Enterprise Solutions • Content Management • XML Labs

XML JOURNAL

THE ULTIMATE XML ENTERPRISE RESOURCE

October 2002 | Volume: 3 Issue: 10

xml-journal.com

web services EDGE world tour 2002

COMING TO A CITY NEAR YOU

2002
SAN JOSE OCTOBER 3
LOS ANGELES NOVEMBER 5
NEW YORK NOVEMBER 18
SAN FRANCISCO DECEMBER 3
BOSTON DECEMBER 12

35

WEB SERVICES RESOURCE CD

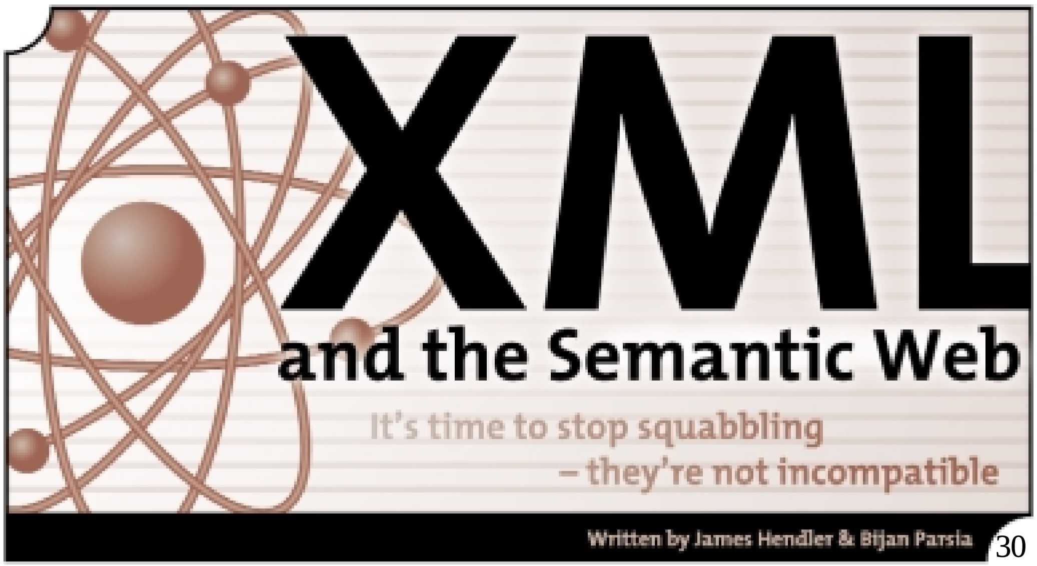
SPECIAL $99 LIMITED PRICE

33

PLEASE DISPLAY UNTIL DECEMBER 31, 2002

$6.99US $7.99CAN

SYS-CON MEDIA


XML and the Semantic Web
It's time to stop squabbling – they're not incompatible
Written by James Hendler & Bijan Parsia

30

BEA

www.dev2dev.bea.com/useworkshop

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
Kal Ahmed, John Evdemon, Jim Gan,
James Hendler, Coco Jaenicke, Bijan Parsia,
Hitesh Seth, Simeon Simeonov,
Mark Ethan Trostler, Rajesh Zade

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.

That's Classified Information

WRITTEN BY **JP MORGENTHAL**

With the advent of computer storage, business has become increasingly more reliant on electronic information as a major source for maintenance and continued growth. The information we store electronically tells us what customers like and dislike, how much material to buy, and where we spend our money. Typically all this information is stored and accessed directly through applications. These applications provide us access to this data through application programming interfaces or embedded reporting systems. Surprisingly, with all this dependency on data and the applications that manage them, few businesses have focused on ensuring that this data is turned into information, the difference being that information has inherent contextual value versus being raw, undefined content.

Of course, with the development of Web services and other integration technologies, we have the ability to turn our data into information on a singular and constrained basis. That is, we can turn a diverse data set into information around a particular use. For example, data from the accounting system can be coupled with data from the inventory system to generate invoices for our customers. However, these exercises don't often take the opportunity to classify the data as it's turned into information so it may be leveraged for other uses, such as vendor-managed inventory or supply-chain management. Instead, each of the aforementioned business functions would most likely use similar aggregation techniques to create the invoice to complete their tasks.

Classification of information is perhaps one of the most underrated and underused processes surrounding business data. Classification offers the business an ability to look for patterns of use surrounding this data, as well as access and search for this data more effectively in other downstream processes. This isn't a novel concept; intelligence organizations have been qualifying and classifying information for years. The difference now is the availability of XML, XML tools, and XML standards, such as Resource Definition Framework (RDF) and XML Topic Maps (XTM). These technologies make classifying and organizing data, turning it into information, affordable and easy.

Moreover, using XML to classify data provides additional opportunities to add important missing contextual metadata that allows businesses to identify the relationships between various sets of information. In our invoice example, we could develop a new data set that represents the relationship between inventory data and accounting data so we can dynamically explore and relate this data on demand. We don't need to have statically defined relationships that are bound to a single purpose, but instead dynamically defined relationships that can be reused for multiple purposes.

So what does it mean to classify information in this manner? XTM, for example, defines a framework that comprises topics, associations, and occurrences. The topic is a real-world subject, such as a book or a person. Topics are named and have an instance that is URI addressable called the occurrence. These topics also participate in associations with other data. Here's an example of a topic map:

```
<topic id="XMLJ">
    <instanceOf><topicRef xlink:href="#maga-
zine"/></instanceOf>
    <baseName>
      <baseNameString>XML
Journal</baseNameString>
    </baseName>
    <occurrence>
      <resourceRef
xlink:href="http://www.sys-con.com/xml"/>
    </occurrence>
  </topic>
```

In this example a map called XMLJ is described as being an instance of a magazine named XML Journal and having a reference http://www.sys-con.com/xml. While this may seem excessive to state a simple assertion, it's extremely powerful to be able to describe the relationships between sets of atomic data in this way. Because of this nomenclature, we can understand the relationship between the name XML-Journal and the Web site http://www.sys-con.com/xml. We can now answer this question of this information regardless of whether the requesting agent wants to understand what the URL points to or what the URL of the magazine is.

If we extend this paradigm to our internal data sets, we have a very powerful facility that will allow us to explore the relationships between sets of data in our stove-piped applications and thus provide effective and useful reuse and integration.

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

ADOS Co.,Ltd

http://www.a-dos.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

The Standards Democracy

WRITTEN BY **SIMEON SIMEONOV**

There's been much recent controversy about the role of Microsoft and IBM in the evolution of Web services standards. At a conference I attended not so long ago a pundit talked about the "standard setting duopoly." Several articles have been written about the "undemocratic" practices of WS-I. Are things really that bad?

Here's a somewhat cynical overview of the Web services standards process. MS and IBM pick an area of standardization. They cooperate privately on core concepts. They select a third partner to help them flesh out a draft specification. The partner is chosen according to two criteria: domain expertise that legitimizes the effort (e.g., VeriSign for security) and/or potential ability to create and successfully promote a competing specification (e.g., BEA for orchestration). A draft specification is released to the public. After some time, the draft is contributed to an official standards body such as W3C or OASIS. IBM, MS, and their partner work actively through the standards process to steer the end result. The final specification is released to the world. WS-I, led by IBM and MS, is positioned as the authority to provide a blessing to the specification through inclusion in WS-I profiles, implementation scenarios, and testing suites.

Obviously, IBM and MS have significant influence from start to finish. However, I don't see the process as necessarily undemocratic, even if it is an unusual blend of democracy and business.

In the abstract, democracy is a system in which everyone has equal representation and decisions are made through "one person, one vote." In the real world things aren't that simple. Not every person has a right to vote. Not every person is directly represented in decision making. Clearly, decision making is not done exactly through the principle of one person, one vote. Activists spend time campaigning for what they believe in on the principle of one person/hour, one vote. The bottom line is that those willing to spend time and money working toward a goal get increased decision-making power.

This suggests three things. First, companies that spend a lot of time and money on standards development should be expected to have more influence than companies that don't. Let's face it, Web services would be nowhere close to their current level of evolution without the efforts of MS and IBM. They were instrumental in creating a huge market that both customers and industry players benefit from.

Second, smaller companies should not bet their business plans on their ability to influence the direction of standards. That's reality; any exceptions just prove the rule.

Third, large companies that aren't significantly investing in standards development should stop complaining about not having enough influence. Whoever works the hardest on a standards working group or technical committee often has the most influence over the end result.

I don't buy the argument that the Web services standards process is undemocratic. It certainly isn't based on the principle of one company, one vote, but I think this is a good thing. In most standards committees 20% of the members do more than 80% of the work. They have the commitment to drive the process forward. The 80% are still valuable as reviewers and helpers that bring meaningful perspectives, but they often lack the time or dedication to engage in a serious and consistent manner. Further, the system is relatively open. Any abusers of power can be identified and dealt with.

It's interesting to ask whether IBM and MS are having too much influence on the evolution of Web services standards. There are two separate aspects to this question. Are they playing fair? (In a democracy there are many examples of inappropriate or illegal influencing practices.) Is what they're doing good for customers in the long run? (Are there more optimal standards development processes that we can use?) These are not yes or no questions.

The balance between standards monopoly, anarchy, and the hell that design-by-committee has proved to be is difficult to maintain without strong leadership. With such leadership comes the responsibility to act in the best interest of customers and the industry as a whole. Only time will tell whether the de facto process we have in place for Web services will yield the desired outcome.

Make sure that your preferred vendors follow both the spirit and the letter of the standards. Only customers and developers should have the final vote, over time!

AUTHOR BIO

Simeon Simeonov is a Boston-based principal at Polaris Venture Partners and a member of XML-J's Editorial Advisory Board.

SIM@POLARISVENTURES.COM

Sonic Software

www.sonicsoftware.com/webs

Reader Feedback

Re: *Eugene Kuznetsov's August article ("XML-Aware Networking")*

It's About Time

I've been turned off by XML because of crappy performance and the time it takes to edit stylesheets to get them to run more quickly. If processing XML in hardware gets me around bottlenecks, I say bring it on.

Alex Chiesi
via e-mail

Reality Check

I think issues around "XML acceleration" as the author puts it are more complex than the way it is described in the article. First, for large XML documents the network issues likely will overshadow any performance improvements in the "accelerator." Try posting a 100K file to a Web server! Second, the article is biased toward presenting pages (mentions page load time) and the Web applications that are (1) using XML/XSLT and (2) need high throughput. It does make some sense (if network issues do not exist!), but it will be useful to understand, other than the top 2-3 sites, where else this will be useful.

The XML firewall discussion is more balanced, but without major Web services adoption this is still [just] a nice concept. Like to see some reality thrown in here too!

Gary Johnson
via e-mail

Re: *JP Morgenthal's August Editorial ("Johnny Got Stuck in the Washing Machine")*

Got Help?

I really got enthused about your implication that the article would be about using XML more effectively in documents. Weekly I sit down with someone who is a database person who has no comprehension about how to manage information inside a document. This includes datapoints that many documents, such as contracts, include.

So your article was disappointing. How about some input as to how we can manage data inside documents using XML?

Barb Race
via e-mail

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

The Object

WRITTEN BY **COCO JAENICKE**

The object is certainly not a new concept, but Web services are considered new, difficult, and intimidating. Since a Web service can be thought of as a glorified object with standard interfaces, why isn't this old hat? Many of the difficulties of implementing services of any kind should be old hat, but in fact we've been cheating all these years.

The original motivation behind objects – going back even before the first object-oriented languages – was to enable reuse. By having self-contained and autonomous chunks of functionality, these chunks can be called by anyone at any time. At least in theory. Many of the limitations of early objects, such as proprietary data formats, sequencing issues, and interdependencies, were solved at the expense of being truly autonomous. Due to the fact that most early objects were small and local, no one really noticed.

Competition is pressuring organizations to integrate more and more across the extended enterprise, and with XML and the Internet, objects are getting bigger and are traveling farther. There is now a clear trend toward making large grains of functionality widely available to other organizations within the corporation and to select partners. Whether SOAP is used or not, this service-oriented approach really is an extension of the age-old object technology. What's different is the added complexity of crossing IT barriers, which makes cheating a nonoption. Connecting services to create business processes appears to be so simple and familiar but, as most architects find, the devil is in the details. What you need to do to make services work are things you should have been doing all along.

For an infrastructure to practically support services, the infrastructure has to support truly autonomous services. Three steps toward autonomy include:

- ***Asynchronous communication:*** Because services may be distant and aren't under local control, their availability is unpredictable. Infrastructure has to allow for services to go offline for whatever purpose without warning. Doing this requires that the invocation command and the payload be safely and reliably stored until the service is ready. In the case of request-reply communication, the reply has to be safely and reliably stored while it's being assembled. This information is referred to as a document, and XML provides the perfect format for information that's going to be transported between diverse locations and services.
- ***Dynamic information model:*** The document that gets passed from service to service needs to be highly dynamic. Since services will evolve independently, the information they accept and share will change independently. For example, the application for credit approval may change as risk models become more sophisticated, but the invoice may stay the same. For services to be truly autonomous they have to give up their dependence on a rigid and predetermined information model. XML plays a key role here. Because XML is extensible, the document passed between services can change to accommodate a change in one service without forcing change in the other services.
- ***Synchronization of services:*** When services are completely independent, they operate at their own pace and make decisions based on the information at hand. Getting services to fire correctly is probably one of the most difficult problems to solve because it goes against our natural tendency to look at processes as simple and linear. The reality is that most processes are filled with exceptions and hiccups along the way and the services that participate in the process are at risk because they don't have a unified view of the state of the entire process. For example, a credit application may be processed differently based on a complex array of risk factors that change over time. The solution is to make sure that the state of the entire process travels with the document. Again, XML is essential in making this possible. State information is variable and changes as processes evolve. By having an extensible information model, the annotations and metadata that describe the current state of the process can be easily tacked on to the original document, giving each service the information it needs to correctly operate within the scope of the larger purpose.

As new technology standardizes and simplifies some of the mechanics of working with objects, the scope of integration broadens the more stringent demands that cause new problems to arise. It does give us cause to wonder, however: How different would things be if the quaint objects of yore were as autonomous as services need to be today?

AUTHOR BIO

Coco Jaenicke was, until recently, the XML evangelist and director of product marketing for eXcelon Corporation. She played a key role in the successful development and introduction of eXcelon, the industry's first application development environment for building and deploying e-business applications. She is a member of XML-J's Editorial Advisory Board.

CJAENICKE@ATTBI.COM

Managing Juniper Networks' Routers Using XML

An XNM-based RPC mechanism

written by Mark Ethan Trostler

XML is the ideal platform for network management. XML's self-describing, text-based syntax is able to encapsulate the complex, hierarchical, and volatile configuration of any networked device. Controlling a network full of devices with proprietary and constantly evolving configuration syntax requires a Herculean effort. XML and XML Schemas help by providing a canonical representation and description of each device's configuration data and model. Even better, the vast and growing array of XML tools makes it even simpler and more intuitive to work with any XML document.

To leverage these in-herent strengths, Juniper Networks implements an XML-based RPC mechanism with which all of Juniper's M- and T-series routers can be queried, configured, and managed. An idea this powerful deserves its own acronym: XNM (XML Network Management).

Quick Tour

Clients send requests encoded within <rpc> elements and receive replies within matching <rpc-reply> elements. It's this simple:

```
<rpc>
     <get-configuration/>
 </rpc>
```

This interaction is shown in Figure 1.

Similarly, loading XML-encoded configuration is performed with the <load-configuration> element. The server responds – assuming no error has occurred – with an empty <rpc-reply/> tag, indicating success.

Supplying additional tags within the <configuration> tag allows a configuration subset to be retrieved. For example, the policy-options statement below the configuration root contains a set of routing policies that our client can request:

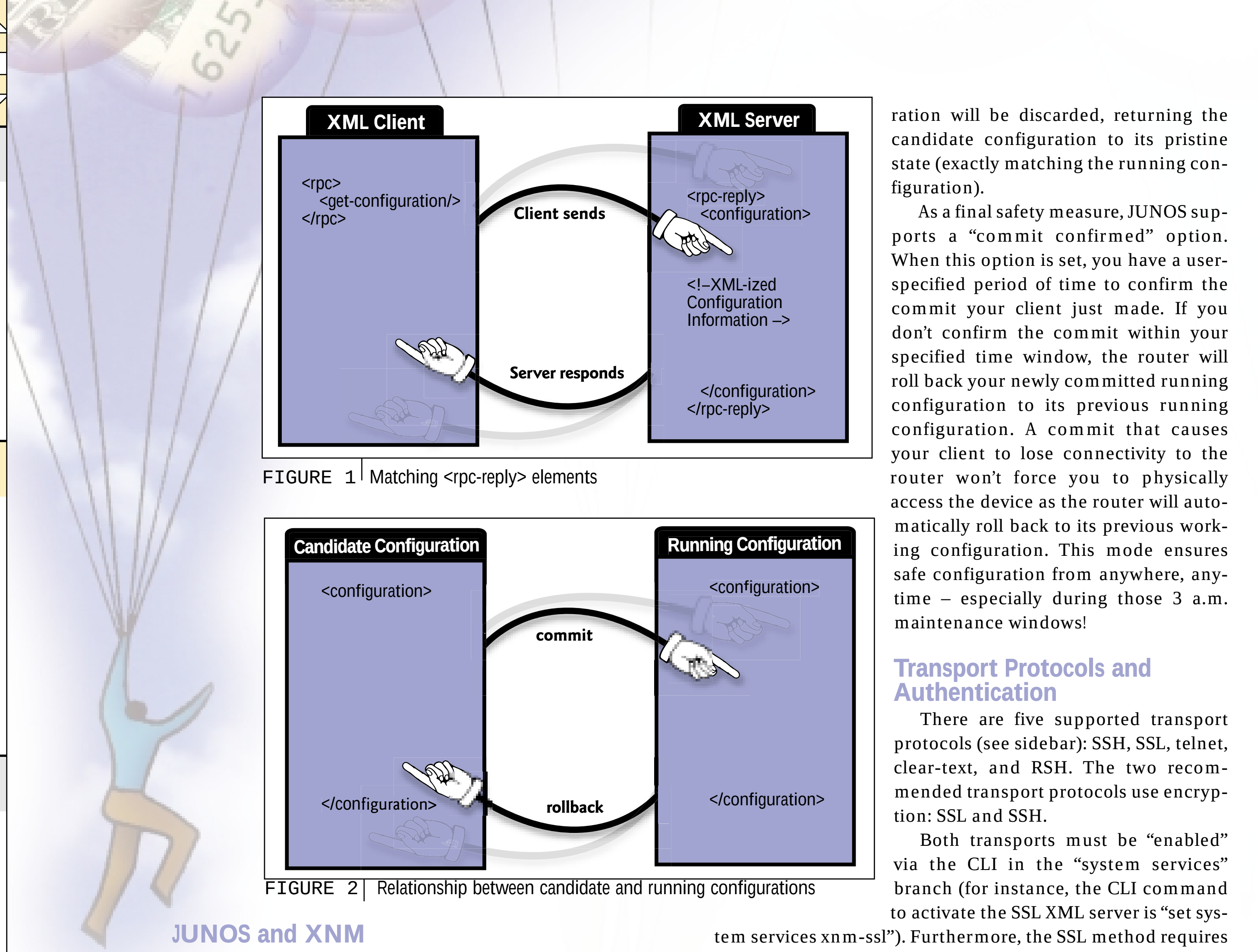
```
<rpc>
 <get-configuration>
  <configuration>
   <policy-options/>
  </configuration>
 </get-configuration>
</rpc>
```

This will retrieve all configured policy statements.

Configuration subsets can be configured on the device in a similar manner. Any configuration snippet supplied within the <configuration> element will be configured on the router. Supplying various attributes determines whether the configuration snippet should be created, updated, or deleted, as we'll see later. Success is indicated when the server responds with an empty <rpc-reply> tag.

Macromedia

www.macromedia.com/goCFMXlight



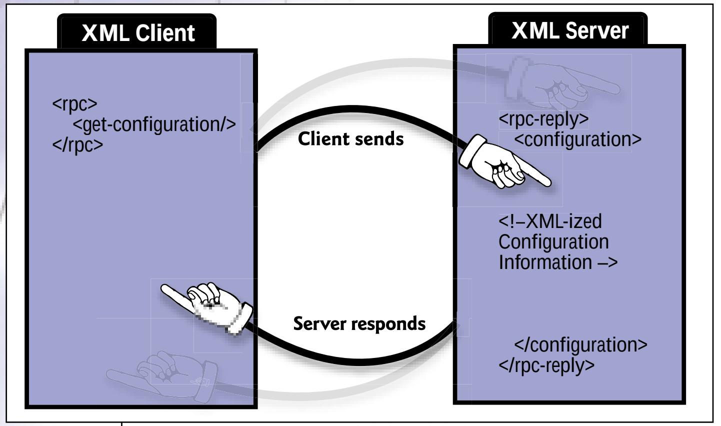


FIGURE 1 | Matching <rpc-reply> elements

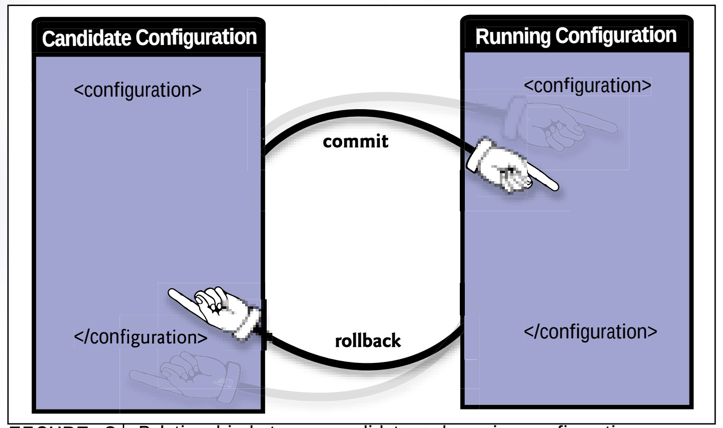


FIGURE 2 | Relationship between candidate and running configurations

JUNOS and XNM

The XNM server software contains several features that make using the XNM API both flexible and robust. Using a "commit-based" configuration model, a client writes configuration data to a "candidate" configuration. A client configuring a router works on a shared or private candidate configuration. Once the configuring is done, a "commit" operation must be issued to transfer the candidate configuration to the router's running configuration. By default, the candidate configuration is shared with anyone who happens to be configuring the router at any given instant.

When the router is configured remotely, it would be nice to lock everyone out of configuration mode except yourself. To accomplish this your client may "lock" the candidate configuration database, ensuring exclusive access only to you. Once your client has exclusive access to the candidate configuration, you can safely alter it without fear of conflicts with other API clients or CLI users. The relationship between the candidate and running configurations can be seen in Figure 2; Figure 3 shows the client accessing the candidate configuration simultaneously. Had any one of the three clients locked the candidate configuration, only that one would be allowed access to it.

JUNOS also supports automatic rollback of the candidate configuration. If you lose network connectivity or your client crashes while loading an XML configuration, all changes to the candidate configuration will be discarded and your exclusive lock on the candidate configuration database will be cleared (if you were holding it). When using this mode your client will not accidentally leave a half-baked candidate configuration for any reason. Any changes it made to the candidate configuration will be discarded, returning the candidate configuration to its pristine state (exactly matching the running configuration).

As a final safety measure, JUNOS supports a "commit confirmed" option. When this option is set, you have a user-specified period of time to confirm the commit your client just made. If you don't confirm the commit within your specified time window, the router will roll back your newly committed running configuration to its previous running configuration. A commit that causes your client to lose connectivity to the router won't force you to physically access the device as the router will automatically roll back to its previous working configuration. This mode ensures safe configuration from anywhere, anytime – especially during those 3 a.m. maintenance windows!

Transport Protocols and Authentication

There are five supported transport protocols (see sidebar): SSH, SSL, telnet, clear-text, and RSH. The two recommended transport protocols use encryption: SSL and SSH.

Both transports must be "enabled" via the CLI in the "system services" branch (for instance, the CLI command to activate the SSL XML server is "set system services xnm-ssl"). Furthermore, the SSL method requires the user to import server keys and certificates into the running configuration.

What really differentiates the access methods is their use of XML during the authentication process. The SSL transport uses XML <rpc> elements to authenticate the client. The SSH transports require a client to authenticate using the standard SSH protocol and then explicitly start the XNM server by issuing the "xml-mode" command.

Once authenticated and in XML mode (when using the SSH access method), the remaining interactions, whether using SSL or SSH, are the same. Figure 4 shows the entire XNM cycle. A client uses SSL to log into the XNM server, which mediates requests and responses between the client and the JUNOS kernel. In Figure 4 the client requests and the server responds with chassis information. A fully featured XNM client can be easily built from the provided XNM Schemas and an XML parser.

Initial Handshake

The XNM server begins its half of the session with the following tags:

```
<?xml version="1.0" encoding="us-ascii"?>
  <junoscript xmlns="http://xml.juniper.net/xnm/1.1/xnm"
version="1.0">
```

Two things to note:

1. ***The standard <?xml?> declaration:*** A session is two interleaved XML documents. The client's consists of possibly multiple <rpc> elements while the server's consists of matching <rpc-reply> elements, both encapsulated within

BEA
www.bea.com/events/dev2devdays

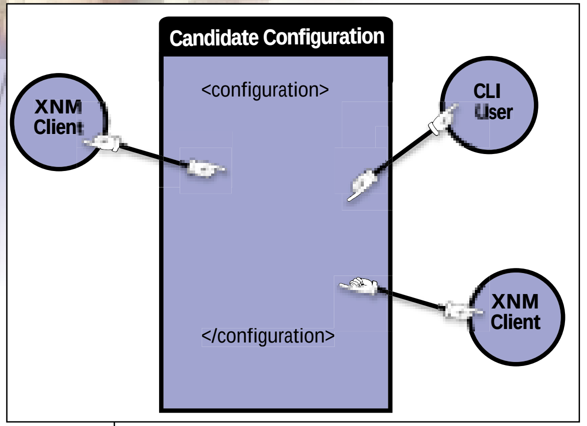


FIGURE 3 Simultaneous client access to the candidate configuration

the top-level <junoscript> element, Juniper's implementation of the XNM protocol.

2. ***The topmost enclosing <junoscript> tag:*** This tag contains two interesting attributes:
   —The default namespace is http://xml.juniper.net/xnm/1.1/xnm (see "The XNM Namespace" section for details).
   —The "version" attribute is the XNM API's version, allowing clients and servers to verify they can understand each other.
   The current and only version is 1.0.

The client is expected to respond with its own XML document. Note that the client's and server's initial handshakes are defined to occur simultaneously. As soon as the connection is open, each side must send its handshake:

```
<?xml version="1.0" encoding="us-ascii"?>
  <junoscript version="1.0" os="my-xml-client">
```

The client begins its own XML document and then specifies the version of XNM it understands. Conveniently, this version is compatible with that of the server, allowing these two to communicate. Servers will also be backward compatible with previous versions of XNM clients.

Authentication RPCs for SSL Connections

Similar to basic HTTP authentication, the RPCs for authentication are semistateless. You provide your username and password in the following manner:

Client sends

```
<rpc>
  <request-login>
    <username>
      <!-- Your username -->
    </username>
  </request-login>
</rpc>
```

Server responds

```
<rpc-reply>
  <login-challenge>
    <challenge echo="no">
      Password: <!-- Server challenge -->
    </challenge>
  </login-challenge>
</rpc-reply>
```

Only one "challenge" is defined – a password. The text in the <challenge> tags is meant to be a prompt for interactive use – the "echo" attribute tells the client whether the user's response to the prompt should be echoed or not. In addition, white space is *not* ignored within the <username> and <challenge-response> tags.

The client should now respond:

```
<rpc>
  <request-login>
    <username>
      <!-- Your username -->
    </username>
    <challenge-response>
      <!-- User's password -->
    </challenge-response>
  </request-login>
</rpc>
```

The client must send the username again, along with the <challenge-response> element (I told you it was "semistateless"!). If the password is valid, the server will respond with:

```
<rpc-reply>
  <authentication-response>
    <status>success</status>
    <message>
      <!-- Name of logged in user -->
    </message>
  </authentication-response>
</rpc-reply>
```

If not, the <status> element will contain "fail" and the appropriate error message will be sent in the <message> element.

Since we know a priori that a "<challenge>" element will be issued, we can simply send the <challenge-response> element with our password in the initial <request-login> RPC to authenticate ourselves immediately.

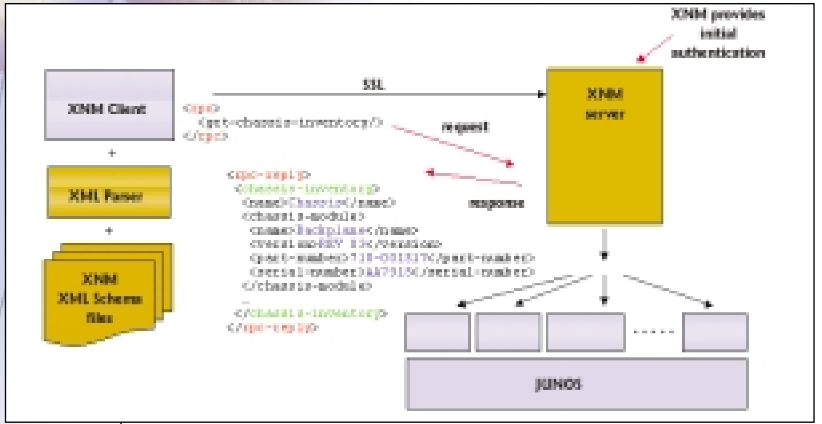
The XNM Namespace

The XNM namespace URI takes the form http://xml.juniper.net/<area>/<release>/<area>. The release version is encoded in the namespace. Although this gives us greater con-

Buzzword Glossary

BGP	Border Gateway Protocol used to exchange routing information
JUNOS	Operating system for Juniper's routers
JUNOScript	Juniper's implementation of the XML Network Management (XNM) API
NMS	Network Management System
RPC	Remote Procedure Call
RSH	Remote Shell, a clear-text transport standard
SSH	Secure Shell, an encrypted transport standard
SSL (TLS)	Secure Sockets Layer (Transport Layer Security), an encrypted transport standard
telnet	A clear-text transport standard
XNM	XML Network Management

PolarLake
www.polarlake.com



```
<rpc>
  <get-chassis-inventory/>
</rpc>
```

```
<rpc-reply>
 <chassis-inventory>
  <name>Chassis</name>
  <chassis-module>
   <name>Backplane</name>
   <version>REV 03</version>
   <part-number>710-001517</part-number>
   <serial-number>AA7919</serial-number>
  </chassis-module>
  ...
 </chassis-inventory>
</rpc-reply>
```

FIGURE 4 | The entire XNM cycle

trol over the contents of each XNM release, it makes using XSLT with our documents painful. Since XSLT documents are namespace specific, a transformation that works with one release will require mangling to work with subsequent releases. Namespaces can be canonicalized at runtime, but admittedly that's a hassle. Such is life....See the "Tools" section for some help.

Requests/Responses, and Closing the Connection

Once the client has been authenticated, it may make <rpc> requests to the server. When the client has no more requests, a </junoscript> tag is sent telling the server that the party is over and it's time to close the connection. The server responds with its own </junoscript> tag and the connection is closed.

Tags of Interest

If an error occurs, the client will receive an <xnm:error> element enclosing a <message> element with a plain-text explanation of the error. There may be other tags within an <xmm:error> element containing more detailed information:

```
<rpc>
  <bogus-request/>
</rpc>
```

will cause the server to respond with:

```
<rpc-reply>
  <xnm:error xmlns="http://xml.juniper.net/xnm/1.1/xnm"
xmlns:xnm="http://xml.juniper.net/xnm/1.1/xnm>
    <message>syntax error</message>
  </xnm:error>
</rpc-reply>
```

Earlier I spoke of locking the candidate configuration so no one steps on your toes while your client is configuring the router. The following RPC accomplishes this:

```
<rpc>
  <lock-configuration/>
</rpc>
```

If no other users are currently in "configuration mode," and the candidate configuration matches the running configuration, you'll receive an empty <rpc-reply> tag indicating success. If another user already has an exclusive lock on the configuration database, or the candidate configuration has been modified, you'll receive an <xnm:error> element.

If, while in configuration mode, your client would like any changes it makes to be automatically rolled back in case it crashes or accidentally loses network connectivity, send this:

```
  <rpc>
    <lock-configuration
rollback="automatic"/>
  </rpc>
```

This automatically rolls back the can-didate configuration to match the running configuration if you lose your connection or log out without committing your candidate configuration as discussed above. To unlock the configuration:

```
<rpc>
  <unlock-configuration/>
</rpc>
```

If your session closes for any reason before you send the <unlock-configuration/> tag, your session's exclusive lock will automatically be released. If you specified 'rollback="automatic"' in your <lock-configuration> tag, any uncommitted changes will be discarded.

Committing Your Changes

Your client has several options when committing its changes:

```
<rpc>
  <commit-configuration/>
</rpc>
```

is the most basic. If any error occurs while attempting to commit the configuration, your client will receive an <xnm:error> tag.

To "test" your configuration's syntax and semantics before attempting to commit it, use:

```
<rpc>
  <commit-configuration>
    <check/>
  </commit-configuration>
</rpc>
```

Again, your client will receive either an <xnm:error> if there's something wrong with the candidate configuration, along with an explanation of the problem, or an empty <rpc-reply> element upon success.

Finally, as mentioned above, your client can place a "confirm" restriction on its commit, necessitating confirmation during the specified time period. If you don't confirm the commit by recommitting the configuration, your confirmed commit will be rolled back to the previous running configuration:

```
<rpc>
  <commit-configuration>
    <confirm/>
    <confirm-timeout>10</confirm-timeout>
  </commit-configuration>
</rpc>
```

This gives you 10 minutes to do another <commit-configuration> RPC before the rollback to the previously running configuration occurs.

Matching Requests and Responses

Any attribute sent in an <rpc> open tag is echoed by the matching <rpc-reply> tag:

```
<rpc date="5/3/2002" op-num="234">
  <get-interface-information/>
</rpc>
```

will return:

```
<rpc-reply date="5/3/2002" op-num=234">
  <!—XML-ized interface information -->
</rpc-reply>
```

This is handy for matching <rpc> tags with their matching <rpc-reply> tags.

XML Schema Validation

XML-encoded configuration is completely described by an XML Schema available to your client:

```
<rpc>
   <get-xnm-information>
     <type>xml-schema</type>
     <namespace>
        junos-configuration
      </namespace>
   </get-xnm-information>
</rpc>
```

This RPC will return the XML Schema without the <?xml?> declaration in an <rpc-reply> element. It's very large – you only want to do this once.

Create, Replace, Update, Delete

CRUD, the magic incantation. Here's how to do each:

Create is supported via the <load-configuration> tag.

To completely replace the configuration with the contents of your load-configuration tag use:

```
<rpc>
  <load-configuration action="override">
    <configuration>
      <!-- Entire router configuration -->
    </configuration>
  </load-configuration>
</rpc>
```

By default the <load-configuration> element simply adds configuration data.

To replace configuration data set the replace attribute to "replace":

```
      <configuration>
        <policy-options>
          <prefix-list replace="replace">
            <name>list1</name>
              <prefix-list-item>
                <name>192.168.0.0/16</name>
              </prefix-list-item>
 </prefix-list>
    </policy-options>
 </configuration>
```

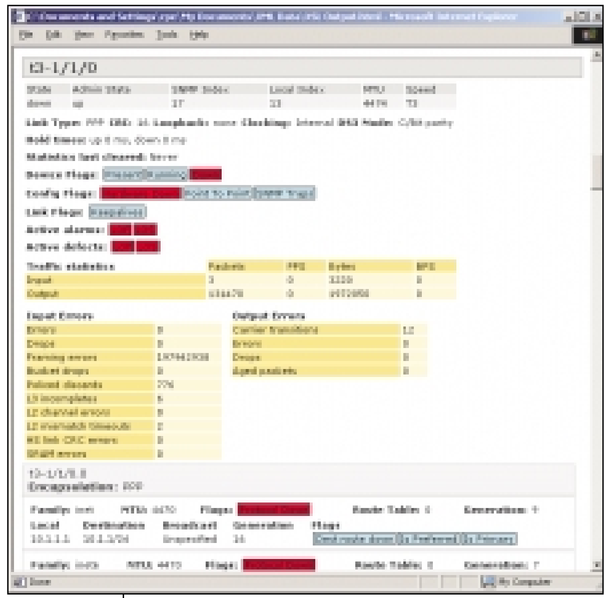


FIGURE 5 | Detailed view of interface parameters

This will replace all of the prefixes once contained in the prefix list "list1" with the single prefix 192.168.0.0/16.

If the attribute was "merge" and was set to "merge", the new prefix would be appended to the list of currently configured prefixes.

You can mix and match merges and replaces within a single configuration snippet. Simply add a 'replace="replace"' or 'merge="merge"' attribute in any enclosing configuration tag.

To delete simply include a "delete" attribute set to "delete". This:

```
      <configuration>
        <policy-options>
          <prefix-list">
            <name>list1</name>
              <prefix-list-item delete="delete">
                <name>192.168.0.0/16</name>
              </prefix-list-item>
  </prefix-list>
     </policy-options>
 </configuration>
```

will delete the prefix 192.168.0.0/16 from the prefix list "list1".

Configuration read is supported via the <get-configuration> element we've already seen:

```
  <rpc>
    <get-configuration>
      <configuration>
        <protocols>
          <bgp/>
        </protocols>
      </configuration>
    </get-configuration>
  </rpc>
```

will retrieve only BGP configuration information.

Putting It All Together

Listing 1 is an entire XNM conversation. For readability the client requests and server responses are shown in separate blocks and white space has been added. In your conversations with an XNM server, the server responses will be "mixed" in the datastream. For instance, as soon as the server sees an <rpc> it will immediately respond with an <rpc-reply> tag.

This example gets the policy-options tree, locks the candidate configuration, applies the "keep-list1-out" policy to the Fast Ethernet interface "fe-0/0/0", and then commits the configuration.

The "junos:key" attribute in various tags provides our client with an important hint. The server is telling us this is the "handle" necessary to delete, replace, or update this object. When I merged the policy-statement "keep-list1-out", I knew the server needed the tag <name> to know exactly what to merge. Think of tags with this attribute of "primary keys" in database-speak, a unique handle that allows us to differentiate among possibly many members of a set with the same type.

Tools

After our quick tour of the raw XNM API and protocol we never need to see it again. As there are many XML tools for many programming languages, as well as standalone applications, we can (and should) hide all these details behind a nice, clean, user-friendly API.

Juniper has created an open-source, object-oriented Perl client. Several examples are included with the Perl client for doing basic operations like getting and setting configuration, as well as the Looking Glass Web application described below. Examples of canonicalizing namespaces are also provided.

This client is meant as a library for programmers to use. An example is a "Looking Glass" Perl CGI script. Figure 5 is a screenshot of the Looking Glass application displaying a detailed view of interface parameters. Using the Perl client and the XSLT language, we provide a pretty Web interface to our router's most common operational parameters using surprisingly little code. Other, higher-level tools could also enhance the raw XNM protocol to provide just about any functionality to a single or large set of routers, from simple query-based tools to a full-blown NMS.

As Mikey says: "Try it, you'll like it!"

Acknowledgments

In January 2001 the Juniper Networks management team released the first version of JUNOScript. Thanks, Rob Enns, Phil Shafer, and the rest of the XML hackers!

Resources

- *In-depth white paper on XML network management:* www.juniper.net/techcenter/techpapers/200017.pdf
- *JUNOScript's home page (download the Perl module from here):* www.juniper.net/support/junoscript
- *All things JUNOScript, in detail:* www.juniper.net/techpubs/software/junos54/junoscript54-guide/html/junoscript54-guideTOC.html
- *Reference/usage information on operational/configuration tags available in JUNOScript (beware...it's a big list!):* www.juniper.net/techpubs/software/junos54/junoscript54-ref/html/junoscript54-refTOC.html

AUTHOR BIO

Mark Ethan Trostler is a software engineer in the Network Management group at Juniper Networks, Inc. In addition to creating proprietary network management solutions for various companies, he has consulted at Qualcomm and the San Diego Supercomputer Center where he exhibited distributed supercomputer projects at both Supercomputing '95 and '98.

TROSTLER@JUNIPER.NET

LISTING 1

```
S: <?xml version="1.0" encoding="us-ascii"?>
S: <junoscript xmlns="http://xml.juniper.net/xnm/1.1/xnm"
xmlns:junos="http://xml.juniper.net/junos/5.4R1/junos"
schemaLocation="http://xml.juniper.net/junos/5.4R1/junos
junos/5.4R1/junos.xsd" os="JUNOS" release="5.4R1 [trostler]"
hostname="ui2" version="1.0">
S: <!-- session start at 2002-08-07 13:13:53 PDT -->

C:<?xml version="1.0" encoding="us-ascii"?>
C:<junoscript version="1.0" os="test-client">
C:<rpc>
C:<request-login>
C:<username>trostler</username>
C:<challenge-response>mypassword</challenge-response>
C:</request-login>
C:</rpc>

S:<rpc-reply>
S:<authentication-response>
S:<status>success</status>
S:<message>trostler</message>
S:</authentication-response>
S:</rpc-reply>
S:<!-- No zombies were killed during the creation of this
user interface -->
S:<!-- user trostler, class super-user -->

C:<rpc>
C:<get-configuration>
C:<configuration>
C:<policy-options/>
C:</configuration>
C:</get-configuration>
C:</rpc>

S:<rpc-reply
xmlns:junos="http://xml.juniper.net/junos/5.4R1/junos">
S:<configuration>
S:<policy-options>
S:<prefix-list>
S:<name junos:key="key">list1</name>
S:<prefix-list-item>
S:<name junos:key="key">192.168.12.0/24</name>
S:</prefix-list-item>
S:</prefix-list>
S:<policy-statement>
S:<name junos:key="key">foo</name>
S:<from>
S:<prefix-list>
S:<name junos:key="key">list1</name>
S:</prefix-list>
S:</from>
S:<to>
S:<protocol junos:key="key">static</protocol>
S:</to>
S:<then>
S:<trace/>
S:<reject/>
S:</then>
S:</policy-statement>
S:</policy-options>


C:<rpc>
C:<lock-configuration/>
C:</rpc>

S:<rpc-reply
xmlns:junos="http://xml.juniper.net/junos/5.4R1/junos">
S:</rpc-reply>

C:<rpc>
C:<load-configuration>
C:<configuration>
C:<policy-options>
C:<policy-statement merge="merge">
C:<name>foo</name>
C:<to>
C:<interface>fe-0/0/0</interface>
C:</to>
C:</policy-statement>
C:</policy-options>
C:</configuration>
C:</load-configuration>
C:</rpc>

S:<rpc-reply
xmlns:junos="http://xml.juniper.net/junos/5.4R1/junos">
S:</rpc-reply>

C:<rpc>
C:<commit-configuration/>
C:</rpc>

S:<rpc-reply
xmlns:junos="http://xml.juniper.net/junos/5.4R1/junos">
S:<commit-results>
S:<routing-engine>
S:<commit-success/>
S:</routing-engine>
S:</commit-results>
S:</rpc-reply>

C:<rpc>
C:<unlock-configuration/>
C:</rpc>

S:<rpc-reply
xmlns:junos="http://xml.juniper.net/junos/5.4R1/junos">
S:</rpc-reply>

C:</junoscript>

S:<!-- session end at 2002-08-07 13:30:53 PDT -->
S:</junoscript>
Connection closed by foreign host.
```

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

WRITTEN BY **SIMEON SIMEONOV**

Web Services Startups

Can they grow big?

You must have seen them – they're everywhere. Despite the market climate, Web services startups are popping up like mushrooms after rain. With no time to lose, these companies are readying themselves to take on the industry gorillas. Who's going to win?

Much has been written about the evolution of the Web services industry, particularly with respect to the big players like Microsoft and IBM. We've all been told countless times how Web services are good for established platform vendors ranging from application server companies to integration and process management vendors. One interesting question is often evaded, however: How can startups make money in this world?

A Matter of Perspective

I wish I had all the answers. I don't, of course. However, my experience does give me a certain perspective. I've been part of the Web services "revolution" since before it had that name and we were just starting to experiment with XML for distributed application integration back in 1998. I've been involved with standards development in that space for several years (SOAP, JAX-RPC, etc.). I've lived through the Internet platform battles and I see a lot of similarities between those days and what's happening now with Web services. And now, as a venture capitalist, I have the luxury of talking to a never-ending streak of entrepreneurs trying to change the world with Web services.

Good VCs stay sharp because they're in constant contact with people who are smarter than they are. Now, having met with dozens of Web services startups and industry pundits, I feel I understand the space much more broadly than when I was at Allaire and Macromedia. Not having a corporate agenda through which to view the world is indeed eye opening. This is the perspective I hope to bring to this and future installments of **XML in Transit**.

Growing Big

It would be nearly impossible to talk in general about opportunities for all types of startups in the Web services space. Some type of filter needs to be applied. From my perspective as a VC, the filter I apply has to do with the magnitude of the opportunity. Venture-backed companies have to meet certain stringent performance requirements. VCs take on so much risk investing in startups that they have to be compensated with very significant returns. After several years of helping a company grow, a VC looks to make a five- to tenfold return on investment.

Here's an example. Imagine that it takes your startup $10 million to $20 million to reach a state where your investors can cash out. This means that you get acquired by a public company or that you go public. Assume that by that time, after several rounds of financing, VCs own two thirds of your company. For them to get that five- to tenfold ROI, your company must be valued at $75 million to $150 million for a $10 million investment and $150 to $300 million for a $20 million investment. Getting these types of valuations was easy several years ago. In the current market it's a lot harder. Given where valuations are at these days and where they're likely to stay in the future, it's easy to say that to be worth this much a startup needs to have revenues in the range of $50 million to $100 million or more. Top-tier VCs are only willing to back big plays.

Therefore, an interesting filter to apply when talking about startup opportunities in the Web services space would be whether the company has reasonable success of getting funded by such a firm. In other words, is the company going to have a significant and sustainable business?

Don't get me wrong. I'm not trying to make any kind of a value judgment about non-VC backed companies. Companies that never seek outside investor funding have equally smart people in them and work with equally exciting technologies. For example, several smart people can get together and build a great narrowly focused product that solves a real customer problem and gets acquired in a couple of years for a good chunk of money. They'll have done us all a service by accelerating development in the industry and will have succeeded financially as well. It's all good, but not something that a VC would typically be interested in. It's just not big enough.

Now that we've narrowed our focus to big and sustainable businesses in the Web services space, let's look at some common patterns they're likely to share.

Web Services Confusion

What does the term *Web services company* mean, anyway? Are we talking hardware or software, appliances or servers, infrastructure or applications, product or service? What, if any, verticals are targeted? Beyond companies working exclusively on core infrastructure related to the Web services technology stack, the term is largely meaningless. It's used imprecisely and often inconsistently, so it's important to get clarification on what exactly vendors mean when they tout Web services support.

This reminds me of how we used the term *XML company* in the late 1990s. Don't worry, it'll pass, just as that one did. For the time being, however, the term is widely used in the industry, so who am I to argue?

AUTHOR BIO

Simeon Simeonov is a Boston-based principal at Polaris Venture Partners with a focus on opportunites in information technology. A founding member and chief architect at Allaire, Sim led the development of ColdFusion, one of the first Web application servers. Following the merger with Macromedia, as chief architect and VP of emerging technologies, he focused on Web services and other platform infrastructure for next-generation Internet applications. Sim was actively involved with standards development through W3C, OASIS, and JCP.

Richard Hale Shaw Group

www.richardhaleshawgroup.com

Patterns for success

It's pretty amazing that, for the first time in computing history, the industry is taking more or less the same approach to connecting and integrating applications through Web services. However, thinking of Web services as a revolution in computing misses the point that, fundamentally, Web services aren't new. They're just the current step in the evolution of our thinking about distributed computing (see Table 1). Since prehistoric times and up to the 1980s we did this through custom RPCs and proprietary messaging platforms. In the 1990s we used COM, CORBA, Java RMI, and a number of XML-based approaches. We're starting the first decade of the new millennium with Web services. Ten years from now further convergence of application interaction patterns and the broad use of meaningful standards – for example, grid-related ones – will undoubtedly lead to new and interesting ways of doing distributed computing.

PERIOD	DISTRIBUTED COMPUTING TECHNOLOGIES
<1990	Proprietary RPCs and messaging systems
1990+	DCOM, CORBA, JAVA RMI, XML over HTTP
2000+	Web services (SOAP, WSDL, UDDI) and more
2010+	???

TABLE 1 | Evolution of distributed computing approaches

It's important to internalize this: Web services, just like XML, are not an end in themselves. They're tools for getting things done. Therefore, a key pattern for successful startups is that they must focus on some core capability that generates significant value by addressing real customer needs, and use Web services to enhance this value proposition. This may seem like an obvious point, but I've talked to several startups that have built great technology focused on Web services themselves without a tie-in to real customer needs. Technology for technology's sake has not made people rich.

Another reason why it's so important to think pragmatically about Web services is that despite all the hype, the Web services marketplace is going to evolve more slowly than most expect. In the current business climate customers are cutting costs. Fewer IT projects are getting funded. This leaves less money to support the efforts of pioneering startups. For your startup to succeed in this climate, it has to find ways to make money sooner rather than later. Some VCs may be willing to fund grand blue-sky ideas, but most will look to back companies that can quickly generate significant customer traction. This can happen only if the companies are building solutions addressing real customer needs. Web services become an enabler and an accelerator of these solutions.

One point of caution: unfortunately, many startups have equated the need for quick revenue growth with a focus on tactical or opportunistic Web services dimensions, typically expressed as following the general path of evolution of the Web services industry to feed on emerging demand, but staying six to 12 months ahead of standards and big industry players. On average, this approach doesn't work. It's too risky. The margin for error is too small. The sustainable competitive differentiation is insignificant. Don't do it.

The second key pattern of building a successful startup is not specific to Web services. You must maintain a strong link to your customers over time. You need sustainable market positioning that will evolve as the market changes. Again, this is common sense, but doing it well requires deep understanding of the Web services industry and solid intuition about its evolution. This is how you can stay ahead of incumbent players that are slowly adopting Web services and position against competing startups, both of which will be trying to take away your customers.

Most startups don't live a long time. History speaks for itself. Few large, successful software companies are more than 10 years old. Most companies either disappear or get acquired for little. Building a large company isn't easy. It's a lot harder to evolve with the market and with customer needs than it is to build a product that does something well at a given point in time. The Web services landscape is rapidly changing. What seemed like a great idea 12 months ago may make little sense today. With product cycles running six to 18 months, startups have little margin for error. Therefore, an extension to the second pattern is the need to build a highly adaptable organization. It takes great leadership to do this.

In short, Web services must enable or enhance a core capability that is key to a product or service that meets a pressing need (see Figure 1). For long-term success, startups must keep delivering significant value over a sustained link to their customers. Here's my litmus test for a startup in this space: If Web services didn't exist, would the core features of its products still provide significant sustained value? If the answer is Yes, then there's a real business there that can be accelerated with Web services. If the answer is No, this doesn't necessarily mean that the startup won't be successful. However, it does suggest that the company faces significant market evolution risk because it may be operating too close to the core Web services infrastructure. Its value proposition is likely to be subsumed over time as incumbents add native Web services support.

Some examples

Let's look at some examples of applying this way of thinking to opportunities in the Web services space. We can start with the most obvious one: connecting applications using Web services. This was all the rage in 2000 and 2001. The customer need is clear: companies have been trying to get their applications to communicate for decades. The core technology is software that can expose applications for remote access. Web services provide a common interaction mechanism. Marry SOAP and WSDL with this technology and you have a Web services engine. Add to that some development and deployment tools and some management capabilities and you have the initial product offerings of companies such as Cape Clear and Systinet (to pick two out of a large pack).

Now, have startups made lots of money selling Web services engines? I don't know of any that have. To see why, we can apply the model we developed. The customer need is still there. The perceived value, however, has been eroded by commoditization. Many products now have the ability to expose components as Web services. On top of this, there are multiple open-source Web services engines, for example, Apache Axis. Most important, all the major platform vendors have this capability built into their products. The value proposition has eroded, and so has the link to the customer. Web services are typically exposed in one of three ways: (1) directly out of packaged applications, (2) via custom application components running on application servers, or (3) through message brokers. Well, the packaged applications vendors are slowly but surely getting on the Web services bandwagon, and, as already mentioned, the application server and messaging vendors are leading the Web services charge. If customers already have Web services capabilities when they buy an enterprise application, an application server, or a message broker, why would they buy a Web services engine from a third party?

This example is a classic case of one of the most common antipatterns of startups: focusing on a short-term opportunity and ignoring the consequences of inevitable changes in the industry. In general, if a startup is in the business of Web service–enabling some capability *X*, and it doesn't own or have control over how *X* is implemented by its customer, it will likely end up in trouble.

It's okay to focus on a short-term opportunity to generate traction, but you have to know when to move on to adjacent markets and you must move quickly. This is what, for example, both Systinet and Cape Clear are trying to do. They started by providing Web services engines and related tools. Now they're moving away from this space toward providing service integration capabilities and deeper infrastructure focused on quality of service (QoS). Will this work? If the companies have strong and insightful leadership, and if the near-term economic environment improves and supports more advanced applications of Web services, the companies may be well positioned.

The same argument that we applied to companies that focused on Web service connectivity can be applied to startups targeting service discovery, integration, and orchestration. These are not new concepts either. They've been around for a long time under various names: directories, naming services, EDI, EAI, workflow, BPA, B2B. There are a number of established players with both capital to defend their position and customers to deliver enhanced products to. Also, the platform vendors that didn't have offerings in this space are using Web services as a means for entering these markets. Sure, a startup now may discover, integrate, and orchestrate applications better with Web services, but what is its long-term sustainable position? Does its embrace of Web services give it a fundamentally defensible position with respect to vendors that are using legacy technology? Will it be able to achieve critical mass in the time it takes for established players to achieve similar capabilities? Will it be able to win and keep new customers even when Web services discovery, integration, and orchestration become mainstream several years down the road? Does it have what it takes to stay three steps ahead at all times?

It's no longer easy to get acquired or go public in a couple of years. In the current climate it can easily take three to seven years for this to happen. We're talking about a marathon-length course that startups have to run at sprinting speeds, a challenge that strains even the strongest of companies. And, of course, the general rule of entrepreneurs is to assume that if they woke up one day with a great idea that would change the world, it's likely that some other people woke up with the same idea that day. Since I'm now in the position to look at many startups, I can assure you that there are many competitors in the race that have more similar approaches than they realize. Companies start the race from different points but are aiming at the same finish line. This phase is going to be great for customers of Web services–enabled products, albeit also very challenging for ISVs and startups trying to leverage these emerging technologies.

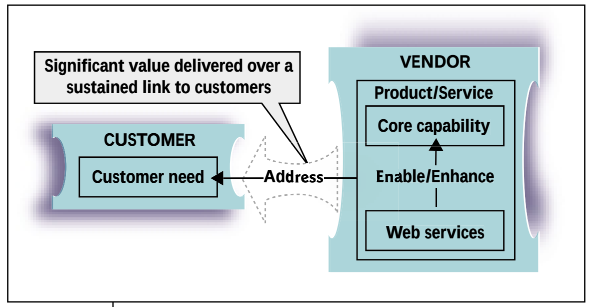
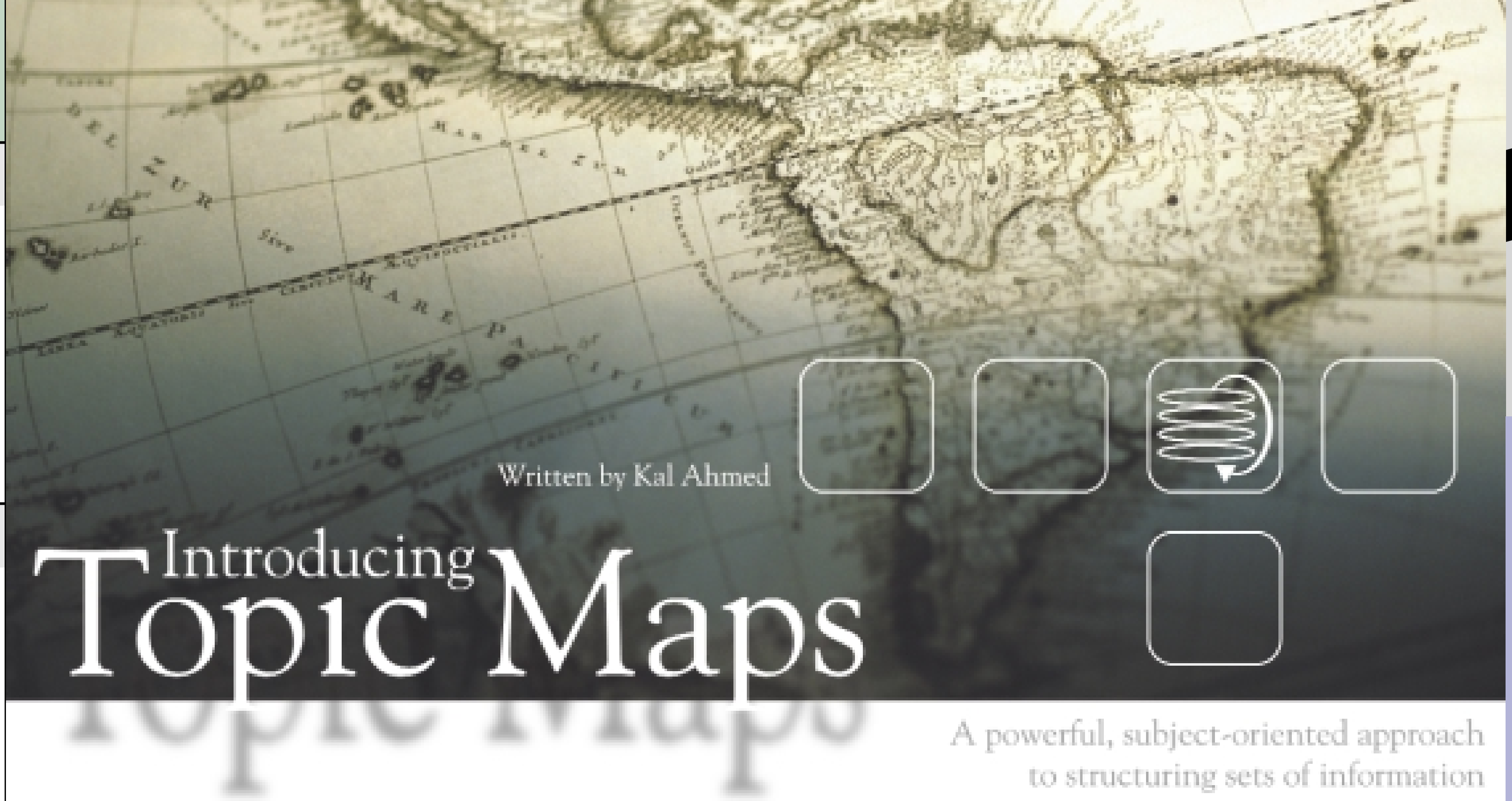


FIGURE 1 | Model for analyzing Web services opportunities

I don't mean to sound pessimistic. There are lots of opportunities for building great companies. The next installment of **XML in Transit** will focus on some of the ones I see. In the meantime, send me an e-mail and let me know what kind of Web services startup you are excited about.

SIM@POLARISVENTURES.COM



Written by Kal Ahmed

Introducing Topic Maps

A powerful, subject-oriented approach to structuring sets of information

Topic maps are a standard way of representing the complex relationships that often exist between the pieces of information that we use in day-to-day business processes. This article begins by discussing what topic maps are, what they can do, and what people are currently using them for. However, my main goal is to introduce the basic concepts of topic maps and their representation in XML.

The Missing Link in Information Management

Almost all XML vocabularies are designed with a single purpose: to describe information in a way that enables automated processing. We use XML to describe document structures so our documents can be rendered as HTML, WML, PDF, or some other presentation format. We also use XML so that business systems can interchange data reliably. Both the ability to render content to different output formats and the reliability of data interchange arise from a predefined agreement about how individual pieces of markup describe the information they wrap.

There's no doubt that information markup is of immense use in automated processes, but alone it can't describe the relationship between different resources or between an information resource and the subject or subjects that the information resource describes. In many systems it's these relationships that enable human beings to make sense of and organize the information they need to work with. In some respects, then, these relationships are the missing link between manageable and unmanageable information.

Topic Maps: Subject-Oriented Markup

If the current set of information-oriented vocabularies based on XML isn't sufficient for information management purposes, it would indicate that what's needed is an alternate view of the information. One such alternate view is provided by topic maps.

In a topic map, rather than focus on the information, you focus on what the information is about. The markup you create is subject oriented rather than information oriented. The fundamental unit of a topic map is a topic, an electronically processible stand-in for a thing that doesn't necessarily have to be processible itself. For example, a topic can represent a person, a building, or a concept, but it can also be used to represent processible things such as Web pages, files, or database cells. In the topic map approach the thing a topic represents is called the subject of the topic.

A single document may describe many different subjects and may establish relationships between those subjects. Using topic map constructs, a topic map author can describe the relationships between the subjects described by different topics and also point to resources that provide information related to the topic in some way. In addition to topics, the two other basic topic map constructs that make this possible are associations, which relate topics to each other, and occurrences, which connect topics to resources that contain some information related to the topic. These three basic concepts are shown in Figure 1.

A Brief History of Topic Maps

The first standard to describe the topic map approach was published by ISO early in 2000 as ISO 13250:2000. In addition to describing the paradigm itself, ISO 13250 describes an interchange syntax making use of SGML Architectural Forms and link structures defined by the HyTime standard (ISO 10744). The interchange syntax described in this first edition of the ISO topic map specification is highly flexible, enabling users to derive their own DTDs for describing their own information structures, and allowing the use of the full range of multimedia linking structures defined by HyTime.

Soon after the publication of ISO 13250, a separate group called TopicMaps.Org was formed to create a topic map interchange standard based on the ISO standard, but with its syntax expressed in XML rather than SGML and with a linking mechanism using the W3C XLink standard rather than HyTime. The result was the XML Topic Maps (XTM) specification, published early in 2001. While some topic map applications support a restricted subset of the ISO 13250 interchange syntax, almost all those available today support XTM 1.0. For this reason (and because this is an XML magazine!), I'll only describe XTM interchange syntax.

Applications of Topic Maps

Before getting into the technical detail about how topic maps are constructed, let's briefly survey the state of the art with regard to the use of topic maps. Current applications can be broken down into three major purposes:

1. A way to improve accessibility to information
2. A flexible, extensible data structure for standalone applications
3. An integration layer between existing applications

Topic maps for information accessibility

This is by far the largest category of current applications of topic maps. Topic maps are being used as the structuring paradigm for portals to all sorts of information. Because they're subject oriented, they provide an intuitive way for users to find their way around large sets of information. Using a topic map, the information provider can create a high-level overview of the concepts covered by the documents. Users can then navigate the overview to find the subject area of interest before accessing the related documents. You can also go the other way – from a document to a list of all subjects in the document and from there to other documents related to the same subject(s). Because all of this structure is explicit in the topic map, navigating from subject to document and back again can be done without requiring the user to construct search terms.

Many commercial topic map tools focus on making information more findable and presenting navigation structures for topic maps. Thus, once you have your information structured using topics maps, it's possible to get a configurable, off-the-shelf application to do the work of presenting that structure to the end users.

Topic maps for programmers

A topic map is a data-driven structure. As you'll see, to create new categories of things and relationships between things, it isn't necessary to modify a fixed schema. Instead, you simply add new data to the topic map. Currently, the use of topic maps for representing data structures is hampered by the lack of a common API for accessing and modifying topic map information, but work is under way to define such an API.

Topic maps for integration

The basic structures of a topic map – topics, their relationship to each other, and their relationship to information resources – can be used to represent information harvested from a wide variety of business systems, from databases through content management systems to workflow systems. By making use of a single standard data structure and by identifying the common business subjects that each of these systems describe, it's possible to create a unified topic map view of the business and its processes.

Topic Maps in Detail

In this section I'll describe the fundamental concepts of topic maps and their expression in the syntax defined by the XTM standard.

Topics

Topics are the building blocks of topic maps. Listing 1 shows a minimal topic map (not quite the smallest possible as the <topic> element is actually optional). (*Note:* All XML listings in this article are slightly abridged. The full source code for each can be downloaded at www.sys-con.com/xml/sourcec.cfm.) The root element of the document is the <topicMap> element. The document uses the XTM 1.0 namespace (http://www.topicmaps.org/xtm/1.0/) and also declares the XLink namespace, which will be used for all linking inside the topic map and to resources external to the topic map. A <topicMap> element may contain any number of <topic>, <association>, and <mergeMap> elements. In this example there's a single topic that has a value only for its (required) ID attribute.

Subject identity

As already discussed, a topic represents a subject that may be some otherwise unprocessible thing. The way in which a topic identifies precisely what subject it represents is by the use of subject identifiers. A subject identifier is some resource that either:

- *Is* the subject that the topic represents

or

- *Describes* the subject that the topic represents

A subject identifier that describes the subject the topic represents is also known as a subject indicator. In XTM such a subject identifier is required to be addressable with a URI. As an example of the difference between the two uses of a URI for identifying a subject, consider the URI http://tm4j.org/. If you visit this address using a standard Web browser, you'll find a Web page describing the TM4J Project, an open source project developing topic map processing tools and applications. So this page can be considered a resource that *describes* the subject "The TM4J Project". The URI is also the root URI for the TM4J Project's Web site, so this same URI could also be considered a resource that *is* the subject "The TM4J Project Web site". These are two distinct usages of the same URI and so have different forms of markup in XTM, as shown in Listing 2.

The URIs used to define the identity of the topic's subject are always contained within the <subjectIdentity> element. In Listing 2 the <topic> with the ID "tm4j-site" represents the TM4J Project's Web site. Because the URI actually points to the subject that the topic represents, the <resourceRef> element is used. However, the <topic> with the ID "tm4j-project" represents the TM4J Project itself, so the URI points to a resource that describes the subject and the <subjectIndicatorRef> element is used instead.

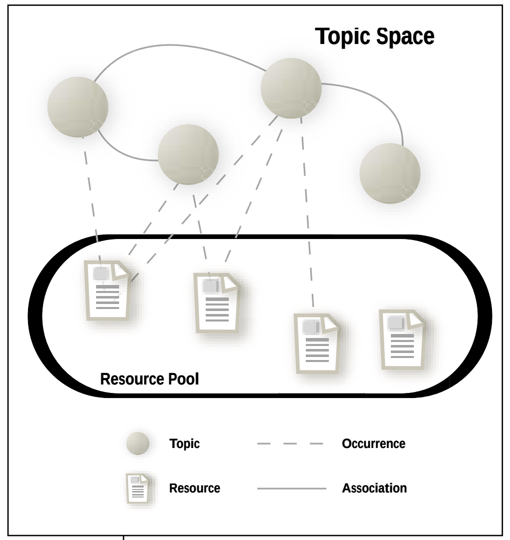


FIGURE 1 Basic topic map concepts

Topic names

The examples we've seen so far say nothing about the subjects they describe. In a topic map the information related to a subject can be provided by either names or occurrences. A topic may have any number of base names, each specified by a separate <baseName> element that contains a <baseNameString> element containing the text of the topic name. Each base name may have any number of variant names, which may be either text or pointers to other resources. This enables the use of icons or multimedia clips for the naming of topics. To distinguish variant names, each variant may declare any number of parameters, each of which is actually just a reference to a topic. The XTM standard provides topics that can be used as parameters to represent sortable variants of a base name. Listing 3 shows this usage – note that the reference to the topic defined by the XTM standard simply makes use of the XLink ability to address an element in another document entirely. This means that it's possible to reuse topics defined in other topic maps without needing to copy the topics into your own topic map.

Occurrences

Occurrences are typed links that point from the topic to related information resources. A topic may have any number of occurrences, each pointing to a single resource. An occurrence may be typed by referencing a topic that defines the type. This reference is made using a <topicRef> element contained within an <instanceOf> element for the <occurrence> being typed. As with the parameters, the topic we reference can even be in a different topic map document! Listing 4 shows the markup for an occurrence.

In addition to pointing to external resources, an occurrence can simply wrap inline data. The current version of the XTM specification limits such inline resources to PCDATA only. To specify an inline resource, use the <resourceData> element in place of the <resourceRef> element, with the occurrence data specified as the <resourceData> element content.

"A topic represents a subject that may be some otherwise unprocessible thing"

Topic types

Just as occurrences can be typed by referencing a topic that defines the type, so topics themselves can be typed by referencing other topics. As with occurrences, the reference to the topic that defines the type is made using a <topicRef> element contained within an <instanceOf> element. However, a topic may have any number of types, so the XTM DTD allows the <instanceOf> element to appear any number of times as a child of the <topic> element. This flexibility allows us to create multiple classification systems and still use the same mechanism (typing) to classify topics. Listing 5 shows a topic with a single type.

Associations

Associations are the mechanism by which a topic map author can express the relationship between topics. An association can be regarded as a grouping of topics that are in some way related. An association consists of one or more roles, each of which may be played by one or more topics. For example, the statement "Kal Ahmed is a member of the TM4J Project" can be expressed in a topic map as an association between the topic representing "Kal Ahmed" and a topic representing "the TM4J Project". In XTM an association is created using the <association> element. Each role in the association is defined using a separate <member> element that contains any number of <topicRef> elements that point to the topics that play that role in the association.

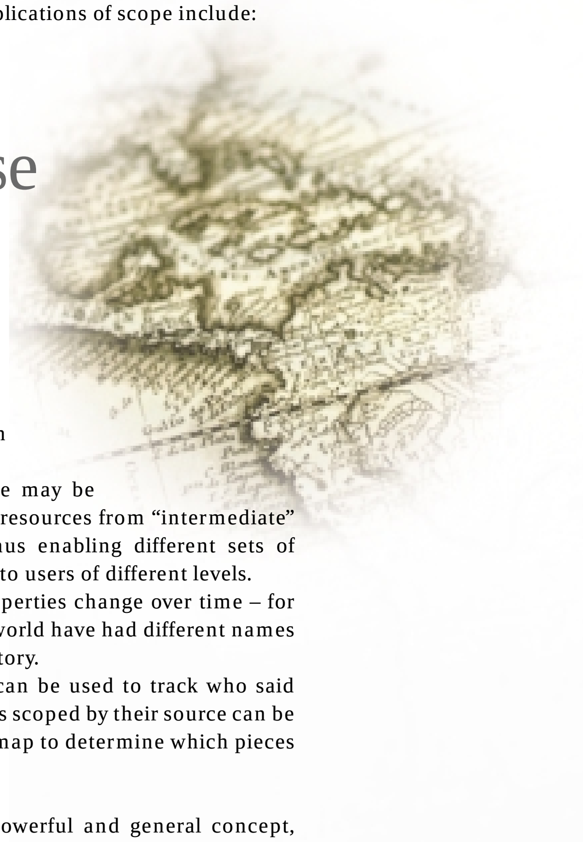
A very simple, uninformative association is shown in Listing 6. It's "uninformative" because it tells us only that there is some sort of relationship between the topics with the IDs "kal" and "tm4j-project". To express the nature of the relationship, we must add a type to the association. Once again, this typing is achieved by reference to a topic that defines the association type, using the <instanceOf> element. As with occurrences, an association can be typed by only one topic. Listing 7 shows this additional type information.

With Listing 7 we now know there is a relationship between the topic "kal" and the topic "tm4j-project" and that the relationship is "member-of". But we don't know which is the individual and which is the group. Is "kal" a member of "tm4j-project" or is it vice versa? In topic maps, associations are groupings without any direction. Instead, each topic plays a specific role in the association, and it's this role that defines the nature of a topic's participation in the association.

In XTM syntax each <member> element has an optional <roleSpec> element that contains a <topicRef> element pointing to the topic that defines the role. To disambiguate the association in Listing 7, we need to create topics for the concept of "individual" and "group" and reference these topics from the <member> elements of the association. This is shown in Listing 8.

Scope

The scope mechanism of topic maps enables any information provided about a topic to be qualified by defining a context within which the information is valid. Scope may be used to define several different perspectives on the same set of information. Some suitable applications of scope include:



- ***Language perspectives:*** You can use scope to differentiate English names and occurrences of a topic from French or German names and occurrences.
- ***Audience perspectives:*** Scope may be used to separate "beginner" resources from "intermediate" or "advanced" resources, thus enabling different sets of information to be presented to users of different levels.
- ***Time perspectives:*** Some properties change over time – for example, many cities in the world have had different names at different times in their history.
- ***Author perspectives:*** Scope can be used to track who said what; names and occurrences scoped by their source can be used by readers of the topic map to determine which pieces of information to trust.

Because scope is such a powerful and general concept, topic maps don't have a predefined set of scopes, but instead allow authors to construct them using topics as the building blocks. Scope may be applied to names (as a child of the <baseName> element), occurrences (as a child of the <occurrence> element), and associations (as a child of the <association> element).

Listing 9 shows the application of scope for differentiating between the English and German names for "The TM4J Project" and for hiding information for programmers (the API Javadoc) from other kinds of users. Note that simply specifying this scoping information is only one part of effective use of scope – the other lies in the application itself. The topic map paradigm has few rules about exactly how scope is applied and allows applications to define their own processing models for deciding how scope controls what a reader of the topic map does and doesn't see.

Merging

We've now covered the fundamental structures required to build a single standalone topic map. However, the "hidden" power of the topic map approach is that there's a simple, well-defined process by which two or more topic maps can be combined to enable a topic map reader to seamlessly view all the information provided by those topics maps simultaneously. The basic rule for topic map merging can be expressed as follows:

Whenever two or more topic maps are merged, the result is a single topic map in which all topics that the processor determines to represent the same subject are merged and duplicates are eliminated.

"The topic map paradigm is a powerful, subject-oriented approach to structuring sets of information"

But how does a processor determine that two topics are about the same thing? The topic map paradigm defines three ways:

1. If the two topics have the same resource as their subject, then they are about the same thing.
2. If two topics use the same resource to describe their subject, then they are about the same thing.
3. If two topics have the same string value for a base name, and if the two base names are in the same scope, then the topics are about the same thing.

(In the last of these rules, "the same scope" means a scope consisting of the same collection of topics.)

In addition to these basic rules, a processor may use any other means to determine whether two topics are about the same thing. For example, a processor with sufficient background knowledge could determine that a topic with the base name "United Kingdom" and a topic with the base name "Royaume Uni" refer to the same subject.

The process of merging two topic maps may be initiated by one of the following means:

- ***Merging of topic maps using <mergeMap>***: In this case a topic map processor must retrieve the topic map document pointed to by the <mergeMap> element and merge it with the topic map containing the <mergeMap> element. This enables modularization of topic maps by allowing one topic map to "include" another and build on the subjects it provides.
- ***Merging of topic maps using <topicRef>***: A topic map may contain a <topicRef> to a topic in another topic map document. In this case a topic map processor must retrieve the topic map document containing the referenced topic and merge it with the topic map containing the reference. This requirement ensures that whenever a topic is referenced from another topic map, the processing application always gets the full context of the referenced topic.
- ***Manual***: A topic map application may allow the user to choose to merge any set of topic maps to create a single unified view of them all.

During merging, whenever a processor has determined that two topics are about the same thing, the topics must be replaced by a single merged topic. The names and occurrences of the resulting topic are the union of the names and occurrences of the topics merged, and the resulting topic replaces the merged topics wherever those topics are referenced (as types, players in associations, members of a scope, or as association role specifications).

The merge process enables a set of topic maps to be created independently by both human beings and automated processes and then combined in a well-defined manner. By combining the merge process with the concept of well-known subject identities, an environment can be created in which human beings can share their knowledge of a domain and contribute their insights on subjects previously identified by automated processing. This approach is actually used by the TM4J.org Web site, where the site itself is constructed from a merge of topic maps created by human beings, a topic map generated from the XML sources of the user documentation, and a topic map generated from the Javadoc comments in the source code. The fully merged Web site enables a user browsing a class definition (from the Javadoc) to see on the same Web page a link to related FAQ entries or documentation.

Conclusions

The topic map paradigm is a powerful, subject-oriented approach to structuring sets of information. The building blocks of topic maps are topics that have names and point to occurrences of the subject in external resources. Associations relate topics to each other. Each topic in an association is the player of a defined role. Topics, occurrences, and associations can all be typed by other topics. In addition, topic names, occurrences, and the roles they play in associations can be specified to be valid only in a given context by the use of a scope, which is defined by a collection of topics.

Further Reading

- *ISO/IEC 13250, 2nd ed:* www.y12.doe. gov/sgml/sc34/document/0322_files/iso13250-2nd-ed-v2.pdf
- *XML Topic Maps 1.0:* www.topicma ps.org/xtm/1.0
- *topicmapmail list (Infoloom):* www. infoloom.com/mailman/listinfo/topicmapmail
- *Web resources for topic maps (Open Directory Project):* http://dmoz.org/Computers/Artificial_Intelligence/Knowledge_Representation/Topic_Maps/
- Park, J., and Hunting, S., eds. (2002). *XML Topic Maps.* Addison-Wesley
- Ancha, S., Cousins, J., et al. (2001). *Professional Java XML.* Wrox
- Dodds, D., Watt, A., et al. (2001). *Professional XML Meta Data.* Wrox

AUTHOR BIO

Kal Ahmed, an independent consultant specializing in XML technologies and information management, was a member of the TopicMaps.Org consortium during the creation of the XTM 1.0 specification. The main developer of the TM4J suite of open-source topic map tools as well as the creator and maintainer of TMTab, a freely available tool for creating topic maps, he has spoken and written widely on the subject of topic maps. More details about TM4J and TMTab can be found at www.techquila.com.

KAL@TECHQUILA.COM

LISTING 1 A minimal XTM topic map

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topicMap SYSTEM "xtm1.dtd">
<topicMap
    xmlns="http://www.topicmaps.org/xtm/1.0/"
    xmlns:xlink="http://www.w3.org/1999/xlink"
>
    <topic id="tm4j"/>
</topicMap>
```

LISTING 2 Identifying topic subjects in XT

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topicMap SYSTEM "xtm1.dtd">
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <topic id="tm4j-site">
        <subjectIdentity>
            <resourceRef xlink:href="http://tm4j.org/"/>
        </subjectIdentity>
    </topic>
    <topic id="tm4j-project">
        <subjectIdentity>
            <subjectIndicatorRef
xlink:href="http://tm4j.org/"/>
        </subjectIdentity>
    </topic>
</topicMap>
```

LISTING 3 Topic base names and variant names

```
<?xml version="1.0" encoding="UTF-8"?>
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <topic id="tm4j-project">
        <subjectIdentity>
            <subjectIndicatorRef
xlink:href="http://tm4j.org/"/>
        </subjectIdentity>
        <baseName>
            <baseNameString>The TM4J
Project</baseNameString>
            <variant>
                <parameters>
                    <topicRef xlink:href="http://www.top-
icmaps.org/xtm/1.0/core.xtm#sort"/>
                </parameters>
                <variantName>
                    <resourceData>tm4j project,
the</resourceData>
                </variantName>
            </variant>
        </baseName>
    </topic>
</topicMap>
```

LISTING 4 Topic occurrence

```
<?xml version="1.0" encoding="UTF-8"?>
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
          xmlns:xlink="http://www.w3.org/1999/xlink">
    <topic id="homepage"></topic>
    <topic id="tm4j-project">
        <subjectIdentity>
            <subjectIndicatorRef
xlink:href="http://tm4j.org/"/>
        </subjectIdentity>
        <baseName>
            <baseNameString>The TM4J
Project</baseNameString>
        </baseName>
        <occurrence>
            <instanceOf><topicRef xlink:href="#home-
page"/></instanceOf>
            <resourceRef xlink:href="http://tm4j.org/"/>
        </occurrence>
    </topic>
</topicMap>
```

LISTING 5 Topic types

```
<?xml version="1.0" encoding="UTF-8"?>
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
          xmlns:xlink="http://www.w3.org/1999/xlink">
    <topic id="person"/>
    <topic id="kal">
        <instanceOf>
            <topicRef xlink:href="#person"/>
        </instanceOf>
        <baseName>
            <baseNameString>Kal Ahmed</baseNameString>
        </baseName>
    </topic>
</topicMap>
```

LISTING 6 A simple association

```
<?xml version="1.0" encoding="UTF-8"?>
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <topic id="kal">...</topic>
    <topic id="tm4j-project">..</topic>
    <association>
        <member>
            <topicRef xlink:href="tm4j-project"/>
        </member>
        <member>
            <topicRef xlink:href="kal"/>
        </member>
    </association>
</topicMap>
```

LISTING 7 A typed association

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topicMap SYSTEM "file:/home/kal/projects/xmljour-
nal/xtm1.dtd">
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <topic id="member-of">...</topic>
    <topic id="kal">...</topic>
    <topic id="tm4j-project">..</topic>
    <association>
        <instanceOf>
            <topicRef xlink:href="#member-of"/>
        </instanceOf>
        <member>
            <topicRef xlink:href="tm4j-project"/>
        </member>
        <member>
            <topicRef xlink:href="kal"/>
        </member>
    </association>
</topicMap>
```

LISTING 8 A fully specified association

```
<?xml version="1.0" encoding="UTF-8"?>
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <topic id="member-of">...</topic>
    <topic id="individual">...</topic>
    <topic id="group">...</topic>
    <topic id="kal">...</topic>
    <topic id="tm4j-project">..</topic>
    <association>
        <instanceOf>
            <topicRef xlink:href="#member-of"/>
        </instanceOf>
        <member>
            <roleSpec><topicRef
xlink:href="#group"/></roleSpec>
            <topicRef xlink:href="tm4j-project"/>
        </member>
        <member>
            <roleSpec><topicRef xlink:href="#individ-
ual"/></roleSpec>
            <topicRef xlink:href="kal"/>
        </member>
    </association>
</topicMap>
```

LISTING 9 Scope

```
<!DOCTYPE topicMap SYSTEM "file:/home/kal/projects/xmljour-
nal/xtm1.dtd">
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <!-- Topics for languages: en = English, de=German -->
    <topic id="en"/>
    <topic id="de"/>
    <!-- Topics for user types -->
    <topic id="programmer"/>
    <!-- Topics for occurrence types -->
    <topic id="javadoc"/>
    <topic id="tm4j-project">
        <baseName>
            <scope>
                <topicRef xlink:href="#en"/>
            </scope>
            <baseNameString>The TM4J
Project</baseNameString>
        </baseName>
        <baseName>
            <scope>
                <topicRef xlink:href="#de"/>
            </scope>
            <baseNameString>das Projeckt
TM4J</baseNameString>
        </baseName>
        <occurrence>
            <instanceOf>
                <topicRef xlink:href="#javadoc"/>
            </instanceOf>
            <scope>
                <topicRef xlink:href="#programmer"/>
            </scope>
            <resourceRef
xlink:href="http://tm4j.org/docs/apiDocs/index.html"/>
        </occurrence>
    </topic>
</topicMap>
```

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

WRITTEN BY **HITESH SETH**

Introduction to CCXML

Call control for VoiceXML applications

When building interactive voice recognition applications, we are inevitably faced with the challenge of providing advanced telephony call-control capabilities. In some scenarios we'd like to bridge two calls for a conferencing application, in others we'd like to provide basic call routing so the caller can be connected to an appropriate customer service agent, make outbound calls, and the like. VoiceXML doesn't really provide these advanced call-control and telephony features.

What's available in VoiceXML is a simple "transfer" element that can be used to connect a VoiceXML application with another phone number/voice application. However, by definition it wasn't meant to provide the sophisticated call-control capabilities.

Introducing CCXML

Call Control eXtensible Markup Language (CCXML), as the name suggests, is an XML-based markup language that provides rich telephony call-control capabilities to an interactive speech application. Whereas the VoiceXML standard is focused primarily on providing semantics for representing conversational dialogs, CCXML provides the much-needed, sophisticated, event-based asynchronous call-control mechanism and tighter integration with the telephony platform.

CCXML and VoiceXML are totally complementary standards. Also, they aren't dependent on each other. For instance, CCXML isn't restricted to VoiceXML; it could be used in the context of other dialog languages (or even without one if the application requires only call routing capabilities). Similarly, VoiceXML could utilize other call-control semantics. The whole idea is to develop CCXML as a complementary standard so it can be leveraged in uniform fashion with dialog markup languages such as VoiceXML.

CCXML Application Model

The CCXML application model is based on an asynchronous event management system. Represented by the root XML node <ccxml>, much of the functionality provided by a CCXML application is around processing asynchronous telephony events. The bulk of CCXML application processing happens within event handlers (represented by the <eventhandler> element) that process incoming events through a series of event-processing triggers (represented by the <transition> element). Transition elements can then invoke a set of corresponding actions, including:

- Accept/reject calls (<accept>/<reject> elements).
- Initiate/terminate a dialog interpreter instance (<dialogstart>, <dialogterminate>).
- Invoke processing logic (<if>, <else>, <elseif>, <assign>, <var>, <exit>).
- Transfer executions to different CCXML documents (<fetch>, <goto>, <submit>).
- Create/destroy conferences and connect/disconnect phone connections (also known as call-legs)(<createconference>, <destroyconference>, <join>, <unjoin>).
- Make an outbound call (<createcall>).
- Generate events (<send>).
- Terminate a call (<disconnect>).

In a nutshell, developing a CCXML application is really writing the handlers, which are executed when certain events arrive. Using the mechanism provided, information is passed back and forth between VoiceXML and CCXML documents. Similar to VoiceXML, a CCXML application can comprise more than one document.

CCXML Events

As is probably clear, asynchronous event handling is a key highlight of CCXML. It's also important to understand that CCXML event handling is quite different from VoiceXML's basic nomatch/filled synchronous style events. Technically speaking, each running CCXML interpreter has a queue, into which it places incoming events and sorts them by arrival time. Events are then available to a CCXML program using an eventhandler.

It's possible for a CCXML application to generate and process any arbitrarily named events. However, most of the CCXML application development is focused around a set of standardized events. Following is a list of commonly used events (the CCXML specification provides an exhaustive list of predefined events):

- ***Call related***
  —call.CALL_INVALID (final state for all calls)
- ***Connection related***
  —connection.CONNECTION_ALERTING (represents an incoming call notification)
  —connection.CONNECTION_CONNECTED (represents a call that's been connected)
  —connection.CONNECTION_FAILED (connection to the other end of call has failed)
- ***Dialog related***
  —dialog.exit (a VoiceXML Interpreter has ended execution)
  —dialog.disconnect (a VoiceXML Interpreter encountered a disconnect tag)
  —dialog.transfer (a VoiceXML Interpreter encountered a transfer tag)

Hello CCXML

Enough talk. Let's get our hands dirty and actually take a look at a CCXML-based application. The code in Listing 1 shows a basic CCXML application that provides the capability to screen calls from selected phone numbers.

What our simple "Hello World" style CCXML application does is provide a basic event handler for the event "CONNECTION_ALERTING", which notifies the application of an incoming call. Once the event is received, we process the caller ID of the user and, based on that, make a decision to accept/reject the call.

Combining VoiceXML, CCXML
A personal secretary

Let's say you've developed an address book application (similar to the one we developed in a previous article [*XML-J*, Vol. 2, issue 2]) that stores your frequent contact information. A VoiceXML dialog recognizes a person in your address book (by name, company, and/or nickname) and returns the phone number of the person (see Listing 2).

It's then possible, using CCXML, to create outbound calls (see Listing 3).

The basic CCXML script in Listing 3 can be modified to handle error-handling scenarios and timeouts, provide call-bridging capabilities, and so on.

Application Scenarios

CCXML is easy enough to develop simple applications like call screening and personal dialer, yet complex enough to develop a multiparty conferencing application. Application scenarios that can use the capabilities provided by CCXML include call screening (filter calls to an application), find-me/follow-me applications, personalized call centers, intelligent routing, call-hold, supervised transfer, multiparty conferencing, calling card applications, and the like.

Let's take an automated self-service interactive voice-based application, for example. Typically, it's desirable to provide the user an easy mechanism to exit out of the recognition mode and be transferred to a human customer service agent. However, we'd also like to inform the customer service agent about any interactions that have already occurred in an automated fashion (for instance, a user's basic profile – telephone number, maybe a credit card number or an account number). Even though it's quite possible to achieve the basic "transfer" functionality using the <transfer> element in VoiceXML, CCXML provides the needed sophistication to achieve an automated "whisper-enabled transfer," in which a CCXML application can actually pass on any information about the caller to the agent before bridging the call.

Implementations

Voxeo Community, a hosted VoiceXML platform, is one of the first implementations of CCXML that I've come across. A number of other vendors have announced support for the upcoming standard as well. Included in the Community application is Voxeo Application Insight, a tool that provides management/analysis/debugging of remote CCXML applications.

Conclusion

CCXML is in its early stages. As a first working draft in the W3C standardization process, it's been pointed out that CCXML may change significantly before it reaches the official "W3C Recommendation" status. So as developers of interactive telephony applications, we need to be aware of that. Actually, a number of vendors today provide implementations to certain features highlighted in CCXML through specific vendor extensions. Apart from standardization, another key development required in the adoption process of CCXML would be for vendors to provide a set of tools that support visual CCXML design, development, and testing. The whole notion of a state machine–based CCXML event model lends itself easily to be generated through a visual call-control application tool.

References
- *CCXML version 1.0 (W3C Working Draft):* www.w3.org/TR/ccxml/
- *Voxeo Community site:* http://community.voxeo.com
- *Voxeo Application Insight:* http://tech-preview.voxeo.com/insight/index.html
- *JTAPI:* http://java.sun.com/products/jtapi/

HKS@HITESHSETH.COM

AUTHOR BIO
Hitesh Seth is chief technology officer of ikigo, Inc., a provider of XML-based Web services monitoring and management software. A freelance writer and well-known speaker, he regularly writes for technology publications on VoiceXML, Web services, J2EE and Microsoft .NET, wireless computing, and enterprise/B2B integration. He is the conference chair for VoiceXML Planet Conference & Expo.

VoiceXML
Standardization Update

VoiceXML inches closer to standardization

The W3C Voice Browser Activity (www.w3.org/voice) recently (June 26, 2002) released Speech Recognition Grammar Specification as a Candidate Recommendation Specification. Candidate Recommendation status indicates that the specification is believed to be stable.

VoiceXML 2.0 itself was released as a renewed Last Call working draft (April 24, 2002). Last call status indicates that the working group believes that this specification is ready and therefore wishes this to be the last call for comments.

Speech Synthesis Markup Language (SSML) also moved ahead, and a new version of the working draft is available.

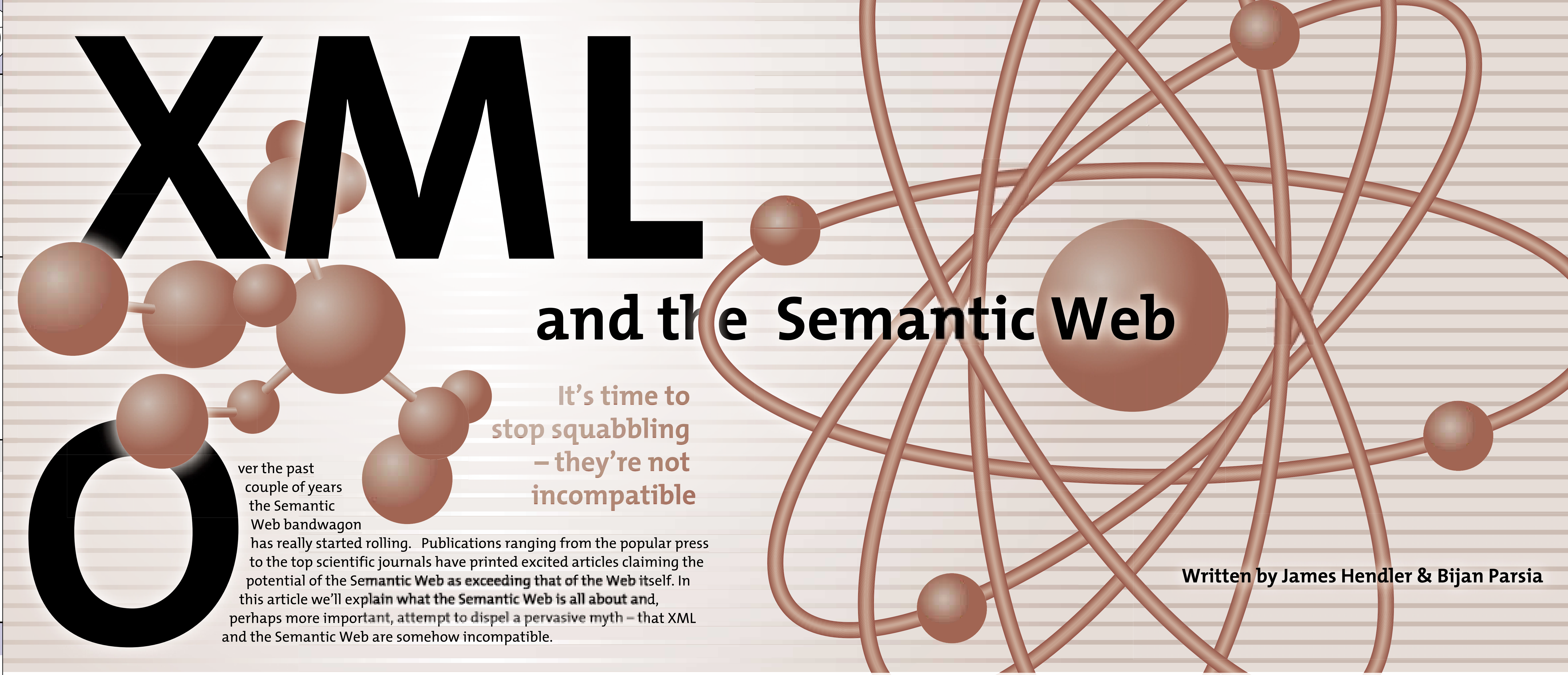
LISTING 1
```
<?xml version="1.0"?>
<ccxml version="1.0">
 <eventhandler>
<transition event="connection.CONNECTION_ALERTING"
      name="evt">
  <if cond="evt.callerid=='AAANNNNNNN'">
  <reject/>
  <else/>
  <accept/>
  </if>
     </transition>
 </eventhandler>
</ccxml>
```

LISTING 2
```
<?xml version="1.0"?>
<vxml version="2.0">
  <form id="MainForm">
    <field name="phone_no">
      <prompt>
  Who would you like to call today?
      </prompt>
      <grammar type="application/srgs+xml"
               src="AddressBook.jsp"/>
      <filled>
        <exit namelist="phone_no"/>
      </filled>
    </field>
  </form>
</vxml>
```

LISTING 3
```
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0">
  <eventhandler>
...
    <transition event=
     "connection.CONNECTION_CONNECTED"
       name="evt">
      <dialogstart src="'AddressBook.jsp'"/>
    </transition>
    <transition event="dialog.exit" name="evt">
      <createcall dest="evt.input"/>
    </transition>
 ...
  </eventhandler>
</ccxml>
```

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

XML and the Semantic Web



It's time to stop squabbling – they're not incompatible

Over the past couple of years the Semantic Web bandwagon has really started rolling. Publications ranging from the popular press to the top scientific journals have printed excited articles claiming the potential of the Semantic Web as exceeding that of the Web itself. In this article we'll explain what the Semantic Web is all about and, perhaps more important, attempt to dispel a pervasive myth – that XML and the Semantic Web are somehow incompatible.

Written by James Hendler & Bijan Parsia

XML-based formats are now becoming the dominant way of marking up data on the Web. There are standard XML languages for hypertext (XHTML), graphics (SVG), syndication (RSS), multimedia (SMIL), service description and discovery (WSDL and UDDI), and many others, not to mention a plethora of more specialized, often ad hoc, XML-based languages.

Furthermore, there are standard ways to query (XPath and XQuery), link into (XLink and XPointer), and parse and program with (SAX, and DOM) your XML documents. Finally, integration with the many other XML formats can be done in a reasonably straightforward manner either by inclusion (using XML Namespaces) or transformation (using XSLT).

XML as an Interchange Mechanism

Given these facts, it's hard to argue against XML as the interchange mechanism for the Web. But the XML model remains primarily rooted in documents – in particular, textual documents with hierarchical structure. For example, a well-formed XML document can have a <US-address> element that contains <US-street-address-line> and <US-zip-code> elements. From the pure XML point of view, there's nothing but a document-based description of the names and current arrangements of these elements: that is, that a <US-address> contains a <US-zip-code>, which is a string. We could, of course, also have a document with two <US-zip-code>s in it. An XML Schema for <US-address>s could go further and state the specific constraints on that structure, for example, that a <US-address> can contain many <US-street-address-line> elements but only one <US-zip-code> element. With this schema a validator can conclude that our <US-street-address> with two <US-zip-code>s was syntactically incorrect.

However, the schema doesn't actually capture much of the semantics of an address. For example, if we were to take several different documents and find two addresses that were identical except for their zip codes, we wouldn't be able to realize, on the basis of the schema, that at least one zip code was wrong. Similarly, there's no automatic way to convert between two very similar encodings of address (for example, to one where there could only be one <US-street-address> element, which itself had many <US-street-address-line>s). Further, if we wanted to do something complex, such as integrating <US-street-address>s with our personnel database, we would have to rely on a human programmer. We'd need someone to provide a specific mapping from the <US-address> to the various fields in our database structure, and this mapping would be vulnerable to even small changes in our XML format or in the database – and just forget about reusing that mapping with unrelated systems.

More strikingly, there's no clear way to connect our various <US-address>s to more general facts about addresses – for example, that they identify locations, that mail can be delivered to them, or that they're associated with people or businesses. And neither the XML model nor XML Schema are going to be much help in determining that a mailing address is like a fax number, at least in the sense that mailing a letter to an address and faxing it to a number both get the text of the missive to its recipient.

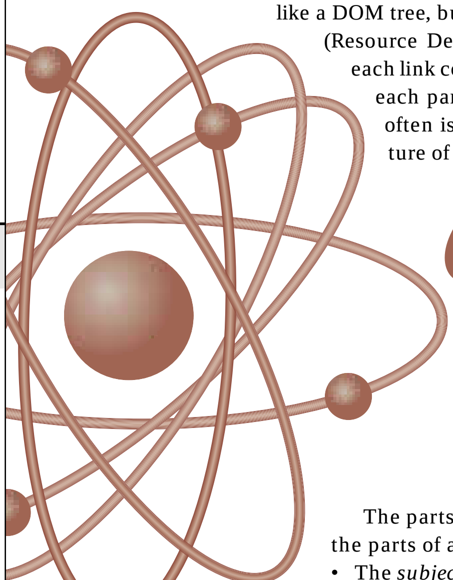
On the Web, however, we really need this semantic information. We want to be able to *connect* an address that we've encoded in XML in some Web document to people, places, concepts, letters, other documents, databases, directories, PDA contact lists, calendars, sensors, services, and all the other resources on the Web. Done right, such links would allow information in various forms and formats to be automatically manipulated in a coherent way by our computer programs. XML is clearly an important part of the answer, but it's very hard to see the XML data model as a feasible way to represent these links and the real-world semantics they encode.

Modeling Links

By examining how links work on the current Web, we can get some idea of what it takes to model them. Web links have two key aspects: first, things *on* the Web are consistently identified by URIs (Uniform Resource Identifiers). Second, a Web link has the following tripartite structure:

- The thing you start with (the source of the link). (In XHTML this is the file containing the "a" tag with an "href" attribute.)
- The connecting, or *linking*, bit between the source and the target of the link.
- The thing you end up with (the target). (In XHTML it's typically named named by a URI placed in the href attribute.)

While, naturally, it's possible to encode Web links in XML (e.g., with <a href="…"/>), note how little in common a Web link has with the XML data model: Web links use URIs instead of tags or QNames; unlike XML, Web links impose no inherent hierarchy, no notion of containment, and no sequencing of the things to which they relate. In fact, a set of Web links doesn't look much like a DOM tree, but does look an awful lot like an RDF (Resource Description Framework) graph, where each link corresponds to an RDF triple. After all, each part of an RDF triple can be, and most often is, identified by a URI, and the structure of a triple is obviously tripartite.



Most notably, RDFS defines class and property relations and provides a mechanism for restricting the domains and ranges of the properties. In RDFS, for example, we can express site-specific Yahoo-like Web site categories as a hierarchy of classes with sets of named (sometimes inherited) properties. This allows other sites to connect their own information to these terms, providing better interoperability for use in B2C, B2B, or other Web transactions.

OWL extends RDFS into a more capable language, usable for thesauruses and domain models. OWL is based on DAML+OIL, a Web language jointly developed by the U.S. Defense Advanced Research Projects Agency (DARPA) and the

"RDFS is a simple language for creating Web-based 'controlled vocabularies' and taxonomies"

The parts of a triple, by design, correspond to the parts of a Web link:

- The *subject* of a triple is where you start.
- The *predicate* connects the subject and the object.
- The *object* corresponds to the target of a Web link.

Indeed, an RDF triple *is* a representation of a Web link, where each part of the link is made explicit. Thus a collection of RDF triples is a way to represent, share, and process chunks of the Web itself. Having the Web in a standardized representation allows us to enrich the semantics not just of information in documents *on* the Web, but of the information expressed *by* the Web. Every Web link is an often vague, usually ambiguous, and almost always underspecified *assertion* about the things it connects. RDF lets us eliminate that vagueness and nail down the ambiguity. RDF Schema (RDFS) and the new Web Ontology Language (OWL) allow us to model the meaning of our assertion links in precise, machine-processible ways.

RDFS is a simple language for creating Web-based "controlled vocabularies" and taxonomies. The language defines several RDF *predicates* for making links between concepts.

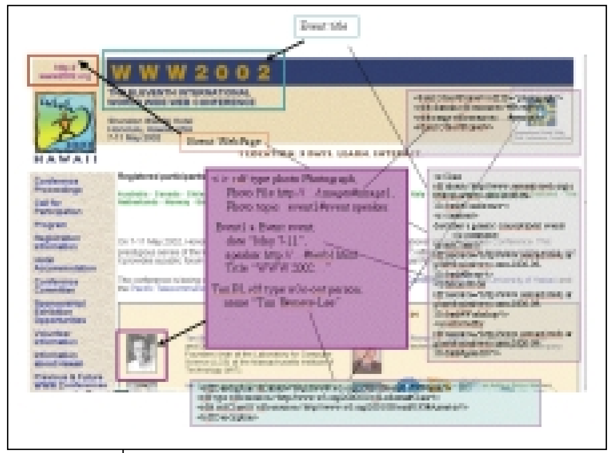


FIGURE 1 | Linking a Web site to ontological information

European Union's Information Science and Technology (IST) program. DAML+OIL has begun to get heavy use in the government, and in November 2001 the W3C chartered a Web Ontology Working Group to refine the DAML+OIL standard into a W3C recommendation language, now called OWL.

OWL extends RDFS with many more constructs for defining the relationships between classes and, more important, placing restrictions on how properties (i.e., predicates) can be used when linking entities. OWL thus allows users to define simple models of their domains using these properties and their constraints. A full discussion of the language is beyond this article (see www.w3.org/2001/sw/WebOnt for more details), but revisiting the "Address" example should make a few things clearer.

XML and OWL

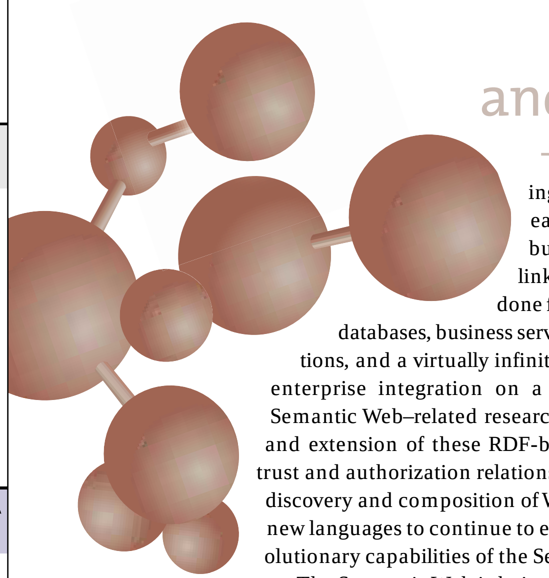
In XML we were able to say that there was a document field called a <US-address> that had subfields of a street address, a city name, a state name, and a zip code. In OWL we can explicitly name these objects as classes and properties, and place constraints on how to legally relate these entities to each other or to entities defined in other documents. Thus, for example, we could mention that cities are in states, and that each city is in one, and only one, state. We could know that a U.S. address is a type of international address where the state field is restricted to be one of Alabama, Maine, New York, and so on, and that these addresses have zip codes that consist of either five or nine numbers. We could also add the information that, in general, international addresses have country codes, and that the country code for U.S. addresses always has the value "USA," and many other such facts. (This sort of specification of these relationships in a formal language is called *ontology*, thus the term *ontology language* for OWL.)

Ontologies let us more precisely link to other documents and resources based on shared use of conceptual terms, even where there is only a partial match (a key difference from current XML-based approaches). Our addresses could thus be linked in turn to other vocabularies – for example, knowing that an address names a location, we could link to other location-based Web resources. These could be databases or Web services that would compute the location of the nearest airport (another kind of location) to a given address, the weather forecast for the city the address is located in, or other such location-specific data. Metadata can also be used to link nontext media to ontologies, expressing, for example, that the photo in a particular picture is of a house at a particular address or that the place to complain about the contents of a particular streaming video is in a particular state (allowing you to compare its location to yours and see if local content restrictions might apply).

Figure 1 provides an example of the linking of a Web site to ontological information. In this case, from a presentation on OWL given at the W3C session of the World Wide Web Conference in May 2002, information about the keynote speaker is linked to information about events, photos, and people.

The example of addresses is an extremely basic one, yet we already see a tremendous number of possible uses. By mapping the implicit semantics inherent in XML DTDs and schemas into the explicit relationships expressible in RDFS and OWL, a whole range of new applications, largely created by the linking of existing Web resources, become easily implementable. In the business world this kind of linking to models could be done for SEC filings, supply-chain databases, business services posting WSDL descriptions, and a virtually infinite range of others, allowing enterprise integration on a Web-wide scale. Current Semantic Web–related research is also exploring the use and extension of these RDF-based languages to express trust and authorization relationships, to do the automated discovery and composition of Web services, and to design new languages to continue to enhance the potentially revolutionary capabilities of the Semantic Web.

“In short, the **Semantic Web** offers **powerful** new possibilities and a **revolution in function**”



The Semantic Web is being built on models based on the RDF representation of Web links. To achieve their full impact, however, the enhanced models enabled by the Semantic Web crucially need to be tied to the document-processing and data-exchange capabilities enabled by the spread of XML technologies. If XML- and RDF-based technologies were incompatible, as some people seem to think they are, it would be a true shame. But, in fact, they aren't. While the underlying models are somewhat different, the normative document exchange format for RDF, RDFS, and OWL is XML. Thus, to those preferring to think of the whole world as XML based, RDF, RDFS, and OWL may simply be thought of as yet another XML language to be managed and manipulated using the standard toolkit. To the RDF purist, the documents and datasets being expressed in XML and XML Schema can anchor their models with interoperable data. To those focused on the world of Web services, SOAP and WSDL can carry, in their XML content, RDF models expressing information that can be easily found, linked, and discovered.

Of course, as is the case with any groups doing overlapping tasks, there is friction between some in the RDF community and some in the XML world. RDF folks often complain about the (for them) superfluous intricacies of XML. XML experts shake their heads at the way the RDF/XML serialization abuses QNames and XML Namespaces and treats certain attributes and child elements as equivalent. However, these kinds of complaints are nothing new. In fact, they're common in the XML community itself: witness the fury that some XML people express over XSLT's use of QNames as attribute content (to pick one example). Similarly, the RDF world has plenty of dark and overcomplicated corners. Both sets of languages are also continuing to evolve, and each is also exploring new non-XML syntaxes (consider Relax-NG, XQuery, and XPath).

Best of the Best

In short, the Semantic Web offers powerful new possibilities and a revolution in function. These capabilities will arrive sooner if we stop squabbling and realize that the rift between XML- and the RDF-based languages is now down to the minor sorts of technical differences easily ironed out in the standards process or kludged by designing interoperable tools. Combining the best of all these languages, and their variants, is easily enabled by the combination of the "documents" of the XML Web with the "links" expressed in RDF. Throw interoperable Web services into the mix and the vision is compelling. The future of the Web can be even more exciting than its past, and pulling all these threads together will get us there.

Useful Links

- *W3C Semantic Web activity:* www.w3.org/2001/sw
- *The RDF Schema Language Specification – Working Draft:* www.w3.org/TR/rdf-schema
- *"Integrating Applications on the Semantic Web" (paper by J. Hendler, T. Berners-Lee, and E. Miller on using the Semantic Web for business applications):* www.w3.org/2002/07/swint
- *W3C Web Ontology Working Group home page:* www.w3.org/2001/sw/WebOnt
- *Feature Synopsis for OWL Lite and OWL – Working Draft:* www.w3.org/TR/owl-features
- *"Why RDF model is different from the XML model," by T. Berners-Lee:* www.w3.org/DesignIssues/RDF-XML.html

AUTHOR BIOS

Jim Hendler, a University of Maryland professor, is the director of Semantic Web and agent technology at the Maryland Information and Network Dynamics Laboratory. A Fellow of the American Association for Artificial Intelligence, Jim was formerly chief scientist for information systems at the U.S. Defense Advanced Research Projects Agency (DARPA) and cochairs the Web Ontology Working Group for the W3C.

Bijan Parsia is a Semantic Web researcher at the Maryland Information and Network Dynamics Laboratory. His research interests include Web logics and rule engines, Semantic Web services, fine-grained, reflective annotation systems, and trust-focused reasoning.

HENDLER@CS.UMD.EDUNET

BPARSIA@ISR.UMD.EDU

WRITTEN BY **RAJESH ZADE**

Using FOP for Industrial Needs

How to ensure accuracy at the client tier

The Web has given us an easy-to-use, easy-to-market, and easy-to-navigate medium for conducting business. We've come to a stage where mainstream people are comfortable shopping on the Web. What it lacks is the ability to ensure that what the end user has browsed or accepted is the same as what was processed on the back end. In other words, how do you make sure the confirmation of a transaction is the same as the transaction itself?

This article discusses how XML and Formatting Objects Processor (FOP) technology can be used to address this problem.

The FOP initiative is a part of the Apache organization and is a subproject of the XML project (http://xml.apache.org/fop/index.html). It is a Java application that enables rendering of XML input data to various output formats such as PDF, PCL, SVG, XML, Print, AWT, TXT, and RTF (not yet supported). Among these formats, PDF is the most popular option. The latest version of FOP supports the XSL-FO version 1.0 W3C recommendation (www.w3.org/TR/2001/REC-xsl-20011015/).

FOP is a Java rendering engine invoked at runtime that applies an XSL-FO stylesheet to XML data and produces an output format based on properties that the runtime engine was initialized with. For example, you can fetch data from the database, represent it in XML format, and then apply an XSL-FO stylesheet to generate a PDF report. This process executes within the server tier. The resulting PDF stream can be presented to the end user in the same way that a PDF document can be set up for download from the Web.

AUTHOR BIO
Rajesh Zade has more than 13 years of experience in the computing field and has been working in various Java and e-commerce technologies since 1996. He is currently chief technical architect for NetCliff, Inc., based in Santa Clara, CA.

Why XML and FOP?

Several rendering mechanisms are available today. HTML, TEXT, Microsoft Word, and other reporting mechanisms such as Crystal Reports are widely used. Most of them can also be used on the Web. Web pages are built mainly using HTML, the most common of the mechanisms. HTML is a primary language for building Web navigations with Web forms to capture user input, and also for displaying results to the end user.

It makes perfect sense to follow this approach until you capture data and navigate from page to page, but not after the input data has been processed and you're ready to show the results. The main flaw with rendering end results in HTML format is that the result page is easily editable. For example, you can get a final receipt from Amazon.com for a purchase in HTML, save it on a local machine, and then edit it to look like a completely different purchase. Or you can get an insurance quote for a car, change its premium by editing the HTML, print it, and fax it back to the company to claim lower premiums. You probably don't stand a chance of actually grabbing that phony deal, but just imagine how much time a sales representative would have to spend to disprove it!

The solution to this problem is that whenever you want to present to a customer what he or she is going to pay – a final receipt or a quote – you probably want to present it in a noneditable form close to the printed receipt you get at the checkout counter, something like a receipt presented as a PDF document. Therefore, it would be more accurate to show a final receipt within a browser as a PDF page instead of an HTML page. You can advise your customer to print that receipt as a proof of purchase.

Another advantage of using XML and FOP: if FOP solves your problem at the client tier, then XML solves it at the server tier. The XSL-FO stylesheet is applied to XML data, and XML data at that instance depends on many parameters a customer must have chosen to finally select that product. For example, a customer may get some discounts that may be applicable for that purchase. Another example: when highly configurable products, such as PCs, are sold on the Web, they vary from customer to customer. Granted, you have information about all the parts in your database, but you don't have information about a particular configuration. That configuration is already represented in the XML data. When the customer actually clicks that final buy button and the response is sent back to the server, your systems can easily work from XML data for that purchase instead of going all the way back to the database.

The main idea here is to make the final proof of purchase noneditable and also keep it in sync with the back-end processes. It makes the hard copy and the electronic copy virtually the same.

Industry	Use
Insurance	To present online insurance quotes, policy details, and claims.
Custom Products	Most highly configurable products need to be assembled on the fly. Information about such a purchase can be stored in XML on the server side while it is presented to the buyer as a PDF document on the client side.
Retail	Most of the time shoppers buy more than one item online. Such a basket of purchase (or the contents of a shopping cart) can be held as one XML file on the server and upon completion of the transaction it can be processed. Also, the final receipt can be presented to the user as a PDF document that looks exactly the same as checkout counter receipts.
Government	Many government documents can be transmitted over the net to the user as a PDF document. Most such documents are generally mailed to the user.
Utilities	Invoices from utility companies can be transmitted to a customer as PDF documents that look similar to printed invoices.
Travel	Many travel documents such as tickets and boarding passes can be transmitted electronically to a traveler.
Banking & Investment	Monthly bank statements and brokerage transaction confirmation notices can be sent to the user as a PDF documents.

TABLE 1 | XML-FOP applications in various industries

Putting it to real use

Before we jump to building a demo application, take a look at Table 1, which describes applications of XML-FOP to various industries.

Case Study: Insurance Quote Processing

Various applications of XML and FOP have already been discussed. Now let's explore how we can apply XML and FOP to a sample auto insurance quote application. I won't delve into how a quote is actually generated from the object tier. That would fall under the J2EE and JDBC areas. We'll assume here that we have all the data needed in the form of XML to process the quote, and will discuss how we can generate a PDF report from the XML data and process the data to complete the transaction.

Figure 1 shows the QuoteSessionBean that's responsible for collecting user-specific data to generate the quote in XML form. The QuoteSessionBean can interact with various business objects or get data straight from the database to generate the XML file (code not provided). Once the XML file is ready, something like a Reporter object that wraps the FOP calls can apply the XSL-FO stylesheet to the XML data and stream an auto policy quote as a PDF report to the user. Once the user accepts the quote and clicks the submit button, the application-tier object, such as QuoteProcessor, can work from the Quote XML file to finalize the quote.

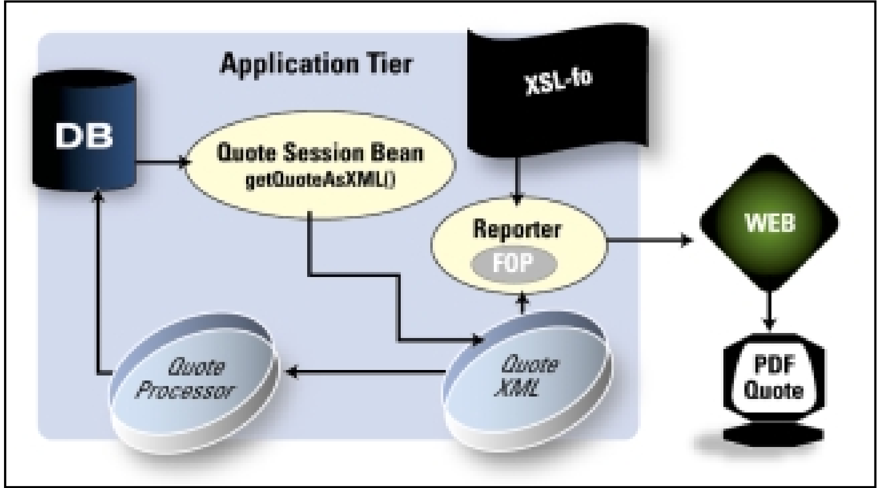


FIGURE 1 | Generating XML and applying XSL-FO

Quote in XML form

The business logic tier generates the QuoteXML. The data needed to generate the report is embedded in the <quotedata> tag in Listing 1. Our report prints a simple auto policy quote that has customer, coverage, and summary data. The customer data has address and contact data within itself. The coverage data has the type of auto defined as an attribute and policy-related information as options. Many option tags appear in the options data definition. Each defines a type of coverage added in the policy. The summary data is just the total amount for the policy. As you can see, the XML data definition is quite simple and can be easily generated using something like Xerces libraries.

XSL-FO stylesheet

At first look the XSL-FO stylesheet looks a little complicated, but with a little experience it becomes as easy as coding HTML manually. Coding HTML or XSLT manually can sometimes get complicated, of course, but tools such as XML Spy help you autofill <fo> tags when you start coding the stylesheet. Also, coding the <fo:root> section becomes a repetitive task that can be copied from one stylesheet to another. Even better, you can have a set of formatting stylesheet libraries that define various styles and flows of PDF documents. Once this section is coded, all you do is match templates (XML node names) from the XML to the XSLT file and apply whatever formats you want in the rest of the section.

> “At first look the XSL-FO stylesheet looks a little complicated, but with a little experience it becomes as easy as coding HTML manually”

In Listing 2 the FOP root section has the simple definition and flow of a PDF document with a title and three sections, one each for Customer, Coverage, and Summary. Following this definition there are matching templates for Customer, Address, Contact, Coverage, Options, and Summary tags.

Note: The code in Listing 2 illustrates only FOP-specific items. See the "Resources" section at the end of this article for directions on obtaining the full stylesheet.

Generating dynamic PDF reports

You can also make the PDF reports dynamic by controlling what section of the report you want to hide at runtime. You can pass a flag to the stylesheet that's defined in another XML file. For example, you can code a common stylesheet for your "preferred" and "new" cus-

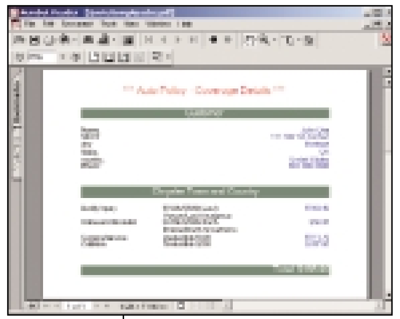


FIGURE 2 | PDF output

tomers and generate different reports at runtime based on the value of the flag.

Say you have a flag called "showdiscount" defined in discount.xml as:

```
<discounts>
 <policy showdiscount="true"/>
</ discounts >
```

and a variable displayControl defined in stylesheet as:

```
<xsl:variable name="displayControl"
select="document(discount.xml')"/>
```

You can have the following statement around any <fo> block to control the display of that block only to your preferred customers:

```
<xsl:if
test="$displayControl@showdiscount =
'true'">
 <fo:block>
  Preferred Customer Discount Details
 </fo:block>
</xsl:if>
```

PDF Output

To get the PDF output, you must have the Apache FOP (http://xml.apache.org/fop/download.html) installed on your computer. Use the following command once you have FOP installed:

```
<<install_dir>>\fop-0.20.3>fop -xml
<<xmlj_files_dir>>\QuoteXML.xml -xsl
<<xmlj_files_dir>>\QuoteXslFopXslt.xs
lt -pdf
<<xmlj_files_dir>>\QuoteUsingApache.pdf
```

You can also invoke the FOP rendering engine from within any Java application or any application server (see Figure 2).

Back-end processing components

If you decide to keep the Quote XML data in memory until the response is obtained or the session is invalidated, you can process the quote from the Quote XML rather than querying for the same data again. This would save several DB calls. Moreover, as described earlier, the presentation layer and the processing layer would be in sync.

Conclusion

The use of XML and FOP together offers two main advantages for industrial applications:

- It allows final confirmation of transactions to be presented to the end user in a noneditable form as a PDF document.
- It keeps the client-tier presentation and the server-tier business processes in sync.

This results in significant cost savings for businesses and reduces processing overheads within the application.

Resources

- *Code for this article:* www.sys-con.com/xml/sourcec.cfm
- *Apache FOP:* http://xml.apache.org/fop/
- *XML SPY:* www.xmlspy.com
- *Adobe Acrobat Reader:* www.adobe.com/products/acrobat/
- *GNU Ghostscript:* www.cs.wisc.edu/~ghost/

RAJESH.ZADE@NETCLIFF.COM

LISTING 1

```
<quotedata>
 <!-- Customer Information -->
 <customer name="John Doe">
  <address street="111 SW 1st Avenue" city="Portland"
state="OR" country="United States"/>
  <contact phone="503-999-9999"/>
 </customer>
 <!-- Coverage information -->
 <coverage auto="Chrysler Town and Country">
  <options>
   <option cvtype="Bodily Injury" cvdescription="$100k/$300k
Each Person/Each Incidence" cvprice="143.40"/>
   <option cvtype="Uninsured Motorist"
cvdescription="$100k/$300k Each Person/Each Occurrence"
cvprice="36.00"/>
   <option cvtype="Comprehensive" cvdescription="Deductible
$500" cvprice="111.70"/>
   <option cvtype="Collision" cvdescription="Deductible $500"
cvprice="197.80"/>
  </options>
 </coverage>
 <!-- summary of coverage -->
 <summary total="488.90"/>
</quotedata>
```

LISTING 2

```
FO namespace and layout settings:
   <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <!-- defines the layout master -->
   <fo:layout-master-set>
    <fo:simple-page-master master-name="first" page-
height="29.7cm" page-width="21cm" …
 <fo:region-body margin-top="1.5cm" margin-bottom="1.5cm"/>
     <fo:region-before extent="1.5cm"/>
     <fo:region-after extent="1.5cm"/>
    </fo:simple-page-master>
   </fo:layout-master-set>

Define and format the title, footer and body sections of the
PDF document (see Figure 2 for output):
   <fo:page-sequence master-reference="first">
    <!-- header -->
    <fo:static-content flow-name="xsl-region-before">
            <fo:block font-family="Helvetica" color="red"
font-size="18pt" text-align="center">
                  *** Auto Policy - Coverage Details ***
               </fo:block>
      </fo:static-content>
      <!-- footer -->
           <fo:static-content flow-name="xsl-region-after">
                                  <fo:block font-
family="Helvetica" font-size="10pt"  text-align="center">
                    Page <fo:page-number />
               </fo:block>
         </fo:static-content>
         <!-- body -->
      <fo:flow flow-name="xsl-region-body">
       <!-- format the pieces (put everything from following
templates here) -->
       <xsl:apply-templates/>
       <!-- end of fo document -->
      </fo:flow>
     </fo:page-sequence>
     </fo:root>

Format the "Customer" section of the document. See Figure 2
for the resulting output (other templates suppressed for
readability):
 <xsl:template match="customer">
  <xsl:if test="@name | contact | address">
Verify customer data exists before trying to format it
   <!-- section heading -->
   <fo:block font-size="16pt" font-family="sans-serif" space-
after.optimum="15pt" text-align="center" …
Customer
         </fo:block>
   <!-- table start -->
   <fo:table space-after.optimum="30pt">
    <fo:table-column column-width="80mm"/>
    <fo:table-column column-width="80mm"/>
    <fo:table-body>
     <xsl:if test="@name">
      <fo:table-row>
       <fo:table-cell>
         <fo:block>Name:</fo:block>
       </fo:table-cell>
       <fo:table-cell>
         <fo:block text-align="right" color="blue">
           <xsl:value-of select="@name"/>
         </fo:block>
       </fo:table-cell>
      </fo:table-row>
     </xsl:if>
     <xsl:apply-templates/>
    </fo:table-body>
   </fo:table>
   <!-- table end -->
  </xsl:if>
 </xsl:template>
```

Format the details of the customer data. The customer data will appear in a blue font right-aligned beneath a a large title bar containing the word "Customer" (The title bar is defined in the section "header" above.)

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

Combining the Power of XQuery & XSLT

Toward fulfilling the promise of XML

Written by **JIM GAN**

People communicate with each other online. People buy goods and services from online stores. People pay bills, file insurance claims, and book travel and flights online. Each of these online activities generates digital information such as e-mail, invoices, statements, hotel reservation confirmations, purchase orders, news reports, and technical documents.

One characteristic of this kind of digital information is that it is semistructured data. E-mail is semistructured; it has structured parts, such as from, to, cc, and subject, and an unstructured part, the e-mail body, which can be any text. Reservation confirmations are semistructured too; they may contain structured information, like room rent and hotel address, and unstructured information, such as the hotel description and cancellation policy.

Another characteristic of this kind of digital information is that it's meant both for people to read and for computers to process automatically. XML (see www.w3.org/TR/2000/REC-xml-20001006) was invented to represent this kind of digital information, ranging from highly structured data like relational tables to semistructured contents like technical documents and news reports.

XML

XML allows people to create their own vocabulary to represent information through simple tagged texts, called *XML documents*. The advantages of using XML are:

- ***XML is simple:*** An XML document is human-readable text. It contains tags, which help computers to process the text automatically. The tagged text looks like this: <city>New York</city>. An XML document is valid as long as it is well formed (in other words, its begin tag matches its end tag).
- ***XML is flexible:*** An XML vocabulary can be easily expanded to add new information. In XML jargon, an XML vocabulary can be defined with a DTD (Document Type Definition) or XML Schema.
- ***XML is self-describing:*** The tagged text tells what the text means. For example, <city>New York</city> says that "New York" is the city while <state>New York</state> says that "New York" is the state. The XML tags provide context so that a search on XML documents becomes more precise.
- ***XML contains content only, no style or formatting information:*** An XML document is different from, for example, Word documents. A Word document contains both content itself and style/formatting information. A correct version of software is needed to read the formatted document. In contrast, since it contains content only, without any style or formatting information, an XML document can always be read without the assistance of software.

These advantages make XML a very popular technology to represent and exchange information. People are creating vocabularies for most vertical industries, creating new XML documents, and converting existing digital contents into XML. I call it information *XMLization*. In this article *XML content* and *XML data* are used interchangeably.

XQuery

To use the massive information in XML intelligently for real business benefits, a powerful query language is required to express sophisticated queries across different kinds of XML data. XQuery (an XML query language) is designed for querying XML data just as SQL is designed for querying relational data. XQuery allows you to navigate the complicated structures in XML, expressing query constraints and constructing the query results. It allows you to query the information stored in XML or view it as XML via integration middleware.

XQuery is still a work in progress. The W3C working draft (see www.w3.org/TR/xquery) is not yet an official W3C recommendation.

XSLT

There will be a multitude of XML documents in different vocabularies. Each organization or vertical industry is defining an XML vocabulary for its own use. EbXML and newsXML are two examples of this kind of XML vocabulary. To exchange XML data such as invoices or orders, the data needs to be transformed from one vocabulary to another. Another benefit of using XML is that it separates data from presentation, so the same data can be repurposed to different devices, such as a Web browser, PDA, and printer. With different stylesheets (formatting), the same XML data is transformed into HTML for presentation on the Web and into PDF for printing. The new XML technology XSLT (Extensible Stylesheet Language [XSL] Transformation) is designed to meet these kinds of requirements.

XSLT 1.0 has been the W3C Recommendation since November 16, 1999 (see www.w3.org/TR/1999/REC-xslt-19991116).

This article describes the differences and similarities between XSLT 1.0 and XQuery, while outlining the pros and cons of each. The various application scenarios that XSLT 1.0 and XQuery are best suited for when used alone, as well as when combined, will be delineated. I'll also explain how to use the standard JAXP transformation API to combine the power of XSLT 1.0 and XQuery. (I'm assuming readers have basic knowledge of XML technologies such as XML Namespace, XSLT, and XQuery.)

"Although XSLT and XQuery are designed to meet different requirements, there are similarities between them"

XQuery and XSLT

As mentioned earlier, the XML vocabulary is often defined in a DTD or XML Schema (see www.w3.org/TR/xmlschema-0). A DTD can define the syntax of an XML vocabulary, while an XML Schema is more powerful and allows the user to define the data type as well as the syntax.

The DTD in Listing 1 defines an XML element, "student". Listing 2 is the student information (stored in the "student.xml" file) in XML, as defined in Listing 1.

An XSLT example

For transformation, an XSLT stylesheet is defined first. An XSLT processor will then use the XSL stylesheet to transform the source XML document into the result document.

An XSLT stylesheet consists of a list of templates. Each template has a match pattern and a template body. The match pattern matches nodes in the source XML document. The template body can be any arbitrary structure, such as plain text, an HTML or SVG tag (see www.w3.org/TR/SVG/), or nodes selected from the source tree.

The following template is used to transform a "name" element into two HTML table rows displaying the firstname and lastname. The match pattern "name" means this template will be applied to any element "name" in the source tree. The template body is the mix of HTML table tag constructs and the selections of nodes from the children of the matched element "name". The template body is shown in blue.

```
<xsl:template match="name">
 <tr><td><b>FirstName </b></td>
 <td><xsl:value-of select="firstname" /></td>
 </tr>
 <tr><td ><b>LastName </b></td>
<td><xsl:value-of select="lastname"/></td>
</tr>
</xsl:template>
```

An XSL stylesheet template is normally defined recursively in a top-down approach. The transformation of the top-level element is accomplished by calling the templates of its child elements recursively.

Listing 3 shows how to define the template body of a template for the element "student" by calling/applying the templates of its child elements "name" and "address" recursively. The instruction <xsl:apply-templates select="name"/> in the template body means selecting the element "name" from all child nodes of the element "student" and applying the template with the match pattern "name" recursively.

An XQuery example

The following XQuery queries the firstname, phone, and e-mail information from the source XML document file "student.xml" and constructs new contact information:

```
let $s := document("student.xml")/student
return <contactinfo>
 { $s/name/firstname, $s/phone, $s/email }
 </contactinfo>
```

The result of the query is:

```
<contactinfo>
    <firstname>Fred</firstname>
    <phone>2626623</phone>
    <email>fred@cs.stratford.edu</email>
</contactinfo>
```

Comparing XSLT and XQuery

Although XSLT and XQuery are designed to meet different requirements, there are similarities between them. In this section I'll briefly compare XQuery and XSLT in terms of data model, content construct, modularization, and expression power. This comparison isn't comprehensive and covers only the interesting parts relevant to this article.

Data model

The XML access data model in the match attribute and select attribute of XSLT instructions is based on the XPath 1.0 data model, which is basically a tree structure with nodes. There are four data types in the XPath 1.0 data model: node-set, string, number, and boolean. There are seven XPath nodes: root node, element, text node, attribute node, processing instruction node, namespace node, and comment node. Note that XPath nodes are slightly different from the familiar W3C DOM nodes. There's no namespace node in DOM, nor is there a CdataSection or EntityReference in XPath nodes. The root node in the XPath node

Construct XML node	In XSLT	In XQuery

Attribute	<xsl:template match="book">
	<book>
	<xsl:attribute name="author" >
	<xsl:text>Andrew Fox</xsl:text>
	</xsl:attribute>
	</book>
		Element book
		{
		attribute author
		{"Andrew Fox" }
		}

Element	<xsl:template match="student"> <contactinfo> <xsl:copy-of select="name/firstname"/> <xsl:copy-of select="phone"/> <xsl:copy-of select="email"/> </contactinfo> </xsl:template>	let $s := document("std.xml")/student return <contactinfo> { $s/name/firstname, $s/phone, $s/email } </contactinfo>

TABLE 1 | Comparison of mechanisms used to construct XML nodes

model corresponds to a document in DOM. Element, attribute, processing instruction, comment, and text in the XPath 1.0 model correspond one to one to their counterparts in DOM.

XQuery is a strongly typed query language, with a data model based on XPath 2.0. It has document node, element node, attribute node, namespace node, processing instruction, comment, and text node. Document node is essentially the same as root node in XPath 1.0. Each node has a node identity concept and a type, which is derived from the corresponding XML Schema definition of the node. Instead of node set, sequence is introduced in the XQuery data model. A sequence is an ordered collection of zero or more items. An item may be an atomic value or a node.

Both data models have the same document order concept. The document order of nodes is defined in order of node appearances in the physical XML document.

Content construct

In an XSLT template body any arbitrary structure can be constructed. But in XQuery only well-formed XML elements can be constructed. In terms of constructing XML nodes, there are similar mechanisms (see Table 1).

Modularization and parameterization

XSLT allows writing a modular stylesheet by providing <xsl:include> and <xsl:import> instructions. You can import or include another stylesheet into a stylesheet. Also, variables and parameters can be defined in a stylesheet to support a parameterized transformation.

XQuery has no include/import support. And there's no mechanism to pass a parameter to an XQuery.

Path expression to access XML data

Both XSLT and XQuery support navigation of an XML document tree structure. A path consists of multiple steps. Each step has an axis indicating the movement direction and a name test matching node name and types. A list of predicates associated with a step is used to further filter the returned nodes from the step.

XQuery supports only six axes: root, child, attribute, descendant, descendant-or-self, and parent. XPath in XSLT supports more axes in addition to those five: ancestor, ancestor-or-self, following, preceding.

Input function

Normally the source document is provided to an XSLT transformer. XSLT has the function document() as an input function to load an external XML document by resolving a URI and return its root node in a stylesheet. The basis document() in XSLT takes a URI as a parameter.

As seen in student.xml, the value of the element bio of each student is a URL to another XML document. The following stylesheet is used to transform the résumé by using document(). The XSLT processor will load the résumé XML document specified by the variable bio, which is a URL, and then apply this stylesheet to the loaded résumé XML document.

```
<xsl:template match="student_directory">
<xsl:for-each select="student">
<xsl:variable name="bio" select="bio"/>
<h3> Resume of <xsl:value-of
select="name/firstname"/></h2>
<xsl:apply-templates select="document($bio)"/>
</xsl:for-each>
</xsl:template>
```

How the URI parameter in document() is resolved depends on the XSLT processor implementation. Some XML platforms, such as a native XML database, enable the user to organize the XML documents into a file system like a collection or folder. The URI format in document() can thus be something like school/computerscience/student/smith.xml to represent an XML document.

The import and include instructions in XSLT stylesheets use document() to load the right component stylesheet to the composed stylesheet.

XQuery has document() and collection() functions. Like document() in XSLT, document() in XQuery takes a URI as a parameter. It is up to each XQuery engine implementation to resolve the URI and return a document node. Collection() is used to query multiple documents in a collection.

Later on I'll show how certain XQuery engine implementations take advantage of this flexibility to provide queries on XML views of different data sources.

Loop and query constraint

In XSLT most query constraints are expressed in predicates of path expression, although there are also conditional processing mechanisms like xsl:if, xsl:when, xsl:choose. The loop is expressed in xsl:for-each.

In XQuery there's a FLWR expression (for-let-where-return clause) to express a loop in the for clause and a query constraint in the where clause. XQuery can easily be used to express joins among different XML data sources.

XQuery is more powerful and flexible than XSLT in expressing query constraints. The differences and similarities between the two are summarized as follows:

- XSLT is better suited for transforming the XML documents as a whole of two different schemas because the stylesheet is normally defined in a top-down way, recursively.
- XSLT is good at formatting XML data into a presentable format like HTML, because any arbitrary structure can be constructed in an XSL template.
- XQuery is better suited for querying over different data sources and constructing new data.

In some situations, however, the two technologies can be used for the same purpose. In the next section I'll describe how to use XQuery to do the transformation between XML documents with two different schemas.

The following code shows a slightly different student record schema, probably used in another school.

```
<student sid="101">
    <name>
      <givenname>Fred</givenname>
      <familyname>Smith</familyname>
    </name>
    <telephone>2626623</telephone>
    <email>fred@cs.stratford.edu</email>
    <homepage>www.cs.stratford.edu/~fred</homepage>
    <score>3.9</score>
</student>
```

These two student XML records have similar structures with different element names. The XQuery in Listing 4 can be used to transform from one kind of student record to another.

Combining XSLT and XQuery

This section describes two major applications where these two promising XML technologies can be used together.

Key technology for content management

Today a large amount of critical business data resides in unstructured documents like Word documents and different application "silos." The unstructured documents need to be *XMLized* by adding XML metadata or converting to XML documents. In addition, the application data in the various applications needs to be published as XML for information exchange.

The traditional form of document, like Word documents, contains both the content text and the style or formatting information. The content itself conveys what the document means. The formatting information is very important to readers because it helps them read the document visually.

The image files on the other hand, like .jpeg and .gif, are in binary format. XML can be used to represent metadata about these traditional non-XML digital assets. Metadata can contain much information about the original content – author, date, format, description, subject, keyword, and so on.

Listing 5 is an example of metadata about an image. It tells the author, creation date, format, description, the actual physical file location, and copyright.

A lot of source documents, like new technical documents and manuals, originate in XML. Application data, like bank statements, invoices, insurance claims, and the like, reside in vertical applications like CRM systems. In order to use this application data for business benefits, the data is extracted and viewed as XML for exchange as well as for presentation. Information residing in different data sources needs to be organized, searched, and combined to generate a new document. Figure 1 diagrams an architectural picture of this kind of application. XQuery is used to aggregate the information from different data sources, while XSLT is used to deliver the resulting information to various targets.

Transformer	Interface to transform the source document into a result document
TransformerFactory	Create a Transformer given a stylesheet
Source	Transformation source
Result	Transformation result
URIResolver	Called by Transformer to convert URL used in document(), xsl:import, xsl:include to a Source object

TABLE 2 | Key transformation concepts defined by JAXP Transform API

Key technology for information integration

XQuery is seen as the key integration technology to aggregate information from various kinds of data sources.

Figure 2 diagrams an integration platform using XQuery and XSLT technologies together. A view mechanism can be created to provide an XML view on top of different data sources: relational tables, Web services, and business applications like CRM and legacy documents. XQuery can be used to search the data across the dynamic XML view data and the XML data itself. XSLT is then used to deliver the query result to meet the business requirements.

I'll use a new application, production of a monthly credit card statement, to illustrate XQuery as an integration technology. The electronic statement is generated in XML format, which is in turn converted to PDF for customers to download, HTML for customers to view online, and a proprietary printer format to print out the paper statement to be mailed to the customer.

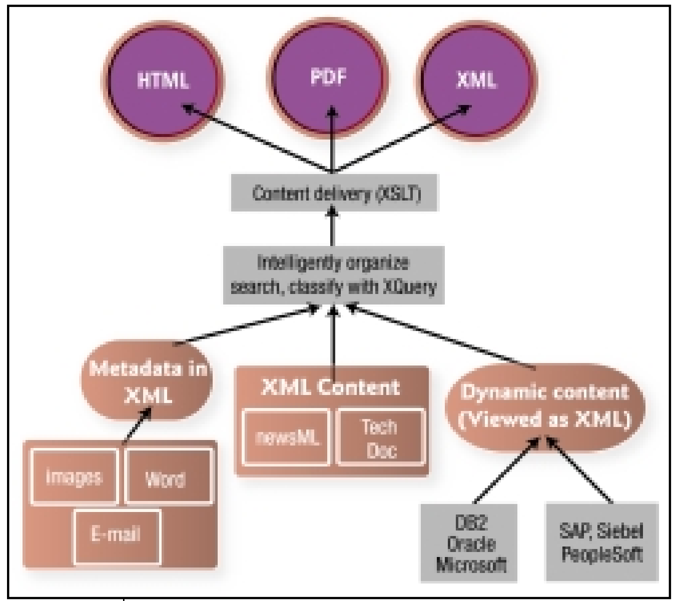


FIGURE 1 | Use XQuery and XSLT in content management.

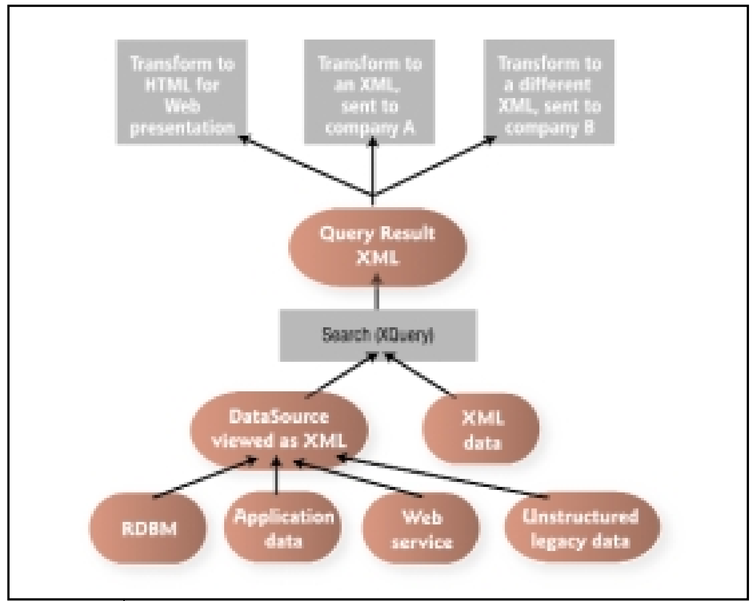

Transform to HTML for Web presentation
Transform to an XML, sent to company A
Transform to a different XML, sent to company B
Query Result XML
Search (XQuery)
DataSource viewed as XML
XML data
RDBM
Application data
Web service
Unstructured legacy data

FIGURE 2 | Use XQuery and XSLT to integrate XML data for exchange and presentation

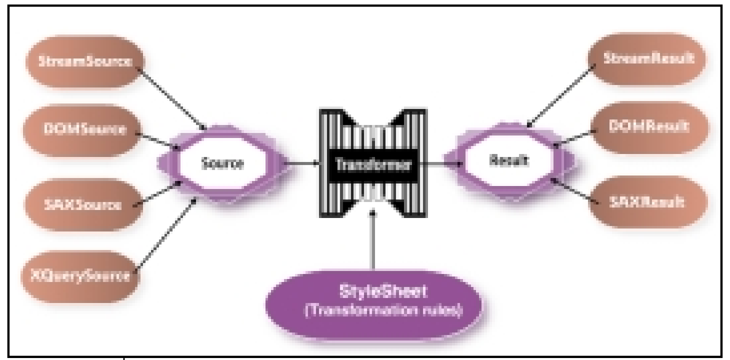


FIGURE 3 | JAXP Transform API

The typical credit card statement consists of an account summary, a credit line summary, payment, new transaction activity, finance charge rate, policies (about payments, finance charges, billing rights, purchases), customer service information (telephone, office hours), and a list of discounts for valued customers (most likely as the result of comarketing between the credit card company and various merchants). The account summary, credit line summary, and new transaction activity are dynamic data, which is fetched from the relational database tables. The policies and customer service information are fetched from the legal credit card agreement document in XML. The discount information is fetched from a CRM application or from marketing materials.

Listing 6 is the incomplete statement DTD. (I won't show all the elements in the statement here.) Listing 7 is an example of the account_summary element.

The statement XML document provides an XML view on top of relational tables so data can be queried through XQuery. A view is provided to map the data stored in relational database tables into a defined XML Schema (see Listing 7). Assuming five views (account summary view, payments view, purchases view, credits view, and finance charges view) that provide the respective mappings, the XQuery in Listing 8 would be used to fetch the statement from the different relational tables. The function Document("account_summary") accesses the XML data mapped into the account summary view.

An XSL stylesheet statement_to_html.xsl would be provided to transform the query result into an HTML file for Web delivery. An XSL:FO (XSL Formatting Objects) would be defined to convert the query result into a PDF.

The query and transformation are combined to produce the electronic credit card statement for delivery formats to meet customers' needs.

Query/Transform with JAXP Transform API

This section, describing the JAXP Transform API for XSLT and the extended new transformation source, shows how XQuery query and XSLT transformation are combined from the point of view of programming.

JAXP (Java API for XML Processing) Transform API is the standard XSLT transformation API for Java programming (see http://java.sun.com/xml/downloads/jaxp.html for details). It defines the key transformation concepts in Table 2.

The key method transform(Source, Result) in the Transformer interface is called to transform a Source into a Result. The interface Transformer also has methods to set parameters in a stylesheet to support parameterized transformations and set output properties to control the result output. Figure 3 shows the JAXP Transform API.

The Source interface represents an abstract transformation source. There are three kinds of transformation sources:

1. ***DOMSource:*** Represents a W3C DOM tree
2. ***SAXSource:*** Represents SAX events as transformation sources
3. ***StreamSource:*** Represents a stream of XML text as a transformation source

The Result interface represents an abstract transformation result. Similarly, there are three kinds of transformation results:

1. Output result as a DOM tree
2. Output result as a list of SAX events
3. Output result as a stream of XML text

So JAXP Transform API is very generic and powerful. Listing 9 shows how a Transformer is created and used to transform the source document student.xml (it is a StreamSource) with the stylesheet student.xsl into an HTML result file.

Now we're going to see how to combine the power of XQuery and XSLT from the point of view of programming. The JAXP Transformation API is very flexible. Both the Source interface and the Result interface are very generic. A new source, XquerySource, is defined to represent an XQuery string.

When an XquerySource is passed into a Transformer, the Transformer will invoke the XQuery engine to first execute the XQuery, and then apply the stylesheet to the query result in order to format or transform it. As shown in Figure 3, with XquerySource, not only are the XQuery query and XSLT transformation of query result combined into one step, but the methods in the JAXP Transform API, such as setting stylesheet parameters, are completely reused.

Listing 10 shows the creation of an XquerySource to represent a query of the XML document student.xml and construct a contactinfo document. The stylesheet contact.xsl is used to transform the query result (ContactInfo) into HTML format. Finally, the output of transformation is written into a file contact.html.

The extension of the JAXP Transform API with Xquery-Source needs the support of both the XSLT transformation engine and XQuery engine, of course.

Summary/Conclusion

XSLT and XQuery are two key technologies for XML processing. XSLT is good for transforming a whole XML document into another XML document with different schema or into a formatted output for presentation. XQuery can be used for XML-to-XML transformation too, but not for formatting XML documents. Like SQL to relational data, XQuery itself is a powerful language to intelligently search the XML data stored in XML. Another key feature of XQuery is that it can query various data sources as long as they can be viewed as XML.

More and more existing non-XML information is being *XMLized* into XML, and all XML technologies can be used together for various applications. To fulfill the promise of XML, a complete XML platform is needed to support these technologies so as to gain real business benefits out of them. This includes XML parsing, XML Schema validation, XSLT, XQuery, XML data storage, and indexing. Supporting some XML technologies at the toolkit level, or one XML technology at a time, just isn't good enough.

AUTHOR BIO

Jim Gan, an engineering manager at Ipedo, led the earlier version of Ipedo XML database development, worked on XPath Query optimization and XSLT, and designed the Ipedo XML database API. He has over nine years of software development experience. Prior to Ipedo, Jim developed the e-commerce software at HP.

JIM@IPEDO.COM

LISTING 1
```
<!ELEMENT student (name, phone, email, address, office, url,
gpa, major, status, program, bio )>
<!ATTLIST student id ID #REQUIRED >
<!ELEMENT name (lastname, firstname )>
<!ELEMENT address (city, state, zip)>
<!ELEMENT phone (#PCDATA )>
<!ELEMENT email (#PCDATA )>
<!ELEMENT firstname (#PCDATA )>
<!ELEMENT lastname (#PCDATA )>
<!ELEMENT office (#PCDATA )>
<!ELEMENT gpa (#PCDATA )>
<!ELEMENT url (#PCDATA )>
<!ELEMENT major (#PCDATA )>
<!ELEMENT status (#PCDATA )>
<!ELEMENT program (#PCDATA )>
<!ELEMENT bio  (#PCDATA) >
```

LISTING 2
```
<?xml version="1.0" ?>
<student id="101">
    <name>
      <lastname>Smith</lastname>
      <firstname>Fred</firstname>
    </name>
    <phone>2626623</phone>
    <email>fred@cs.stratford.edu</email>
    <address>
      <city>Redwood City</city>
      <state>CA</state>
      <zip>94063</zip>
    </address>
    <office>Room 205</office>
    <url>www.cs.stratford.edu/~fred</url>
    <gpa>3.9</gpa>
    <major>Computer Science</major>
    <status>enrolled</status>
    <program>graduate</program>
     <bio>http://cs.stratford.edu/~fred/fred.xml</bio>
</student>
```

LISTING 3
```
<xsl:template match="student">
<table width="70%" border="0" cellspacing="0" cell-
padding="4">
   <xsl:apply-templates select="name"/>
 <tr><td>Phone</td><td> <xsl:value-of select="phone"/>
</td></tr>
 <tr><td>Email</td><td> <xsl:value-of select="email"/>
</td></tr>
 <xsl:apply-templates select="address"/>
 <tr><td>office</td><td> <xsl:value-of select="office"/></td>
</tr>
 <tr><td>URL</td><td> <xsl:value-of select="url"/></td> </tr>
 <tr><td>GPA</td><td> <xsl:value-of select="gpa"/></td> </tr>
</table>
</xsl:template>
```

LISTING 4
```
for $s in document("student-directory-of-schoolA.xml")/stu-
dent
return <student sid="$s/@id " />
 <name> <givenname> {$s/name/firstname } </givenname>
  <familyname> { $s/name/lastname } </familyname>
 </name>
 <telephone>{$s/phone}</telephone>
 {$s/email}
 <homepage> {$s/url }</homepage>
 <score> {$/gpa } </score>
 </student>
```

LISTING 5
```
<image_asset>
<author>Joe Smith</author>
<date>Nov 12, 1998</date>
<documentname>sfnightview.jpeg</documentname>
<format>jpeg</format>
<url>http://imagestore/photo/san
Francisco/sfnightview.jpeg</url>
<description>This image catches the beautiful night view of
San Francisco downtown. </description>
<copyright>National City Associations</copyright>
</image_asset>
```

LISTING 6
```
<!ELEMENT statement ( closingdate, accountnumber, cardholder,
account_summary, payments, credits, finance_charges, dis-
count_list, card_policy_list, service_info)>
<!ELEMENT cardholder (name, address)>
<!ELEMENT account_summary (previous_balance,
payment_received, new_charges, new_balance,
minimum_amount_due, payment_due_date)>
<!ELEMENT payments ( payment+)>
<!ELEMENT payment ( amount, date, referencenumber)>
<!ELEMENT credits ( credit+) >
<!ELEMENT purchases ( purchase+)>
<!ELEMENT purchase ( amount, merchant, referencenumber)>
<!ELEMENT card_policy_list ( payment_policy, credit_policy,
purchase_policy)>
<!ELEMENT service_info ( customer_service_telephone, office-
hours>
<!ELEMENT payment_policy (#PCDATA)>
```

LISTING 7
```
<account_summary>
<previous_balance>1234.45</previous_balance>
<payment_received>1234.45</payment_received>
<new_charges>1023.44</new_charges>
<new_balance>1023.44</new_balance>
<minimum_amount_due>40.00</minimum_amount_due>
<payment_due_date>August 22, 2002</payment_due_date>
</account_summary>
```

LISTING 8
```
let $sc
:=document("account_summary")/account_summary[accountnum-
ber='123456789']
let $payment :=document("payments")/payments[accountnum-
ber="123456789']
let $purchases :=document("purchases")/purchases[accountnum-
ber="123456789']
let $credits :=document("credits")/credits[accountnum-
ber="123456789']
let $finance_charges
:=document("finance_charge")/finance_charges[accountnum-
ber='123456789']
let $policy_docs :=document("credit_policy.xml")
let $service_info :=document("customer_service_info.xml")
return <statement>
 {$sc, $payment, $purchases, $credits, $finance_charges,
$discounts, $policy_docs, $service_info }
 </statement>
```

LISTING 9
```
TransformerFactory tFactory =
TransformerFactory.newInstance();

// Use the TransformerFactory to create a Transformer given
a stylesheet
Transformer transformer = tFactory.newTransformer(new
StreamSource("student.xsl"));

// Use the Transformer to apply the associated stylesheet to
an source XML
// document(student.xml) and write the output to a file
(student.html).
transformer.transform(new StreamSource("student.xml"), new
StreamResult(new FileOutputStream("student.html
")));
```

LISTING 10
```
// create a XquerySource to represent a XQuery
XquerySource xqSource = new XquerySource("let $s := docu-
ment('std.xml')/student
return <contactinfo>{ $s/name/firstname, $s/phone, $s/email
}</contactinfo>");
TransformerFactory tFactory =
TransformerFactory.newInstance();

// Use the TransformerFactory to create a Transformer given
a stylesheet
Transformer transformer = tFactory.newTransformer(new
StreamSource("contact.xsl"));

// the Transformer invokes XQuery engine to execute XQuery
and then apply the associated stylesheet to the query result
and write the output to a file (contact.html).
transformer.transform(xqSource, new StreamResult(new
FileOutputStream("contact.html")));
```

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

XML NEWS

NeoCore Updates XML Database

(Colorado Springs, CO) – NeoCore Inc. is now shipping version 2.6 of its core product, NeoCore XML Information Management System (XMS), a self-constructing native XML database. NeoCore XMS 2.6 includes an optimized standalone server architecture, support of XQuery, and the option for a Solaris 64-bit platform.
www.neocore.com

Ipedo Introduces XML Views

(Boston) – Ipedo is shipping Ipedo XML Views, a new integration technology that unifies information management for portals, custom Web applications, and Web services initiatives.

Features include composite data and content integration, simple connect architecture, multisource search and update, index and cache acceleration, and intelligent query optimization.

The XML Views architecture works across existing relational databases, content management systems, and Web services, and can be tailored using an adapter development kit. The all-Java design includes support for JDBC, JMS, and SOAP, and can be deployed in conjunction with J2EE application servers.
www.ipedo.com

New Advisory Group at OASIS

(Boston) – The OASIS consortium also announced the formation of the OASIS Technical Advisory Board, a group of appointed and elected industry experts who provide guidance on issues related to strategy, process, interoperability, and scope of OASIS technical work.
www.oasis-open.org

Altova Announces New Product Line of XML Tools

(Beverly, MA) – Altova has released three new XML tools – XMLSpy 5, Authentic 5, and Stylevision 5 – to facilitate and advance the adoption of XML technologies.

Authentic 5 is a browser-enabled document editor, Stylevision 5 is a Web developer tool with support for HTML importing, and XMLSpy 5 is an XML development environment. New features in XMLSpy 5 include XSLT debugging, WSDL editing, code generation, and integrated support for Software AG's Tamino XML server.
www.altova.com

New Book, *XPath and XPointer*, from O'Reilly

(Sebastopol, CA) – XPath and XPointer, two related languages key to XML processing, allow developers to manipulate embedded information. XPath is used to locate XML content within an XML document; XPointer is the standard for addressing such content, once located. Developers can find what they need to begin using these two technologies in John Simpson's *XPath and XPointer*, published by O'Reilly.

The book assumes a working knowledge of XML and XSLT. *XPath and XPointer* contains material on the forthcoming XPath 2.0 spec as well as XPath 1.0 and XPointer 1.0.
www.oreilly.com/catalog/xpathpointer/

Enosys Integration Server on Tap

(Redwood Shores, CA) – Enosys Software announced the general availability of the Enosys Integration Server, the industry's first XQuery-based Enterprise Information Integration (EII) server. Enosys also released a 30-day free trial software and demonstration package that includes the full integration suite and a hands-on tutorial that allows users to try the product and walk through a business scenario.
www.enosyssoftware.com

VoiceGenie Revamps VoiceXML Gateway

(Toronto) – VoiceGenie Technologies Inc. has unveiled a major new version of its VoiceXML Gateway. Version 5.5 marks the complete integration of all supported VoiceGenie technologies in one platform, including simultaneous support for PSTN and VoIP, using SIP-based call control, ASR engines), and TTS engines.
www.voicegenie.com

Datawatch Updates VorteXML Designer

(Boston) – Version 2 of VorteXML Designer software by Datawatch Corporation allows users to visually extract, transform, and map data from structured text output, including invoices, purchase orders, reports, log files, and HTML documents into valid XML. It also provides XML conversions without programming; correctly interprets hundreds of data type formats from text data through its Recognition engine; enables transformation and derivation of data through its Calculation engine; works with output from any host system and requires no changes to existing applications; and reduces the length and expense of XML conversion projects.
www.datawatch.com

Nimble Ships Nimble Integration Suite 2.0

(Seattle) – Nimble Technology, Inc., announced the release of the Nimble Integration Suite 2.0, the first data and information integration platform to provide both Open Database Connectivity (ODBC) and XQuery access.

New features also include enhanced concordance support, access to LDAP data, and front-end reporting.
www.nimble.com

Swingtide, New Software Company, Launched to Manage XML Proliferation

(Portsmouth, NH) – Three veteran entrepreneurs have formally launched a new software company, Swingtide, to protect, measure, and maximize companies' quality of business amid the proliferation of XML. The company has raised $4 million in first-round financing from Pequot Ventures, the investment arm of Pequot Capital Management, and private investors. Cofounders Jack Serfass and David Sweet were founders of Bowstreet, an XML and Web services infrastructure company, and Preferred Systems Inc., which sold to Computer Associates. Cofounder David Wroe was previously chief technology officer at CNA, a commercial insurer, and chairman and CEO of Agency Management Services, a software company. Swingtide was conceived with the guidance of a group of executives concerned with the emerging complexity of XML as it spreads unchecked within their corporations. Swingtide is developing software designed to address this problem; their first products are expected to be available later this year.
www.swingtide.com

XML-J ADVERTISERINDEX

ADVERTISER	URL	PHONE	PAGE
ADOS Co., Ltd.	http://www.a-dos.com	81-3-5475-1551	4
Altova	www.altova.com	978-816-1600	52
BEA	dev2dev.bea.com/useworkshop	404-240-5506	2
BEA	www.bea.com/events/dev2devdays	404-240-5506	11
IBM	ibm.com/developerworks/linux/cd		50, 51
JDJ Store	www.jdjstore.com	888-303-JAVA	33
Macromedia	www.macromedia.com/goCFMXlight	888-939-2545	9
NET Developer's Journal	www.sys-con.com/dotnet	888-303-5282	23
PolarLake	www.polarlake.com	353-1-449-1010	13
Richard Hale Shaw Group	www.richardhaleshawgroup.com		19
Sonic Software	www.sonicsoftware.com/websj	781--999-7000	6
SYS-CON Media	www.sys-con.com/suboffer.cfm	888-303-5282	15, 21
Web Services Edge 2003	www.sys-con.com		49
WebServices Edge World Tour	www.sys-con.com	201-802-3069	35
WebSphere Developer's Journal	www.sys-con.com/websphere	888-303-5282	38

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.

REST & Pneumatic Tube Systems

Between SOAP and REST, we get the message

BY JOHN EVDEMON

John Evdemon is CTO of Discovery.Logic (www.discoverylogic.com) and coeditor-in-chief of XML-Journal.

It's often said that history repeats itself – and by studying history we gain better insight into our current (and future) society. In the late 1800s the telegraph was immensely popular, but telegraphs only connected telegraph offices. Messages still had to be transcribed into a paper format and delivered to the appropriate person. Delivering them was a problem due to transportation costs, personnel costs, and time lost between transcription and delivery.

Pneumatic tube systems were developed to transport documents within a building or across an entire city (London had a tube system that traversed well over 50 miles). They were quite successful because they were easy to use and fairly reliable. Senders placed their document(s) within a small capsule that they deposited into the tube, and the system transported the capsule using compressed air. These systems had several built-in benefits — simplicity, security, and shared semantics — and they provided closed, point-to-point connectivity. Most of the tubes ran underground or within walls and were frequently shielded by cement or thick iron ducts; intercepting a message required a far different set of hacking skills (some of which may have involved explosives and pick axes). As with the Web today, documents were most vulnerable at the points of transmission and receipt. Semantics weren't an issue since senders and recipients usually worked for the same company or resided within the same locale (the fact that senders and recipients were humans capable of thinking, reasoning, and interpretation was also a big plus). Pneumatic tube systems proved to be so popular that they were used until the mid-1980s in France. Smaller systems are still in use today at hospitals, banks, and large grocery outlets.

Although tube systems have become virtually obsolete, we can still draw some interesting comparisons to today's popular view of Web services. Most implementations require SOAP for transport, WSDL for describing interfaces, and (optionally) UDDI for publishing/locating the service. Like the old pneumatic tube systems, messages must be placed into a "capsule" (in this case a SOAP envelope) prior to transmission. Secure messaging via HTTPS helps ensure that messages aren't tampered with prior to delivery (no cement or ironwork needed). While SOAP has become a de facto standard for Web services communications, it's far more difficult to understand than a simple pneumatic tube system capsule. SOAP is frequently referred to as a highly flexible application-level protocol that supports virtually any message exchange pattern (MEP), any protocol, any method or type of data. Highly flexible initiatives can be a lot like the paper clip – easy to understand and use until someone bends it out of shape, negatively impacting its interoperability.

A set of simple, universally understood and implemented methods can provide a viable alternative (or extension of) a SOAP infrastructure. The pneumatic tube system was so popular because of its simplicity and ease of use – all tube implementers shared a common set of core methods. The Web shares this trait and is based on a core set of simple methods (POST, GET, PUT, DELETE) that have been widely deployed and used on a global basis – you probably use most of them daily (via your browser) without a second thought. The wonderful thing about these methods is that they're universally agreed upon and highly scalable (the Web itself is built on them). Since we all agree on how these methods operate, why not utilize them for Web services?

This is, in essence, the point made by REST advocates. REST (REpresentational State Transfer) is an architectural style, not an application-level protocol (like SOAP), first described by Roy Fielding (see www.ics.uci.edu/~fielding/pubs/dissertation/top.htm). The REST architectural style recommends using the common HTTP methods (POST, GET, PUT, DELETE) used by the Web itself. This style is highly interoperable since clients need only understand HTTP – there's no longer a requirement (or expectation) to use specific enveloping rules (e.g., SOAP) or invoke proprietary methods (e.g., getStockPrice). Instead of building and sending a SOAP message, clients simply POST a simple XML message (containing only relevant application data) to a specific URI. The URI is linked to an internal method that processes the XML and returns an appropriate response that contains one or more additional URIs, enabling the client to drill down and obtain more detailed information or gain access to additional resources and methods. Sound simple? It is – REST architectural styles can simplify and extend SOAP infrastructures to organizations unfamiliar with or unable to use SOAP. A REST architectural style can also be used to wrap repositories that require knowledge of both SOAP and repository-specific APIs.

While REST may provide some insights into designing highly interoperable systems, some of the limitations of HTTP (such as WSDL-like descriptions, multihop routing, and reliability) have yet to be addressed. Providing SOAP-enabled methods within a RESTful architectural style appears to be a good approach for ensuring the highest levels of interoperability.

Pneumatic tube systems were a success because they provided a simple tool for effective message distribution. Initiatives such as SOAP and REST will continue to be successful due to their simplicity and ease of use (they're also immune to lost messages due to a rat's nest stuck in the pipe). Like the Web, tube systems continue to evolve – some scientists propose them for transporting people instead of documents. Might we see something similar using the Web or Weblike technologies? Beam me up, Scotty. Oops – 404: Person Not Found.

JEVDEMON@SYS-CON.COM

IBM

ibm.com/developerworks/linux/cd

Altova

www.altova.com