Content Management • Data Management • XML Labs

XML JOURNAL

THE ULTIMATE XML ENTERPRISE RESOURCE

August 2002 | Volume:3 Issue:8

IBM

www.ibm/com/websphere/winning

XML JOURNAL

FOUNDING EDITOR
Ajit Sagar ajit@sys-con.com

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

EDITORIAL
Coeditor-in-Chief
John Evdemon jevdemon@sys-con.com
Coeditor-in-Chief
JP Morgenthal jpm@sys-con.com
Editorial Director
Jeremy Geelan jeremy@sys-con.com
Executive Editor
M'lou Pinkham mpinkham@sys-con.com
Managing Editor
Cheryl Van Sise cheryl@sys-con.com
Editor
Nancy Valentine nancy@sys-con.com
Associate Editors
Jamie Matusow jamie@sys-con.com
Gail Schultz gail@sys-con.com
Jean Cassidy jean@sys-con.com
Assistant Editor
Jennifer Stilley jennifer@sys-con.com

PRODUCTION
VP, Production & Design
Jim Morgan jim@sys-con.com
Lead Designer
Aarathi Venkataraman aarathi@sys-con.com
Art Director
Alex Botero alex@sys-con.com
Associate Art Directors
Cathryn Burak cathyb@sys-con.com
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
Ron Ben-Natan, Steven berkowitz, Michael Champion, Tim Clark, Klaus Fittges, Eugene Kuznetsov, JP Morgenthal, Mike Morris, Paul Prescod, Soumen Sarkar, Barry A. Schaeffer, Ron Schmelzer, Travis Vandersypen

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.

Johnny Got Stuck in the Washing Machine

WRITTEN BY **JP MORGENTHAL**

While I understand that technology adoption occurs in steps, moving from simple to more complex, I'm amazed by how many people in the computing industry still don't have an understanding of what XML is and what problems it enables solutions for. I'm even more amazed by the people who are still using XML as a data format for systems integration.

But before diving too deeply into the problems of XML adoption, let me explain the title of this editorial. I recently got to thinking about the issue of context. Context provides a perspective from which other ideas and concepts may be better understood. It's a relative position. For example, with a context of discussion around a sunny day, you could easily gather that we were talking about "weather" and not "whether."

Context is one of the most beneficial components of representing information in XML. Each level of hierarchy in an XML document provides greater and greater context over the information that follows. For example:

```
<Person>
   <Name>
      <First/>
      <Last/>
   </Name>
</Person>
```

As you descend the hierarchy you reach an element identified as "First." First without context of Person/Name would be totally ambiguous, but within the defined context is quite easily understood to mean a person's first name.

This brings us back to the title, and to the subject of this editorial: information. Months from now, after the Web indexing agents have had their go at this piece, the title will have absolutely nothing to do with it, and only the full-text index of the article will present any real-world value for someone looking to better understand the power of XML. Thus we can garner information value only by identifying the key words that will unlock the information we seek, much the way the right ordering of teeth on a key will open a door.

The title of this piece offers no context for the information held within. Is the piece diminished in value? Hopefully, the answer is no. Would this piece offer more value if it were represented in XML with a properly defined context and an associated abstract and keyword list? Absolutely! Chances are unlikely for this scenario, as is expecting those creating our world's content to start marking up their content in XML.

This brings us back to the adoption issue. I view using XML for integration – as it's defined today – as one of most immature uses for this technology to date. Effectively, it amounts to a data serialization format, which is why integration projects are still failing in large numbers. The idea behind using XML for integration was not just a new data format. The goal was to make the information more usable by providing context to the legacy information so it can be used more effectively by other applications.

Those who are simply creating elements called "Amount" to replace the database column name "Amt" have missed the boat. Remember, integration is about making multiple systems operate as one. The types of information that should be defined in XML should answer questions such as: "What does that amount mean to System A relative to System B?" and "What other systems contribute to the creation of this amount?" That is, XML should provide the context for integration rather than act as the interim carrier of information between systems.

When asked recently what I thought the key benefits were to representing data in XML over relational technologies, my answer wasn't as straightforward as I first thought. My answer was that it was better suited for hierarchical data, which, on reflection, didn't answer this person's question. The key benefit is that hierarchy provides direction and context for the information, whereas relational data is arbitrarily ordered based on the user of the information.

After all I've seen, I still believe we've barely scratched the surface of what we can do with XML. As we get better at developing structures for data that make them usable to more processes than the one currently identified, we'll begin to push the envelope of this technology and realize greater information gains than ever imagined.

ABOUT THE EDITOR

JP Morgenthal is coeditor-in-chief of XML-Journal *and chief services architect at SoftwareAG. He has been writing and speaking about XML since 1997.*

JPM@SYS-CON.COM

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

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

BUSINESS DEVELOPMENT
VP, Business Development
Grisha Davida grisha@sys-con.com
COO/CFO
Mark Harabedian mark@sys-con.com

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

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

CUSTOMER RELATIONS
Manager, Customer Relations
Anthony D. Spitzer tony@sys-con.com
Customer Service Representative
Margie Downs margie@sys-con.com

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

WEB SERVICES
Webmaster
Robert Diamond robert@sys-con.com
Web Designers
Stephen Kilmurray stephen@sys-con.com
Christopher Croce chris@sys-con.com
Catalin Stancescco catalin@sys-con.com
Online Editor
Lin Goetz lin@sys-con.com

ACCOUNTING
Assistant Controller
Judith Calnan judith@sys-con.com
Accounts Receivable
Kerri Von Achen kerri@sys-con.com
Accounts Payable
Joan LaRose joan@sys-con.com
Accounting Clerk
Betty White betty@sys-con.com

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

A New Direction for Web Services

WRITTEN BY **PAUL PRESCOD**

Web services is a glamorous term for a very old and established idea. The core Web services technologies, SOAP and WSDL, allow developers to define new protocols. A protocol is nothing more than a way for computer programs to talk to each other over computer networks. As long as there have been networks there have been protocols. What makes Web services unique is that they're the first mainstream technologies, supported by all major vendors, intended to make the creation of new protocols so easy that ordinary business developers can do it.

Developers who, for instance, build a SOAP wrapper around their payroll systems may not think that they're building their own protocol, but in fact they are. Web service developers must decide how to group their logical objects into services. They must create some technique to correlate messages sent to the service with particular objects. They must choose which messages may be sent to which objects. For each message they must choose whether or not it should be sent in a manner that expects a reply. They must choose an XML representation for the message. These are the decisions that would usually be made by a so-called "application protocol" like SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), or HTTP (HyperText Transfer Protocol).

I claim that there are two problems with this picture. First, making new protocols is still not easy enough for ordinary developers to do. Today's tools only make it seem easy by removing or hiding many options necessary to actually make these new protocols succeed. Second, making many new protocols isn't necessarily a good idea. Let's drill into each of these assertions in depth.

Visual Studio.NET is probably the most popular, widespread Web services creation toolkit. It seems to make Web service construction ridiculously easy. A developer takes an existing class, clicks a few buttons, and has a Web service. The developer doesn't need to learn anything about either XML or network programming.

The problem is that this technique obviates any advantages that Web services were to provide over DCOM and CORBA. In particular, Web services were supposed to more cleanly separate a service's interface from its implementation. This aspect of Web services, termed *loose binding*, requires a thorough understanding of XML. Loose binding allows your implementations to evolve while your public interface is static. In order to make Web services easy, the tool vendors ripped out its central benefit.

As if that isn't bad enough, the Web services specifications themselves had already ripped out one of the core values of the Web. The Web is first and foremost an environment in which any object (traditionally, but not necessarily, a Web page) can reference any other object. It was the first global information system through which any organization's data objects could reference and thereby incorporate data objects from any other organization. The Web performed this amazing feat by moving all information into a single namespace addressable through URLs (known to the intelligentsia as URIs). XML technologies have always built on this feature: XPointer, XLink, XInclude, and even XSLT are all organized around the concept of URLs. SOAP-based Web services are never in practice organized around URLs. Some use GUIDs. Some use numeric identifiers. Some use RPC parameters. There's no standardization as there is on the Web.

Even if we ignore this large problem, it's debatable whether scalable, robust, fault-tolerant Web service construction can be made as easy as clicking buttons on a dialog box. Networked applications must intrinsically deal with many issues that don't affect local programs: latency, race conditions, topology changes, network outages, obsolete client and server software, denial of service attacks, and so forth. The dialog boxes don't offer any strategies to deal with these problems. When today's Web services are deployed in the real world, I expect we'll find that ignoring these issues will have serious consequences for the reliability of our services.

Let's move on to the second problem. Does it make sense to encourage people to create all of these new protocols? Metcalfe's Law states that the value of a network grows by the square of the size of the network. So a network that is twice as large will be four times as valuable because there are four times as many things that can be done due to the larger number of interconnections. A network a hundred times as large is ten thousand times more valuable.

Consider how this applies to protocols. According to Metcalfe's Law, the open, public, SMTP-based e-mail system is thousands of times more valuable than even the most sophisticated proprietary e-mail system at even the largest company. This explains why today almost every corporate executive has e-mail whereas 10 years ago they would have left that unimportant chore to a secretary…if they bothered with it at all. It

Continued on page 8

Reader Feedback

Persuasive Article

Reading this article ["Desperately Seeking…Help for XML Schema" by Tom Gaven, June 2002] makes me think that this is the beginning of a much larger critique of XML Schema. Tom, keep your illuminating articles flowing. Good job!

PRASHANTH RAO
via e-mail

Subj Access/ Identification

Noticed your journal in a local Barnes & Noble and purchased the June copy.

We are interested in learning who is using XML in the area of intellectual property. More precisely, we are wondering if anyone is tagging patents with XML and working with a subject schema to enhance/improve subj searching/retrieval. We have looked at and spoken to the U.S. Patent and Trademark Office. They do have an XML coding/tagging solution being offered, PASAT, that can be used by those submitting patent applications. But this XML tagging is not changing the subj access/identification process.

Do you have any recommendations/-suggestions as to how we can stay on top of the players in this specific niche of XML application?

Your insights/experience sharing is most appreciated.

PEGGY LUCERO
Director of Marketing and Human Capital
4iP, Inc.
webwiseus@speakeasy.net

From the Editor

I am not aware of any specific companies or standards focusing on this particular vertical application of XML. I assume you were attracted to the publication because of the article on patent mining ["Patent Mining Using XML," by Joseph Moeller]. I also cannot claim to understand the intricacies of this particular niche. For example, you refer to "changing the subj access/identification process"; could you explain the issues with this further?

I have forwarded your message to someone who works with software developers in the XML space who might be better able to help you with your particular interests.

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

Continued from page 7

was the ability of anybody to talk to anybody that made e-mail so powerful. The same can be said for the Web and its popular HTTP protocol.

But SOAP isn't like SMTP or HTTP. If General Motors exposes a SOAP interface with one set of messages and Ford exposes a different SOAP interface with other messages, a supplier trying to integrate with both of them is only marginally better off than if one company had exposed a CORBA interface and one a COM interface. Mapping between their different method semantics is the tricky part. In any modern programming language the difference between calling a COM versus a CORBA method is trivial in comparison.

Corporations that use Web services technologies to invent new protocols are going to get very poor economies of scale and essentially no Metcalfe-value multiplier. Inventing new protocols should therefore be a last resort.

A small group of Web service developers has recently begun to promote a radically different strategy. We argue that protocols are essentially means of moving information around a network and coordinating processes based on that information. In this sense they're not unlike file systems and databases. Where file systems and databases are generic holders of heterogeneous data, protocols could be generic movers of such data.

We could stop making protocols optimized for particular types of data and particular problem domains and instead focus on protocols that can work with any kind of data in almost any application domain. Just as we don't have special file systems for image files, or special databases for customer addresses, we don't need different protocols when moving images instead of customer addresses. No matter what the content type, we could transfer representations of it between machines. This model is therefore called "REpresentational State Transfer," or REST.

File systems have a relatively simple, generic model: "read file", "write file", "delete file", "extend file". Databases do too: "SELECT", "INSERT", "UPDATE", "DELETE". In the protocol world the protocol that is most generic and has the best testing is HTTP. Its "GET", "PUT", "POST", and "DELETE" methods are quite analogous to the SQL and file system methods.

Consider how much HTTP infrastructure and knowledge is already out there. The software industry has spent the past five years selling people tools for working with HTTP. It's stable and has excellent reliability and scalability features, such as load balancing, failover, and transparent caching. The vendors, of course, want to convince you that you now need new tools because of the rise of Web services. But those tools will need years to catch up, and in fact may never catch up because, arguably, the SOAP Web services model itself makes tool development much more difficult.

Every resource exposed through HTTP (as opposed to SOAP) has a URI. That means that every resource can refer to every other resource. You can see how Metcalfe's Law applies. In a Web of REST Web services the notion of application just drops away and is replaced by the much finer-grained idea of "resource." There are no concrete boundaries between applications – there is just a "Web" of linked resources.

HTTP's very generic methods can work on any kind of data. In most cases this will be some XML vocabulary. This is also where incompatibility will inevitably creep back into the system. Going back to our analogies, storing Word documents on a common file system does not automatically make them compatible with WordPerfect, and using SQL does not automatically make a program designed for one database schema work with another. Similarly, programs expecting different XML vocabularies will not magically work together.

But just as there are filters for office file formats and views for relational databases, there are tools and techniques for mapping between XML vocabularies. XSLT is the most pervasive and famous, but we should also pay attention to the so-called Semantic Web technologies: the Resource Description Framework (RDF) and the Darpa Agent Markup Language (DAML). Another technique for managing XML interoperability issues is the standardization and sharing of schemas through registries, repositories, and standards bodies.

Even in the SOAP model, we'll need to fight incompatibility through transformations and XML vocabulary registries. REST just helps to take protocol interoperability issues out of the equation so we can focus on these pressing XML issues.

The existing Web protocol, HTTP, already makes the transfer and addressing of networked data as natural, standardized, and well understood as the storing of data in file systems and databases. It provides a more solid foundation for the future of Web services.

• • •

To learn more about how to build Web services with XML and HTTP, please see:

- *Paul Prescod's REST Page:* www.prescod.net/rest
- *REST discussion wiki:* http://internet.conveyor.com/RESTwiki

PAUL@PRESCOD.NET

AUTHOR BIO

Paul Prescod, coauthor with Charles F. Goldfarb of the XML Handbook, *worked with the W3C to develop the XML family of standards. A proponent of open source technologies, Paul is known for his implementation of systems using XML and the open source Python programming language. He also has experience in C, C++, and Java. Before founding Constant Revolution, Paul worked for ActiveState, the leading vendor of tools for open source programming languages.*

WRITTEN BY **TIM CLARK**

Orchestration: The Missing Link

New tools are needed to keep up the pace

Tools to create Web services and to put Web services "wrappers" around existing software features are proliferating rapidly. But tools to help enterprises keep track of these new components remain relatively rare.

Earlier this year Greg Clark, chief operating officer of E2open, described such tools for managing Web services as the biggest missing aspect of Web services. At the time he saw relatively few solutions. E2open, a venture capital and industry-backed collaboration network for high-tech and electronics manufacturers, is building its infrastructure on XML Web services for easier integration with the many legacy systems its owners and clients use.

Since then, both startups and established software tool vendors that provide similar features for other software components are moving to offer these "orchestration" capabilities for Web services. But the tools remain immature because the needs of corporate users are evolving as the core technologies of Web services evolve.

The FactPoint Group uses the term *orchestration* to describe the types of services required to track, utilize, and assemble Web services components into broader applications. Others use *manageability*, but we prefer orchestration because it implies not just managing the software bits (which would be "manageability") but the business processes that are encapsulated in Web services. Fundamentally, Web services is about business processes, not simply software.

In research published in June, The FactPoint Group found that early adopters of Web services are relatively satisfied with today's Web services tools and protocols for the fairly simple Web services applications that have been built to date.

But as these users look ahead, they believe future Web services projects will outstrip today's Web services tools and standards. As one user put it, "The depth and breadth of implementation is questionable right now."

That voice sets the tone for many early adopters, who may not have immediate needs but are closely monitoring the standards processes and how vendors evolve their offerings over the next year, especially in this area of orchestration.

Web services allow monolithic pieces of software to be served up in smaller units, allowing "granularity" or the use of smaller bits of functionality. That has two positive benefits: easier access to back-end business functions and reuse of Web services components as building blocks to assemble larger applications.

But having smaller bits of software code creates complexity in tracking Web services so they can be reused. These users clearly don't see standards such as UDDI (the Universal Description, Discovery, and Integration standard) as ready for the challenge, nor do they think vendors have fully grasped the problem. They cited seven areas of orchestration still lacking for next-generation Web services implementations:

- ***Monitoring tools***
- ***Reporting***
- ***Version control***
- ***Mapping metadata***
- ***Messaging requirements***
- ***Discovery***
- ***Workflow integration***

Overall, these seven identified needs for orchestration suggest a world of more complex applications than most Web services implementations today, which consist largely of pilot projects, simple internal application integrations, and external implementations with trusted partners. These corporate early adopters are already looking to the day when they will build proprietary applications by assembling Web services created by third parties. As a strategy manager for a Fortune 500 bank told us: "We don't want to develop applications anymore. We want to buy more off-the-shelf, buy apps that are architecturally compliant and loosely coupled."

Orchestration Needs

Before Web services will be ready for that sort of corporate "prime time," certain orchestration needs will have to be addressed.

Monitoring tools

Monitoring tools give corporate IT departments the ability to check the performance and reliability of Web services tools and applications. The CIO of a Midwestern chain of health clubs considers monitoring a key requirement before he can move his entire applications infrastructure onto XML: "We don't yet have some of these monitoring tools and we need them. What's exciting is that a year from now, we'll have more tools to select from."

Reporting

In several industries, including pharmaceuticals and healthcare, reporting requirements from government regulators or other entities are more stringent than can be met today by Web services. That has kept more than one firm from even experimenting with Web services. "Web services don't get address reporting, and I have a big reporting need," said a senior Internet architect with a Fortune 500 pharmaceutical firm.

Version control

This becomes an issue when a company is running multiple instances of the same application or Web service. "How can I be sure all are in sync? How do I bring

AUTHOR BIO

Tim Clark is a partner in The FactPoint Group (www.factpointgroup.com), a Silicon Valley research, consulting, and publishing firm. He was lead analyst on the firm's Web services research, which interviewed 50 enterprises and surveyed nearly 800 corporate users in collaboration with Outsource Research Consulting. For five years Tim was a reporter and e-commerce columnist for CNET Network's News.com and for Inter@ctive Week.

them up and shut them down?" asks a technical director with a major financial services firm. Version control issues also may emerge as vendors that offer "Web services for rent" periodically upgrade individual Web services in ways that affect how those components perform as they're assembled into larger applications.

Mapping metadata

This becomes a requirement as application architects design how Web services are assembled in new applications. "I need a way to design a service layout, with metadata about Web services," a technology analyst at a telecommunications equipment company told us.

Messaging requirements

These are akin to having an old-fashioned telephone switchboard and a switchboard operator to manage Web services. The same Midwestern health club chain quoted above needs "an engine that manages the transaction asynchronously as all the different systems are being connected. Web services allow all these connections we never had before."

Discovery

This refers to how an enterprise locates a Web service it would use in an application. The UDDI standard is supposed to address this need, but today few Web services are listed on public UDDI directories. E2open thinks industry-specific UDDI directories will be part of the answer, and providing a UDDI directory for high-tech manufacturing is a key element of E2open's business model.

More broadly, UDDI directories are designed to enable businesses to quickly, easily, and dynamically find Web services and transact business with one another. The CIO of a major airline's loyalty card unit describes his needs:

> *We need a better handle on how you advertise Web services. You have to know what the Web service expects and what to expect from it. We want to make sure the flight schedule quoted is accurate so we can brand it so that it looks like it's coming from us and not a third party. So we need a way to brand that information and for people to be able to find it.*

Workflow integration

This refers to mapping Web services into an enterprise's existing workflow processes. This IT manager at a big telecommunication firm is already implementing Web services for simpler applications but wants more:

> *What is missing from the current Web services vendor offerings are robust workflow management systems – meaning the ability to easily take your process and map it to a workflow pattern. I already have workflow processes. Why do I have to reconvert it to their environment? If I had that, I could deliver functionality faster.*

Response to the Needs

Vendors are beginning to respond to the need for orchestration tools. At **SYS-CON Media**'s XMLEdge 2002 East–International in June in New York City, the most prominent exhibitor at the trade show was Attachmate, whose main business is mainframe connectivity and data transformation. It grabbed the premier spot at the only entrance to the show floor. That was fittingly symbolic: Web services are a valuable tool for Attachmate's legacy integrations.

Legacy vendors Seagull, Serena, WebPutty, and others also made showings at the SYS-CON show, underscoring a move by software tool firms with real businesses outside Web services into the XML "orchestration" space. Pure-play Web services firms such as Actional, LogicLibrary, WestGlobal, and others also talked up their "orchestration" solutions.

We expect both EAI vendors (Tibco, etc.) and other software toolmakers (Borland, webMethods) likewise to "orchestrate" Web services into their offerings.

Ultimately, we expect orchestration features will be built into offerings of middleware vendors (IBM's Tivoli, HP's OpenView, even Computer Associates, etc.) In the Darwinian world of enterprise software, some of these big-fish middleware firms are likely to swallow smaller Web services minnows. Witness Novell's recent purchase of SilverStream and Sybase's acquisition last year of New Era of Networks, both plays to buy an XML story.

• • •

Data in this article comes from research on Web services by The FactPoint Group and Outsource Research Consulting (ORC) (www.orconsulting.com). The key finding: North American companies are adopting Web services at a faster rate than anticipated. We believe that companies holding back on Web services risk being bypassed by nimbler competitors. Almost half of the respondents to our survey this spring are already piloting or deploying live applications based on Web services. Fortune 1000 companies are the most aggressive adopters of Web services applications. However, smaller firms are braver in venturing beyond the firewall to use Web services applications to link to trusted trading partners.

Bottom Line

Early corporate adopters of Web services need new tools to orchestrate granular Web services into broader applications, thus capturing the benefits of reusing Web services. Until these orchestration tools are available, even the most venturesome corporate adopters will be unable to capture the full benefits of using Web services.

TCLARK@FACTPOINTGROUP.COM

WHO'S PLAYING IN THE ORCHESTRA?

Moving into Web Services Orchestration

Attachmate
www.attachmate.com

Mercury Interactive
www-svca.mercuryinteractive.com

Novell/Silverstream
www.silverstream.com

Seagull
www.seagull.com/

Serena
www.serena.com

Sybase/New Era of Networks
www.sybase.com

WebPutty
www.webputty.com

Startups for Web Services Orchestration

Actional
www.actional.com

Amberpoint (formerly Edgility)
www.amberpoint.com

Dirig
www.dirig.com

LogicLibrary
www.logiclibrary.com

WestGlobal
www.westglobal.com

Expected to Move into Web Services Orchestration

IBM (Tivoli)
www.ibm.com

HP (OpenView)
www.hp.com

Computer Associates (CA Unicenter)
www.ca.com

WRITTEN BY **JP MORGENTHAL**

Transactions Aplenty

Classifying transactions to set expectations

In the world of automation, the ambiguous can be a beautiful thing or it can be a nightmare. To those responsible for delivering a solution, ambiguity leads to missed expectations, higher costs for delivery, and delays in completion. To those providing solutions, ambiguity leads to opportunity.

One of the most overused and ambiguous words in the IT industry today is *transaction*. Over the years, the transaction has come to mean so many different things depending on whether you have a business perspective or a technology perspective. Recently, however, the business and technology perspectives have begun to coincide at management levels. This makes the goal of communicating the vision and goal of a project even more difficult and complex than previously.

Let's look at some of the different uses of the word in the enterprise world and isolate a few distinctions in its meaning that we'll be able to use downstream to ensure clear delivery of intent.

Exchange Transaction

From a purely business perspective any exchange of goods or merchandise constitutes a transaction. If you purchase a new pair of shoes from the store, the purchase itself is a transaction. If you purchase the shoes with a credit card, there are actually two transactions (although they look like one): the exchange of money for shoes between the customer and store and the financial reimbursement to the store from the credit card company.

Database Transaction

ACID is the acronym for atomic, consistent, isolated, and durable. *Atomic* means that the transaction is unitized and encompassing. *Consistent* means that the data from the transaction is kept valid, by either completing successfully or terminating and returning the data to its original state. *Isolated* means that the data from the transaction is protected from corruption by any other transaction. *Durable* means that once the transaction has executed, it is permanently recorded.

This definition tends to be most closely associated with the properties of transaction management with regard to database management. However, I think that ACID captures the intent of any transaction, whether it occurs in the database or over the Web.

Compensating Transaction

Compensating transactions effectively undo transactions that have already occurred. For example, when a customer returns merchandise to a store, the store has to return the merchandise to stock (or to the manufacturer or distributor) and return money that was already booked. The interesting thing about such transactions is that they often need to be coupled with the original transaction in order to gain insight into transaction patterns.

Transactional Messaging

As data moves from one location to another, we often use the word transaction to characterize that it does so in a guaranteed manner. Transactional messaging leverages the principles of ACID to ensure that a message is delivered to its intended recipient or that a proper level of exception is created so alternative processing can occur. Transactional messaging also ensures that a message is delivered once and only once, which is critical for proper message handling. It would be problematic if two messages to remove $1,000 from your bank account resulted from the same, single ATM operation.

Transaction Processing Monitor

The TPM is actually a poorly named piece of software that has responsibility for ensuring that processes are scalable and running. Most often TPMs will watch for a process to fail and then restart that process. TPMs do use the concepts of ACID-ity to manage the processes, which is why the word transaction is associated with them.

• • •

As you can see from the above definitions, it's possible to classify transactions for purposes of setting proper expectations. If you're developing system requirements and want to represent that the transaction is initiated by the user purchasing merchandise in the store, it would be best to call it an exchange. Likewise, if you want to represent the credit portion of that transaction, you might identify that as an ACID or transactional message, depending on whether it's with the store's back office or directly with the credit card company.

Clarifications like this may seem rudimentary and overly basic, but between those gathering requirements and those implementing the solution I've seen confusion mount quickly because the terms weren't defined explicitly.

In one very poorly defined situation, the system requirements described what amounted to an exchange transaction, which led the implementers to believe they could build their system against the executed transactions in the database. One month before expected completion of the project, the misunderstanding surfaced and the team realized they were a good two months away from completion because they hadn't built the mechanism to record the transaction in the database the point of sale.

JPM@SYS-CON.COM

XML Computation Trees

Written by Ron Ben-Natan

Every computer science undergraduate program in the world has two important foundation courses: data structures and algorithms. Open any book on these subjects and you'll see immediately that almost a third of it is devoted to graphs. Graphs are used to model a very large number of real-world problems: the traveling salesman problem, efficient routing of a package, network flows, and more — all are modeled as graphs and often solved by graph-based algorithms.

An approach to managing computation graphs that uses XML

A common use of a graph-based representation is that of a computation graph. Simply put, it's a graph that models a set of functions F1, F2, …, Fk that take a set of inputs I1, I2, …, In and computes a set of outputs O1, O2, …, Om. As an example, Figure 1 shows a simple computation graph with a small number of nodes. This simple example represents the following equations:

```
O1 = F5(F1(I1, I2), F4(F1(I1, I2), F2(I2, I3)))
O2 = F6(F4(F1(I1, I2), F2(I2, I3)), I4)
O3 = F3(I3, I4)
```

Note that by managing the computation as a graph traversal you can eliminate computing F4(F1(I1, I2), F2(I2, I3)) twice. The nodes modeling both F5 and F6 access the value computed by the node representing F4.

Computation graphs arise in many environments and tend to be the preferred way to model very complex computations and dependencies. In fact, most computationally intensive environments use graph-based computation structures. The most prominent example of such environments occurs in financial services, including computer-based portfolio management, computer trading, pricing systems, risk management systems, hedge systems, and insurance quoting systems. These structures also arise in contract management systems, constraint-based optimization, impact analysis, root cause determination, and simulation systems. The reason is that a graph is the perfect way to model the dependencies between computation elements as well as the perfect metaphor to program to.

Most current systems managing computation graphs make use of proprietary environments to model and manage such structures. In this article I'll walk you through a different approach, one that makes use of XML to represent computation graphs. The approach is based on an XML representation that can be handed to almost context-free computation engines that cooperate to compute very large computation structures. While there would seem to be overhead and inefficiency in using XML instead of proprietary methods, this is offset by the fact that the full computation graph is partitioned so that many CPUs can work in tandem to compute the full result set. This approach has additional advantages in areas involving the maintainability of the system and the total cost of ownership.

Since the full framework involves some very advanced, complex, and sometimes confidential graph-partitioning methods, this article focuses on the general framework and stays away from the mathematical aspects of the approach.

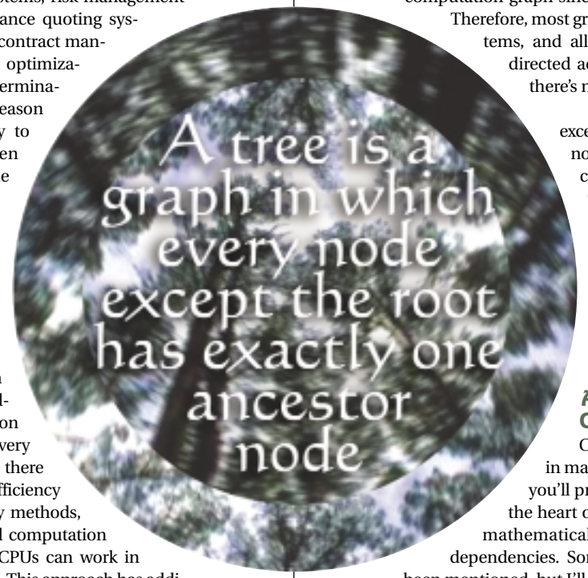
Graphs and Trees

Data structures representing graphs (and hierarchies, which are also graphs) have been in use since the dawn of computers. For example, the major database technologies before relational databases were network and hierarchical databases. Efficient algorithms on graphs are certainly one of the most researched areas within computer science, and the need for such algorithms stems from every business vertical that uses computing systems. This isn't just a passing trend. Actually, over the years it has become more and more prominent. The current object-oriented development paradigm is based on objects, and it's most natural to represent them as a graph in which the nodes are the objects and the edges are the object references. In fact, object-oriented databases use a native graph-based representation, and hierarchies (both type and instance hierarchies) are best managed as graphs and sometimes as trees (a special family of graphs).

XML also follows this pattern. It organizes data according to a containment hierarchy, and any XML document can be viewed as a containment tree. More important in the context of this article, any tree can be encoded as an XML document, and graphs can be encoded in XML by adding attributes that represent references.

Formally, a graph G=G(V, E) includes a set of nodes (or vertices) {V1, V2, …, Vn} and a set of edges {E1, E2, …, Em} where for each i, Ei={Vk,Vl} where Vk and Vl are both nodes in the node set. A directed graph is one in which each edge has a source node and a target node; in this case an edge is sometimes written as Ei=<Vk,Vl>. Computation graphs are always represented by directed graphs in which the edges represent the computation dependencies (i.e., the value computed by the source node is an input to the computation of the target node).

Graphs can have cycles. A cycle within a directed graph is a set of edges E1, E2, …, Ek where for each i the target node of Ei is the same as the source node of Ei+1, and the target node of Ek is the same as the source node of E1. Cycles are clearly a bad thing for a program traversing the computation graph since the cycle implies an infinite loop. Therefore, most graphs managed within computer systems, and all computation graphs, tend to be directed acyclic graphs (DAGs), meaning that there's no directed cycle within the graph.



A tree is a graph in which every node except the root has exactly one ancestor node. Trees are also heavily used in computation graphs; the reason is that the computation dependencies are more limited. Thus the propagation of values through the structure is simpler and faster. In many cases a computation DAG can be reduced to a computation tree by creating duplicate nodes, a common practice in efficient computation solutions.

Applications of Computation Graphs

Computation graphs are heavily used in many domains. If you dig deep enough, you'll probably find a computation graph at the heart of any system that includes complex mathematical calculations and computational dependencies. Some of these domains have already been mentioned, but I'll elaborate a bit on two that I've seen.

The first involves constraint optimization. Imagine that you're building a system that calls for optimal assignment of resources (e.g., workers or crews) to tasks. Each task has various components, such as where it has to be performed and in what time frame, and the skills required. Given a set of tasks and resources, it's your job to create the best possible schedule for your workforce to perform all tasks within the defined constraints.

There are many approaches for performing such optimizations, and the use of computation graphs may not be immediately obvious. Computation graphs come into play because optimization problems such as that described above need constant recalculation. This is often called *continuous optimization*. The problem is that as time goes by reality changes and constraints need to be reapplied. New tasks are created, tasks may be completed ahead of schedule, resources can become unavailable.

All this creates a reality in which it's not good enough to compute a good schedule once a day. The schedule needs continuous updating in real time as new information comes in. To do this efficiently, a computation graph has to be maintained to enable continuous reoptimization. New inputs ripple through the graph, updating the schedule, thus avoiding recomputing from scratch.

The key is real-time recomputation, the common thread in many of today's advanced uses of computation graphs. The second example is also heavily biased toward real-time recomputation. The domain is real-time pricing and real-time risk calculations for financial products and portfolios. Today's investment banks create and manage highly customized and very complex financial instruments. This is not about stocks, and not even about simple derivatives such as options and swaps. It sometimes involves hybrid derivatives, or synthetics. Investment banks take positions in one instrument or portfolio in order to make a commission or a margin, but don't want to carry the risk. In such a scenario they will hedge the portfolio with another instrument or portfolio that can be correlated with the original one. All of these instruments and portfolios can be represented as computation graphs managed for both pricing and risk management purposes. The inputs to these computation graphs include, for example, spot interest rates, exchange rates, and stock prices.

There are two important characteristics of these graphs: (1) they need to be very flexible and dynamic to allow traders to create the best instruments for the bank, and (2) they're very sensitive to constantly changing conditions. As an example, risk is sensitive to many factors, one of which is time itself. Because time is continuously changing, risk calculations can potentially require continuous recomputations that can trigger adjustments to a portfolio.

It's not unusual to see computation graphs with thousands and even tens of thousands of nodes that need to be recomputed in real time. Such computation graphs place a huge requirement on computation power. It's typical to see such computation engines run on very powerful (and expensive) Unix machines with a large number of CPUs. It's still never enough. These systems also tend to be huge memory hogs, requiring 2 and even 4GB of memory to hold the graph structure in memory and perform the computations.

The nature of such environments is that they always require more CPUs and more memory than is readily available. Some sample causes follow.

- The computation graphs are truly complex. There's no way to get around that. The functions in the nodes are elaborate and often require complex computations including solving partial differential equations and performing iterative "guessing."
- Nodes in the computation graph often involve Monte Carlo simulation. Computing a value is often not deterministic, but rather requires a sampling from a probability distribution, multiple (up to many thousands) runs, and an averaging in order to compute a single value.
- Traders and quantitative analysts (quants) often need to work off a single computation graph but create permutations trying to simulate a certain environment or instrument. In effect, they create copies of subgraphs and can quickly grow the computation burden by a factor.

One way to overcome the problem is to use a large set of computers on a network for real-time recomputation of such graphs. While this has been recognized by many, it hasn't been easy to do. The main reason is that most environments doing such calculations are highly proprietary and often include a home-grown object database and a proprietary communication and distribution layer. While such proprietary environments have proved themselves in operation, they've also proved to be very expensive to maintain, and the total cost of ownership (including the need to continuously maintain and improve the proprietary environment) tends to be so high that only investment banks have been able to afford it.

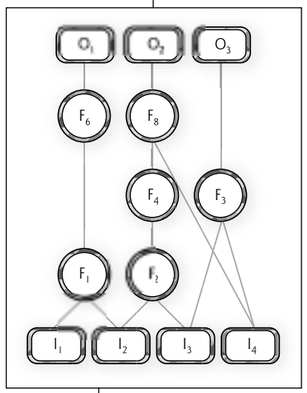


FIGURE 1 A simple computation graph

Computation Graphs in XML

A new paradigm aimed at lowering this total cost of ownership is emerging. It's based on the use of standard tools and techniques as well as partitioning the work and distributing it to a set of low-cost computers (running Linux or Windows). Much of the savings results from the fact that the bulk of the work is delegated to XML tools and less proprietary code needs to be written. The framework is based on representing the computation graph in XML, partitioning the graph into a large number of subgraphs (each also represented in XML), and distributing this computation to many computers on the network.

To use XML easily (and not require HREF, XPath expression, or other types of references), the computation graph may be converted into a tree. This is done by duplicating nodes and maintaining multiple nodes that subscribe to an underlying value. While not the most elegant solution, it simplifies the algorithms for partitioning and managing the computations. Alternatively, the graph can be partitioned first; only then is each subgraph converted to a tree.

The computation graphs are built using standard XML editors. This means that you can use off-the-shelf tools for building the graphs instead of writing proprietary editors, helping to reduce cost of maintenance. To make sure that the computation trees make sense in terms of the inputs and outputs, XML schemas must be employed to represent typing information.

Assume for example that F is a function that takes as input a value of type Ta and a value of type Tb and computes a value of type Tc (i.e., Tc F(Ta, Tb)). Assume that G is a function that takes as input a value of type Tc and calculates a value of type Td (i.e., Td G(Tc)). Also assume that ta is an input value of type Ta, tb is an input value of type Tb, and tc is an input value of type Tc. The schema needs to be built so as to easily represent G(F(ta, tb)) as well as G(tc) in a way that can be type-checked by the XML editor.

The "trick" in defining the schema is in the introduction of elements that can be either data or return values from the recursive activation of a function. Such elements are defined as compositors using choice elements that include the basic type as well as any function node that returns that value. As an example, the TcVal that becomes the type of the parameter of the function G can be defined using the following compositor:

```
<xsd:group name="TcVal">
 <xsd:choice>
<xsd:element ref="Tc"/>
<xsd:element ref="F"/>
… and any other function that
 returns a Tc
 </xsd:choice>
</xsd:group>
```

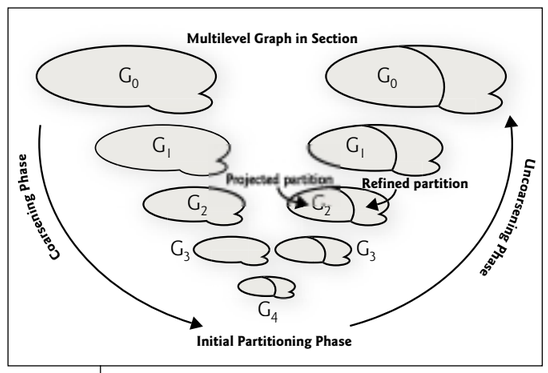


FIGURE 2 Multilevel graph bisection

Partitioning the Computation Tree

The most important phase in the process is the partitioning phase. In this phase you need to partition either the computation tree or the graph, depending on whether conversion to trees happens before or after the partitioning. Good partitions mean that parallel computations can be performed efficiently. A good partitioning is also the key to continuous real-time recomputation.

It's impossible here to survey all the methods used for partitioning. Partitioning can be proved to be NP-hard (meaning that there is probably no deterministic fast algorithm for doing general partitioning), but many approaches have been used in the past, some producing very good results. These range from simple greedy algorithms to complex spectral methods using eigenvectors of the Laplacian of the graph. Also, partitioning is often based on domain-specific nodes. The simplest example involves Monte Carlo simulation nodes since it's clear that multiple runs can be done in parallel.

The partition is normally kept static, at least for a while. Although the dynamic nature of the computation tree is such that continuous partitioning may seem to be required, in practice this is overkill and takes much too long. Therefore, new partitions are created daily at most, and the subtrees are maintained throughout a relatively long period of time.

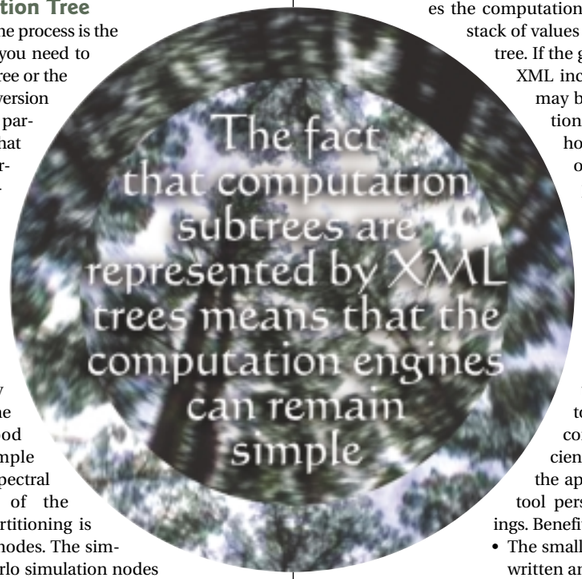
Representing the computation graph in XML complicates the partitioning phase, and is the main area in which more research is required. If partitioning is done while looking at the entire graph, then the graph needs to be built in memory, which means that the partitioning process will be a memory superhog. In fact, building the tree using DOM is impossible for large graphs and shouldn't be attempted. Instead, current approaches for very large computation graphs use partitioning algorithms that can be implemented using a SAX parser with one or more passes over the graph. Multilevel bisectional algorithms (the general concept of these algorithms is illustrated in Figure 2) lend themselves to such an implementation.

It's possible that JAXB, the new Java Architecture for XML Binding, may prove to be a good alternative to SAX processing for the partitioning phase. JAXB (formerly called Adelard) provides a framework for supporting a two-way mapping between XML documents based on schemas and Java objects. Schemas are compiled into Java classes that create a specialized parser based on the schema. The result is a parser that often operates as fast as a SAX parser but builds objects in memory while maintaining a small memory footprint. The JAXB documentation claims that early JAXB prototypes have shown that JAXB can be faster than SAX because the generated Java classes are precompiled and contain the schema logic, thereby avoiding the dynamic interpretation that a SAX parser performs. Using a JAXB-based approach, it may be possible to create better partitions, but whether such an approach is feasible from a memory footprint perspective has yet to be demonstrated.

Computing Subtrees

The fact that computation subtrees are represented by XML trees means that the computation engines can remain simple. Indeed, all they need to do is parse the XML and compute it based on the functions defined in the XML. The functional engines are therefore nothing more than tree traversal programs.

The simplest approach to the actual computation is one that parses the computation tree using SAX and maintains a stack of values that reflect a DFS traversal of the tree. If the graph isn't converted to a tree, the XML includes references and two passes may be required. In this case the partitioning method needs to reduce (and hopefully eliminate) dependencies on other subgraphs. A family of partitioning methods called bounded contractions uses this as the primary target.



Summary

XML parsing is overhead, pure and simple. It's thus a bit surprising that an XML-based scheme for managing computation graphs can be used. But XML tools (parsers and other tools) are constantly being made more efficient, and the inherent simplicity of the approach from an architectural and tool perspective leads to significant savings. Benefits include:

- The small amount of code that needs to be written and maintained
- The use of XML editors rather than self-built editors
- The flexibility afforded by schemas
- The inherent attributes of XML that make the computation graph descriptions less fragile
- The fact that graph serialization is implicitly taken care of by the very nature of XML-based communication

There's one additional advantage that I believe is the most important one:

- Because the framework is based on XML documents, it's inherently traceable.

It's easy to debug, tune, and improve the process because you can see what's going on within this complex network of computation engines. Dependencies are implicitly described in a form that is readable by both humans and programs. These descriptions can be easily used to generate reports and perform queries using tools such as XSL and XQL – again saving a lot of complex code that would otherwise have to be written. Current approaches based on in-memory proprietary structures are very hard to debug and trace. If you're into computation graphs, this advantage alone should make you at least curious about whether XML computation graphs are appropriate for you.

Finally, most of the mainstream database vendors seem to be moving toward native support for XML. As an example, the next major release of Microsoft's SQL Server supports XML as a "first-class citizen" (i.e., the database internals optimize XML handling and don't rely on a "bolt-on"). This means that mainstream databases can be used for these advanced environments instead of continuous use of proprietary or niche-vendor environments.

RBENNATA@HOTMAIL.COM

AUTHOR BIO

Ron Ben-Natan has been developing object-oriented software for over 15 years using Java, Smalltalk, and C++. Currently chief technical officer at ViryaNet, a company specializing in service optimization, Ron previously worked for Intel, AT&T Bell Laboratories, Merrill Lynch, and J.P. Morgan, among others. He holds a PhD and an MsC in computer science (in the fields of distributed computing and probabilistic algorithms, respectively), and has authored numerous technical books.

WRITTEN BY **BARRY A. SCHAEFFER**

Getting Serious About Content Management

A brief review of best practices

For some time the text information world has known instinctively that something called "content management" should be part of its planning and operations. Prior to the Internet and Web, however, managing one's content, while perhaps valuable, often wasn't perceived as critical to the success of information efforts, and where it was, technological "cats and dogs" could be made to work well enough to get by.

Things have changed. Today an organization's very survival can depend on its ability to create or acquire, manage, and deliver rich content on ever-tightening schedules to audiences increasingly impatient with any lapse in currency or usability of presented information products. In the face of this growing demand that providers better manage and share what they know, the need for a closer look at what works well is making itself increasingly felt. The demand is further escalated as more content is designed and created in XML – enhanced by the rich structures and tagging conventions made possible by this rapidly growing worldwide family of standards, but complicated by the growing complexity of the XML movement itself. Indeed, this article is intended for organizations using or considering XML to create, manage, and deliver content in an information world increasingly impacted by the Internet. Organizations with data in RDBMS software have already developed a structure and discipline that may be used to "wrap" content on the way out. While this approach doesn't work well for richly structured document content, it's an effective way to manage and deliver simpler, less complex data.

Take a Functional View of CM

The very nature of the term *content management* is often misunderstood. Content management is something one *does, not* something one buys or builds. This misunderstanding has created a CM marketplace in which every vendor claims to do CM, and many prospective users, with no working definition of what CM means to them, have no idea whether the claims are true or false, often with serious and costly results. Moreover, with the growing trend among Web management vendors to advertise themselves (often with little merit) as "end-to-end" content management providers, the stakes have become even higher and the penalties for error even more costly. Gartner, for example, categorizes CM systems into three functional groups:

1. ***Web content management:*** Managing content ready for delivery
2. ***Digital assets management:*** Controlling the source, consumption, and rights to content
3. ***Document content management:*** Managing the process of creation, review, and finalization of content for delivery

For the most part, vendors in one area don't understand or do a superlative job in the others, although you'd never know it from their presentations.

Develop a Detailed CMS Function List

A key best practice related to content management is to carefully investigate your own content life cycle, developing a detailed description of the functions required to fully support its information objectives. These descriptions will become the road map you need to confront a world in which many vendors want to sell you content management, but in which few are selling what's described above. Indeed, for any particular organization, there's no adequate definition of CM outside this list of required functions. While some CM functions – check-in and -out, and locking, for example – are common to most organizations, many other functions appear in differing patterns across organizations, even within the same industry. For instance, you should include in your CM function list such items as:

- *How your content is designed for presentation to your target audiences and media:* Your target deliverable will determine your optimum CM approach.
- *How you want your authors and contributors to mark and manage revisions:* Some CM architectures are incapable of developing a coherent record of revisions.
- *How you author and represent links and associations among portions of content:* Your CMS must be able to link at the levels you need, but not all products can.
- *How you support your authors' productivity and accuracy:* They spend only 5% of their time in the CMS, so they shouldn't have to fight it to get their jobs done.
- *How you capture, store, and communicate information about your content and its status:* Some CMS approaches are forced into a tortured process to link "metadata" to the content it describes.
- *How much and how soon you can expect your environment to evolve into something different from the form in which it was originally set up:* Scalability covers more than volume and traffic.

AUTHOR BIO

Barry A. Schaeffer is president of X.Systems, Inc. (www.xsystemsinc.com), a consulting and system development firm specializing in the conception and design of text-based information systems. A frequent speaker and contributor on subjects related to information management, Barry previously held management and technical positions with Bell, Xerox, Planning Research, and U.S. News & World Report.

Armed with a carefully developed functional CM list, you'll be able to plan and communicate with the vendor community from a position of strength. While most vendors will answer in the affirmative if the question is, "Do you offer CM?" the answer may be very different if the question includes a list of specific functions and the request that vendors sign up for each one or explain why they don't offer it and include their proposed alternative.

The most successful CM project I've ever seen, for example, used an RFI comprising nearly 400 pages. Among the 56 vendors contacted, over half opted out immediately after reading the requirements. The end result was a system that met every functional goal, supporting nearly a thousand authors producing over 200 information products in up to five media simultaneously.

Take a Holistic View of CM

Organizations seeking to establish CM in their environment often view it as a discrete component that can be acquired and dropped in, largely on its own merits and without consideration for other portions of their information life cycle. This isolation of CM from the information flow it's supposed to support can have serious implications for how (or whether) the resulting architecture works. Viewing CM as "plug and play" benefits the vendor community, so software salespeople virtually never raise this issue, preferring instead to get the sale and let the buyer deal with the resulting problems later.

A more holistic approach to information and its management would suggest that because value in the endeavor begins with the design and capture of rich content and matures in its easy and successful use by target audiences, management of the entire process is part of the content management equation. For example, functions like linking, reuse/variant management, and content modularization, although normally thought to be outside the realm of content management, will significantly impact and be impacted by the CM system acquired to support them. Moreover, the system's architectural characteristics will inevitably impact the ability of content contributors to perform their critical functions, and likewise the ability of delivery tools to organize and create the output products users want and will accept as successful.

> "Content management is something one does, not something one buys or builds"

Given this interdependence of phases in the information life cycle, CM tools should be evaluated and acquired only with a clear understanding of the complete set of challenges they will be asked to address. Because every tool is different and every manufacturer has personal technological preferences, you can't assume that because a system is advertised as "Content Management" it will meet a particular set of needs.

This approach, admittedly, makes for somewhat more work on the front end, but it pays rich dividends, not the least of which is a working environment that meets its own and its customers' needs. Most everyone has heard at least one horror story about what can happen if CM is acquired in a vacuum: poor performance, out-of-control costs, dissatisfied users and customers, even complete failure.

When Choosing Technology, Keep It Simple

Every design component and every tool acquired to support it presupposes a technological approach. Where CM is involved, a key technology component is the underlying repository in which the content is stored and on which the CM functionality is based. A valuable precept in this choice, inherent in any CM planning or acquisition, is simplicity. While technology can be argued in detail, in general the best approach, especially where CM is involved, is the simplest one (with thanks to William of Occam, who clearly stated this principle nearly 700 years ago).

There are several discrete approaches to the storage and processing of content; four of the most often encountered follow:

1. Fragment the content for storage in RDBMS tables, keeping metadata in other, linked, tables.
2. Store document content "as is" in database BLOBs (binary large objects), keeping some metadata in the content and some in linked DBMS tables.
3. Store document content on the file system, using RDBMS to store metadata.
4. Store XML content in a native XML repository, storing metadata in the content, keeping metadata in the content itself, and using DOM (Document Object Model) software to access both.

Each approach and the software based on it has characteristics and limitations that will materially affect its ability to properly support your CM requirements. Through application of the functional and holistic views of CM described above, it's possible to understand how a particular CM approach and product will function in your environment. That accomplished, you'll be able to choose the simplest, most straightforward option consistent with your needs. Moreover, coming to a choice on functional, holistic, and simplicity grounds will allow you to make the oft-required trade-offs between functionality and cost on a rational basis.

Recap

- Design your overall environment first; *you can't decide how to manage until you know what you're managing.*
- Flesh out your CM function list and send it to prospective vendors as an RFI; *you want their thoughts before you ask them to bid.*
- Ask vendors to detail how they plan to support your requirements; *only vendors with an idea of how to solve your problems will want to run this gauntlet.*
- Call in only those who return direct answers to your questions and ask them to explain their answers in even more detail; *don't allow canned demos, they always work…and almost always muddy the water; if you feel a demo is appropriate, ask each vendor to demo with your data and to show only those functions on your list.*
- Base your acquisition plans on which vendor does the best overall job of meeting your needs; *beyond prudence, don't worry about who is the largest or the most aggressive, but look carefully at a vendor's existing clients in circumstances similar to yours.*

> "While technology can be argued in detail, in general the best approach, especially where CM is involved, is the simplest one"

BARRY@XSYSTEMSINC.COM

WRITTEN BY **EUGENE KUZNETSOV**

XML-Aware Networking

XML as a family of new protocols

XML is, among many other things, a data-encoding standard for network protocols. What's known as "XML" in the community or the trade press is actually a large collection of protocols and data-handling systems that use XML-encoded packets or instructions.

SOAP, XML-RPC, UDDI, BXXP, XSLT/XPath, XML DSig, XML Enc, SAML, XSD – despite their disparate purposes and higher-level complexities, the one requirement they share is the need to parse, and process, XML. If XML is a new protocol, just as HTTP was in the Web's infancy, it's not hard to see that a new breed of intelligent network infrastructure can be created and harnessed to overcome XML's own growing pains. XML network equipment hardware is already being sold.

Keeping that in mind, this article provides an overview of some of the types of XML-aware network devices on the horizon and how they may be able to alleviate some of the barriers to XML's success.

Existing HTTP-Aware Network Equipment

The rapid growth of the Web in its early days quickly meant that performance and reliability needs escalated well past those offered by a single server. Server load balancing (SLB) provided the ability to have a group of servers appear as a single Web server to users. This virtualization of Web servers was done initially by special software deployed on the Web servers themselves. But SLB functionality moved very quickly from software into content-aware network hardware, both appliance- and switch-grade products delivered by companies such as F5 Networks, ArrowPoint Communications, and Alteon WebSystems. Hardware load balancers provided the performance and reliability demanded by the ever-expanding Web server farms running the Internet's largest sites.

In a typical installation, incoming page requests were load balanced among dozens of individual Web servers, and the load-balancing algorithms themselves quickly became more complex: URL switching, cookie redirection, and SSL ID tracking are a few examples. Web application developers became aware of load-balancer functionality, and sometimes the infrastructure's capability influenced application design so as to make the use of an SLB possible or more beneficial.

Both content switching and content caching grew up with the Web, evolving their features in response to the new applications and protocols, and enabling the tremendous growth of the Web. Content caching, load balancing, and other elements of the HTTP-aware network infrastructure helped overcome the early Web's performance, reliability, and security problems, and enabled it to be used for the kinds of applications (e.g., large news portals or online stores) that could fuel its growth even further. While SOAP and XML-RPC are built on top of HTTP and can therefore flow through these old load balancers and accelerators, the content-aware network equipment designed for Web traffic doesn't know anything about XML-encoded data or new protocols accompanying it. To be really useful for next-generation protocols, network devices must be "XML-aware," or capable of looking further up the network stack, past TCP, SSL, HTTP, and parsing the XML messages themselves.

XML-Aware Network Equipment

XML redirectors offer basic routing functionality for XML and are also sometimes referred to as XML *switches* or XML *routers*. Because they're the simplest kind of XML-aware "box," they were the first ones to become commercially available (Intel offered the now discontinued NetStructure XML redirector two years ago). Such a device would be placed between the servers and the incoming XML traffic and can be very useful in redirecting XML messages between different servers based on their content. For example, all purchase orders over $50,000 could be redirected to the fast application server, while all others could be sent to the slower server. In addition to handling this kind of trivial QoS (quality of service) issue, the same device could send messages in a particular XML vocabulary to the server capable of processing them, or it could send separate XML-RPC and SOAP messages. The routing rules are specified using either proprietary pattern-matching languages or a limited subset of XPath.

XML *accelerators* are network devices that aim to speed up either the flow or the processing of XML messages. XML processing performance can be a barrier to adoption of XML technologies for business-critical applications. An XML accelerator has the ability to offload one or more resource-intensive XML processing functions onto purpose-built hardware. For example, bottlenecks associated with XML parsing and XSLT are a frequent and well-known problem. XML accelerators can perform XSLT transformations outside the application and greatly improve transaction throughput and reduce page load times. Using standards such as xml-stylesheet processing instructions can make it very easy to drop an accelerator into an existing environment.

Other types of XML processing can also be performed "in the network," with an XML accelerator taking over the bottlenecked functionality from the application server. In order to take over processing, the accelerator can't implement just a subset of a specification, of course; it must be fully conformant to the standard.

To be considered an accelerator, the device should offer much better performance than a general-purpose server. In fact, the best accelerators will be quite unlike general-purpose servers: custom-built gear lacking hard disks, keyboard, or screens, and controlled through existing network management systems and interfaces. This is similar to SSL accelerators or Web caches, which have been speeding up Web sites and e-commerce applications for years by providing a single-purpose offload function. Since SOAP and other Web services protocols are all XML-based, their performance is intrinsically dependent on XML performance. Expect the XML acceleration space to get quite crowded with a number of very sophisticated devices offering better XSLT performance, high-speed schema validation, XML parsing, or other functions.

XML *firewalls* are true to their name and play the same role for XML traffic that traditional IP firewalls do for the rest of the enterprise network. Security is quickly becoming a hot topic with respect to XML Web services, and with good reason: if there's one issue that can stop or roll back the advance of XML messaging systems, it's security. An XML (or SOAP) firewall intercepts incoming and outgoing XML messages and has one or more of the following security functions:

- Performs XML well-formedness checks to screen out invalid XML
- Provides XML denial-of-service attack protection
- Validates against standard (e.g., SOAP) or custom user–specified XML schemas or DTDs
- Verifies XML signatures
- Encrypts sensitive message fields as necessary

It would take only one well-publicized security fiasco in a Web services app at a Fortune 500 company to stall the Web services efforts at the other 499. An XML firewall is on the wish list of many network administrators because it picks up where the existing firewall leaves off, and offers a measure of XML security and access control at the network level. Without it, his or her only options are either to attempt to block all XML network traffic or rely on the growing number of XML-enabled applications to be the sole layer of security. As a result, even today, deployment of externally facing Web services applications is often held back by concerns of the corporate network security group about opening up port 80 to incoming SOAP traffic, or just the very notion of offering RPC access to the back end of the enterprise's system.

Technical Challenges

Designing and building a truly useful XML-aware network device is an extremely difficult technical undertaking. Because enterprise network equipment is expected to function at wirespeed (at least Fast Ethernet or 100 megabits per second), the same is required of the XML processing core embedded in the device. Smart decisions also need to be made about what kind of XML functionality can be put into the network without introducing additional complexity into the overall architecture.

For example, manual setup of transformation rules can be cumbersome if not designed carefully. The complexity of configuration can be greatly eased by offering integration with XML development tools. A good XML network device should integrate seamlessly with application software, preserving XML standards compliance while delivering the kind of performance expected of network devices.

XML Network Services

It's important to note that the network infrastructure is composed of both network equipment and network services. Although XML-aware network equipment is much further along than network services, we can look forward to some changes in how complex applications are built. Looking again to the HTTP-aware network, during the Web's early growth it became clear that performance and reliability needs of the largest Web sites were well past those offered by one server cluster. Flash crowds spurred by major events could overwhelm any single network or server group, and the Internet's growing global reach meant even longer waits and growing bandwidth costs. Distributed edge caches moved content closer to users, reducing latency, load on origin servers, and ISP bandwidth costs all at once. Content Distribution Network (CDN) is a network service employing thousands of caching servers located at ISPs across the globe and first offered by startups: Akamai, Digital Island, and several others.

Similar kinds of "overlay" networks are now being created to offer guaranteed delivery of business documents or transactions through the Internet. Such a service is designed to allow the customer to send a purchase order "into the network cloud" and be assured that it will be delivered to its destination despite any failed network links, security challenges, or other disruptions.

Online machine-readable directories for XML Web services are another obvious network service, and it's no accident that some of the biggest existing network service providers are involved in UDDI and similar initiatives. The hope is that UDDI or a similar directory initiative can become a next-generation domain name system of sorts for Web services endpoints. Of course, just because a resource is publicly listed doesn't mean that it should be accessible to all or without charge. In the security and payment area, innovative startups are creating automated access brokering services based on signed cryptographic assertions, often using SAML, which could enable both pay-per-use XML Web services and new business models for existing content providers.

These are merely examples of the kinds of XML-aware network services an XML developer may be developing for in a year or so.

• • •

I've attempted to offer a view of XML messaging and Web services from a different perspective – not from the "top down" point of view of an application, but from the "bottom up" perspective of the network. These two points of view are always different enough to be interesting, and are sometimes even in conflict with each other. Yet, armed with the realization that XML Web services aren't just about applications, they're also about the network, you can look to the Web's early days for the role HTTP-aware network equipment and services played in overcoming the obstacles to its success.

While it may be too early to tell what kind of new "xml-aware" network gear or services will be most successful, XML developers should expect that the network infrastructure a year from now will have the next level of intelligence, with purpose-built "XML-aware" hardware for XML acceleration and security.

EUGENE@DATAPOWER.COM

AUTHOR BIO

Eugene Kuznetsov is the founder of DataPower Technology, Inc., a provider of XML-aware network infrastructure. Previously he led the Java JIT compiler effort for the Macintosh version of Microsoft Internet Explorer 4.0. Eugene has also been involved in three different optimized Java VM implementations.

XML PROTOCOLS

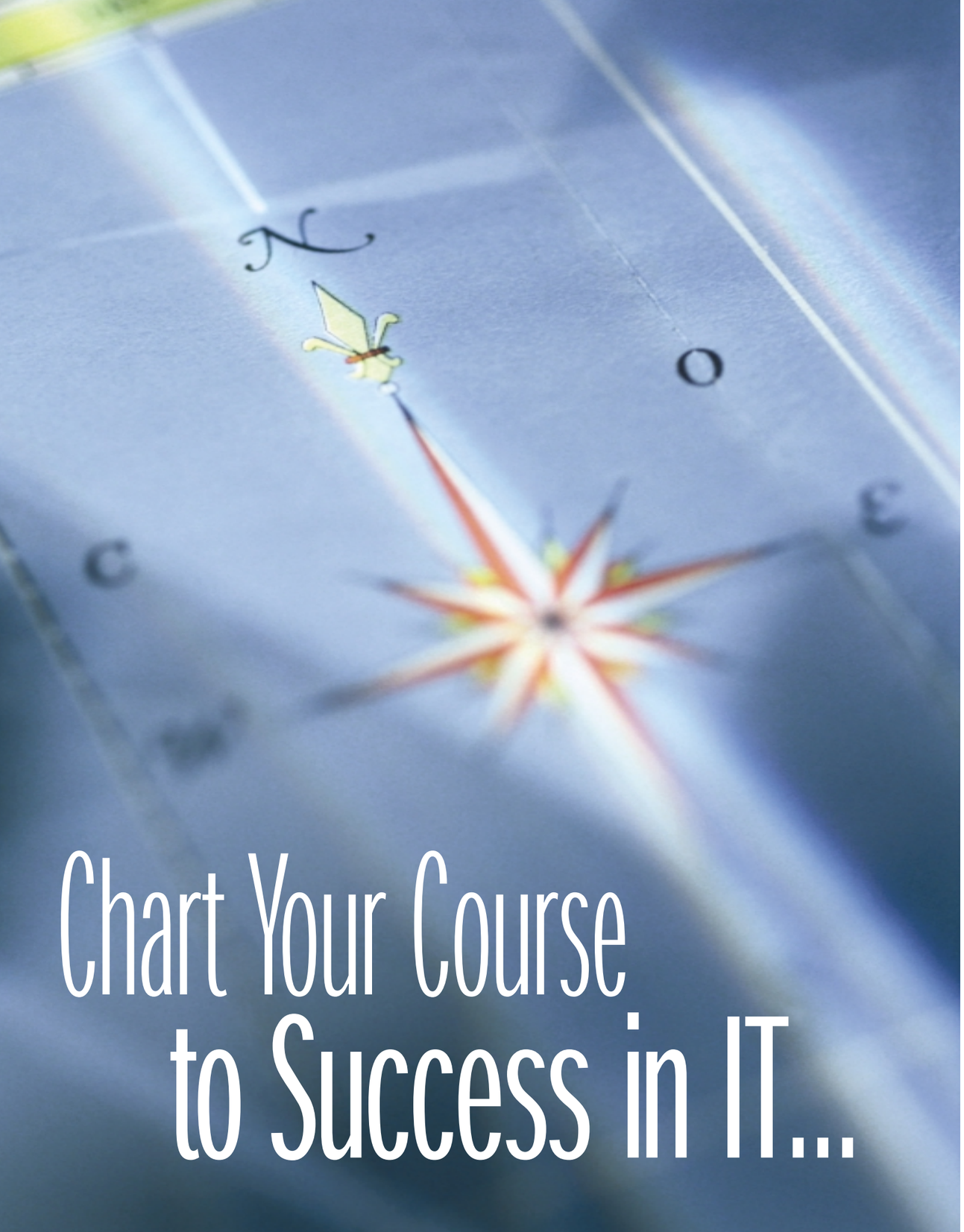
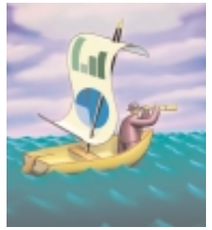


Chart Your Course to Success in IT...

...order your copy of

Java Trends: 2003

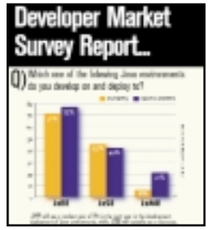
Don't go astray. In the vast sea of Internet technology, market conditions change constantly. Will Java remain the hot platform it is today? Will C# rapidly gain ground? What are the world's foremost Java developers aiming their sights toward? Which companies come to mind when planning their next project? How can their thinking direct your efforts?



EnginData Research has studied the IT industry extensively, spotting many key trends and alerting industry leaders along the way. In the first quarter of 2002, they embarked on their most challenging mission ever: the most in-depth study ever done of the Java development marketplace.

After months of analyzing and cross-referencing more than 10,500 comprehensive questionnaires, the results are now available.

EnginData's research is invaluable to leading IT vendors, integrators, Fortune 500 companies, trade-show organizers, publishers, and e-business organizations worldwide.



Here's just a sample of the critical data points the Java report delivers...

- ✔ Current and projected usage of languages and platforms
- ✔ Multiple rankings of hundreds of software vendors and tools
- ✔ Types of applications being developed
- ✔ Databases being used
- ✔ Purchasing patterns
- ✔ Company size
- ✔ Development and purchasing timelines
- ✔ Perceptions and usage of Web services
- ✔ Preferred Java IDE
- ✔ J2EE, J2ME, J2SE usage comparisons

As an IT specialist, **EnginData Research** focuses exclusively on three leading drivers in IT today – Java, Web services, and wireless development. Coming soon, our Web services survey will deliver such critical knowledge as:

- ✔ Time frame for developing Web services – enabled apps
- ✔ Percentage of apps with Web services today
- ✔ Sourcing and hosting Web services
- ✔ Perceived leaders in Web services tools and solutions

Navigate your company through the challenging IT marketplace with the trusted knowledge and intelligence of **EnginData Research**. Order your copy of the 2002–2003 Java market study by calling Margie Downs at 201-802-3082, or visiting our Web site.

engindata RESEARCH

www.engindata.com


WRITTEN BY **KLAUS FITTGES & MICHAEL CHAMPION**

Native XML Databases for Hierarchical Data

Preserving the value of XML's hierarchical structure

XML is widely accepted as the standard for describing data to be exchanged between applications and over the Web. But there is considerably less agreement on how it should be stored and queried.

XML and its predecessor, SGML, have been in existence for a long time without the world of "ordinary" data processing paying too much attention. XML brings with it a whole set of data modeling assumptions, processing mechanisms, and extensibility considerations that have proved useful in document-centric communities, but have had little mindshare in the database world for 20 years.

Hierarchical Structures and the XML Approach

Generally, XML addresses the demand for modeling hierarchical structures in a very elegant way. Modeling information in terms of a component/subcomponent model of the world is made very straightforward. XML elements are containers: they have the ability to contain information content as well as other containers. These containers can contain other containers, and so on.

XML enables its users to deal with highly recursive structures, well suited to concisely express complicated nested relations. Typically, "bill-of-material"–like structures are easily expressed via an XML data model. The containership model, in which containers contain other containers of the same type, has always been familiar to people working with document models, but isn't so familiar to "classical" database designers.

XML designers make use of its ability to deal with the "self-similarity" of document structures. Imagine a data model for scientific research papers that's intended to be open for any kind of publication. If a library user wants to search for a specific paper, it shouldn't matter whether the paper was published as a separate work, or in some institution's quarterly reports, or as part of a book or volumes of books. If we model this environment not with different element types ("tags") but with one type (e.g., "section"), we can express the actual mode of this section by an attribute (e.g., "section_type=book" or "section_type=chapter").

Another facet of extensibility is the ability of XML to deal with "open content." We can direct an XML application to ignore parts of an element. This is important when you deal with developing data models, in which one data model for a certain community is deployed to be further – and differently – developed in different areas of this community, only to be consolidated later in a more precise schema. Such communities could be internationalized standardization bodies as well as different developers' teams in large companies.

Storing XML

Thus the hierarchical structure of XML is an important part of its power to effectively model many real-world data management problems. As more and more data is created and transferred in XML form, it'll become increasingly important to store it in a scalable, queryable, robust manner, that is, in a database management system. How can you preserve the value of XML's hierarchical structure when storing and querying XML, given that the database industry has lost interest in hierarchical DBMS systems?

A number of options are now available:

- ***Store documents as BLOBs in the file system or in an RDBMS.***

This conserves the document in its entirety, but requires a time-consuming parsing process each time it's accessed. Metadata for this document and index information could be stored separately, but in order to find certain elements we'd have to parse again and again.

For the purpose of having XML documents directly available to applications such as Web services that need them – and in a reliable and transaction-oriented way – we can exclude file system–based solutions. When we deal with millions of documents in these environments, we need typical DBMS mechanisms such as backup, restore, recovery, locking, and transactionality. Also, DBMSs are characterized by their ability to handle most of the access cases (retrieval, queries, deletions) via powerful indexes rather than by inspecting the primary data ("table scan," in RDBMS lingo).

AUTHOR BIO

Klaus Fittges, CTO of Software AG, Inc. (USA), is one of the architects and a patent holder of Tamino, Software AG's native XML database. He is interested in data modeling, fault-tolerant searches and information retrieval.

Michael Champion, an R&D specialist at Software AG in Ann Arbor, MI, has been a software developer in the U.S. for more than 20 years. He currently works in the Technology Enablement group, focusing on technical business development activities, and represents Software AG on several W3C groups.

- ***Store the elements of a document as columns of tables in an RDBMS.***

This "shredding" process uses the results of the parsing to generate values for columns of tables in an RDBMS. While several products support this capability, it creates several difficulties: large schemas create a potentially huge number of linked tables, hierarchies have to be modeled by foreign-key relations, resulting in processor-intensive Join operations when reassembling the document; and by not storing the document inertially, the reassembled document can logically be in a different form than the one in which it originally arrived. RDBMSs have their own internal mechanisms that are suited for relational data but are weaknesses when it comes to XML.

- ***Store documents in an object-oriented DBMS.***

The OO approach should be expressive enough to easily map any XML structure into an OO model. In fact, some of the existing approaches for XML databases are based on object-oriented DBMSs (OODBMSs). As a practical matter, there are significant mismatches between the XML model and the OO approach. For example, some OODBMSs demand all objects to be available in memory, which limits scala-

HIERARCHICAL EXAMPLE

A classic example of a hierarchical structure is the "bill of materials" describing some object as a collection of parts, each of which is made up of smaller parts, until one gets to the most basic components.

Consider a very simple example in XML format that describes a computer system consisting of a computer, monitor, keyboard, and mouse. The computer consists of a motherboard, disk drive, and power supply, and the motherboard contains a CPU and memory.

```
<item model="L1234123">
<part desc="computer" partnum=
 "5423452345">
<part desc="motherboard" partnum=
 "5423452345">
<part desc="CPU" partnum=
 "6109486697">
</part>
<part desc="memory" partnum=
 "545454232">
</part>
</part>
<part desc="diskdrive" partnum=
 "6345634563456>
</part>
<part desc="powersupply" partnum=
 "098765343">
</part>
</part>
<part desc="monitor" partnum=
 "898234234">
</part>
<part desc="keyboard" partnum=
 "191986123">
</part>
<part desc="mouse" partnum= "98798734">
</part>
</item>
```

A description of a real computer, of course, would have more information on each part, far more parts, and more depth to the component hierarchy. And a product catalog would contain a collection of these item elements, each containing a different component hierarchy, but with many of the standard components appearing at different places in different items.

A classic challenge for SQL (addressed in the SQL 99 standard, but not widely implemented in a standard way) is to find all the top-level items that contain a specific part somewhere in the hierarchy. For example, we might want to find all systems that use the power supply with the part number 098765343. This is very easy in an XML system that supports XPath:

```
item//part [@descr="powersupply"
and @partnum="098765343"]
```

> Native XML databases generally use familiar XML mechanisms

bility. Likewise, there's a conceptual mismatch between the OO approach of encapsulating data behind access methods and the XML approach of exposing the data itself as a first-class "object." It's certainly possible to implement an XML store on top of OODBMS technology, but it's not a simple application.

- ***Store documents in a native XML database.***

Storing and indexing XML documents in their native form eliminates the problems related to the additional layers needed by the nonnative methods, especially the time- and space-consuming, processor-intensive chopping up of documents into separate tables and then putting them back together again. Having XML documents available in a native XML database generally increases the performance and reduces the complexity of applications that depend on the data.

A native XML database will generally provide an optimized search for the occurrence of hierarchical XML structures. For example, we might need to search through millions of financial stock recommendation documents in order to find the few *not* containing certain optional subelements (such as warnings).

Native XML databases generally use familiar XML mechanisms such as (1) DOM to provide the documents as objects in an OO language, and (2) XPath and/or XQuery to query the data or to specify which document(s) to delete or update. This minimizes the amount of proprietary information that a developer must learn to employ the database, and minimizes the "impedance mismatch" with other XML applications.

Native XML DBMSs Exploit XML's Hierarchical Structure

Any XML-enabled DBMS will provide what you'd expect from a database management system: a query and transaction language, a way to build indices and data structures, and interfaces for other applications. The more faithfully a DBMS can expose the hierarchical structure of XML, however, the better applications can exploit those situations in which the hierarchical model best fits the data.

1. Native XML DBMSs provide a method of traversing the data within an XML document in a way that coheres to the document's structure, that is, it's more than a string-matching function.
2. Native XML DBMSs can easily deal with data models defined in an XML schema language or a DTD, with no analysis of what "metadata" is or what needs to be efficiently queried.
3. Native XML DBMSs support recursion quite naturally; the "Bill of Materials Problem" is very challenging for SQL, but quite simple to handle in XPath/XQuery.
4. Since most "documents" are much easier to model as hierarchies than as linked tables, it's easier and more efficient in many document-oriented use cases to store the data in a native XML DBMS.

KLAUS.FITTGES@SOFTWAREAGUSA.COM

MICHAEL.CHAMPION@SOFTWAREAGUSA.COM

XML in the Enterprise

Dynamic SQL generation using XSLT

Written by **Mike Morris**

XML has gained wide acceptance in organizations large and small over the past several years. Part of the reason for this rapid adoption is that XML is intrinsically flexible and simple enough to be used in a variety of problem domains.

Vendor support for standards-based implementations of platform tools and utilities has paved the way for individuals and organizations to find new ways to apply this technology. This is further proof that sometimes the simplest of ideas, in this case self-describing data, are the most elegant and powerful.

The majority of XML applications today fall into several broad categories. These include application integration, business-to-business document exchange, and more recently Web services. Additionally, there are more specialized implementations of XML, such as VoiceXML and IBM's Bean Markup Language, that conform to specific vocabularies and have runtime engines that interpret the tags to drive tangible software processes.

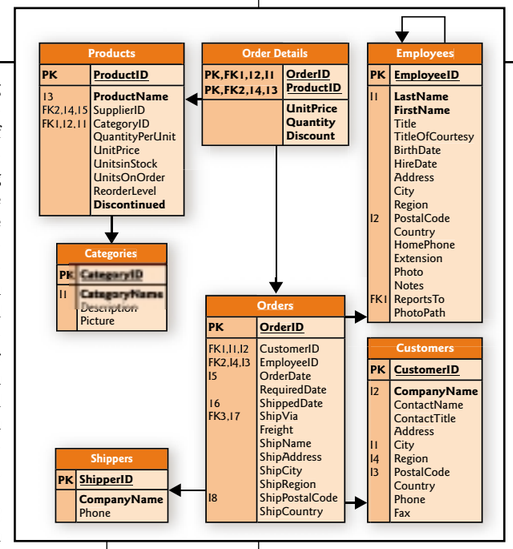


FIGURE 1 Northwind database

In spite of all the hype surrounding Web services and wireless application development, a large portion of new development continues to be traditional online transaction processing (OLTP) systems that form the backbone of corporate information systems. These systems usually have *n*-tier architectures with a browser-based user interface, some middle-tier business/data access components, and a relational database back-end. That's right, relational databases, not XML documents or XML-derivative databases. There are a multitude of reasons for using relational databases over these alternatives. It is a mature technology, handles concurrency issues well, supports transactions, and boasts robust security models.

One characteristic that most OLTP, data entry–type applications have in common is the requirement to query the database and retrieve some subset of data based on a user-defined filter. Many of these systems contain millions of rows of data and would be of little use if there weren't a straightforward way to get at these rows. This functionality is usually provided in the form of search/filter pages that allow the user to specify certain criteria. These criteria are then used as the basis for a dynamically created SQL statement that's executed against the database. As the number fields to query on increases, the number of possible SQL statements increases exponentially.

This is where XML and XSLT come into play.

The simple nature of XML and the recursive qualities of XSLT can greatly reduce the complexity of SQL statement generation and facilitate future maintenance as requirements change.

Search Form Example

The example uses a subset of the Northwind database that ships with Microsoft SQL Server. The partial schema is shown in Figure 1. The Northwind database provides a good reference implementation of a conventional OLTP order entry system. The requirement is to allow filtering of orders based on any combination of the following: Customer Name, Product Category, Product, Shipper, and Salesperson. To this end, Figure 2 depicts the Advanced Search form that will be used to specify the criteria. This page is an ASP.NET form that uses server-side controls to populate the various form controls with the actual reference data. This form and supporting files are included in the downloadable code.

Several steps must be performed to produce the results we want using XML and XSLT:

1. Serialize form data to XML.
2. Run XSLT to dynamically produce SQL statement.
3. Execute SQL command against RDBMs.
4. Return results.

The first two steps are of the most interest and are discussed in more detail below.

Serialize form data to XML

The first thing we need to do is convert the form data to XML. This can be done on the client side or the server side, depending on the specific environment, browser restrictions, or simply developer preference. For this example I've opted for a client-side implementation using a generic function named SerializeForm(). This code is available for download at www.sys-con.com/xml/source.cfm.

The downloadable code includes all files necessary to run the sample application cited in this article. There are options to view the XML generated by the client-side JavaScript, the SQL code generated via the XSL transformation, and finally the Orders that match the specified criteria.

The routine simply steps through the elements collection of the form, determines what type of tag it is, and calls the BuildTag() function to construct the actual tag. The root element for this XML fragment is <form>. You'll notice that the name property of the HTML form tag is used as the newly created tag name. Another note of interest is that each tag value is wrapped in a CDATA block, so we don't have to concern ourselves with manually converting &, <, and > characters to their entity reference equivalents. Once the form variables have been serialized, the resultant XML is placed into a hidden form variable and posted to the server.

Transform form data to produce SQL statement

At this point the XML is ready to be transformed into a SQL statement that can be run against the Northwind database. For this example we'll assume that we've specified the filter criteria as shown in Figure 2 and clicked the Perform Query button. The serialized XML version of the form is shown in Listing 1.

The XSLT used to produce the query is shown in Listing 2. Note that the <xsl:output> element specifies a method attribute of text since our output will be SQL, not HTML or XML. Let's analyze the structure of this document to get a better understanding of how this XSL produces the desired results.

There are three distinct operations being performed. The first is the specification of the SELECT clause. This is followed by the generation of the FROM clause and finally the WHERE clause.

SELECT clause generation

In our example the SELECT clause is static. We always want to retrieve the OrderID and the OrderDate. There's no reason why the SELECT clause can't be dynamically generated similarly to the FROM and WHERE clauses, but for our purposes this part of the XSL is straightforward.

FROM clause generation

As we get into the construction of the FROM clause, things get a bit more interesting. This is where we dynamically build our JOIN conditions. Since not all tables are included in every possible permutation of the query, we use <xsl:if> statements to determine their inclusion. For example, let's look at the following excerpt of our XSL:

```
<xsl:if test = "form/Salesperson != '0'">
  INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
</xsl:if>
```

What this piece of XSL does is check whether the user made a selection specifying orders placed by a particular Salesperson. If they specified the default, which is "All" (with an Option value of "0"), then the Employees table shouldn't be included in the FROM clause, as we don't care to filter on it and there will be no corresponding WHERE condition specified. If they have specified a Salesperson, we want the XSL to emit the appropriate INNER JOIN clause and relate the Employees table to the Orders table based on the EmployeeID column.

WHERE clause generation

The generation of the WHERE clause is similar to that of the FROM clause. A series of <xsl:if> statements determines whether a particular

Data needs to be imported into the database in the fastest, least disruptive way possible

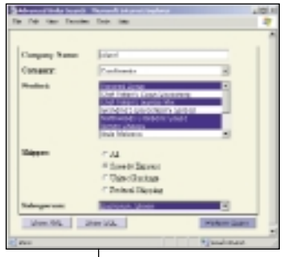


FIGURE 2 Search form

clause is included based on the value of a node or simply the existence of one. Take a look at the following excerpt:

```
 <xsl:if test = "Salesperson !=
   '0'">
    AND Employees.EmployeeID =
<xsl:value-of select=
 "Salesperson"/>
 </xsl:if>
```

This is the WHERE clause counterpart to the example cited in the FROM clause generation above. It states that if the user has selected anything other than the default "All", the condition needs to be included.

The question of which Products to include is slightly more complicated. This is because the form allows for multiple selections and there's no default value as in most of the other cases. The XSL fragment to do this is:

```
<xsl:if test = "count(//AvailableProducts) != 0">
   AND Products.ProductID IN
 (<xsl:for-each select = "AvailableProducts">
  <xsl:value-of select="node()" />
  <xsl:if test = "position() != count(//AvailableProducts)">,</xsl:if>
 </xsl:for-each>)
</xsl:if>
```

Once again we use a <xsl:if> element to determine inclusion in the resultant WHERE clause, but this time we don't check for a default value but rather the existence of one or more AvailableProducts nodes with the count() function. If one or more do exist, we step through the AvailableProducts nodes collection using a <xsl:for-each> element and build the IN condition. A peculiarity in building the IN clause is that possible values must be separated by commas. This is achieved by the line:

```
<xsl:if test = "position() != count(//AvailableProducts)">,</xsl:if>
```

This uses the position() function to determine whether we've reached the final AvailableProducts node, and if not, emits a comma. The end result of this XSL transformation is the SQL statement shown in Listing 3.

Additional Uses

The XSL dynamic SQL generation programming pattern isn't limited to the ad hoc search functionality discussed above. The techniques employed there are applicable in many other corporate data scenarios as well. One of these is the use of batch interfaces for importing and updating of relational data.

Batch data load and update

The batch interface is still an integral part of many RDBM systems. It remains a solid and proven way to integrate both legacy systems and outside vendor applications. More recent technologies such as message-oriented middleware have assumed some of this burden, but the task at hand remains the same. Data needs to be imported into the database in the fastest, least disruptive way possible. The traditional implementation is to write a custom program that reads the data file(s), constructs SQL strings, and executes the create, update, and delete actions. Many of the popular RDBMs in use today also have proprietary transformation utilities that use a combination of script code and declarative programming to facilitate this process.

If the data being imported is in XML format, the XSLT dynamic SQL generation pattern can provide a simpler, potentially platform-neutral alternative to the traditional mechanisms. Here's how.

Assume the following XML excerpt:

```
<customer>
  <id>6541</id>
  <name>BiblioPhile</name>
  <contact>John Quick</contact>
   <address>1344 Elm Street</address>
   ......
</customer>
```

This represents a fragment of the file exported from the fictitious Customer Relationship database on a nightly basis. This new customer needs to be imported into the Customers table of the Northwind database depicted in Figure 1. We can produce the corresponding INSERT statement(s) using the following XSL excerpt:

```
<xsl:template match="customer">
 INSERT INTO Customers (CustomerID, CompanyName, ContactName, Address)
 VALUES (<xsl:value-of select="id" />, '<xsl:value-of select="name" />',
         '<xsl:value-of select="contact" />', '<xsl:value-of select=
           "address" />' )
</xsl:template>
```

This example illustrates a very simple INSERT statement, but it doesn't take a broad stretch of the imagination to envision how a carefully crafted XSLT stylesheet could accommodate UPDATES and DELETES as well. Additionally, transaction and commit logic could also be embedded in the resultant SQL code, depending on the capabilities of your database.

Once the SQL file has been produced, it can be submitted in batch mode using the command-line processor of your database. For example, if the results of the XSLT were placed in a file named inserts.sql, the following command-line statement would submit the file for execution on a Microsoft SQL Server database with the results being output to results.txt:

```
osql /U sa /P /i inserts.sql /o results.txt
```

If this were used in conjunction with the scheduling tool bundled with the OS, along with the command-line utility of the XSLT processor, the whole import function could be performed without a line of compiled custom code or proprietary import utilities.

Summary

Many of the applications being built today that give companies a competitive advantage and facilitate day-to-day operations continue to be traditional browser-based solutions that communicate with relational database back-ends via server-side technologies such as JSP, ASP, and, more recently, ASP.NET. A general requirement for most, if not all, of these systems is a mechanism to intelligently filter content based on some user-specified criteria.

It's often been said that XML has become the lingua franca of the Web. This may be true but there is little doubt that SQL is the lingua franca of relational databases. This is a scenario in which the flexible nature of XML, in conjunction with the power of XSLT, can make creating these dynamic SQL statements easier to develop as well as maintain.

MIKEM@DEVELOPERBOX.COM

AUTHOR BIO

Mike Morris is cofounder of Developer Box LLC, a software development and consulting company located in Middletown, CT. Mike has been developing enterprise solutions for over 12 years on a variety of platforms.

LISTING 1

```
<form>
   <CompanyName><![CDATA[island]]></CompanyName>
   <Category><![CDATA[2]]></Category>
   <AvailableProducts><![CDATA[3]]></AvailableProducts>
   <AvailableProducts><![CDATA[5]]></AvailableProducts>
   <AvailableProducts><![CDATA[8]]></AvailableProducts>
   <AvailableProducts><![CDATA[15]]></AvailableProducts>
   <Shipper><![CDATA[1]]></Shipper>
   <Salesperson><![CDATA[5]]></Salesperson>
</form>
```

LISTING 2

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
 "http://www.w3.org/1999/XSL/Transform">
<xsl:output method = "text" />

 <xsl:template match="/">
     SELECT Orders.OrderID, Orders.OrderDate
     FROM Orders
     <xsl:call-template name = "BuildInnerJoins" />
     WHERE 1 = 1
     <xsl:apply-templates />
 </xsl:template>

 <xsl:template name="BuildInnerJoins">

  <xsl:if test="form/CompanyName != ''">
   INNER JOIN Customers ON Orders.CustomerID =
    Customers.CustomerID
  </xsl:if>

  <xsl:if test = "form/Salesperson != '0'">
   INNER JOIN Employees ON Orders.EmployeeID =
    Employees.EmployeeID
  </xsl:if>

  <xsl:if test = "count(form/AvailableProducts) !=
   0 or form/Category != '0'">
   INNER JOIN [Order Details] ON Orders.OrderID =
    [Order Details].OrderID
   INNER JOIN Products ON [Order Details].ProductID =
    Products.ProductID
  </xsl:if>

  <xsl:if test = "form/Category != '0'">
   INNER JOIN Categories ON Products.CategoryID =
    Categories.CategoryID
  </xsl:if>

 </xsl:template>

 <!-- This template builds the where clause -->
 <xsl:template match="form">

   <!-- Company Name -->

  <xsl:if test="CompanyName != ''">
   AND Customers.CompanyName LIKE '%<xsl:value-of select=
    "CompanyName"/>%'
  </xsl:if>

  <!-- Salesperson -->
  <xsl:if test = "Salesperson != '0'">
   AND Employees.EmployeeID =
    <xsl:value-of select="Salesperson"/>
  </xsl:if>

  <!-- Products -->
  <xsl:if test = "count(//AvailableProducts) != 0">
   AND Products.ProductID IN
  (<xsl:for-each select = "AvailableProducts">
   <xsl:value-of select="node()" />
   <xsl:if test = "position() !=
    count(//AvailableProducts)">, </xsl:if>
  </xsl:for-each>)
  </xsl:if>

  <!-- Product Category-->
  <xsl:if test = "Category != '0'">
   AND Categories.CategoryID = <xsl:value-of select="Category" />
  </xsl:if>

  <!-- Shipper-->
  <xsl:if test = "Shipper != '0'">
  AND Orders.ShipVia = <xsl:value-of select="Shipper" />

   </xsl:if>

 </xsl:template>

 <xsl:template match="*|text()|@*" >

 </xsl:template>

</xsl:stylesheet>
```

LISTING 3

```
SELECT Orders.OrderID, Orders.OrderDate
  FROM Orders
  INNER JOIN Customers ON Orders.CustomerID =
   Customers.CustomerID
  INNER JOIN Employees ON Orders.EmployeeID =
   Employees.EmployeeID
  INNER JOIN [Order Details] ON Orders.OrderID =
   [Order Details].OrderID
  INNER JOIN Products ON [Order Details].ProductID =
   Products.ProductID
  INNER JOIN Categories ON Products.CategoryID =
   Categories.CategoryID
  WHERE 1 = 1
  AND Customers.CompanyName LIKE '%island%'
  AND Employees.EmployeeID =  5
  AND Products.ProductID IN (3, 5, 8, 15)
  AND Categories.CategoryID =  2
  AND Orders.ShipVia =  1
```

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

FREE TUTORIAL
JUNE 27TH WITH WEB SERVICES EDGE
2002 EAST REGISTRATION. LIMITED OFFER!

TO REGISTER: www.sys-con.com or Call 201 802-3069

web services world tour EDGE 2002

$195 REGISTRATION FOR SYS-CON SUBSCRIBERS
BEST EDUCATIONAL VALUE FOR THE HOTTEST TECHNOLOGY SKILLS!

Take Your Career to the Next Level!

SEATING IS LIMITED. REGISTER NOW FOR THE CITY NEAREST YOU! WWW.SYS-CON.COM

Learn How to Create, Test and Deploy Enterprise-Class Web Services Applications

TAUGHT BY THE INNOVATORS AND THOUGHT LEADERS IN WEB SERVICES

EXPERT PRACTITIONERS TAKING AN APPLIED APPROACH WILL PRESENT TOPICS INCLUDING BASIC TECHNOLOGIES SUCH AS SOAP, WSDL, UDDI AND XML, PLUS MORE ADVANCED ISSUES SUCH AS SECURITY, EXPOSING LEGACY SYSTEMS AND REMOTE REFERENCES.

SPONSOR A CITY!

POSITION YOUR COMPANY AS A LEADER IN WEB SERVICES
Act today!
Call 201 802-3066 today to ask how.

SHARPEN YOUR PROFESSIONAL SKILLS.

KEEP UP WITH THE TECHNOLOGY EVOLUTION!

"Presented an excellent overview of Web services. Great inside knowledge of both the new and old protocols. Great insight into the code piece."
– Rodrigo Frontecilla

"Very articulate on the Web services SOAP topic and well-prepared for many questions. I've learned a lot from this seminar and I appreciate this seminar for my job. Thank you!"
– Kenneth Unpingco, Southern Wine & Spirits of America

"I liked the overview of Web services and the use of specific tools to display ways to distribute Web services. Good for getting up to speed on the concepts."
– B. Ashton, Stopjetlag.com

Echoed over and over by Web Services Edge World Tour Attendees:

"Good balance of theory and demonstration."

"Excellent scope and depth for my background at this time. Use of examples was good."

"It was tailored toward my needs as a novice to SOAP Web services – and they explained everything."

WHO SHOULD ATTEND:

- Architects
- Developers
- Programmers
- IS/IT Managers
- C-Level Executives
- *i*-Technology Professionals

IF YOU MISSED THESE...

BOSTON, MA (Boston Marriott Newton) SOLD OUT!
WASHINGTON, DC (Tysons Corner Marriott) SOLD OUT!
NEW YORK, NY (Doubletree Guest Suites) SOLD OUT!
SAN FRANCISCO, CA (Marriott San Francisco) SOLD OUT! CLASSES ADDED

BE SURE NOT TO MISS THESE...

Each city will be sponsored by a leading Web services company

...COMING TO A CITY NEAR YOU

2002

SEATTLE	AUGUST 27
AUSTIN	SEPTEMBER 10
LOS ANGELES	SEPTEMBER 19
SAN JOSE AT WEBSERVICES EDGE 2002: WEST	OCTOBER 8
CHICAGO	OCTOBER 17
ATLANTA	OCTOBER 29
MINNEAPOLIS	NOVEMBER 7
NEW YORK	NOVEMBER 18
SAN FRANCISCO	DECEMBER 3
BOSTON	DECEMBER 12

2003

CHARLOTTE	JANUARY 7
MIAMI	JANUARY 14
DALLAS	FEBRUARY 4
BALTIMORE	FEBRUARY 20
BOSTON	MARCH 11

REGISTER WITH A COLLEAGUE AND SAVE 15% OFF THE LOWEST REGISTRATION FEE.

TOPICS HAVE INCLUDED:
Developing SOAP Web Services
Architecting J2EE Web Services

On May 13th, the San Francisco tutorial drew a record 601 registrations.

SYS-CON *i*-TECHNOLOGY EDUCATION

REGISTRATION FOR EACH CITY CLOSES THREE BUSINESS DAYS BEFORE EACH TUTORIAL DATE. DON'T DELAY. SEATING IS LIMITED.
NON-SUBSCRIBERS: REGISTER FOR $245 AND RECEIVE THREE FREE ONE-YEAR SUBSCRIPTIONS TO *WEB SERVICES JOURNAL*, *JAVA DEVELOPER'S JOURNAL*, AND *XML-JOURNAL*, PLUS YOUR CHOICE OF *BEA WEBLOGIC DEVELOPER'S JOURNAL* OR *WEBSPHERE DEVELOPER'S JOURNAL*, A $345 VALUE!

TO REGISTER: www.sys-con.com or Call 201 802-3069

WRITTEN BY **RON SCHMELZER & TRAVIS VANDERSYPEN**

MANAGING YOUR XML DOCUMENTS WITH SCHEMAS

A powerful – and flexible – way to validate XML documents

THE XML SCHEMA DEFINITION LANGUAGE SOLVES A NUMBER OF PROBLEMS POSED WITH DOCUMENT TYPE DEFINITIONS. BECAUSE DTDS PROMPTED MUCH CONFUSION AND COMPLAINING AMONG XML DEVELOPERS, THE W3C SET ABOUT CREATING A NEW STANDARD FOR DEFINING A DOCUMENT'S STRUCTURE.

WHAT THE W3C CREATED IS SOMETHING EVEN MORE COMPLEX AND FLEXIBLE THAN DTDS: THE XML SCHEMA DEFINITION LANGUAGE. IN THIS ARTICLE WE'LL LOOK AT MANY ASPECTS OF SCHEMAS AND HOW YOU CAN BUILD AND USE THEM.

A Little Background

Schemas, while more complex than DTDs, give an individual much more power and control over how XML documents are validated. For instance, with the new W3C standard a document definition can specify the data type of an element's contents, the range of values for elements, the minimum as well as maximum number of times an element may occur, annotations to schemas, and much more.

In May of 2001 the W3C finalized its recommendation for the XML Schema Definition Language. This standard allows an author to define simple and complex elements as well as the rules governing how those elements and their attributes may show up within an instance document. The author has a large amount of control over how the structure of a conforming XML document must be created. The author can apply various restrictions to the elements and attributes within the document, from specifying the length to specifying an enumerated set of acceptable values for the element or attribute. With the XML Schema Definition Language, an XML schema author possesses an incredible amount of control over the conformance of an associated XML document to the specified schema.

Sample XML Document

The remainder of this article is devoted to creating and understanding the XML schema for the XML document shown in Listing 1, which details a purchase order for various items that can commonly be found in a grocery store. This document allows one individual to receive the shipment of the goods and an entirely different individual to pay for the purchase. This document also contains specific information about the products ordered, such as how much each product cost, how many were ordered, and so on.

As you can see, the listing represents a fairly small and simple order that could be placed online. It contains the necessary information regarding how payment is to be made, how the order is to be shipped, and what day delivery should be. The listing should by no means be construed as an all-inclusive document for an online grocery store order; it has been constructed only for use as an example.

For the listing, an author might construct a DTD to describe the XML document. While such a DTD might require only 30 lines or so, it would provide a relatively inflexible definition of the XML document.

A Sample Schema

Creating an XML schema to describe this document is somewhat more complex than building a DTD. However, in exchange for the extra complexity, the schema gives the author virtually limitless control over how an XML document can be validated against it.

Authoring an XML schema consists of declaring elements and attributes as well as the "properties" of those elements and attributes. We will begin our look at authoring XML schemas by working our way from the least complex to the most complex example. Because attributes may not contain other attributes or elements, we will start there.

Declaring attributes

Attributes in an XML document are contained by elements. To indicate that a complex element has an attribute, use the <attribute> element of the XML Schema Definition Language. For instance, Listing 2 is from a hypothetical PurchaseOrder schema based on the XML document shown in Listing 1. You can see the basics for declaring an attribute.

From this you can see that, when declaring an attribute, you must specify a type. This type must be one of the simple types: anyURI, base64Binary, boolean, byte, date, dateTime, decimal, double, duration, ENTITIES, ENTITY, float, gDay, gMonth, gMonthDay, gYear, gYearMonth, hexBinary, ID, IDREF, IDREFS, int, integer, language, long, Name, NCName, negativeInteger, NMTOKEN, NMTOKENS, nonNegativeInteger, nonPositiveInteger, normalizedString, NOTATION, positiveInteger, QName, short, string, time, token, unsignedByte, unsignedInt, unsignedLong, unsignedShort. Each type can be further categorized as a "primitive" data type or a "derived" data type. The derived data types are "primitive" or other "derived" data types with restrictions placed on them, such as integer, positiveInteger, and byte.

From the simple types you may notice what appears to be a group of duplicate or unnecessary types, such as nonNegativeInteger and positiveInteger. If you look closely, you'll see that nonNegativeInteger is an integer whose value is greater than or equal to zero, whereas the positiveInteger type is an integer whose value is greater than zero, which means a positiveInteger type cannot be zero. Keep this in mind when deciding on the base data type for your elements and attributes – these small details can greatly influence their acceptable value ranges.

Aside from defining the type of an attribute, the <attribute> element within the XML Schema Definition Language contains attributes to assist in defining when an attribute is optional, whether its value is fixed, what its default value is, and so on. Here's the basic syntax for the <attribute> element:

```
<attribute name="" type="" [use=""] [fixed=""] [default=""] [ref=""]/>
```

The use attribute can contain one of the following possible values:

- ***Optional***
- ***Prohibited***
- ***Required***

If the use attribute is set to required, the parent element must have the attribute; otherwise the document will be considered invalid. A value of optional indicates the attribute may or may not occur in the document and the attribute may contain any value. By assigning a value of prohibited to the use attribute, you can indicate that the attribute may not appear at all within the parent element.

Specifying a value for the default attribute indicates that if the attribute does not appear within the specified element of the XML document, it is assumed to have the value. A value within the fixed attribute indicates the attribute has a constant value.

It's important to remember that if you specify a value for the fixed attribute of the <attribute> element, the resulting attribute must have the value specified for the attribute to be valid. If you mean to indicate that the attribute should have a default value of some sort, use the default attribute instead. It should be noted that the default and fixed attributes are mutually exclusive.

The ref attribute for the <attribute> element indicates that the attribute declaration exists somewhere else within the schema. This allows complex attribute declarations to be defined once and referenced when necessary. For instance, let's say you've "inherited" elements and attributes from another schema and would simply like to reuse one of the attribute declarations within the current schema; this would provide the perfect opportunity to take advantage of the ref attribute.

Just as attributes can be defined based on the simple data types included in the XML Schema Definition Language, they can also be defined based on <simpleType> elements. This can easily be accomplished by declaring an attribute that contains a <simpleType> element, as the following example demonstrates:

```
<xsd:attribute name="exampleattribute">
   <xsd:simpleType base="string">
      <xsd:length value="2"/>
   </xsd:simpleType>
</xsd:attribute>

<xsd:complexType name="exampleelement">
   <xsd:attribute ref="exampleattribute"/>
</xsd:complexType>
```

From this example you can see that the XML Schema Definition Language gives the schema author a great deal of control over how attributes are validated. One of the wonderful side effects of the XML Schema Definition Language is the similarity to object-oriented programming. Consider each attribute definition and element definition to be a class definition. These class definitions describe complex structures and behaviors among various different classes, so each individual class definition, whether it's a simple class or complex class, encapsulates everything necessary to perform its job. The same holds true for the declaration of attributes and elements within an XML document. Each item completely describes itself.

Declaring elements

Elements within an XML schema can be declared using the <element> element from the XML Schema Definition Language. The example in Listing 3 shows a simple element declaration using the XML Schema Definition Language.

From the example you can see that an element's type may be defined elsewhere within the schema. The location at which an element is defined determines certain characteristics about its availability within the schema. For instance, an element defined as a child of the <schema> element can be referenced anywhere within the schema document, whereas an element that is defined when it's declared can have that definition used only once.

An element's type can be defined with a <complexType> element, a <simpleType> element, a <complexContent> element, or a <simpleContent> element. The validation requirements for the document will influence the choice of an element's type. For instance, going back to our object-oriented analogy, let's say you define a high-level abstract class and then need to refine its definition for certain situations. In that case you would create a new class based on the existing one and change its definition as needed. The <complexContent> and <simpleContent> elements work much the same way: they provide a way to extend or restrict the existing simple or complex type definition as needed by the specific instance of the element declaration.

The basic construction of an element declaration using the <element> element within the XML Schema Definition Language is as follows:

```
<element name="" [type=""] [abstract=""] [block=""]  [default=""]
 [final=""] [fixed=""] [minOccurs=""]  [maxOccurs=""]
 [nillable=""] [ref=""]  [substitutionGroup=""]/>
```

From this you can see that element declarations offer a myriad of possibilities to the author. For instance, the abstract attribute indicates whether the element being declared may show up directly within the XML document. If this attribute is true, the declared element may not show up directly. Instead, this element must be referenced by another element using the substitutionGroup attribute. This substitution works only if the element utilizing the substitutionGroup attribute occurs directly beneath the <schema> element.

In other words, for one element declaration to be substituted for another, the element using the substitutionGroup attribute must be a top-level element. Why would anyone in his right mind declare an element as abstract? The answer is really quite simple. Let's say you need to have multiple elements that have the same basic values specified for the attributes on the <element> element. A <complexType> element definition does not allow for those attributes. So, rather than define and set those attribute values for each element, you could make an "abstract" element declaration, set the values once, and substitute the abstract element definition as needed.

You may omit the type attribute from the <element> element, but you should have either the ref attribute or the substitutionGroup attribute specified.

AUTHOR BIOS

Ron Schmelzer is founder and senior analyst of ZapThink. A well-known expert in the field of XML and XML-based standards and initiatives, Ron has been featured in and written for periodicals and has spoken on the subject of XML at numerous industry conferences.

Travis Vandersypen, a programmer with EPS Software Corporation, has five years' development experience in XML, UML, XSLT, FoxPro, HTML, and other tools. He has authored a number of articles and is a frequent speaker at conferences.

This article is based on Chapter 4 of XML and Web Services Unleashed, *by Ron Schmelzer, Travis Vandersypen, et al., published (2002) by Sams Publishing.*

The type attribute indicates that the element should be based on a complexType, simpleType, complexContent, or simpleContent element definition. By defining an element's structure using one of these other elements, the author can gain an incredible amount of control over the element's definition. We will cover these various element definitions in the "Declaring Complex Elements" and "Declaring Simple Types" sections later in this article.

The block attribute prevents any element with the specified derivation type from being used in place of the element. The block attribute may contain any of the following values:

- #all
- extension
- restriction
- substitution

If the value #all is specified within the block attribute, no elements derived from this element declaration may appear in place of this element. A value of extension prevents any element whose definition has been derived by extension from appearing in place on this element. If a value of restriction is assigned, an element derived by restriction from this element declaration is prevented from appearing in place of this element. Finally, a value of substitution indicates that an element derived through substitution cannot be used in place of this element.

The default attribute may be specified only for an element based on a simpleType or whose content is text only. This attribute assigns a default value to an element.

You cannot specify a value for both a default attribute and a fixed attribute; they are mutually exclusive. Also, if the element definition is based on a simpleType, the value must be a valid type of the data type.

The minOccurs and maxOccurs attributes specify the minimum and maximum number of times this element may appear within a valid XML document. Although you may explicitly set these attributes, they are not required. To indicate that an element's appearance within the parent element is optional, set the minOccurs attribute to 0. To indicate that the element may occur an unlimited number of times within the parent element, set the maxOccurs attribute to the string "unbounded". However, you may not specify the minOccurs attribute for an element whose parent element is the <schema> element.

The nillable attribute indicates whether an explicit NULL value can be assigned to the element. If this particular attribute is omitted, it is assumed to be false. If this attribute has a value of true, the nil attribute for the element will be true. So what exactly does this do for you, this nillable attribute? Well, let's say you are writing an application that uses a database that supports NULL values for fields and you are representing your data as XML. Now let's say you request the data from your database and convert it into some XML grammar. How do you tell the difference between those elements that are empty and those elements that are NULL? That's where the nillable attribute comes into play. By appending an attribute of nil to the element, you can tell whether it is empty or is actually NULL. Remember, the nillable attribute applies only to an element's contents and not the attributes of the element.

The fixed attribute specifies that the element has a constant, predetermined value. This attribute applies only to those elements whose type definitions are based on simpleType or whose content is text only.

Declaring complex elements

Many times within an XML document an element may contain child elements and/or attributes. To indicate this within the XML Schema Definition Language, you'll use the <complexType> element. If you examine the sample section from Listing 4, you'll see the basics used to define a complex element within an XML schema.

The sample section specifies the definition of PurchaseOrderType. This particular element contains three child elements – ShippingInformation, BillingInformation, and Order – as well as two attributes – Tax and Total. You should also notice the use of the maxOccurs and minOccurs attributes on the element declarations. With a value of 1 indicated for both attributes, the element declarations specify that they must occur one time within the PurchaseOrderType element.

The basic syntax for the <complexType> element is as follows:

```
<xsd:complexType name='' [abstract=''] [base=''] [block='']
 [final=''] [mixed='']/>
```

The abstract attribute indicates whether an element may define its content directly from this type definition or from a type derived from this type definition. If this attribute is true, an element must define its content from a derived type definition. If this attribute is omitted or its value is false, an element may define its content directly based on this type definition.

The base attribute specifies the data type for the element. This attribute may hold any value from the included simple XML data types.

The block attribute indicates what types of derivation are prevented for this element definition. This attribute can contain any of the following values:

- #all
- extension
- restriction

A value of #all prevents all complex types derived from this type definition from being used in place of this type definition. A value of extension prevents complex type definitions derived through extension from being used in place of this type definition. Assigning a value of restriction prevents a complex type definition derived through restriction from being used in place of this type definition. If this attribute is omitted, any type definition derived from this type definition may be used in place of this type definition.

The mixed attribute indicates whether character data is permitted to appear between the child elements of this type definition. If this attribute is false or is omitted, no character may appear. If the type definition contains a simpleContent type element, this value must be false. If the complexContent element appears as a child element, the mixed attribute on the complexContent element can override the value specified in the current type definition.

A <complexType> element in the XML Schema Definition Language may contain only one of the following elements:

- all
- choice
- complexContent
- group
- sequence
- simpleContent

Declaring simple types

Sometimes it's not necessary to declare a complex element type within an XML schema. In these cases you can use the <simpleType> element of the XML Schema Definition Language. These element type definitions support an element based on the simple XML data types or any simpleType declaration within the current schema. For example, consider the following example:

```
<xsd:simpleType name="PaymentMethodType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Check"/>
    <xsd:enumeration value="Cash"/>
    <xsd:enumeration value="Credit Card"/>
    <xsd:enumeration value="Debit Card"/>
    <xsd:enumeration value="Other"/>
  </xsd:restriction>
</xsd:simpleType>
```

This type definition defines the PaymentMethodType element definition, which is based on the string data type included in the XML Schema Definition Language. You may notice the use of the <enumeration> element. This particular element is referred to as a *facet*, which we'll cover in the next section.

The basic syntax for defining a simpleType element definition is as follows:

```
<xsd:simpleType name=''>
   <xsd:restriction base=''/>
</xsd:simpleType>
```

The base attribute type may contain any simple XML data type or any simpleType declared within the schema. Specifying the value of this attribute determines the type of data it may contain. A simpleType may contain only a value, not other elements or attributes.

You may also notice the inclusion of the <restriction> element. This is probably the most common method in which to declare types, and it helps to set more stringent boundaries on the values an element or attribute based on this type definition may hold. So, to indicate that a type definition's value may hold only string values, you would declare a type definition as follows:

```
<xsd:simpleType name='mySimpleType'>
   <xsd:restriction base='xsd:string'/>
</xsd:simpleType>
```

Two other methods are available to an XML schema author to "refine" a simple type definition: <list> and <union>. The <list> element allows an element or attribute based on the type definition to contain a list of values of a specified simple data type. The <union> element allows you to combine two or more simple type definitions to create a collection of values.

Putting It All Together

Now let's look at Listing 5, a complete schema for the document shown in Listing 1. You may notice the use of the <xsd:choice> element. This element can be used to indicate when one of a group of elements or attributes may show up, but not all, as is the case with the DeliveryDate and BillingDate attributes. Also, notice the use of the xsd namespace. This namespace can be anything, but we'll use xsd to indicate an XML Schema Definition Language element.

As we indicated earlier, the listing is substantially more complex than a DTD would be, but it provides much better control over your XML document. There are many additional facets to an XML schema, but the information and examples here should be enough to get your feet wet.

Summary

The XML Schema Definition Language provides a very powerful and flexible way in which to validate XML documents. It includes everything from declaring elements and attributes to "inheriting" elements from other schemas, from defining complex element definitions to defining restrictions for even the simplest of data types. This gives the XML schema author such control over specifying a valid construction for an XML document that there is almost nothing that cannot be defined with an XML schema.

Further Reading

- Schmelzer, R., Vandersypen T., et al. (2002). *XML and Web Services Unleashed*. Sams Publishing.
- Savourel, Y. (2001). *XML Internationalization and Localization*. Sams Publishing.
- Rambhia, A.M. (2002). *XML Distributed Systems Design*. Sams Publishing.

RSCHMELZER@ZAPTHINK.COM

TRAVIS@DILMAD.COM

LISTING 1 PurchaseOrder.xml

```
<PurchaseOrder Tax="5.76" Total="75.77">

  <ShippingInformation>
    <Name>Dillon Larsen</Name>
    <Address>
      <Street>123 Jones Rd.</Street>
      <City>Houston</City>
      <State>TX</State>
      <Zip>77381</Zip>
    </Address>
    <Method>USPS</Method>
    <DeliveryDate>2001-08-12</DeliveryDate>
  </ShippingInformation>

  <BillingInformation>
    <Name>Madi Larsen</Name>
    <Address>
      <Street>123 Jones Rd.</Street>
      <City>Houston</City>
      <State>TX</State>
      <Zip>77381</Zip>
    </Address>
    <PaymentMethod>Credit Card</PaymentMethod>
    <BillingDate>2001-08-09</BillingDate>
  </BillingInformation>

  <Order SubTotal="70.01" ItemsSold="17">
    <Product Name="Baby Swiss" Id="702890" Price="2.89"
     Quantity="1"/>
    <Product Name="Hard Salami" Id="302340" Price="2.34"
     Quantity="1"/>
    <Product Name="Turkey" Id="905800" Price="5.80"
     Quantity="1"/>
    <Product Name="Caesar Salad" Id="991687" Price="2.38"
     Quantity="2"/>
    <Product Name="Chicken Strips" Id="133382" Price="2.50"
     Quantity="1"/>
    <Product Name="Bread" Id="298678" Price="1.08"
     Quantity="1"/>
    <Product Name="Rolls" Id="002399" Price="2.24"
     Quantity="1"/>
    <Product Name="Cereal" Id="066510" Price="2.18"
     Quantity="1"/>
    <Product Name="Jalapenos" Id="101005" Price="1.97"
     Quantity="1"/>
    <Product Name="Tuna" Id="000118" Price="0.92"
     Quantity="3"/>
    <Product Name="Mayonnaise" Id="126860" Price="1.98"
     Quantity="1"/>
    <Product Name="Top Sirloin" Id="290502" Price="9.97"
     Quantity="2"/>
    <Product Name="Soup" Id="001254" Price="1.33"
     Quantity="1"/>
    <Product Name="Granola Bar" Id="026460" Price="2.14"
     Quantity="2"/>
    <Product Name="Chocolate Milk" Id="024620" Price="1.58"
     Quantity="2"/>
    <Product Name="Spaghetti" Id="000265" Price="1.98"
     Quantity="1"/>
    <Product Name="Laundry Detergent" Id="148202"
     Price="8.82" Quantity="1"/>
  </Order>
</PurchaseOrder>
```

LISTING 2 Excerpt from hypothetical PurchaseOrder schema

```
<xsd:complexType name="ProductType">
  <xsd:attribute name="Name" type="xsd:string"/>
  <xsd:attribute name="Id" type="xsd:positiveInteger"/>
  <xsd:attribute name="Price">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Quantity" type=
   "xsd:positiveInteger"/>
</xsd:complexType>
```

LISTING 3 Simple element declaration

```
<xsd:element name='PurchaseOrder' type=
 'PurchaseOrderType'/>

<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
    <xsd:element name="ShippingInformation" type=
     "InfoType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="BillingInformation" type="InfoType"
     minOccurs="1" maxOccurs="1"/>
    <xsd:element name="Order" type="OrderType" minOccurs=
     "1" maxOccurs="1"/>
  </xsd:all>
  <xsd:attribute name="Tax">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Total">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
```

LISTING 4 Complex element

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
    <xsd:element name="ShippingInformation" type=
     "InfoType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="BillingInformation" type="InfoType"
     minOccurs="1" maxOccurs="1"/>
    <xsd:element name="Order" type="OrderType" minOccurs=
     "1" maxOccurs="1"/>
  </xsd:all>
  <xsd:attribute name="Tax">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="Total">
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
```

LISTING 5 Complete schema for Listing 1 document

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

  <xsd:annotation>
    <xsd:documentation>
      Purchase Order schema for an online grocery store.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="PurchaseOrder" type=
   "PurchaseOrderType"/>

  <xsd:complexType name="PurchaseOrderType">
    <xsd:all>
      <xsd:element name="ShippingInformation" type=
       "InfoType" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="BillingInformation" type=
       "InfoType" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="Order" type="OrderType" minOccurs=
       "1" maxOccurs="1"/>
    </xsd:all>
    <xsd:attribute name="Tax">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Total">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>

  <xsd:group name="ShippingInfoGroup">
    <xsd:all>
      <xsd:element name="DeliveryDate" type="DateType"/>
      <xsd:element name="Method" type="DeliveryMethodType"/>
    </xsd:all>
  </xsd:group>

  <xsd:group name="BillingInfoGroup">
    <xsd:all>
      <xsd:element name="BillingDate" type="DateType"/>
      <xsd:element name="PaymentMethod" type=
       "PaymentMethodType"/>
    </xsd:all>
  </xsd:group>

  <xsd:complexType name="InfoType">
    <xsd:sequence>
      <xsd:element name="Name" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="Address" type="AddressType" minOccurs=
       "1" maxOccurs="1"/>
      <xsd:choice minOccurs="1" maxOccurs="1">
        <xsd:group ref="BillingInfoGroup"/>
        <xsd:group ref="ShippingInfoGroup"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="DateType">
    <xsd:restriction base="xsd:date"/>
  </xsd:simpleType>

  <xsd:simpleType name="DeliveryMethodType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="USPS"/>
      <xsd:enumeration value="UPS"/>
      <xsd:enumeration value="FedEx"/>
      <xsd:enumeration value="DHL"/>
      <xsd:enumeration value="Other"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="PaymentMethodType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Check"/>
      <xsd:enumeration value="Cash"/>
      <xsd:enumeration value="Credit Card"/>
      <xsd:enumeration value="Debit Card"/>
      <xsd:enumeration value="Other"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="AddressType">
    <xsd:all>
      <xsd:element name="Street" minOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="City" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="State" type="StateType" minOccurs=
       "1" maxOccurs="1"/>
      <xsd:element name="Zip" type="ZipType" minOccurs="1"
       maxOccurs="1"/>
    </xsd:all>
  </xsd:complexType>

  <xsd:simpleType name="ZipType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="5"/>
      <xsd:maxLength value="10"/>
      <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="StateType">
    <xsd:restriction base="xsd:string">
      <xsd:length value="2"/>
      <xsd:enumeration value="AR"/>
      <xsd:enumeration value="LA"/>
      <xsd:enumeration value="MS"/>
      <xsd:enumeration value="OK"/>
      <xsd:enumeration value="TX"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="OrderType">
    <xsd:sequence>
      <xsd:element name="Product" type="ProductType" minOccurs=
       "1" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="SubTotal">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="ItemsSold" type=
     "xsd:positiveInteger"/>
  </xsd:complexType>

  <xsd:complexType name="ProductType">
    <xsd:attribute name="Name" type="xsd:string"/>
    <xsd:attribute name="Id" type="xsd:positiveInteger"/>
    <xsd:attribute name="Price">
      <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
          <xsd:fractionDigits value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Quantity" type=
     "xsd:positiveInteger"/>
  </xsd:complexType>

</xsd:schema>
```

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

Model-Driven Programming Using XSLT

WRITTEN BY SOUMEN SARKAR

AN APPROACH TO RAPID DEVELOPMENT OF DOMAIN-SPECIFIC PROGRAM GENERATORS

Model-driven programming is a software development paradigm that strives to bring out the abstract model manipulation that we're trying to achieve through a body of programming language code. This approach focuses first on *what* is being achieved in a computing system and then on *how* it's being achieved. It's the responsibility of a software subsystem to translate the *what* to *how*.

Typically, the purpose of the software subsystem is to generate a concrete implementation from declarative models. This could be viewed as an extension of MVC (Model-View-Controller) architecture by incorporating a generator component (i.e., MVCG).

Adopting a generative approach in software development is a goal cherished by many application developers. Why write repetitive code when a single source of specification can generate the needed variations? Generic programming partially addresses this by allowing us to write template-based code in which the variability is expressed in the template parameter. For example, in C++ Standard Template Library (STL), the concepts of collections, iterators, and algorithms operating over iterators are all implemented in terms of parametric types T_i. When we use STL with a concrete type such as a C++ class, the code is generated by a C++ compiler/linker. In software development projects it's desirable to replace repetitive coding with some form of automation, and many times we see a need to achieve this. Custom program generators could replace the volume of manually written code by generating code from domain-specific models. Less manually written code is a good thing.

Plenty of software products around us apply the concept of model-driven development. Consider, for example, a professional HTML editor. The user interface assists the user in developing the Web site through a look-and-feel visual approach. The editor will generate HTML, JavaScript, JSP/ASP, JDBC programs, and other software artifacts based on the model of the Web site that the user has built through the highly intuitive visual interface. However, these products are built to perform specific tasks that may not be of use in general software development. My point is that there's a dearth of tools to facilitate the rapid translation of domain-specific models into desired programming language constructs.

Compiler-compilers (CC) are tools that help in the building of program generators. The input specification to CC is typically language grammar. Based on the input syntax, the CC generates a parsing/lexing framework, which then builds a custom program generator. You can build a C compiler by using a CC. In a CC-based code generation approach, the focus on code generation logic is lost due to the distraction of building various infrastructures, such as an abstract syntax tree (AST), syntax and semantic validation, and so on. A rapid approach to program generator development needs tools supporting scripting capabilities with reference to parsing and syntax tree building/visiting. We don't want to spend time and resources in program generator acquisition. Since code generation invariably involves tree walking, the entire focus should be on visiting the syntax tree of the model and generating the output syntax tree.

One view of XML is that it's a metalanguage used to define other domain-specific languages. XML tags, attributes, and namespaces could be used to capture models of application domain concepts. XML model syntactic validity can be checked by tools that enforce XML schema. In-memory representation of XML documents is standardized with the DOM (Document Object Model).

With XSLT you have scripted access to the XML DOM. The XSLT processor parses the XML document to construct the DOM, and XSLT scripts manipulate the DOM input to construct the DOM output. It's possible to construct plaintext source code files from the DOM output. The plaintext output forms one to many source code files.

These facts make the source code generation task extremely easy since the tasks of parsing, validating, and generating code can all be performed with freely available tools in the manner of rapid scripting-based development. Development of a visual modeling interface may require significant effort, but it's not a must-have. Plaintext editors can be used to capture domain-specific models in XML.

In a source code generation scheme a central concern is how to integrate generated code with manually written code. Typically, a framework is generated that is then customized manually. Regeneration caused by model change shouldn't wipe out past customizations. In other words, there should be a clear physical separation of custom code and generated code.

I've found that the inheritance and encapsulation mechanisms of object-oriented languages are a neat way to address this separation concern. More specifically, manually written code could inherit or use generated base classes or interfaces. In the other direction, generated code could also inherit or use handwritten classes. A generated code module/package is kept separate from manually developed modules/packages. In procedural languages the same effect could be achieved by effectively mimicking OO languages. It isn't good practice to mix generated text with handwritten text physically since this approach is brittle from an automation standpoint. Code generation examples in this article are in Java.

Though we're focusing on source code generation in this article, many kinds of documents can be generated out of XML models. For example, from the XML object relational model of domain entities you can generate a PDF file capturing the model in pictorial form. The notable point is that plaintext that can be used for various purposes is generated, source code being one of them.

This article documents experience gained in the application of a model-driven programming ideal in software development projects. To achieve the ideal in a given application domain, we must provide three things: (1) a modeling language specific to that domain, (2) a visual tool for building models in that language, and (3) automatic code generation from models in that language to the appropriate implementation code. Software projects would like to have a model-driven program generator without investing significantly in the program generator acquisition. This article demonstrates an approach to rapid development of domain-specific program generators by using XML as the domain modeling language and XML Stylesheet Language Transformations (XSLT) as a syntax-tree scripting mechanism.

The benefits of domain model–based software development are well known; they include a higher level of development abstraction, solid consistency, resiliency to behavior changes, rapid application prototyping, and increased focus on application logic development.

In my opinion, complexities in the current approach to source code generation have impeded widespread adoption of the generative approach in software development projects. For a reasonably sophisticated source code generation scheme, the learning curve to achieve XML/XSLT-based source code generation is appreciably less compared to the benefits obtained. This article shows how freely available XSLT processors supporting the W3C standard can be used as a custom program generator in many software development projects.

Our example is drawn from a real-world software project (refer to "Application in a Software Development Project" section) in which 60% of the source code is generated. The purpose of the project was to develop a management system for a large network of telecommunication equipment. The project used source code generation extensively in the areas of EJB, SNMP-based network management, Java helper utilities, and the SQL schema for relational databases. Though the project didn't call for it, other kinds of documents such as SVG pictures or PDF documents could have been generated from the XML models the project developed.

The remainder of this article discusses the following four topics:

1. Source code generation by XML/XSLT compared to the CC approach
2. A simple example showing the details of source code generation step by step
3. An application in a software development project
4. Things to be aware of

While the XML/XSLT-based code generation approach is easy enough for it to be considered in many projects, it isn't a general approach. There are some limitations. The third item mentioned above addresses this and puts the applicability of the current XML transformation approach into proper perspective.

Code Generation via XML/XSLT vs CC-Based Approaches

Code generation processing generally consists of a number of distinct phases, as shown in Figure 1.

Figure 2 shows the building blocks in CC (like Yacc, JavaCC) based program generators. The input to the CC is a grammar specification tagged with target language code (like Java/C++) for semantic actions and values. Typically, complementary tools have to be used in conjunction for tree building and semantic analysis. A lot of glue code has to be written to integrate output generated by a CC to produce the program generator. If the grammar allows things to be referenced before the declaration is encountered, syntactic checks must be replaced with semantic checks; thus a CC's value as a development tool is reduced.

The logic for generating output from the in-memory representation of input is some sort of tree walk processing. The output may also be produced in tree form in memory before being written in source code in one or more files. The output has to conform to the target language grammar; otherwise target language compilation will fail. The target language compilation errors will generally provide feedback to correct source code generation in the iterative development process.

Examination of the CC-based source code generation process reveals a number of things not central to the objective:

- The need for an internal abstract syntax tree
- The need for glue code and data structure
- The need to build the program generator first

The central processing step of code generation logic is tree walking. The key to rapid development of a source code generator is to have some kind of scripting capability for tree walking. Everything else is there to support these steps. With this background we enter the XML/XSLT-based code generation scheme. Code generation using XML/XSLT requires the following abilities on the part of the program generator developer:

1. The ability to define the domain specifications in XML, which involves the determination of XML elements, XML element attribute names, values, and the nesting structure
2. The ability to visualize the tree that will result from the input XML documents containing domain specifications
3. The ability to use XSLT and XPATH facilities to browse through the XML tree and generate output text that gives rise to one or more source code files

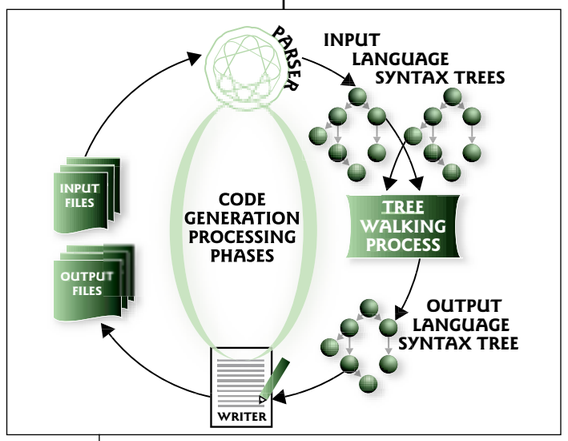


FIGURE 1 | Code generation processing phases

Please note that the XML syntax for the domain modeling language is definitely inferior to a natural language syntax that could be supported in the CC-based approach. However, in a software development project this may not be an issue. On the other hand, the gains derived from adopting XML are many. With the input language in XML, parsing and in-memory representation aren't a development concern since this part is supported by the XSLT processor.

The developer writes code generation logic in terms of the XSLT and XPATH language facilities (see Figure 3). XPATH is a language that specifies how to denote the expression accessing the in-memory tree representation. XSLT is the specification for transforming the input tree parts selected by XPATH. The result is that through XSLT/XPATH scripts the developer is constructing an output tree. When serialized to plaintext this output tree gives rise to one or more source code files. The compile correctness of the generated output files provides feedback to correct the source code generation scheme in an iterative fashion. Tree-oriented assertion statements may validate the input.

It's worthwhile to note that in an XML/XSLT-based approach:

- Building a syntax tree isn't a concern.
- The program generator logic is nothing but a collection of XSLT/XPATH scripts that are interpreted by an XSLT processor.

Scripting means rapid development. Having this capability to browse a syntax tree means rapid development of program generators. Thus it becomes extremely easy to change code generation logic or to accommodate a structural change in the input language. Table 1 summarizes the benefits obtained.

	COMPILER-COMPILER BASED	XML / XSLT BASED
Pros	• Input language can have intuitive and elegant syntax • Logic for code generation is written in powerful languages like C++ or Java	• Easy to take care of changes in code generation specification • Scripted, thus aiding rapid development • Little effort spent in support infrastructure. Focus mostly in code generation logic
Cons	• Changes in grammar specification cause widespread changes • Not scripted, i.e., program generator executable needs to be built first. Not a rapid approach • Appreciable effort spent in building support infrastructure used by actual code generation (tree walking) logic • Depending on language structure, syntactic validation may not be in proportion, i.e., not much help from parser generator • Steeper learning curve	• Input language has to be well-formed XML, whose syntax is considered ugly by some. Further validation performed by tree-oriented assertions • XSLT is a specialized programming language having functional and logic flavors. Not an imperative language like C++/Java

TABLE 1 | Comparison of code generation approaches

Steps to Generate Source Code

This section shows the steps involved in XML/XSLT-based source code generation with a simple contrived example. The purpose is to highlight the steps while omitting irrelevant details. Let's consider Java code generation. Java doesn't have an enumeration type. A project using Java may use the following XML specification to generate Java utility classes having enumeration functionality:

```
<EnumDef name = 'DayOfWeek'>
    <choices>
        <choice name = 'SUNDAY'
          value = '0' />
        <choice name = 'MONDAY'
          value = '1' >
        .....
        <choice name = 'SATURDAY'
          value = '6' />
    </choices>
</EnumDef>
```

This specification is intuitive enough to convey the fact that DayOfWeek is an enumerated type with seven distinct values. It also shows that the process of defining a domain-specific language in XML consists of defining the markup elements and element attributes. Where domain language elements clash with preexisting XML elements, the XML namespace facility should be used to distinguish the domain language elements. The set of XML namespace, elements, and tag attributes defines the XML-based domain language.

The specification has the following constraints arising out of the fact that the target language is Java. These constraints are not expressible formally with XML syntax:

- The <EnumDef> element attribute name, <choice> element attribute name, can have values only according to Java language specifications for variable names.
- The <choice> element attribute value can only have distinct integral values. At least one <choice> element has to be nested within the <choices> element.

Once the input language document is defined in XML, XSLT scripts can be written to process documents conforming to the input language and generate output documents in various forms. Processors supporting the XSLT language standard are used to translate XML documents into other XML or text documents. A number of free XSLT processors are available; this study used the SAXON XSLT processor developed by Michael Kay.

XSLT is used for document transformation in, for example, a multiple-line publishing scenario including HTML for Web clients, WML (Wireless Markup Language) for wireless clients, and PDF (Portable Document Format) for print documentation. The concern about one document content, yet multiple presentation formats, is neatly addressed by using XSLT transformation in the Web-based information architecture. In my opinion, however, the preceding use of XSLT has received the most attention in Web document publishing scenarios, and not for its utility of custom code/document generation in application software development projects. This article emphasizes that XSLT offers an easy approach to code generation.

Please note that this strategy of code generation is perfectly in accordance with the processing model in Figure 1. Compared to the traditional approach of code generation using a CC, it has many features aiding the faster development of code generators, as follows:

- ***Parser:*** XML document parsing is implemented in the XSLT processor. With a custom parser you need to implement the parser or understand how to use a program generator such as Lex/Yacc to generate the parsing framework.
- ***Tree:*** The XSLT processor constructs the tree and provides access to it according to the XPATH specification. The user doesn't have to construct the tree at all. With the custom parser the user needs to populate the tree as the parsing progresses.
- ***Tree processing:*** The XSLT processor provides access to the tree through the XPATH expression as well as many programmatic constructs and functions to perform tree processing. In other words, XSLT users write XSLT scripts to perform operations on the tree. On the other hand, this part needs to be implemented programmatically by code generator writers following other approaches. XSLT programming for code generation programming is at a high level – namely, at the level of tree abstraction. Note that with XSLT, even though the code generation programming is at the tree level abstraction, the programmer never needs to worry about tree data structure implementation details.
- ***Writer:*** The XSLT processor implements this part too.

To write a code generator, the developer writes XSLT scripts using XSLT and XPATH facilities. Another benefit of XSLT-based code generation is the flexibility that's allowed in the change of input document grammar: additional elements and attributes can be introduced in the input language specification without affecting code generation scripts. Contrast this to a custom parser–driven approach in which major change propagation occurs throughout the code generation system. These are complexities in the application development of custom code generation not present in the use of XSLT.

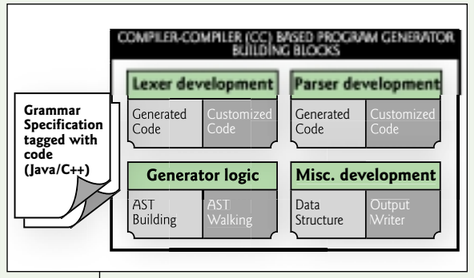


FIGURE 2 | CC-based code generation

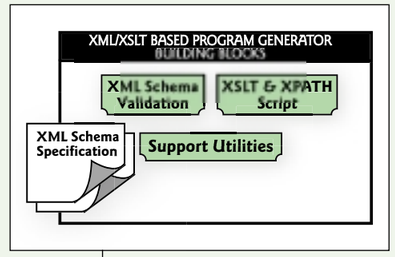


FIGURE 3 | XML/XSLT-based code generation

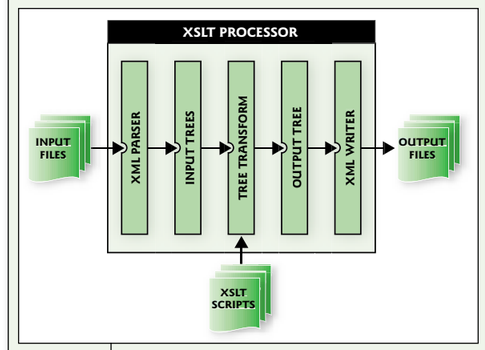


FIGURE 4 | XSLT-based document generation process

Figure 4 shows the process of program generation using XSLT. Generally, XSLT processors produce one output file, although an external utility could break the single output file into multiple files. For example, in the case of source code the output file can delimit the beginning and end of each file with markers not likely to occur as part of the source code. Furthermore, the location of each file could be indicated as part of the generation. The external utility then processes the single output file to produce multiple files in multiple locations. Similarly, input to the XSLT processor should preferably be one XML file including other XML files. There are techniques to achieve XML include, and W3C is working on a standardized XML include mechanism.

Details, Details, Details

The detail lies in the use of XSLT and XPATH to implement the desired transformation from the XML source document to the destination document. XPATH is a separate W3C standard that's intimately related to XSLT. XSLT defines a transformation language that operates on the tree representation of XML documents. XPATH defines the syntax of a language that's used within XSLT scripts to address parts of the XML document tree being transformed. XPATH is the "tree addressing" language. An XSLT processor implements XSLT and XPATH specifications and, optionally, some extension functions to make the life of an XSLT scriptwriter easier.

Let's demonstrate the Java enumeration utility class code generation. We first need to decide how we'd like to specify the code generation (Ability 1). We decided that a particular enumeration specification would look like this:

```
<EnumDef name = 'name of the enum'>
<choices>
<choice name = 'name of the choice' value = 'integer value'/>
</choices>
</EnumDef>
```

The specification has the following statements, not expressible formally with XML syntax:

- The <EnumDef> element attribute name, <choice> element attribute name, can have values only according to Java language specifications for variable names. This is because the target language is Java.
- The <choice> element attribute value can only have distinct integral values. There has to be at least one <choice> element nested within the <choices> element.

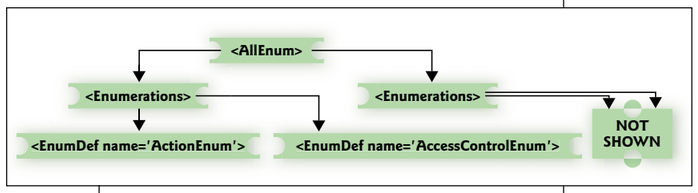


FIGURE 5 Part 1 of the tree structure

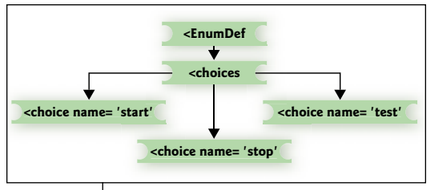


FIGURE 6 Part 2 of the tree structure

With this specification in mind, we start defining XML documents containing Java enumeration code generation specifications. The document in Listing 1 was authored by project software developers, whereas the one in Listing 2 was generated by a utility program from some data source.

One technique of XML inclusion (using an external parseable entity reference) is used to present only one XML input file at XSLT processing time. The XML file in effect has all the enumeration definitions. The XML file showing the technique for XML include is shown in Listing 3.

The second step is to visualize the internal tree structure that will be created inside the XSLT processor. The tree structure is depicted in two parts. The first part shows the overall tree structure (see Figure 5); the second, the tree structure rooted at the first <EnumDef> element (see Figure 6).

For the sake of accuracy, it should be emphasized here that the actual tree structure is much more detailed than what is depicted here. For example, attributes are nodes, text values are nodes, and so on. However, Figures 5 and 6 are sufficient to continue our present discussion.

The third step is to use XSLT and XPATH facilities to process this tree to produce the output files. The XSLT script processes the AllEnums.xml file and produces one output file named enum_codegen.snp that's processed (sniped) by a utility program (file snipper) to produce four Java files named AccessEnum.java, SeverityEnum.java, ActionEnum-.java, and AccessControlEnum.java. For the sake of simplicity, each Java class has only one method, whose purpose is to return the string value of the enum, given the integral value. After setting the classpath environment variable to contain saxon6.4.3.jar, the following command will execute the XSLT script to generate code for the enumerations:

```
java com.icl.saxon.StyleSheet -o gen_enum.snp AllEnums.xml
enum_codegen.xslt
```

What this command is saying is "execute the XSLT processor com.icl.saxon.StyleSheet to produce output file gen_enum.snp from input file AllEnums.xml by using XSLT script file enum_codegen.xslt."

Listing 4 shows the XSLT script; the output it produces is shown in Listing 5. The output is shortened for space considerations.

The code generation script enum_codegen.xslt, version 1, can be explained as follows:

1. The first four lines are declarations, which aren't very relevant from a code generation point of view.
2. The line <xsl:template match = '/'> instructs the XSLT processor to find the root node in the tree and apply the rules contained in the template body. The body works as follows:
   —The xsl:for-each loop selects <EnumDef> child nodes within the root node or from its descendants. Four <EnumDef> children are found within the node hierarchy, starting from the root node. Thus the xsl:for-each loop will execute four times. Please refer to Figure 5.
   —For each execution of the xsl:for-each loop, all non-XSL text is copied to the output. That means the first line within the xsl:for-each loop copies //@@@BEGIN_FILE to the output. The XSLT processor then processes the instruction <xsl:value-of select='@name'/>.java, which outputs the value of the name attribute of the current <Enumdef> child followed by the .java extension. Similarly, you can analyze what happens for the other lines within the xsl:for-each loop.

This step illustrates why it's important to visualize the input tree. At every instruction we're using the XSLT or XPATH facility to navigate the input tree and select content from it to mix with our Java bits and pieces to produce the output Java files. Without a clear idea of the input tree, it wouldn't be possible to use XSLT effectively to generate the desired code.

I hope this explains the code generation development process with XML/XSLT. We're now in a position to show the XSLT script fragment that completes getEnumValueAsString() (see Listings 6 and 7).

Application in a Software Development Project

The purpose of the project was to develop a network management system (NMS). Network management systems are like typical IT applications except that they have to manage the information of a communication network. As a result, a major portion of the NMS goes into providing network device access through some communication protocol. Another core part of the NMS is the object infrastructure, which captures the information of the managed network based on an object and relational model. The project benefited heavily from the use of a generative approach in these core areas. Although we didn't undertake to do so, we could have generated other kinds of documents from the project's XML data.

Network Access Code Generation

Simple Network Management Protocol is a UDP-based network management protocol applied extensively in the data communication industry. Data exchanged over SNMP is defined in a management information base (MIB) with a language called Abstract Syntax Notation One (ASN.1). ASN.1 defines protocol data units (PDU) at the abstract level of data type semantics. ASN.1 doesn't care how the PDU is digitally encoded; this aspect is taken care of by the transfer syntax specification. The name *abstract syntax notation* derives from this fact. A commercial ASN.1 MIB compiler was used to "dump" the MIB in text form. The MIB dump was then processed by a Java utility program to convert the ASCII MIB dump to XML. Essentially, the XML form has all the information that's available in the ASN.1-based management information base.

Once the SNMP MIB was available in XML form, XSLT processing was applied to generate object-oriented Java APIs on a very high level (i.e., far removed from the drudgery of programming according to low-level SNMP APIs). The high-level SNMP APIs were used without difficulty by all project software developers. This approach can be compared with CORBA. In CORBA, programs communicate with each other over TCP protocol without knowing anything about TCP network programming. Before CORBA, distributed application programming used to be done by expert TCP/IP programmers. Now, with CORBA, TCP/IP-based code is generated from a contract language called IDL that specifies what messages and parameters are exchanged between communicating programs. Thus distributed application programming today no longer requires TCP/IP experts. Likewise, SNMP code generation in my project – device-control code – no longer required SNMP experts.

There's another benefit of generating high-level APIs for network access. Two types of implementation were generated. One provides network access by way of SNMP. The other simulates it by storing/retrieving data from local files. Application code using the high-level API remains unaffected when one implementation is switched with another. Code generation for file-based SNMP simulation allowed the project to proceed without waiting for the actual device to be ready. This is a tremendous advantage since network management development can proceed in parallel and can be tested with a large, simulated network.

Server-Side Code Generation

An object-based server-side infrastructure was used in this project. Application servers complying with the Enterprise JavaBean standard provide concurrency, transaction, security, persistence, and naming services for the objects. A server-side system development consists of implementing the information model by using EJBs and providing interfaces for remote access to the information, which satisfies graphical user interface use cases. The project specified the information model along with a number of relationships in XML. A fictitious object showing object and relational model capability is shown in Listing 8. The model specifies the following:

- The Employee object has attributes like salary, job title, join date. It will inherit other attributes from the Person object.
- The Employee object can't exist without a containing Division object.
- If you have access to the Employee object, you can get access to all its subordinates, which are objects of class Employee.
- If you have access to the Employee object, you can get access to the supervisor, which is an object of class Employee.

The foregoing specifications were implemented by autogeneration. Similarly, a relational database schema in SQL was generated to implement the persistence of object and relational model information. XML deployment descriptors were generated for runtime deployment of EJBs. In addition, semantics were mixed in by the use of application classes inheriting generated classes, and application-specific behavior was coded in derived classes.

Object and relational model code generation, coupled with middleware services provided by the EJB framework, created a powerful paradigm of server-side infrastructure development for the project. The project had a tremendous lead by being able to build further on this sophisticated server-side infrastructure rather than spend time on building the infrastructure itself. The project focused totally on building application logic and delivering functionality.

Things to Be Aware Of

I experienced some limitations in starting from XML and applying XSLT in real application software projects.

- The input language is XML, whose syntax may be abhorrent to some. XML should be considered an underlying data representation language. Tools such as editors, both textual and visual, provide high-level browsing and editing capabilities and should be used while working with XML. If XML is generated by some tools (like SNMP MIB browsers), this concern doesn't arise.
- XSLT is a purely functional language, which means no side effects. For example, to implement a for-loop of fixed count you need to implement a tail-recursive template with a suitable termination condition. Application programmers have to make many more "habit adjustments" before becoming comfortable with XSLT programming. Moreover, XSLT has a "logic programming flavor," a paradigm that some application developers may not be familiar with. XSLT syntax is quite verbose, which is bothersome to say the least. XML escape mechanisms have to be used to generate symbols like "<" (less than), ">" (greater than), and others that are heavily used in source code. Some XSLT workbenches are available to mitigate this inconvenience.
- Unlike the IDEs we have for C++/Java development that have graphical debugging capabilities, no such IDE is available yet for XSLT. The debugging technique we used was to print part of the tree in the output document itself, i.e., by using <xsl:value-of ..>. We then inspected the printed tree to understand how a desired selection/transformation could be created using XSLT/XPATH.
- The XSLT processor is quite permissive with reference to mistakes/omissions made in inputting the document. For example, missing elements or attributes don't cause the XSLT processor to fail. They cause missing output! Consequently, for source code generation, there will be compiler errors somewhere down the line. There are a number of solutions to this problem of input document valida-

LISTING 1 AuthoredEnums.xml

```
<Enumerations>
<EnumDef name = 'AccessEnum'>
<choices>
<choice name = 'read_only' value = '1'/>
<choice name = 'read_write' value = '2'/>
</choices>
</EnumDef>
<EnumDef name = 'SeverityEnum'>
<choices>
<choice name = 'critical' value = '1'/>
<choice name = 'major' value = '2'/>
<choice name = 'minor' value = '3'/>
<choice name = 'warning' value = '4'/>
</choices>
</EnumDef>
</Enumerations>
```

LISTING 2 GeneratedEnums.xml

```
<Enumerations>
<EnumDef name = "ActionEnum">
<choices>
<choice name = "start" value = "1"/>
<choice name = "stop" value = "2"/>
<choice name = "test" value = "3"/>
</choices>
</EnumDef>
<EnumDef name = "AccessControlEnum">
<choices>
<choice name = "permit" value = "1"/>
<choice name = "deny" value = "2"/>
</choices>
</EnumDef>
</Enumerations>
```

LISTING 3 AllEnums.xml

```
<?xml version = "1.0"?>
<!DOCTYPE AllEnums[
<!ENTITY include_authored_enums SYSTEM "AuthoredEnums.xml">
<!ENTITY include_generated_enums SYSTEM
"GeneratedEnums.xml">
]>
<AllEnums>
&include_authored_enums;
&include_generated_enums;
</AllEnums>
```

LISTING 4 Enum_codegen.xslt(version 1)

```
<?xml version='1.0'?>
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:output omit-xml-declaration='yes'/>

<xsl:template match = '/'>
<xsl:for-each select='//EnumDef'>
//@@@BEGIN_FILE <xsl:value-of select='@name'/>.java
//@@@LOCATION common.gencode.enums
//*************************************
//********** Generated code.  ***********
//*************************************
public class <xsl:value-of select='@name'/>
{
    public static String getEnumValueAsString
     (int enumValue)
    {
    }
}
```

tion. For example, you can use Schematron, an XSLT-based assertion facility for XML document validation. XML schema–enabled parsers may be able to do some validation based on the schema description of the document.

Regardless of these limitations, it can be argued that XSLT is being used to support application software development rather than used directly in application software development. Consequently, project members' exposure to XSLT will be minimal once the correct code generation has been figured out. For most code generation purposes, an XSLT-based solution is feasible. I believe that application software developers need to become more familiar with this XML-based document transformation approach so they too can achieve the benefits set forth in this article.

Summary/Conclusion

This article demonstrated how XML-based document transformation technology can benefit application software development projects. Here are some software metrics that show how my project benefited:

- Out of approximately 2,480 Java files in the project, 1,520 were generated. In other words, 61% of the project Java code was generated. SQL and XML files were generated too.
- SNMP coding was extremely easy. For example, it took only one line of Java code to display the RFC1213 ipRouteTable:

```
System.out.println(new
IpRouteTableUtils().getIpRouteTable(deviceInfo));
```

IpRouteTableUtils is generated from RFC1213 ASN.1 MIB. The method getIpRouteTable returns the table object, which is displayed in the System.out.println call. The deviceInfo object has the device IP address and the password for read-only SNMP access (also known as read community name).

With the standardization of XML transformation by XSLT/XPATH, and with the increasing availability of good literature on the subject, it's my firm belief that code generation by XML transformation using XSLT will give application software development projects tremendous leverage in attaining increased productivity without sacrificing quality. One developer's expertise in XSLT and code generation could dramatically lessen the burden on other software developers by providing semantically rich, high-level APIs through code generation. With XSLT 2.0 features the task of code generation will become easier.

References

- Cleveland, J.C. (2001). *Program Generators with XML and Java.* Prentice-Hall.
- Kay, M. (2001). *XSLT Programmer's Reference*, 2nd edition. Wrox.
- *SAXON XSLT Processor:* http://saxon.sourceforge.net
- *GSLgen:* www.imatix.com/html/gslgen/index.htm
- *Fxt, generator of XML document transformers:* www.informatik.uni trier.de/~aberlea/Fxt/
- *The Schematron: An XML Structure Validation Language Using Patterns in Trees:* www.asccnet/xml/resource/schematron/schematron.html
- *XML in 10 points:* www.w3.org/XML/1999/XML-in-10-points
- *Extensible Markup Language (XML) 1.0:* www.w3.org/TR/REC-xml
- *XSL Transformation (XSLT) Version 1.0:* www.w3.org/TR/xslt
- *XML Path Language (XPATH) Version 1.0:* www.w3.org/TR/xpath

AUTHOR BIO

Soumen Sarkar, currently lead software engineer at Ätoga Systems, Inc., specializes in building large-scale distributed object systems for telecommunications network management. His interests include programming languages, CORBA and EJB, XML and related technologies, communication protocols, and software architectures. He has more than 10 years of experience in building object-oriented software.

SARKAR_SOUMEN@YAHOO.COM

```
//@@@END_FILE AccessEnum.java
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

LISTING 5 gen_enum.snp

```
 //@@@BEGIN_FILE ActionEnum.java
//@@@LOCATION common.gencode.enums
//**************************************
//*********** Generated code.  ***********
//**************************************
public class ActionEnum
{
    public static String getEnumValueAsString
     (int enumValue)
    {
    }
}
//@@@END_FILE ActionEnum.java
…….
Repetition of above pattern for SeverityEnum.java,
AccessEnum.java, AccessControlEnum.java
…….
```

LISTING 6 XSLT code fragment

```
Public static String getEnumValueAsString(int enumValue)
{
    switch(enumValue)
    {<xsl:for-each select='choices/choice'>
        case <xsl:value-of select='@value'/>:
        return "<xsl:value-of select='@name'/>;
    </xsl:for-each>

        default:
        return null;
    }
}
```

LISTING 7 Corresponding generated code fragment

```
7 public static String getEnumValueAsString(int enumValue)
{
    switch(enumValue)
    {
        case 1:
        return "start";
        case 2:
        return "stop";
        case 3:
        return "test";

        default:
        return null;
    }
}
```

LISTING 8 Example of object and relational modeling in XML

```
<managed-object class-name = "Employee">
<base-object class-name = "Person"/>
<containing-object class-name = "Division"/>
< one-to-many-relation  role-name = "my-subordinates"
class-name = "Employee"/>
<one-to-one-relation  role-name = "my-boss" class-name =
"Employee"/>
<attribute name = "salary" type = "float"/>
<attribute name = "job-title" type = "string"/>
<attribute name = "join-date" type = "java.util.Date"/>
```

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

XML-J ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
ADOS Co., Ltd.	http://www.a-dos.com	81-3-5475-1551	59
Altova	www.xmlspy.com		60
Borland			11
engindata Research	www.engindata.com	201-802-3082	24, 25
HitSoftware	www.hitsw.com/xmlrdb	408-345-4001	15
IBM	www.ibm/com/websphere/winning		2, 3
JDJ Store	www.jdjstore.com	888-303-JAVA	55
Macromedia	www.macromedia.com/go/cfmxad	888-939-2545	9
Sonic Software	www.sonicsoftware.com/websj		6
Sprint	http://developer.sprintpcs.com		4
SYS-CON Events	www.sys-con.com	201-802-3069	27, 28
SYS-CON Media	www.sys-con.com/suboffer.cfm	888-303-5282	48
SYS-CON Media	www.sys-con.com	888-303-5282	47
WebLogic Developer's Journal	www.weblogicdevelopersjournal.com	888-303-5282	51
WebServices Edge World Tour	www.sys-con.com	201-802-3069	34, 35
WebSphere Developer's Journal	www.sys-con.com/websphere	888-303-5282	48
XML for Financial Services	www.worldrg.com/fw251	800-647-7600	55

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.

web services conference&expo EDGE

Show Report

WRITTEN BY **STEVEN BERKOWITZ**

AUTHOR BIO

Steven Berkowitz has done development and project management for Fortune 100 companies, startups, and nonprofit organizations.

While I wandered, head down, among the detritus of the post dot–com era, a new world order was forming. When I looked up, I found the technical world rallying under a new banner – Web services. Savior or hype? I had to know. What better place to learn than the world's largest Web services event? So off I went to SYS-CON's Web Services Edge 2002 East – International Web Services Conference & Expo. While this was also the JDJEdge and XML Edge Expos, the main thrust was Web services. The keynotes all centered on Web services and most of the Java and XML seminars also had a Web services tilt.

In addition to an expo floor with dozens of vendors offering products and services either supporting Web services or claiming actually to enable them, there were 60 hour-long seminars divided among five tracks – Java, Web Services, XML, IT Strategy, and .NET. On top of all this, Sun ran six day-long courses from their Java University Program.

Monday was dedicated solely to Java University. I took the "Web Component Developer: Certification Fast Path" course about servlet and JSP technology. Any code-level course where you don't get to write code is at a disadvantage. But, for what it was, I found the course very useful. Its stated goal was to cover the material that's required on the certification exam and give pointers for further study. It met this goal admirably, and clarified, for me, a number of issues I was unclear on in these technologies.

Among the technologies that had passed me by was XML. Sure, I've dabbled in it here and there, and edited my deployment descriptor files, but I never did anything meaty with it. So Tuesday found me in the day-long "Developing Solutions Using Java Technology and XML" course, presented by Todd Greanier of Compass Technologies. This course was fantastic. It started with a very thorough XML primer, which was exactly what I needed. Then it detailed which Java technologies are available for use with XML. Overall, I was very impressed with this course and its presentation.

The trade-off for taking this XML course was that I missed the first day of the expo, including the keynotes and all the seminars. Initially, I was a bit concerned about this, but unless day one was very different from the next two, I don't believe I missed too much. While I learned a lot of specific things on the Wednesday and Thursday of the expo, a common thread ran through the keynotes and seminars. Summed up, it might read like this: Web services holds a lot of promise, but there are vast areas that remain undefined. I'm getting a little ahead of myself here, so bear with me awhile and we'll come back to this point.

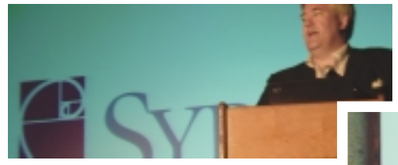
> **"I took the 'Web Component Developer: Certification Fast Path' course...It met its goal** admirably, and clarified, for me, a number of issues I was unclear on in these technologies"

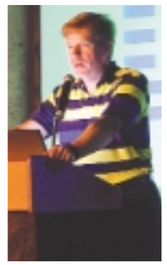
My original question was: Is the concept of Web services just hype? The simple fact that this conference existed inclined me to believe that there had to be something more. The dozens of vendors I mentioned earlier spent big money developing supporting products. While it is possible that they are merely chasing the hype machine, I sense otherwise. Please don't get me wrong. I'm in no position to declare Web services the be-all and end-all. But there is something there.

Eric Newcomer of IONA gave a very powerful keynote on Wednesday entitled "Web Services: Integration Technology for the 21st Century." He led with the statement that there are currently too many definitions of precisely what Web services is. While this may be strictly true, a fairly standard definition emerged during Newcomer's speech and in all the other talks I attended. Web services deals with the space between applications. It is a service-oriented architecture using XML and XML-based standards to identify, describe, define, and execute business processes. Newcomer's holy grail is the built-on-the-fly business applications assembled from pluggable Web services sitting out there on the Web ready to go.

Now, this new world of interoperability will have monetary benefits. Without increased ROI, why would our bosses want to spend money on this stuff? Make no mistake, they will be spending the money. A slide from the "Practical Experiences with Web Services and J2EE" seminar quotes surveys from groups like Gartner, Jupiter Media Metrix, and InfoWorld showing that money is heading this way.

One of the primary cost savings with Web services, according to Newcomer, is that it will deskill programmers. We won't need mAd SkILz to integrate applications anymore. Business users can just pick from a menu of functionality and, voilà, a new application. We as developers may bristle at that; you may say that as a developer myself, I'm in outraged denial. But, this review will look at the undefined portions of Web services and the competing standards under development that are meant to address them. Look at that and then tell me how your accounting department is going to make their own applications Web services–enabled. We will still be needed. There really is no alternative.





Another problem with Newcomer's pluggable application vision is that we are nowhere near it. When talking about companies already using Web services, you've made them talk with each other. There is value in that.

However, adoption of interenterprise Web services is hampered when you consider what is *not* defined by Web services. Little things like security, workflow, and transaction support. Web services currently does not address these issues in a standard way. Anne Thomas Manes of Systinet gave a talk on "Developments in Web Services Standards." Manes said that Web services needs to address the following issues before it can be truly ready for prime time: management, security, context, delivery, workflow, choreography, service agreements, service characters, discovering, business system registration, and standards and conventions. That's a pretty hefty list of things in the way of widespread adoption. There are standards bodies addressing these things: W3C, Web Services Interoperability Organization (WS-I), and Organization for the Advancement of Structured Information Systems (OASIS) are just three. OASIS has 30 tech committees alone working on Web services issues.

Manes wasn't the only one to make the point about competing standards. Even "Practical Experiences with Web Services and J2EE," by Hugh Grant of Cape Clear, stressed these issues. The J2EE thrust of that hour was that you can translate your EJBs into Web services using WSDL to describe them, and a Façade pattern to cover multiple beans if they combine to a single service. Beyond that, his talk centered around the standards, which I found very telling.

I can give you the alphabet soup of proposed standards, but I think that would cloud the real issue. You have to realize that many of the standards under development compete with one another. As we know, multiple standards to address a single issue mean there is no standardization. This is a very risky place to be, because it allows vendors to fill in the gaps. This is where Web services is now. There are too many holes, and vendors, such as Actional, SilverStream, and Sonic, all have products that will come in and fill many of those gaps. Many other vendors

web services EDGE conference&expo

offer more tailored solutions, some addressing security, others workflow.

Phillip Merrick of webMethods gave a keynote entitled "Enterprise Web Services: An Evolutionary View of Web Services." He made a crucial point – the history of many technological issues is this: proprietary solutions (immature), proprietary solutions (mature), standardized solutions (immature). From here, there is a crossroad. We can either go to standards (mature), which would be ideal, or standards plus proprietary adds from vendors, which would be bad. It would fracture the Web services market and more or less kill the interoperability promise.

Web services is at this crossroad. If what I saw on the expo floor is any indication, we are going down the bad road. If you don't want to see that, take Merrick's advice. Demand standardization. Keep your vendors honest. If we can do that, the promise of Web services can be met. And that would be a fine thing indeed.

Even in the face of all the seminars and keynotes, I managed to spend some time on the expo floor, speaking with the vendors, learning a bit about what they had to offer. The two products I found most fascinating deal with XML in general, giving them a broader applicability than the Web services–focused products.

First, there is Allora from HiT Software. Coming in winAllora (COM) and jAllora (Java) flavors, this is a tool for mapping XML to relational data. Even my limited dabbling in XML showed just how different these beasts can be. While it is easy enough to take a JDBC ResultSet and write it all to an XML document, I can see that this would get very tedious for more than the odd table or two. The Graphical Mapper shows your XML Schema or DTD on one side and your database schema on the other, allowing you to map your relationships at design time. Plus, Allora comes with mapping, databinding, and messaging APIs to make it useful at runtime. These two features alone might be worth the price of admission, but Allora goes one step further – it includes a set of SOAP interfaces so you can expose it as a Web service. The holiday weekend and publishing schedules have kept me from installing and running it at home, but I was very impressed with the hands-on demo of the Graphical Mapper.

The other product that caught my fancy was Forum System's XML Web Services Security Appliance, which deals specifically with securing XML documents. This is a hardware device that plugs into your network between the firewall and the application server. As we know, XML is basically self-describing data. An element called "password" or "card number" kind of screams out to the world that there's good juicy data there. This is rather a gaping vulnerability that the security appliance addresses by encrypting the data, either in totality or on an element-by-element basis, and protects it from end-to-end.

> "One thing is clear—
> XML is the lingua franca of Web services.
> **Learn it. Live it. Love it.**"

I would be remiss if, in the focus on Web services, I failed to point out that the conference also had a Java focus. The best Java presentation I went to was given by Rick Hightower of Trivera. His talk on "Java Tools for Extreme Programming" was well presented and rather informative. Rick focused on the testing and continuous integration portions of XP, discussing JUnit, Ant, and Cactus in detail. His talk proved to me that no matter what you are doing, no matter what you call your methodology, testing is invaluable and these tools will make your life easier.

On Thursday there was a panel discussion on .NET versus J2EE. The stars were Simon Phipps, chief technical evangelist at Sun, and Barry Gregory, principal technology specialist from Microsoft. Sharing the stage with them were Sean Rhody, editor-in-chief of ***Web Services Journal***, Mike Kovach of FullTilt Systems, and the editor-in-chief of these very pages, Alan Williamson. Because this conference had a .NET track (which, admittedly, I did not take advantage of), I had hoped for a fairly well-reasoned discussion. Some good points were made on both sides. Sadly, most good points were buried in bickering and the entire discussion was peppered with not-so-thinly-veiled snipes between the two camps. Maybe it was naïve of me to expect more, maybe it would have been better to leave representatives of Microsoft and Sun out of the discussion, but as it stood, the whole thing left a sour taste in my mouth.

Finally, right next to the Web Services Edge was PC EXPO/TECHXNY, and attendees of one got admission to the other. While this may seem like a great deal, free admission to PC EXPO after all, it really took away from the power of Web Services Edge. At the risk of sounding horribly l33t, the net effect was hordes of mere users, wearing "Gimme Free Stuff" buttons, gawping like tourists at booths displaying things they wouldn't understand even if they wanted to. They clogged the aisles, wasted exhibitors' time, and in some cases kept us Web Services Edge attendees from speaking to the vendors. What's more, they glommed up more than their share of the goodies. And let's face it, to tech folk like us, corporate toys are second in importance only to free food.

Summing up: Web services exists in some form. It is coming. You can either help define it or live with what happens out there. But if you're reading this magazine, you'll end up working with it on some level. Whatever shape this mess takes, one thing is clear – XML is the lingua franca of Web services. Learn it. Live it. Love it. I know I spent a lot of time talking about what Web services isn't. Sometimes, though, that's more important than learning what something is. Which made my four days at the Web Services Edge Conference & Expo invaluable.

SJB47@YAHOO.COM

XML NEWS

New Release of GoXML Transform 3.0

(New York) – XML Global Technologies, Inc., has released GoXML Transform 3.0, an easy-to-use data transformation product focused on transforming legacy and other structured data into a variety of data formats, including XML.

New features include support for HL7 data format and data type, and improved support for EDI HIPAA, SWIFT, and W3C schema.
www.xmlglobal.com

HiT Software Delivers Allora v3.0

(San Jose, CA) – HiT Software has released Allora v3.0, its XML-relational database bidirectional transformation platform that now includes technology-leading transformation control as well as integration with Microsoft BizTalk Server. The Allora product line includes jAllora, an all-Java solution, and winAllora, a native Windows-based product.

New control features include scripting, dynamic parameter support, and XML predicate query support.
www.hitsw.com

New BizTalk Server 2002 Editions

(Redmond, WA) – Microsoft has announced the availability of the Partner and Standard editions of BizTalk Server 2002. Both editions deliver the full capabilities of BizTalk Server Enterprise Edition, but are tailored to companies that require minimal integration capabilities and processing power.
www.microsoft.com

BizTalk Accelerator for RosettaNet

(Mountain View, CA) – The BizTalk Accelerator for RosettaNet version 2.0 is available from Microsoft. Built on top of Microsoft BizTalk Server, version 2.0 includes all the functionality of the first version plus full support of the RosettaNet Implementation Framework (RNIF) versions 1.1 and 2.0; all published Partner Interface Processes; Partner Agreement Wizard for rapid importing, deploying, and testing of new and customized PIPs; and support for additional industry standards.
www.microsoft.com

Natural Engineer WebStar from Software AG

(Reston, VA) – Software AG has debuted Natural Engineer WebStar, an XML-based solution designed to transform legacy applications to Web applications. The solution combines Software AG's professional services expertise (SAGe) and the power of EntireX Communicator, Software AG's communication broker.

Key benefits of WebStar: it transforms a terminal application into a full-fledged Web application automatically; builds on Natural, a programming language companies are already using; protects investment in current applications; and supports modularization of code by creating subprograms for specific functions.
www.softwareagusa.com

X-Hive Launches X-Hive / DB 3.0

(Rotterdam, The Netherlands) – X-Hive Corporation has released version 3.0 of X-Hive/DB, a native XML database designed for software developers who need to process and store XML data in their applications.

X-Hive/DB supports all major XML standards and offers a transaction mechanism, versioning with branching, BLOB storage, and various indexing methods, as well as support for J2EE and WebDAV.

A free 30-day evaluation license can be obtained from www.x-hive.com.

IFX Forum Publishes XML Spec Version 1.3

(Alexandria, VA) – The Interactive Financial eXchange (IFX) Forum has published the latest IFX specification, version 1.3, providing an XML-based communication protocol enabling the exchange of information between financial institutions and their customers, service providers, and other financial institutions.

Enhancements to v1.3 include multiple addresses in contact info, payment credit status, improved check ordering, direct debit feature via the payment messages, deposit account application, and status code to the additional status aggregate.
www.IFXForum.org

Self-Paced, Online XML Courses for Professionals

(Newport Beach, CA) – Professionals can now work at their own pace to learn about EAN.UCC XML Standards through online education jointly developed by Drake Certivo Inc. and the Uniform Code Council Inc. (UCC).

The three-course components to EAN.UCC XML Fundamentals (Parts I, II, and III) are designed to provide different levels of education in EAN.UCC XML, and are the only XML courses based on the globally recognized EAN.UCC XML Schemas.

The modules consist of presentation materials, interactive exercises, and code examples. Optional hands-on learning labs are also available in Parts II and III.

Completion of all three parts prepares students for a certification exam, available in August. The course components are accessible online at http://elearning.certivo.net/ucc.

Adobe InCopy 2.0 Now Shipping

(San Jose, CA) – Adobe Systems has announced the immediate availability of Adobe InCopy 2.0 software, a professional copy editor tightly integrated with Adobe InDesign software.

Designed for writers, editors, and copy fitters at magazines and newspapers, InCopy includes support for XML, enhanced editing tools such as dynamic spell-check, and an improved interface that lets users work on multiple views of a story simultaneously.
www.adobe.com

BPMI.org Releases BPML 1.0

(Denver) – The first public draft for release 1.0 of the Business Process Modeling Language (BPML 1.0) has been issued by the Business Process Management Initiative (BPMI.org)

BPML 1.0 supports the modeling of end-to-end processes including private implementations and public interfaces for transactional and collaborative business processes. It is available royalty free for download from BPMI.org's Web site at www.bpmi.org/bpml-spec.esp.

Updated Blaze Advisor from HNC

(*New York*) – HNC Software has announced the immediate availability of Blaze Advisor from HNC, version 4.2, featuring improved performance in large-volume batch operations, automated support for iPlanet (Sun ONE) 6.5 Application Server deployments, advanced support for XPath processing of XML documents, and reuse of rule conditions for faster and more efficient rules definition.

Customers also have more options available for source code control with the addition of Rational ClearCase command menus and management of authentication and authorization strategies for the Blaze Advisor rules repository.
www.blazesoft.com

Content@ Web from XyEnterprise

(Reading, MA) – Content@ (con-tent-a) Web interface to XyEnterprise's Content@ XML content management software is now available. Content@ Web offers browser interfaces to Content@'s robust XML reuse, versioning, metadata, and workflow functions as well as a development environment for integrators, resellers, and OEMs. The Content@ Web capabilities expand the impact of Content@'s single-source multichannel delivery capability for customers and partners in the technical documentation, financial, e-learning, and other publishing-related industries.
www.XyEnterprise.com

PureEdge Updates XML E-Forms Standards

(Victoria, BC) – PureEdge Solutions has upgraded their published XML language, the first XML language to define electronic forms. XFDL 5.0 brings PureEdge products into full compliance with Section 508 accessibility requirements, and offers enhanced digital signature information.
www.PureEdge.com

Rapidfire Data Introduces LINX

(*New York*) – Rapidfire Data has unveiled the industry's first peer-to-peer enterprise database connectivity solution. LINX (Linked Information Network in XML) from Rapidfire Data enables an improvement in the ways that dispersed data sets are accessed across the corporate enterprise. LINX has no centralized hub, and offers simplified integration of front-end applications.
www.rapidfiredata.com

Sentori Offers New Version of Billing and Customer Care

(Baltimore) – Significant functional enhancements specifically tailored to the wireless market, including an improved customer care platform and m-commerce support, have been announced by Sentori.
www.sentori.com

Corel Introduces Corel Ventura 10

(Ottawa) – Corel Corporation recently announced the upcoming availability of Corel Ventura 10, a comprehensive page-layout and publishing application. New features and enhancements: XML import, table tags, publish to PDF, integrated preflight engine, and enhanced graphics capabilities.
www.corel.com

Reactivity Introduces Service Firewall

(Belmont, CA) – Reactivity, Inc., a provider of products for distributed application security, debuted the Reactivity Service Firewall, a drop-in solution that makes secure XML and Web services–based application integration fast and affordable. The Service Firewall supports both XML-based message integration and the latest Web services standards and tools.
www.reactivity.com

New Tutorial / Reference Book from O'Reilly

(Sebastopol, CA) – XML Schema: The W3C's Object-Oriented Descriptions for XML, by Eric van der Vlist, is available from O'Reilly. Primarily designed as a tutorial – with design choices, best practices, and limitations – the book also serves as a reference to many aspects of XML Schema creation and processing. Schemas, it explains, effectively serve as design tools for an array of XML-based applications that enable developers to automate tasks such as validation, code generation, documentation, data binding, and query optimization. Validation is the most common use for schemas, ensuring that XML documents conform to expectations, simplifying the code needed to process them.

Two sample chapters are available free online at www.oreilly.com/catalog/xmlschema/chapter/index.html.

Exclusive Canonical XML Now a W3C Recommendation

(Cambridge, MA) – The World Wide Web Consortium has released Exclusive XML Canonicalization as a W3C Recommendation. The specification augments the previous Canonical XML Recommendation to better enable a portion of an XML document (i.e., a fragment) to be as portable as possible while preserving the digital signature. It works in combination with XML Signatures, the W3C Recommendation produced jointly by W3C and the IETF in February, representing cross-industry agreement on an XML-based language for digital signatures.
www.w3.org

WINNERS OF WSJ'S READERS' CHOICE AWARDS

(New York) – ***Web Services Journal*** has announced the results of its first annual **Readers' Choice Awards**. Winners and finalists were honored on June 25 at an awards ceremony at **SYS-CON Media**'s Web Services Edge 2002 East – International Web Services Conference & Expo (June 24–27) in New York City.

More than 10,000 *WSJ* readers cast their votes to select the best Web services products of the year in 17 award categories. For a list of winners see the August issue of *Web Services Journal* or go to www.sys-con.com/webservices.

TAMINO XML SERVER INTEGRATED WITH STELLENT CM SYSTEM

(Eden Prairie, MN / Reston, VA) – Software AG's Tamino XML Server has been integrated with the Stellent Content Management system. This integration offers the first content management system tightly integrated with an XML database and provides users with a content repository optimized for managing and storing large amounts of pure XML content.

The integration provides highly streamlined content exchange processes, enhanced search-and-retrieval capabilities, and enterprise-level scalability.
www.stellent.com
www.softwareagusa.com

The End of E-Business As We Thought We Knew It

The night of the living B2B exchanges

BY **JOHN EVDEMON**
Coeditor-in-Chief, XML-Journal

John Evdemon has been a CTO at both large and small XML vendors. Currently CTO of DiscoveryLogic (www.discoverylogic.com). John is an Invited Expert with the W3C XML Core Working Group.

During the late '90s and the early part of 2000, many people were busily working in startup *X* or *Y*, gleefully anticipating an initial public offering and the promise of cashing in on the New Economy.

Then the bubble burst.

Many people are out of work due to poor stock performance, unrealistic business models, or "vulture" capitalists. Several successful e-business companies watched their stock values plummet and began laying off employees, attempting to lower costs while winning the favor of Wall Street analysts.

We may not have hit bottom yet – several integration vendors' stock values are hovering at (or below) a dollar per share. Rather than address the entire tech sector, let's take a closer look at a specific strategy that promised (yet failed) to revolutionize how companies transact business with one another.

B2B exchanges were heavily promoted as a new, low-cost approach for transacting business electronically. Several startups appeared; some provided tools for building and maintaining exchanges, while others provided resources to host exchanges. Bold analyst predictions continued to surface, fueling a blazing inferno of B2B activity. B2B exchanges would be capable of generating billions of dollars – clearly someone was going to capitalize on this emerging opportunity and become very rich.

Or not.

Implementing successful B2B exchanges required attracting high levels of participation from both buyers and sellers, a delicate, if difficult process – exchanges can't attract buyers without sellers, and attracting sellers is impossible without buyers. Sellers in a B2B exchange can become frustrated since there is no clear approach for differentiating their services from other sellers within the exchange.

Technical and procedural challenges also proved to be a major issue. Many companies lacked the technical expertise or business process flexibility required to connect and succeed within the highly competitive environment of a B2B exchange. Companies were expected to abandon their internal metadata/naming conventions to accommodate proprietary XML vocabularies. These vendor-specific vocabularies were designed to represent a wide range of business transactions. Companies soon realized that these vocabularies were incapable of modeling all of their transactions, forcing them to develop custom extensions. This caused additional integration issues because one company's vocabulary was rarely shared by another. B2B exchanges that managed to attract buyers and sellers soon realized that very large volumes of consistent transactions were necessary for the exchange to generate a profit. Large organizations broke off and began to establish private exchanges to serve their own supply chain needs. These smaller, private exchanges proved to be easier to implement and operate since they served a single company's supply chain. As the apparent demand for public exchanges waned, many B2B vendors either went bankrupt or retreated into safer, more mature areas of the e-business model (such as supply chain management).

Not all public exchanges have ended in failure – several industry-specific exchanges continue to operate today:

- ChemConnect serves the chemical and plastics industries.
- Global Net Exchange (GNX), Worldwide Retail Exchange (WWRE), and Transora serve the retail industries.
- ExoStar serves the defense and aerospace industries.
- Covisint serves the auto industry.

While the B2B exchanges mentioned above were still in operation at the time this piece was written, some of them are operating at a loss and have had to lay off numerous employees. Additionally, several large e-business vendors are continuing to experience problems. General Electric recently sold off 90% of its e-business unit (Global eXchange Services – GXS). Note, however, that GXS has been very successful – it continues to serve over 60% of the Fortune 500.

So what does the future hold for XML in e-business? XML-based e-business vendors and users should analyze the success of EDI prior to moving ahead. EDI has been very successful in linking companies' business processes together for decades. EDI also provides a rich set of metadata and business rules for defining, maintaining, and conducting e-business. Despite what you may have read, EDI does not necessarily require the use of an expensive value-added network (VAN) or high-priced vendor tools – EDI can also be used over the Internet, much like XML.

The popularity of instant messaging has introduced the concept of Business Activity Monitoring (BAM) – real-time agents capable of detecting and notifying all participants to exceptions that may arise across the value chain.

Web services provide the promise of low-cost, standards-based connectivity, but fail to address the issue of semantic integration. Despite the hype, Web services alone won't cut it.

Last, the gradual awareness of RDF and the Semantic Web has forced many vendors to reconsider traditional approaches to mapping and data transformation.

Perhaps an ideal e-business initiative will knit these disparate technologies together, enabling the vision of a collaborative, business process–driven Semantic Web to be realized. This is the next killer app – some company will surely capitalize on this emerging opportunity and become very rich.

Or not.

JEVDEMON@SYS-CON.COM