to think in terms of the function of the code as it relates to the manipulation of a document: In other words, what will happen to the document as it passes through this code? The easiest way to demonstrate this is by thinking in terms of a real-life purchase order scenario for a retailer organization that uses external suppliers for product fulfillment. First, the incoming purchase order document is checked over to make sure it has the correct authorization and that the expenditure is within given ranges – validation. This may be countersigned or stamped to say that it's been checked before it's passed to someone else to handle the next stage of the process (i.e., the ordering process). This new person will fill out an order form for the company selling the product using the content of the first document to populate the relevant fields to the new document. This document will then be given to the company that supplies the goods, and they will in turn return a new document in the form of a receipt after it's been processed.

When modeling out a document-based application, it makes business sense to think of these processes being performed by someone or something because this mirrors the real-world process in a way that's clear and easy to understand for both business and development users. Each individual step can be broken down into a series of entities performing business tasks on the document, and the term most often used to describe this entity is an "agent."

Checking the Content

Once the information has reached the agent, it must be checked to ensure that it can be used appropriately by the agent for the business task that it provides or delivers. With XML documents, there are several ways that this content can be checked. The first (and most obvious) of these is by schema validation. This can ensure that the incoming document is syntactically valid. This means that incoming data is processed on a best-effort basis, and that the incoming document can be rejected if it does not contain the expected data.

If the application reacts to the data in the incoming document, then the application can be extended iteratively by adding processing that reacts to the product of the previous results. The application might initially begin by validating the content in some way, and returning a message to let the user know if the data is valid or not. To extend this processing we now only need to react to this new piece of data. This ensures that each stage of processing is clearly separated and can be added iteratively to make development and maintenance much simpler, as well as allowing for easy maintenance and extensions to functionality in the future.

Document Manipulation

Working directly with documents means that operations must be defined to make alterations to them. Ideally, these processing actions will mirror the actions that a human operative would make when processing a document manually – store it, read some information from it, write to it, send it somewhere else, transform it into a new document for another business service to use, and so on. This makes it easier to see what is happening throughout the process, and debugging is simplified by the ability to save the document out at each processing stage to ensure that the processing is performed correctly. This approach also allows each agent to be tested and debugged individually, and for the resultant application to be constructed in a modular and iterative manner promoting reuse and improving maintainability.

Of course, these maintainability issues also apply to the incoming data. What happens if the incoming data structures are altered, perhaps by the need to incorporate a new standard for a business document into the application? By making use of XML documents we can ensure that the processing is flexible, easily extensible, and above all else maintainable. By handling XML documents directly and acting on their content, converting an incoming document into the expected structure for processing is as simple as adding a new rule to transform the incoming data when it matches the new structure. Without this new rule, the new incoming document will simply not be processed by the application, making the application more robust and resilient.

Conclusion

We've looked at using XML documents directly when developing business logic. This document-oriented development approach is ideally suited to Web service and service-oriented solutions, and also provides massive benefits in maintainability. Making use of documents in software echoes the human, document-based world in which most business processes exist, and by echoing this we can leverage the understanding that we already have of these processes. This will simplify both the construction of new applications and the maintenance of existing applications constructed using this model.

Business stakeholders understand business documents but often don't understand object models, UML, relational database models, and so on. Business and IT development can now converse based on the same business models, and this will deliver far greater collaborative benefits and effective communication between them.

As the adoption of XML Web services and service-oriented architecture solutions accelerates, any edge we can gain when constructing new applications or altering existing applications can translate rapidly into commercial gain. Any improvements we can make to the speed of assembly and construction of new applications, or alterations to existing applications in a response to the changing world around us, can impact this, and making the most effective use of data through documents becomes an essential evolutionary step.

By making the information available to both humans and machines simultaneously, we enable the rapid evolution of business processes in a world where Darwin's evolutionary principles apply. Can we really afford to be without the benefits of reacting dynamically to document content within applications?

Native XML applications will pay long-term dividends as the use of XML becomes pervasive. If companies are eager to evolve and to react quickly in response to the changing business world around them, they must use new approaches that allow this evolution to occur in a natural, progressive, manner that echoes the evolution of the business world.

Resources

To see an example of this approach visit our Web Services Developer Centre at www.hyfinity.net and click through "Intelligent Web Services in Minutes."

- *Document Object Model (DOM)*: www.w3.org/DOM/
- *Java Binding (JAXB)*: http://java.sun.com/xml/jaxb/
- *XPath Specification*: www.w3.org/TR/xpath

AUTHOR BIOS

Steve Bailey is cofounder and chief e-business architect at hyfinity. Before hyfinity, Steve worked for Software AG for 12 years, where he focused on delivering XML-based e-business solutions. Steve's focus was on systems integration projects, specializing in enterprise-scale e-business systems.

Allan Jones is a senior systems engineer within R&D at hyfinity. His specialization is W3C XML architecture domains and expert systems. Before hyfinity, Allan worked for Wrox Publishing as a technologist focused on XML, .NET, and J2EE architectures.

STEVE.BAILEY@HYFINITY.COM
ALLAN.JONES@HYFINITY.COM

WRITTEN BY **DARE OBASANJO**

Can One Size Fit All?

Exploring the possibility of one API for XML processing

Traditionally, APIs for processing XML have been categorized according to whether they're designed for processing entire XML documents loaded in memory, such as the W3C DOM, or for processing XML in a streaming, forward-only fashion, such as SAX. However, these divisions do not fully represent the various classes of APIs for processing XML.

In a recent article entitled "A Survey of APIs and Techniques for Processing XML," I describe six primary methodologies for processing XML.

1. Push-model APIs such as SAX
2. Pull-model APIs such as the .NET Framework's XmlReader class
3. Tree-model APIs such as DOM
4. Cursor-model APIs such as the .NET Framework's XPathNavigator class
5. Object–XML mapping technologies such as the .NET Framework's XmlSerializer class
6. XML-specific languages such as XQuery

This list highlights that the range of considerations when choosing an API or technique for processing XML extends beyond forward-only access over XML streams versus random access over XML documents stored in memory. Other considerations include whether the XML being processed is used to represent semi-structured documents versus rigidly structured data, whether the XML is considered to be strongly or weakly typed, and ease of use of the API.

The purpose of this article is to explore whether a single API could be designed that satisfies the various needs that warrant the existence of six different categories of technologies for processing XML.

Rigidly Structured Data and Semi-Structured Documents

One of the main reasons for XML's rise to prominence as the lingua franca for information interchange is that, unlike prior data interchange formats, XML can easily represent both rigidly structured tabular data (e.g., relational data or serialized objects) and semi-structured data (e.g., office documents). However, applications that utilize XML typically produce or consume XML that is primarily either rigidly structured data or semi-structured documents. Several defining characteristics distinguish both XML usage patterns.

Software applications are usually the primary consumers of XML documents that represent rigidly structured data. Such XML documents usually have content that is meant primarily for machine processing that is labeled with markup targeted for human consumption. XML configuration files, log files, and relational database dumps are examples of rigidly structured data that are meant primarily for machine processing. The markup in these documents is mainly of use to human readers who are either editing or debugging an XML application. Such XML documents typically comprise elements and attributes where only the deepest subelements – the leaf nodes – contain character data. Although XML considers the order of elements to be significant, the order of sibling elements in such documents is often not important to the semantics of the document (e.g., the order of the rows in a database dump is often not significant). The following is an example of an XML document representing rigidly structured data:

```
<items>
   <compact-disc>
     <price>16.95</price>
     <artist>Nelly</artist>
     <title>Nellyville</title>
   </compact-disc>
   <compact-disc>
     <price>17.55</price>
      <artist>Baby D</artist>
      <title>Lil Chopper Toy</title>
   </compact-disc>
</items>
```

Human readers are usually the primary consumers of semi-structured XML documents. In this case, the XML markup assists software applications to process the data. Web pages and business documents are examples of semi-structured documents that are meant primarily for human consumption. Their markup is mainly of use to programs that are processing or displaying the information within the documents. Such XML documents typically comprise elements and attributes where character data appears alongside subelements, and character data is not confined to the leaf nodes. The interleaving of character data with subelements is often described as mixed content. The order of elements in semi-structured documents is often significant (e.g., the order of chapter elements within a book element matters). Features such as entities, processing instructions, and comments are more likely to be used in semi-structured documents to aid authors and readers of the XML. The following is an example of a typical semi-structured XML document:

```
 <p
xmlns="http://www.w3.org/1999/xhtml/">
If customer is not available at the
address then attempt to
   leave package at one of the
```

Author Bio

Dare Obasanjo is a program manager on Microsoft's WebData team, and is responsible for core components within the System.Xml and System.Data namespace of the .NET Framework.

```
 following locations listed in
 order of
which should be attempted first
   <ol>
    <li>Next Door</li>
    <li>Front Desk</li>
    <li>On Doorstep</li>
   </ol>
   <b>Note</b> Remember to leave
     a note detailing where to
     pick up the package.
 </p>
```

In reality, many uses of XML fall somewhere in the middle, where there is an island of rigid structure within a semi-structured document or an area with "open content" in a rigidly structured document. XML easily accommodates these scenarios because the choice of which model of document to exchange is not mutually exclusive.

The Relationship Between Data Typing and XML Usage Patterns

The different XML usage patterns, semi-structured documents and rigidly structured data, typically have different requirements when it comes to accessing the content within XML documents as typed data.

Consumers of such XML documents that contain rigidly structured data often want to consume the documents as strongly typed XML. Specifically, such applications tend to map the elements, attributes, and character data within the XML document to programming language primitives and data structures so that they can better perform operations on them. This mapping is usually done using either an XML Schema or a mapping language. Listing 1 is an example of a W3C XML Schema document that describes the strongly typed view of the XML document.

Consumers of semi-structured XML documents typically want to consume the documents as weakly typed or untyped content presented as an XML data model. In such cases XML APIs that emphasize an XML-centric data model, such as DOM and SAX, are used to process the document. An XML-centric view of such semi-structured documents is preferable to an object-centric view because such documents typically use features peculiar to XML, such as mixed content, processing instructions, and the order of occurrence of elements within the document is significant.

Choosing a Data Model

The first question to ask is whether it's feasible for an API that is meant to process both rigidly structured XML data and semi-structured XML documents to be based on the same XML data model. XML documents containing rigidly structured data typically consist of XML elements and attributes with character data. Semi-structured XML documents also typically consist of elements and attributes with character data but also utilize other aspects of XML, such as processing instructions, comments, CDATA sections, and entities. However, given that CDATA sections are just a syntactic shortcut around encoding certain types of text and entities are just placeholders for elements or character data, they actually do not have to be represented in the data model. XML documents that contain rigidly structured data as well as semi-structured documents may utilize XML namespaces, which should also be accounted for in the data model.

There are already a number of existing abstractions for XML documents, including the XML DOM, the XML infoset, and the XPath 1.0 data model. Of these the XPath 1.0 data model best meets the requirements set forth in the previous paragraph. The XPath 1.0 data model provides a simple and consistent view of an XML document that is loosely coupled to the text-based nature of the XML 1.0 recommendation. The fact that the XPath 1.0 data model ignores certain aspects of the XML 1.0 recommendation makes it easier to map other domain models to the XPath 1.0 data model. For instance, information such as which quotation characters are used in an attribute or whether character data was directly entered or represented as an entity is not directly exposed in the XPath 1.0 data model. Thus, when exposing a relational database, file system, or in-memory object graph as XML it is easier to do so if the API doesn't require you to expose information that is only pertinent to XML text documents. Examples of such "virtual XML" views of relational and object-oriented data based on the XPath 1.0 data model are Microsoft's SQLXML and the ObjectXPathNavigator on MSDN, respectively.

There is one limitation of the XPath 1.0 data model that makes it less than ideal for use in representing rigidly structured data within XML documents: the lack of support for strong typing. The XQuery and XPath 2.0 data model is the next iteration of the XPath 1.0 data model. The data model is the XPath 1.0 data model with the addition that the data types associated with elements and attributes can be identified using an expanded name (i.e., the xs:QName type). The ability to identify the data type of nodes via the namespace URI and a local name (i.e., an expanded name) provides a loosely coupled mechanism for supporting W3C XML Schema data types and potentially any other type system in which individual types can be identified by an expanded name.

Thus, we have arrived at the XQuery/XPath 2.0 data model as the data model suitable for an API that is meant for processing both rigidly structured XML data and semi-structured XML documents.

A Single Model for Forward-Only Access to XML

In "A Survey of APIs and Techniques for Processing XML," I pointed out that pull-model APIs can handle streaming, forward-only access to XML, as well as push-model APIs. Thus, both categories of XML APIs can be collapsed into a single one: streaming, forward-only access to XML.

Listing 2 is an example of using the pull-based XmlReader class in the .NET Framework to obtain the artist name and title of the first compact disc in an items element (Listings 2–7 can be found at www.sys-con/xml/sourcec.cfm).

In the same article I pointed out that on close inspection a pull-model parser is a cursor that happens to be restricted to being able only to move forward and not back. Listing 3 is an example that utilizes the .NET Framework's XPathNavigator class to obtain the artist name and title of the first compact disc in an items element

From the examples in Listings 2 and 3 it doesn't seem that there is much difference between accessing the contents of an XML document using a cursor-model API and using a pull-based API. But looks can be deceiving. In simple cases there is not much difference between the two, but it does get slightly more difficult in complex cases.

The primary programming idiom when using a pull-based parser is to create a loop that continually reads from the XML document until the end of the document is reached and to act solely upon items of interest as they are seen. The same effect can be achieved using a traditional cursor-model API as shown in Listing 4.

The output of both DumpTree() methods when passed one of the XML fragments from earlier in the article is shown in Listing 5.

From Listing 5 it can be seen that a cursor-model API can be used to walk all the nodes in an XML document in document order in much the same way as a push- or pull-based API. However, in the example above, the code using the .NET Framework's cursor-based XPathNavigator class is more cumbersome than equivalent code using the XmlReader class. This has less to do with the nature of cursor-based APIs and more to do with the fact that the XPathNavigator class does not have helper methods that make it friendly towards traversing nodes in document order.

To make the .NET Framework's XPathNavigator more suitable as an API for pull-based processing of XML, you could introduce a base class called ForwardOnlyPathNavigator, which would possess only the forward-only access methods from the XPathNavigator and possibly an additional MoveToNextInDocumentOrder() method that would make it equivalent to most pull-based APIs. This would then unify the streaming, forward-only access model for XML documents with the cursor model.

A Single Model for Random Access to XML

In "A Survey of APIs and Techniques for Processing XML," I pointed out that cursor-model APIs could be used to traverse in-memory XML documents just as well as tree-model APIs. Cursor-model APIs have an added advantage over tree-model APIs in that an XML cursor need not require the heavyweight interface of a traditional tree-model API where every significant token in the underlying XML must map to an object.

Listing 6 is an example of using the XmlDocument class in the .NET Framework to obtain the artist name and title of the first compact disc in an items element.

Listing 6 shows a common idiom when accessing XML through a tree-model API; the nodes of interest are requested through a query mechanism, then processed as needed. A similar usage pattern is evident in cursor-based APIs as Listing 7 using the .NET Framework's XPathNavigator class shows.

From these examples it can be seen that a cursor-model API is a satisfactory access mechanism for processing XML documents in-memory.

Strongly Typed XML

A large number of consumers of rigidly structured XML data tend to prefer it to be strongly typed so it can be mapped to objects and data structures native to the programming environment. The growing popularity of Object-XML mapping technologies such as JAXB, the .NET Framework's XmlSerializer, and Castor is a testament to this trend. In such cases the consumer is usually not interested in the fact that the data is provided as XML as long as it can be mapped to objects and data structures in the target programming environment.

You would then be justified in expecting that such consumers would be uninterested in XML APIs. However, this is not the case. A number of people have seen the growing benefits of being able to access objects as XML infosets when necessary since it gives them access to a wide range of technologies for processing XML such as rich queries using XPath (this is the basis of the ObjectXPathNavigator described in an Extreme XML column on MSDN). In such cases, a cursor-model API that provides an XML view of an object graph turns out to be quite beneficial. This approach was taken by the aforementioned ObjectXPathNavigator as well as BEA's XML Beans technology.

In some cases this means the ability to nest cursors is important. For instance, many XML Schemas written using the W3C XML Schema Definition Language (XSD) use the wildcards (xs:any and xs:anyAttribute) to enable extensibility of the XML messages being sent. This often leads to some parts of the document being strongly typed while others are untyped. The XmlSerializer in the .NET Framework maps such untyped content to one or more instances of the XmlNode class. Given that an instance of XmlNode can itself provide a cursor over its contents, the cursor over an object that contains one or more XmlNode objects as fields or properties needs to know how to handle nested items that provide their own XML cursors.

LISTING 1 •

```
<xs:schema xmlns:xs="http://www.
  w3.org/2001/XMLSchema">

<xs:element name="items">
 <xs:complexType>
  <xs:sequence>
    <xs:element ref="compact-
     disc" minOccurs="0"
     maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="compact-disc">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="price"
    type="xs:decimal" />
   <xs:element name="artist"
    type="xs:string" />
   <xs:element name="title"
    type="xs:string" />
  </xs:sequence>
 </xs:complexType>
</xs:element>

</xs:schema>
```

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

Conclusion

The purpose of this article was to explore whether it was possible for a single API to satisfy the major usage scenarios for consumers of XML documents. When I first started this article I assumed the answer was no, but once I actually started to investigate the issue I was surprised to find out that my initial impressions were mistaken. It is actually possible for a cursor-model API based on the XPath/XQuery data model to satisfy the needs of the users of several categories of XML data access technologies. The cursor-model API would be factored into forward-only and random-access versions to balance the needs of those who want streaming access to XML versus those processing entire XML documents in memory. A cursor-model API is also a nice compliment to XML<->Object mapping technologies because it enables users to transform the data within an XML document into primitives and data structures within their target programming language while retaining the ability to access the data as XML nodes.

The primary criticism of such an API is that it would be a compromise across widely differing usage scenarios and thus may not be optimized for the specifics of a given scenario. If such an API was designed and intended to replace existing models for processing XML, it would have to be carefully designed not to have too little functionality by being focused on the lowest common denominator nor too much by trying to have all the functionality of existing API models, thus making it bloated and difficult to use.

It will be interesting to see where the future takes us.

Reference

- Obasanjo, Dare. (2003). "A Survey of APIs and Techniques for Processing XML." www.xml.com/pub/a/2003/07/09/xmlapis.html

DAREO@MICROSOFT.COM