Content Management • Data Management • XML Labs

XML JOURNAL

THE ULTIMATE XML ENTERPRISE RESOURCE

November 2002 | Volume: 3 Issue: 11

From the Editor
by John Evdemon pg. 3

Guest Editorial
by Matt Campbell & Carmine Marino pg. 5

Business Documents
by Coco Jaenicke pg. 7

READER FEEDBACK
pg. 7

NEWS
pg. 50

Feature: The Importance of Information Asset Governance *XML is not a silver bullet* — Ram Kumar & George Langley — 8

Optimizing Content: Models of Content Management *Using the cookbook model as a guide* — Jack Danaher — 14

XML & CM: The Role of XML in Content Management *Addressing the need for information sharing* — Dan Ryan — 16

Feature: The Myths of "Standard" Data Semantics *Faulty assumptions must be rooted out* — William C. Burkett — 18

Automation: Intelligent XML Switching *Streamlining global customer support with Sarvega XPE Switches* — Chandru Bolaki — 22

Xindice: Working with Xindice *Moving to an XML database enables more XML-driven apps* — Roy C. Hoobler — 24

XML DB Apps: Writing a Native XML Database Application *Using XSLT and XUpdate to modify data* — Mike Jasnowski — 30

Feature: Using OAGIS for Integration *For enabling everywhere-to-everywhere integration, OAGIS simply works* — Michael Rowell — 34

Automated Scheduling: Distributed AI, Schedules, and the Semantic Web *A glimpse of what's to come as intelligent agents navigate the Web* — Rahul Singh, Katia Sycara, & Terry Payne — 40

XML JOURNAL

FOUNDING EDITOR
Ajit Sagar ajit@sys-con.com

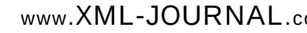
EDITORIAL ADVISORY BOARD
Graham Glass graham@themindelectric.com
Coco Jaenicke cjaenicke@attbi.com
Sean McGrath sean.mcgrath@propylon.com
Simeon Simeonov sim@polarisventures.com

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

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

CONTRIBUTORS TO THIS ISSUE
Chandru Bolaki, William C. Burkett, Matt Campbell, Jack Danaher, Roy C. Hoobler, Coco Jaenicke, Mike Jasnowski, Ram Kumar, George Langley, Carmine Marino, Terry Payne, Dan Ryan, Michael Rowell, Rahul Singh, Katia Sycara

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.



Tracking the Evolution of XML

WRITTEN BY **JOHN EVDEMON**

The XML 1.0 Technical Recommendation was approved in 1998, with a corrective release (termed the "Second Edition") in 2000. The popularity of XML and XML-related initiatives has far surpassed the expectations of the original XML Working Group. The XML 1.0 Technical Recommendation (TR) has been, without a doubt, one of the most popular and successful developments of the W3C. At the time of this writing, the XML 1.1 Candidate Recommendation (CR) had just been released (you can find it at www.w3.org/TR/xml11).

A CR is a W3C Working Draft's final step before being submitted to the director and advisory committee for approval as a W3C TR. CRs are designed to solicit feedback, comments, and criticism from companies, groups, or individuals who are not members of the W3C. A CR signals to the larger technical community that a Working Draft is now considered stable and implementations are highly encouraged. The technical community is also encouraged to share their opinions and critiques of the CR during their implementation process.

The XML 1.1 CR is an update to the original XML 1.0 TR. XML 1.1 is the first potential change to the XML TR in four years. However, it does not introduce any major changes to the original TR – the concepts of well-formedness and validity still apply, and those old familiar angle brackets are still omnipresent. XML 1.1 was designed to provide better Unicode support. At the time of the XML 1.0 TR, the Unicode standard was at version 2.0. The XML 1.0 TR was tied to the current Unicode standard to ensure support for international character sets.

While the XML 1.0 TR has remained stable and unchanging, the Unicode standard has not. The current version of Unicode is 3.2.0 (see www.unicode.org) – several new character sets have been introduced that are not supported within XML names (e.g., element names, attribute names and values, etc.). While many of these character sets may be used within an XML document, they cannot be used to construct XML tag names under XML 1.0. XML 1.0's strict rules regarding XML names may have prevented some countries from creating markup using their native languages. XML 1.1 eliminates this issue by removing XML's tight dependency on a given version of Unicode. In other words, XML 1.0's rules regarding the construction of XML names has been relaxed – unless a particular character is forbidden by XML 1.1, it can be used in the creation of XML names. This "loosening" of the rules enables XML to keep pace with future releases of Unicode, without having to issue a new XML TR each time there is a new version of Unicode.

XML 1.1 also adds two additional enhancements: better support for legacy line-end codes (XML generated in mainframe environments do not utilize "\n"), and support for arbitrary Unicode characters. Why the new version number – why not simply issue XML 1.0 "Third Edition"? A new version number is needed because XML 1.0 and XML 1.1 define slightly different rules for well-formedness. An XML name in XML 1.1 is not necessarily valid under XML 1.0 (XML 1.1–compliant processors will continue to support XML 1.0 naming rules). This obviously makes things a bit more complicated, especially when combining multiple XML data sources using XSLT or XInclude.

I encourage all ***XML-Journal*** readers to take a look at the XML 1.1 CR, if only to better understand the reasons why XML 1.1 is needed. Be forewarned, however – the XML 1.1 CR assumes knowledge of the Unicode standard. If you're new to Unicode (or need to relearn it), I highly recommend Tony Graham's excellent book *Unicode: A Primer* (John Wiley & Sons); *The Unicode Standard* is also available, published by the Unicode Consortium. With the publishing of the XML 1.1 CR, XML continues to evolve – what next? XML has provided us with a data model, and XML APIs provide us with several object models…how about a processing model?

The W3C XML Core WG has begun working on this important issue – you can track their progress via their public home page at www.w3.org/XML/Core. The Web itself is also evolving – the topic of the Semantic Web is one that you'll be reading more about, here and elsewhere. This issue contains an article entitled "Distributed AI, Schedules, and the Semantic Web" that provides a real-world application of the Semantic Web that you can download and begin using today. We also take a look at XML and metadata, XML databases, XML-based content management, and the increasing popularity of XML within switching networks.

As XML continues to evolve to support the future, so does *XML-Journal*. I hope you can join us – it's going to be a wild ride.

ABOUT THE EDITOR

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

JEVDEMON@SYS-CON.COM

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

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

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

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

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

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

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

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

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

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

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

IM – a Viable Tool for the Enterprise

WRITTEN BY **MATT CAMPBELL & CARMINE MARINO**

Instant Messaging (IM) systems have historically leveraged XML as a messaging protocol. The power and flexibility of XML allow IM systems to clearly identify the content and meaning of messages moving through the system. Enterprises are beginning to explore the potential of IM systems as a means of dramatically improving productivity. IM systems will utilize XML not only as a messaging protocol to allow person-to-person communication, but as a means of allowing applications to interact with humans or other applications.

In spite of the powerful capabilities that IM systems provide, the technology has not been rapidly accepted by businesses as an enterprise tool. The technology itself has gained the reputation of being non-productive for an enterprise – even counter-productive. IM technology is overwhelmingly used today by individuals for personal use – the IM community comprises millions of users primarily engaged in personal communication. Many businesses restrict internal usage of popular chat systems to limit the amount of personal communication going on during the workday. Furthermore, IM technology has been commonly identified with chat rooms as a tool for entertaining, personal collaboration. Finally, there haven't been many serious business cases made for use of IM within the enterprise, regardless of the fact that it carries significant advantages over other tried-and-true enterprise technologies like e-mail.

IM technology incorporates many unique and powerful capabilities that are being looked at, indeed implemented, for use within the modern enterprise. Here are just a few:

1. ***Presence:*** Always knowing who is available right now.
2. ***Conferencing:*** The ability to support many people in a collaborative, real-time fashion.
3. ***Archival:*** Easy tracking of all messaging.
4. ***Security:*** Encrypted communication, for example, SSL, can be used.

With the powerful capabilities mentioned above, it's easy to see how IM technology can be used to improve productivity within the enterprise. Presence and conferencing in particular can bring people together to solve problems much more quickly than conventional tools such as e-mail or paging. The next step involves facilitating interaction between not only people, but between people and enterprise applications. Consider enterprise applications with respect to IM technology. What are they? What should they be? Ideally, they would be just like any other "buddy" (with associated presence) that can interact with other entities within the IM layer. If we view enterprise applications as "buddies," it follows that we should be able to define a way of communicating with them through IM to get things done. For example, a purchase order is sent to a vendor and rejected because it contains an invalid part number. The supply chain management (SCM) system notifies the appropriate person via IM, and that person can then modify the part number and resubmit the purchase order entirely by interacting with the SCM "buddy." Also, applications should be able to utilize IM presence to determine which people are available for a variety of interactions, most notably alerts (to facilitate problem resolution) or notifications.

XML is the natural candidate for enabling IM interfaces to a variety of back-end systems. Allowing people to communicate with applications involves exposing a defined set of functionality that can be invoked through the IM layer. When the IM server identifies a message for delivery to an application, the application must be prepared to execute as expected. Through the use of Web services, the IM server is able to indicate to the application what action(s) should be taken.

It's likely that IM will be increasingly used within enterprises as a tool to improve productivity. We're already seeing systems that leverage key IM concepts such as presence to accomplish tasks more quickly than ever before. Furthermore, the trend of exposing enterprise applications using a Web services model will make it easier to allow interaction between humans and applications via an IM layer. With IM capabilities expanding to enable interaction with enterprise applications as well as online friends, don't be surprised if you find yourself chatting with J.D. Edwards or Seibel one day soon.

AUTHOR BIOS

Matt Campbell is president of Bowline Solutions, Inc., a focused group of developers and architects specializing in application integration leveraging IM technology. Bowline Solutions has designed and deployed integrated solutions for commercial and federal customers.

Carmine Marino is CTO of Bowline Solutions, Inc. Carmine's research and development efforts have yielded high-quality IM and XML architectures capable of facilitating person-to-machine interaction via wireless devices.

MCAMPBELL@BOWLINESOLUTIONS.COM

CMARINO@BOWLINESOLUTIONS.COM

Reader Feedback

A Few Resources

Re: "*XML Excellence at Ford Motor Company*"

I would like to know to what extent is that platform planned to control the execution of business processes? Also, here are a few resources for Business Process Management information:

- *Business Process Management*: www.cscrsearchservices.com/process
- *CSC Research Service Report on Business Process Management*: www.cscresearchservices.com/foundtion/library/reports02.asp

Klaus Loehnert
via e-mail

Microsoft Woke Up?

Re: "*Generating Preformatted Reports in Excel*"

Does Microsoft now provide a readable form of Excel's data format?

The case: Microsoft Excel is a substantial reporting tool in almost every company. Creating reports with Excel makes business decisions more visible, and in many cases easier.

The problem: Until now it hasn't been possible to integrate Excel in an automated data workflow in a reasonable way. OLE and COM programming aren't flexible enough for most use cases. Using Visual Basic provides more flexibility but means poor performance, and there is no simple, automated way to handle errors.

The solution? If access to Excel's raw data were simple,Excel would be able to get a more integrated tool in business applications. It would be possible to generate reports automatically, by generating Excel's raw data, in a fast and reliable manner.

Until now, the available API described by the Spreadsheet Reference covered only a few possibilities of Excel's data format. If this API will cover more sophisticated functionality and the new Excel versions are widely in use, this feature will be a great invention!

Salvador Richter
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)

Don't Define a Data Model

WRITTEN BY **COCO JAENICKE**

Traditional application development dictates that you define your data model first, and then design your applications around the data. As information changes and moves at ever-increasing speeds, being dependent on a rigid data model places a limit on how sophisticated integrated applications can get. But with extensible XML, that limitation is removed – applications can be separated from the information model in the same way that HTML and XML separate presentation from content.

Complex Applications

The Internet and XML have given us the promise of ubiquitous connectivity and a zero-latency universe with instant access to all information. The trouble is, we now want and often expect ubiquitous connectivity and a zero-latency universe. Applications are required to be more complex and broader than ever before. But this is old news.

The only way to build these complex applications is to decouple the entities. Just as the client/server movement decoupled the back end from the application and Web servers have decoupled presentation for greater flexibility, the task architects now face is decoupling diverse chunks of business logic. These chunks have many names: if you are hip you call them Web services, if you are an analyst you call them silos, if you are an IT manager you call them legacy systems, and if you are in marketing you call them stovepipes. Whatever you call them, they need to cooperate without being coercive.

When System A is integrated with System B, traditional architectures impose dependencies that have to be hardwired into the application. One of the few remaining dependencies is the shared information mode, which prevents Systems A and B from changing independently and therefore limits the scope of practical integration to a sphere of coercion.

The turnaround that XML enables is that instead of defining a rigid data model and then forcing all applications to conform to it, you can build applications around an extensible information container that allows the data within it to change throughout the life of the business application.

That container is called an XML business document; it acts as flexible "glue," allowing systems to cooperate without coercion. Because XML business documents can extend to accommodate change, systems aren't forced to conform to a predefined data model.

New Conundrums

While working with business documents sounds straightforward, the devil is always in the details. Many of the platform vendors have solved (or are solving) the problem of integrating services from a processing point of view, but there are many additional issues surrounding data sharing in an asynchronous environment.

Because a services paradigm allows for broader integration, business processes that are executed within these applications take significantly longer. Long-running processes that are fully integrated can't assume that the world stands still until the process is over. In addition, because services must remain loosely coupled, the sender of the data can't make any assumptions about the receiver of the data. This leaves a few things to consider:

- ***Unscheduled downtime:*** Services may be remotely located, may be offline, or may even involve human input. In each case, the subsequent service in the process may not be available when it is being called. Where is the data stored while the process is waiting?
- ***Managing temporary data:*** As data is being shared (or is waiting) between services, it may need to be changed. For example, customers may add or subtract from their orders, or perhaps the supply chain has a new member. Where do you store the new data, and what if it changes the schema of the data file?
- ***Managing process state data:*** In a long-running complex process, there is often state data, or temporary data that only has relevance during the process, such as pending approvals or a shipping status. This isn't data that would be stored in a record in the back-end database, so where do you store data that is needed throughout a long-running process?
- ***Transformations of data:*** Given that services are loosely coupled and can be running any software, data may need to be in different formats for different services. How is the data transformed, and how do you guarantee not losing this data in the event of a crash?

These considerations can be irritants, especially since we typically have a procedural view of things, but the business document is a simple metaphor that pulls it all together.

–continued on page 48

CJAENICKE@ATTBI.COM

The Importance of Information Asset Governance

XML is not a silver bullet

WRITTEN BY
RAM KUMAR &
GEORGE LANGLEY

The greatest benefit of XML lies in its potential for managing "islands of data" locked into proprietary tools, systems, applications, and technologies through the use of a common data format that's understandable to any process and system that use it for data representation, data exchange, and application integration. In fact, if XML doesn't achieve this potential, there is little point in using it at all.

Following a trend all too familiar to the IT industry, XML is being seen as a silver bullet that will magically solve all the problems of data integrity, instantly remove islands of data, and disentangle the spaghetti of data interfaces across the enterprise. Typically, those who have willingly embraced this "Next Big Thing" approach will be sadly disappointed, and another real opportunity to move forward could be lost.

XML's greatest strength is also its biggest weakness. It is so flexible that different vocabularies and structures can be written to define the same type of data. Left unchecked, this can lead to a confused set of fragmented, point-to-point XML formats that simply move the problems of data islands and integration around, instead of actually addressing them. Evidence from early (and very expensive) XML implementations confirms this problem. Many software vendors are implementing proprietary layers of XML that can give the appearance of providing a common format for data exchange, but in fact are designed to lock users into proprietary platforms. As a result, eXtensible Markup Language is fast becoming the "eXtensively Misused Language."

XML will only succeed when businesses truly embrace the principles of openness, transparency, and reusability that XML can support. This will require moving away from point-to-point negotiations for exchanging data toward an agreement on standard ways of defining and expressing XML vocabularies, and on common guidelines for the implementation and ongoing use of these standards.

The Need for an XML Clearinghouse

The effective application of XML to overcome the "islands of data" problem requires proper implementation strategies to ensure consistent exchange and reuse of information assets using XML. All projects using XML should have uniform access to appropriate XML documents and their related metadata, which can be only be achieved through the development and deployment of an XML Clearinghouse (also known as XML registry and repository) for the enterprise that provides a single location to store and manage the common XML standards and vocabularies used in the enterprise.

An enterprise XML Clearinghouse allows users to:

1. Discover and use pertinent XML components in the enterprise
2. Register additional XML components that can be reused (e.g., projects, applications) by other projects within the enterprise

What is an XML Clearinghouse?

An XML Clearinghouse is a software model that consists of two key components:

1. ***Registry:*** Stores relevant descriptive information, or metadata, about registered information components and their associated objects, and allows the metadata to be operated on in various ways.
2. ***Repository:*** Stores the registered information components and their associates. It also provides interfaces to retrieve and use registered information components and associated objects.

Registered information components and their associated objects provide users with the information necessary to effectively use them. The key information components registered in the Clearinghouse are the XML Schemas and XML DTDs that can be reused within the enterprise by XML developers, projects, applications, etc. Other information components may also be registered.

Examples of associated objects of registered information components include:

- XML documents
- XML specifications
- XML specifications usage
- UML models
- Other projects that use the information components and the implementation details
- XML design and implementation guidelines
- Usage conflicts and collisions, including details of their resolution
- Business process descriptions
- Trading partner agreements

The relationship between an XML registry and an XML repository is shown in Figure 1.

The XML Clearinghouse can store different XML-specific

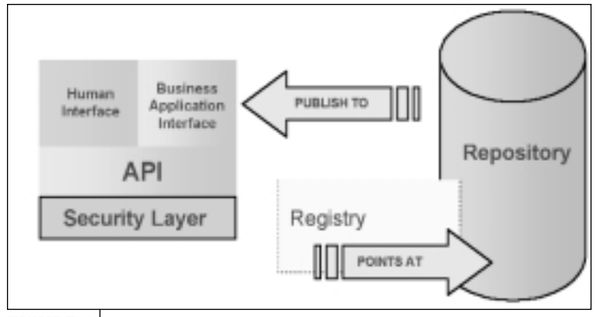


FIGURE 1 The relationship between an XML registry and an XML repository

files: XML, XSL, XSLT, XSD, and many others from the XML family. The main goal of the registry and repository is to share XML vocabularies between interested parties so they can discover and understand each other's vocabulary. The XML Clearinghouse ensures consistency, quality, and compatibility among XML projects within the enterprise and also helps to ensure that the XML standards used are controlled and monitored effectively.

Using the XML Clearinghouse

The following step by step example illustrates how the XML Clearinghouse works:

- Project A of an enterprise develops an XML vocabulary (DTD or Schema) for structuring the customer name and address data that it uses.
- Project A registers the vocabulary (information component) with other appropriate and relevant information (associated objects) as needed, such as description of the vocabulary, implementation guidelines, and so on, with the registry of its enterprise's XML Clearinghouse.
- Project A wants to communicate with Project B by submitting name and address data electronically as XML documents that are validated against the registered XML vocabulary. Instead of writing its own XML vocabulary for name and address data, Project B should seek to use an XML vocabulary that has already been registered with the XML Clearinghouse of its enterprise for structuring their submission, as per the policy implemented by the enterprise.
- Project B then searches the registry of the XML Clearinghouse for vocabularies that are designed for name and address data, to see how they have been used by other projects, and in particular to verify what has been adopted by Project A, and selects and downloads the proper vocabulary.

Note that the registry should also provide facilities to store multiple versions of the same XML vocabulary. However, it is up to each project to decide which versions of the XML vocabulary it will support.

Figure 2 shows how an XML developer can hypothetically use an XML Clearinghouse in a government environment consisting of many agencies and business units.

Benefits of an XML Clearinghouse

The XML Clearinghouse is an essential part of the infrastructure that supports the reuse of common information components across business applications. It has four main benefits:

1. ***Helps protect enterprise information assets:*** The information assets represented in the form of XML Schemas and DTDs that are commonly and consistently used across the enterprise are preserved, maintained, and protected from being locked into proprietary or custom-built applications that result in new data islands.
2. ***Promotes cross-project activities:*** The XML Clearinghouse stores information on the use of the XML Schemas and DTDs by different projects, which gives projects insight into XML-related activities and provides opportunities for collaboration.
3. ***Promotes a common understanding of information components:*** Storing the information models and metadata for subject areas (such as customer, product, etc.) as XML Schema in a single location promotes a common understanding of the purpose of the registered information components.
4. ***Facilitates B2B communication:*** External business partners can access the XML Clearinghouse so they can exchange data with the enterprise in the same standard ways used by internal business systems. This can expedite B2B communication and greatly reduces the cost of supporting it.

The main thrust of the XML Standards Clearinghouse work is to provide visibility and awareness of XML data elements in use within various functional areas across an enterprise. This awareness is important to application integrators and system interfaces.

Use of an XML Clearinghouse needs governance

Despite the benefits of having an XML Clearinghouse for an enterprise, if the XML Clearinghouse were left uncontrolled, every point-to-point XML vocabulary would be registered, each reflecting the narrow interests of application areas. This would result in the proliferation of many diverse data formats within an enterprise to represent the same piece of data. New data islands of information would be created, resulting in new and expensive one-off formatting problems. This would seriously threaten the credibility and integrity of the XML Clearinghouse and defeat the purpose of using XML in the first place.

An XML Clearinghouse will only be effective if policies are set, usage guidelines established, and a management and funding process put in place for effective operation. Policies and procedures should include:

- Registration procedures for new XML components
- Verification procedures for input to the registry
- The extent to which developers will be required to consult the standards registry when deploying XML data structures
- Classes of compliance for categorizing how rigorously projects within the enterprise adhere to standard data structures and definitions
- A configuration management process to keep track of successive versions of each registered component

Benefits of a governance model

The benefits of a governance model to manage XML Clearinghouse use in an enterprise are numerous. Some of the key benefits are listed below:

- Facilitates the convergence of standards within an enterprise, and reduces the costly divergence and fragmentation of XML used by different business units/projects.
- Supports the management and control of the information assets of an enterprise by ensuring consistency, quality, and compatibility of the information assets used across the enterprise.
- Enables an enterprise to track and monitor the progress of different XML projects, thereby avoiding duplication, conflicts, and inconsistency in the XML initiatives.

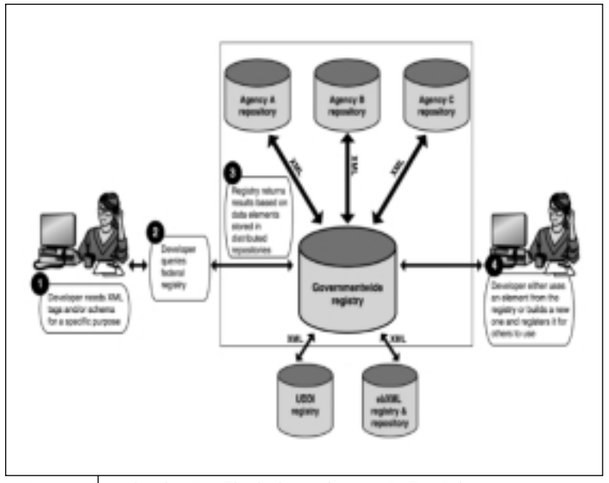


FIGURE 2 Using the XML Clearinghouse (Source: GAO, USA)

- Provides a set of guidelines, policies, and procedures for implementing XML standards in an enterprise. This will provide a common understanding of data and a standard way of implementing XML that will enable the interoperability of information and business processes.
- Enables an enterprise to monitor the evolution of the XML standards at national and international levels.
- Resolves conflicts between different projects within the enterprise regarding the definition, creation, usage, and implementation of XML standards.

Potential XML-Related Work in an Enterprise and the Importance of Governance

This section describes the different types of XML-related work experienced implementing XML strategies for enterprises and the importance of governance for such projects.

Projects that use existing XML standards

Projects could be using existing industry XML standards. It's therefore important for the enterprise to monitor and govern this type of project for the following reasons:

- Several projects in an enterprise could be planning to use the same XML standard, which could result in duplication of effort in evaluation and implementation.
- A single mechanism should monitor the changes in industry standards on behalf of the enterprise and document how to apply any changes within applications.
- It's important to have a common agreement on which XML standards will be adopted, since there could be many flavors of similar standards.
- It's important to have a common agreement on how XML standards are implemented since an XML standard could be implemented in different ways and still meet its purpose.

Projects that need to extend or develop XML standards

There will be situations where existing industry XML standards don't meet the requirements of a project. In such circumstances the project may decide to cover identified gaps by extending existing standards, or by developing a new standard. It's important for an enterprise to monitor and govern this activity in order to:

- Produce a common set of guidelines for adopting, changing, or building XML standards. This will encourage interoperability and foster a common understanding and use of the standards in an enterprise.
- Ensure that effort is not duplicated where two projects share the same requirements.
- Encourage projects with similar requirements to collaborate to extend or build common standards that meet all requirements, thereby producing more robust XML standards and fostering interoperability and common understanding.
- Minimize the impacts that changes to existing enterprise XML standards might have on projects and business applications.

Developing enterprise XML standards

Projects in an enterprise may be building an enterprise XML standard for a subject area not addressed by existing standards, and the enterprise must govern this type of activity. It's important to:

- Gather all inputs and requirements in a central place in order to expedite consensus across business units about the development and deployment of XML standards
- Enable different projects to communicate using a common language when developing or adopting XML standards
- Coordinate the enterprise XML standards initiative to avoid version control problems, conflicts and collisions, confusion, and misunderstandings about developing and deploying XML standards

Turn locally developed XML into open industry standards

There may be times when an enterprise decides to submit an XML standard it has developed to a global XML standards body or a local (e.g., national) standards body to ratify it as an "open" XML standard. For example, our organization submitted three XML standards for customer information management to OASIS to develop and promote them as open industry standards free of royalty and IP issues:

- ***NAML:*** Name and Address Markup Language
- ***CIML:*** Customer Identity Markup Language
- ***CRML:*** Customer Relationships Markup Language

This has resulted in five open industry standards under the control of the OASIS Customer Information Quality Technical Committee:

- ***xNL:*** Extensible Name Language
- ***xAL:*** Extensible Address Language
- ***xNAL:*** Extensible Name and Address Language
- ***xCIL:*** Extensible Customer Information Language
- ***xCRL:*** Extensible Customer Relationships Language

It's important for an enterprise to monitor and govern this type of activity in order to:

- Ensure that the enterprise's interests are strongly and accurately represented at the global or local XML standards level when modifications and extensions to the standards are being considered
- Position the enterprise as a world leader in the development of global XML standards for industry (this requires active sponsorship by the enterprise)
- Ensure that changes and extensions ratified by the global or local XML standards body to standards submitted by the enterprise are reflected in their deployment within the enterprise

The XML Standards Governance model will enable enterprises to form closer relationships with global/local standards

–continued on page 48

WRITTEN BY **JACK DANAHER**

Models of Content Management

Using the cookbook model as a guide

Organizations have a variety of content throughout their enterprise, ranging from simple files, including word processing documents, PDFs, and spreadsheets, to more sophisticated documents such as Web pages or those filled with digital graphics or complex formulas and tables. Whether data is basic or complex, organizations can make the most of their valuable intellectual assets by implementing a content management system that weaves this disparate content into a meaningful content network model.

Traditional business information, including contracts, sales reports, accounting statements, and status reports, is often stored in simple applications such as spreadsheets or word processing documents. However, if these documents are not networked and are not centralized in a repository for multiple users to access, they can be limiting. Organizing this type of data in a content management system is simple and quick and provides organizations with immediate benefits, including security, easy access to content, linking in workflow, and advanced searches for information (see Figure 1).

The example shown in Figure 1 isn't robust enough to enable the simultaneous update of the spreadsheet by authorized users. For example, if one person in the organization is updating the spreadsheet, this creates a bottleneck because other users, although they can view it, can't update this spreadsheet at the same time. This can restrain the whole spreadsheet update process if there are many users who need to perform updates.

While this poses a challenge, organizations can easily create an ad hoc XML content management model to resolve this issue. An ad hoc XML model can represent the columns in the spreadsheet and capture the data as components, which would allow salesman A to update his prospects in the spreadsheet while saleswoman B is updating a separate component for her prospects (see Figure 2). Through this solution, organizations become empowered because they have control of their content at a component level, which makes advanced processing, validation, and more refined workflow possible.

However, because some organizations have a plethora of detailed and complex content, such as product documentation, repair manuals, and highly technical engineering information, they need to go a step beyond an ad hoc XML content management model and, instead, implement a networked content management model. This model encompasses the content shown in Figures 1 and 2, but also includes complex documents that have extensively defined document type definitions (DTDs). This networked content management model provides more advanced processes, such as decomposition and cloning, to make an organization's intricate information more manageable and digestible. Additionally, a content management system should provide repurposing, workflow, and cross-media publishing across the content network model.

An organization can also build a network of information by linking data as shown in Figures 1 and 2. For example, a saleswoman can access a request for proposal (RFP) that links to her client's original RFP, and when she is looking at her prospects list she can simultaneously click on a link to open the RFP (see Figure 3).

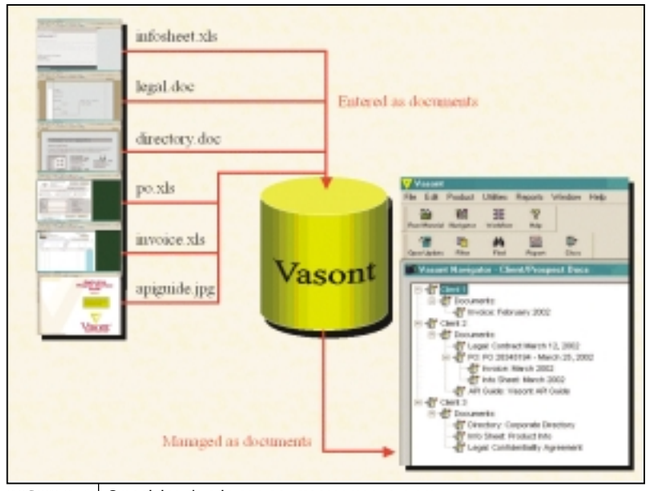


FIGURE 1 | Organizing data in content management system

AUTHOR BIO

Jack Danaher is the director of database development of Progressive Information Technologies and has more than 33 years of experience in the IT database field. Jack is responsible for the ongoing development of Progressive Information Technologies' flagship Vasont product, a powerful cross-media publishing system that enables enterprises to create, manage, and publish content across numerous media channels.

Users can also click on a link that would bring them to a client or prospect demographic list. This component-level content would contain all of the contact information about a client or prospect, including address, phone number, e-mail, and other pertinent information.

The most efficient way, though, for organizations to manage, access, and publish their content enterprise-wide for multiple uses is to leverage the full capabilities of a content management system. A cross-media publishing system for content management, like Vasont, offers advanced modeling capabilities that automate the separation of the content into more digestible modules.

One way to visualize how Vasont creates and manages large volumes of dispersed content is by thinking of a cookbook and the structure of its content. A cookbook contains chapters, and within the chapters are many recipes. And the recipes can be further broken down into ingredients. Additionally, within the cookbook are graphics and charts used to represent the recipes, ingredients, and prepared dishes. Vasont can store the cookbook's content in the following four distinct models (see Figure 4):

- A cookbook model that houses recipes and their graphics in containers
- A recipe model that holds all of the recipes, regardless of which cookbook(s) they appear in
- An ingredients model that stores all of the ingredients that were used in any of the recipes, regardless of the cookbook(s) they appear in
- A graphics model that keeps all of the graphics, no matter where they're used

Because the system categorizes and stores data in modules, it provides numerous benefits for organizations.

Maximizing Content Reuse

Vasont stores only one copy of existing content in its repository for usage enterprise-wide. This saves organizations time because they have to update or edit the content only once, and it's automatically applied to all relevant content uses.

Referring back to the cookbook example, a number of recipes throughout several cookbooks may require a tablespoon or some unit of butter. But if the organization decides to produce several low-fat cookbooks and wants to replace butter with margarine, it can easily make this change one time, and it will be updated in every applicable recipe within every cookbook. Additionally, an organization can easily and quickly develop and cross-media publish new and varied cookbooks on demand in multiple media formats, including print, CD-ROM, and the Web.

Simplified Workflow

Through a content management system's workflow application, users can easily build a process that controls the creation, review, editing, and approval of the information. The integrated workflow application provides a set of checkpoints and notifies users when a task has been or needs to be completed.

Using the cookbook example, there can be one workflow that coordinates the creation of graphics, another workflow that manages the creation and testing of recipes, a third that allows for up-to-the-minute checking of the latest nutritional information on ingredients, and, of course, a master workflow that coordinates these subworkflows so the assembly of finished cookbook products can be coordinated.

This workflow can be expressed in a graphical representation, which the system then interprets and implements (see Figure 5).

Availability and Accessibility

An organization's content can be made available enterprise-wide through a content management system. This system can make content available on a local network or over the Web.

Security

Organizations can specify which users are permitted to access the content, and only authorized users can manipulate the data. Additionally, because a content management system comprises a database that is continually backed up, this ensures that an organization's content will not be lost and that when documents are updated, earlier versions can be recovered.

Similar to a cookbook, organizations comprise an assortment of "recipes" and "ingredients" that need to be categorized, continuously updated, and stored. A content management system allows organizations to take advantage of their disparate data by networking it, thus making it accessible throughout an entire enterprise. With this dynamic technology application, organizations of all sizes can optimize their content and achieve a newfound productivity.

JDANAHER@PIT-MAGNUS.COM

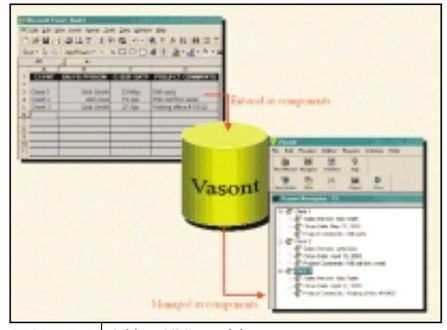


FIGURE 2 | Ad hoc XML model

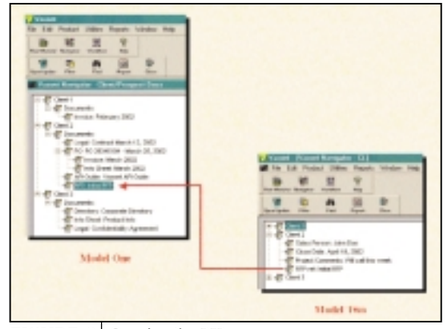


FIGURE 3 | Opening the RFP

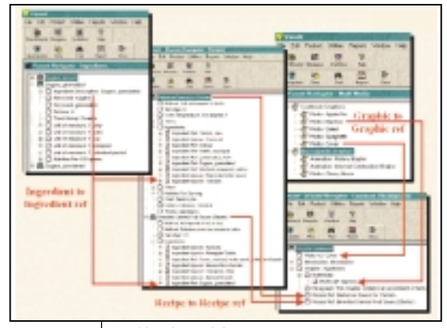


FIGURE 4 | Cookbook model

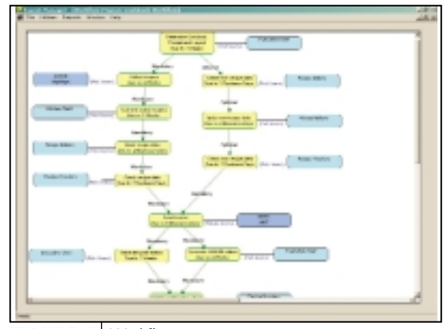


FIGURE 5 | Workflow

WRITTEN BY **DAN RYAN**

The Role of XML in Content Management

Addressing the need for information sharing

The data inside a corporation doubles every six to eight months, according to META Group. As a result, content management systems (CMSs) have become a critical component of organizations' IT infrastructures, managing all enterprise content for a variety of applications.

In the past, exchanging information between content repositories and data-oriented applications within and across organizations was extremely difficult, as many of these systems are incompatible with one another. Consequently, users were often forced to manually convert native content into Web-viewable formats or to cut and paste data into Web-based templates before distribution or publishing. These inefficient processes drove the development of CMSs, offering robust conversion technology that provided users a way to automatically convert and distribute native content via the Web.

XML has recently emerged as a popular mechanism in the content management industry for creating, managing, and exchanging data. As a standard, common format, it can be easily converted to and distributed in numerous other Web formats to best suit the needs of end users, applications, and devices (see Figure 1). XML has quickly become a de facto tool for sharing content between disparate enterprise systems.

It's therefore essential now for organizations to consider the integration of basic XML functionalities when implementing a CMS.

The Content Management System

Providing critical business information on the Web is no longer an option for today's enterprise – it's a prerequisite to success. Consequently, the CMS has evolved into a must-have business application.

These systems allow enterprises to rapidly deploy line-of-business Web sites, such as partner portals and intranets, and manage content on an enterprise-wide basis for use by multiple sites and applications, including enterprise portals. In either case the system automates the cumbersome task of contributing, managing, and publishing business information to the Web. Content management solutions provide a secure, scalable method for checking in, checking out, revising, routing, and approving a wide range of business and Web content, including documents, product catalogs, Web assets, marketing materials, CAD drawings, and regulatory documentation.

CMSs facilitate information exchange between an organization and its customers, partners, and employees and increase the amount, timeliness, and accuracy of shared content. In some cases they have evolved to provide tightly integrated team collaboration functionality. As a result, these implementations generate significant return on investment for companies, such as cost savings and increased productivity.

The content management product suites on the market today continue to evolve and expand, bringing content management capabilities further into the enterprise to reach more users and applications and manage more content.

XML Functionality in Content Management Systems

As companies increasingly rely on CMSs to manage and deliver larger amounts of data to a greater number and wider spectrum of users and applications while also integrating with other IT infrastructure, the benefits of XML functionality within the content management application grow in importance. Following are XML-based features to consider during a content management implementation.

Content contribution and conversion

In addition to accepting XML content, robust CMSs enable users to contribute information into a content server in its native format – such as a word processing document, spreadsheet, or graphic file – and automatically convert this content to XML. By converting and storing content in XML, companies can maintain one single authoritative source of the data and transform it into a variety of other formats, such as HTML, WML, and other XML flavors, for reuse by multiple applications and devices.

Organizations can leverage their investments in existing applications and user expertise by implementing a CMS that automatically converts native content to XML. These systems allow users to create content in the formats they prefer and easily share it via applications they're most familiar with.

For instance, an insurance agent can create and complete a claims form in Microsoft Word and contribute it into a CMS that automatically converts the document to XML for content reuse by other applications or Web sites.

Content access and exchange

XML content can be easily merged from disparate sources and represented in one common manner in content management repositories. Because XML is a standard language companies use to share information, XML data can be easily accessed from content repositories and exchanged between applications and organizations.

AUTHOR BIO

Dan Ryan is senior vice president of marketing and business development for Stellent, Inc., a global provider of content management solutions. Stellent has been involved with XML and its predecessor, SGML, for more than 10 years.

As an example, a brokerage firm can store information on individual stocks as XML in a centralized content repository. From there, the information can be integrated with an e-business portal or business intelligence system.

Content formatting and presentation

With XML content, CMSs can separate formatting and display information from structure in content. This separation of content and presentation allows different formatting to be applied to the same content in different situations using XSL stylesheets. As a result, one XML file becomes the single authoritative source from which users can manipulate, edit, and present information with different looks and feels.

For example, two different stylesheets can be applied to a product fact sheet published to both an Internet site and an intranet. One stylesheet would format the document to match the look and feel of the Internet site, while the other would format the content to correspond with the intranet.

Content storage

According to ZapThink, the market for XML data storage technologies will grow to more than $4.1 billion by 2005. CMSs that integrate with and store content in XML databases provide a number of unique benefits that content management solutions integrated with relational databases do not.

A content management/XML database integration provides a single, common repository where XML data can be stored and accessed by users for dynamic assembly in any presentation. XML content can be stored in its original form instead of being broken up into specific components, which is a necessary step for storage in relational databases where XML content can often lose some of its meaning.

XML content stored in an XML database can be more easily searched. However, organizations should integrate an XML database that isn't proprietary to ensure that it's searchable using standard XML search languages endorsed by the W3C. Integration with an XML database enables CMSs to separately manage and store XML content and XSL stylesheets, combine them, and convert and deliver them in any format.

For instance, a publishing company can separately store book pricing information as well as a number of formatting options for price lists in an XML database. If a user requests a price list that matches a specific marketing packet, the CMS retrieves the XML pricing information, combines it with the XSL stylesheet that provides the requested formatting, and delivers the final version in a PDF document the user can easily print out.

Content personalization

XML data can be manipulated and customized during its assembly on a content server to target a specific user or device. Once the CMS identifies the user profiles and the type of device the content is being delivered to, the appropriate XSL stylesheet is selected. The XML files are then quickly broken down and recombined on the fly, and tailored content is delivered to the user.

For example, content that comprises a product price list can be broken down and recombined on the fly to show wholesale prices to one customer and retail prices to another, depending on their user profiles.

Content Management Web Services

XML also plays an important role in content management Web services. When building sophisticated content-centric Web applications, content and functionality frequently must be shared among many applications. Web services are increasingly becoming a preferred method for integrating these applications.

Growing in popularity are Web services that rely on XML for exchanging and accessing data and applications. The self-defining nature of the XML language allows disparate systems to understand each other with little or no custom coding, which is a substantial benefit over previous distributed computing approaches.

Many CMSs leverage Web services to share and deliver data and specific content management features on the Internet, within an organization, or across corporate boundaries. The components of these Web services are:

- ***SOAP (Simple Object Access Protocol):*** XML-based message format used to communicate or deliver requests between Web services
- ***WSDL (Web Services Description Language):*** XML-based description that defines the way a Web service is to be accessed and used
- ***UDDI (Universal Description, Discovery and Integration):*** XML-based directory used for the registration and real-time lookup of Web services (by looking up a service in the UDDI directory, a Web service can find another Web service to dynamically integrate)

Web services are componentized and provide platform-independent application access that enables customers, suppliers, and trading partners to access content management Web services regardless of their particular hardware, operating system, or even programming environments.

Web services reduce the costs and time involved in integrating content and content management functionality with other applications. They can be used as a cost-effective replacement for expensive Electronic Data Interchange (EDI) integrations as well as pure application-to-application integration projects such as a CMS with an enterprise portal.

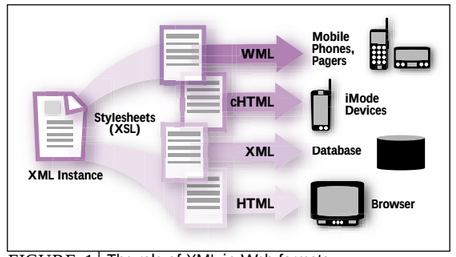


FIGURE 1 The role of XML in Web formats

A variety of options are available to choose from when it comes to deploying and managing content-centric enterprise Web sites and applications. In many cases XML is a critical part of the solution. Whether an organization's XML strategy involves converting and publishing content to XML, providing robust content management features to file types such as XML and XSL, dynamically transforming XML documents to a presentable HTML format, or sharing data among other enterprise applications, content management implementations can be greatly enhanced with the following basic XML functionalities:

- *Automatic conversion of native content into XML*
- *Integration with an XML database to enable easy access to XML content stored there*
- *Availability of Web services to easily access and share data and content management features*

The increased need for companies to effectively share data between internal and external audiences and applications via the Web has made content management technology a top priority for many IT executives. The enhanced functionality that XML-based features and Web services can bring to content management implementations enables companies to further increase and leverage the value of these IT investments.

DAN.RYAN@STELLENT.COM

The Myths of "Standard" Data Semantics

Faulty assumptions must be rooted out

WRITTEN BY
WILLIAM C. BURKETT

Much of the literature heralding the benefits of XML has focused on its application as a medium for application interoperability. With (a) the Internet as a platform, (b) Web services as the functional building block components of an orchestrated application, and (c) XML as a common data format, applications will be able to communicate and collaborate seamlessly and transparently, without human intervention. All that's needed to make a reality is (d) for everyone to agree on and use XML tags the same way so that when an application sees a tag such as <firstName> it will know what it means.

This intuitive understanding makes a lot of sense, which is why so many organizations have sprung into existence to create their own vocabularies (sets of tags) to serve as the "lingua franca for data exchange in <insert your favorite industry, application, or domain>." This intuitive understanding is so pervasive that it's even a key part of the U.S. GAO recommendations to Senator Joseph Leiberman (chairman of the Committee on Governmental Affairs, U.S. Senate) on the application of XML in the federal government. This report warns of the risk that:

…markup languages, data definitions, and data structures will proliferate. If organizations develop their systems using unique, nonstandard data definitions and structures, they will be unable to share their data externally without providing additional instructions to translate data structures from one organization and system to another, thus defeating one of XML's major benefits.

The perspective of these efforts is that the standardization and promotion of the data element definitions and standard data vocabularies (SDV) will solve the application interoperability problem. Unfortunately, this intuitive understanding – like many intuitive understandings – doesn't survive the trials of real-life application because important (and seemingly trivial) assumptions are poorly conceived. This article will examine some of these assumptions and articulate several myths of "standard" data semantics.

For the purpose of this article, the term "namespace" will be used to denote a domain in which a vocabulary (i.e., a set of data element names or tag names) is used with a particular meaning and purpose.

Context of Position

"Context" plays an important and largely unrecognized role in data semantics. Similarly, context plays an important role in understanding the position expressed in this article. To set the context for the comments expressed herein, I'll summarize the perspective from which they originate.

My background is in industrial and systems engineering rather than computer science. I'm more concerned about the overall system behavior of integration and interoperability, and about the human element within the system. I also have 20 years of experience in the development of schemas and the standardization of data exchange schemas within the International Standard Organization (ISO). This latter experience, in particular, has made me acutely aware of the pitfalls confronting schema developers and the "standardization of semantics."

Myths of Data Semantics

Several authors have described the dangers and problems of standardizing data semantics. Perry warns of the willful misuse of SDV by "gamesters" who exploit the semantic play in language (e.g., Bill Clinton's use of the term "sexual relations"). Knox cites the same problem with the GAO report previously described: "Except for the most rudimentary concepts, the assumption that XML-defined models will be the same from one industry to the next, from one country to the next, or from one point in time to the next is naïve."

The notion that data semantics can be standardized through the creation and promulgation of data element names/definitions or vocabularies is based on several assumptions that are actually myths:

- ***Myth 1:*** Uniquely named data elements will enable, or are enough for, effective exchange of data semantics (i.e., information).
- ***Myth 2:*** Uniquely named data elements will be used consistently by everybody to mean the same thing.
- ***Myth 3:*** Uniquely named data elements can exist (uniquely named as opposed to uniquely identified data elements – see note on namespaces and URIs below).

Many will readily acknowledge that these are, in fact, myths and that they don't really hold these assumptions. However, it seems that users of namespaces and developers of SDVs and metadata registries are pursuing their work as if these assumptions were true. No mechanisms or strategies have appeared in the extant literature that acknowledge, explain, or address the challenges that arise due to these faulty assumptions.

The reasons that these assumptions are faulty fall into the following three areas of SDV development and use:

- Scope
- Natural language use
- Schema evolution

Scope

Scope of control is a principal issue making these assumptions myths as opposed to realities. The assumptions can be made true with the proper oversight, such as the assignment of a data steward with the authority and scope of control to police the uniqueness of the names and the semantics of their use. However, the scope of control of such a steward is necessarily limited. There will never be a data steward with a global reach. In fact, it's unlikely that a data steward with these responsibilities can be effective (or even exist) in any but small data-intensive enterprises.

Standards development organizations such as OASIS (www.oasis-open.org) and the use of registries (e.g., www.xml.org) will not avert this problem. Not only will there be competing organizations developing and registering their own SDVs (e.g., OAGIS Canonical Model [www.openapplication.org], the ebXML Core Components [www.ebxml.org/specs], the IEEE SUO [suo.ieee.org], and the unrelated *xxx*MLS]), but many organizations won't know about, or even think to search out, registries or SDVs – they will naturally assume an insular view of the problem they're trying to solve and develop their own vocabularies for their own use. In addition, most newcomers to the field of standards have no idea how difficult it is to achieve consensus on meaning (let alone the set of terms) of the elements in a vocabulary (an observation originating from almost 20 years in the field of data standards development).

There will be no authoritative agent with the global scope of control to enforce constraints on the naming and use of data elements and vocabularies. Vocabulary and registry development efforts must recognize and accept that independent vocabulary "namespaces" (i.e., scopes or domains) will arise in the world, which is why Myth #3 is a myth: uniquely named data elements will not exist.

Note: The use of URIs ensures the uniqueness of designations across the World Wide Web, but it has no effect on the purpose of unique names: to convey a conventional (widely accepted) meaning.

"Rules of engagement" are necessary for meeting and dealing with the semantics and vocabularies of the different namespaces. The first rule of engagement should be a semantic "Prime Directive": each namespace has a perfectly valid and legitimate right to, and ownership of, its own semantics and vocabulary. No standards organization has the right or ability to dictate how another organization chooses to use or name its data.

Natural language use

In natural languages, the meanings of words and sentences are conventions coupled with innate linguistic capabilities that have arisen over the course of human evolution. Among the linguistic capabilities is the use of extra-linguistic perceptions to interpret the correct meaning of uttered sounds (i.e., words, sentences); these extra-linguistic perceptions are typically referred to as the *context* of an utterance. Because of contextual differences, the same word is able to convey different meanings.

Although the innate linguistic capabilities are (presumably) the same across the human species, the *conventions* vary regionally and by usage community. Words acquire meaning by *convention*: the constant reinforcement of the meaning of a sound/word through repeated use in a language community is what gives a word its meaning. (And it is why there are different languages, and why languages evolve over time.) It's a mistake to assert that a dictionary defines a word in the sense of specifying its meaning – it doesn't. A dictionary documents the *conventional meaning* of a word.

There have been no arguments made and no evidence presented in the literature that the (human) assignment or (human) interpretation of the "meaning" of data elements/terms in an XML document or schema will be any different than the use of individual words in natural language. Quite the contrary: the fact that technical discussion of issues often devolves into philosophical arguments provides strong evidence that they are the same. Thus, the differences in context and in the conventions of different usage communities are the reasons that Myth #2 is a myth: humans will not assign/interpret data element semantics consistently. (This is also the reason that the EDI standard requires volumes of federal regulations to use/interpret them consistently.) This phenomenon is amplified by the fact that language communities arise independently and without knowledge of one another.

Schema evolution

Schemas are typically used not only as a data validation mechanism, but also as a mechanism – when coupled with natural language prose definitions – for specifying the semantics of the data. As the size, scope, and number of purposes (i.e., the objectives, missions, requirements) of a community increase, the schema must evolve in response: the size, scope, and number of purposes served by the schema will also increase. There are a number of competing and conflicting forces in schema evolution that are not easily reconciled, managed, or even recognized.

The evolutionary forces that prompt changes in business and business computing environments will require schemas to be changed frequently to adapt to the new environment or face becoming obsolete. Schemas are typically designed as if they are a static declaration that can be objectively completed, as if there were some criteria by which one could say that the schema is "done." This is a mistake because the performance of a schema always degrades over time because of changes in the usage environment. (This observation applies, of course, to disembodied schemas that are not bound to a particular application.)

In addition, as the scope of the schema grows (i.e., the number of things it has "do" or "be able to say"), the schema will have to accommodate a very large number of primitive concepts in order to remain semantically precise (unambiguous). This schema will evolve in one of two directions:

- It will grow in size, yielding a very large number of element types and "interpretation flags."
- It will become very "abstract," "neutral," or "generalized" to embrace the wide variety of semantics.

The first option is impractical for the very reasons that enterprise schemas (or global data dictionaries) are impractical. Not only is a huge monolithic schema difficult to understand and manage – it's also far too cumbersome to adaptively respond to changes in the usage environment. This is one of the primary reasons that Myth #1 is a myth: it is not possible to uniquely identify, name, and define all (or even enough) data elements necessary for effective information exchange for a large, many-purposed community.

The second option is problematic in a fuzzier way: an abstract, neutral schema is by necessity more ambiguous, thus defeating the need for semantic precision. Abstract terms with general definitions readily and validly admit varied and often incompatible uses. This is one of the primary reasons why Myth #2 is a myth.

Recommendations

The problems described here are not insurmountable, but they certainly challenge "standard data vocabulary for application interoperability" proponents to answer some hard questions or rethink their direction. The following are some thoughts and recommendations for the SDVs and registry development projects to consider as they define the problem they're solving and design solutions for it.

Islands, bridges, and the role of the data steward

The first principle that the SDV/development projects should accept and adopt is a recognition that there are multiple independent and autonomous "namespaces" that service different communities, and that it isn't possible to bring them all under the same managerial umbrella. Each community has the perfect right to define its own semantic requirements and solve them in whatever manner it sees most fit. Therefore, an important component of the project work should be devoted to the analysis and design of how "bridges" between information community "islands" can be designed and built. A bridge may be a simple equivalency mapping between data elements, or a complex trading partner agreement. The "rules of engagement" mentioned above include construction of bridges.

The role of the data steward, then, is not as a policing agent for data element names, definition, and use within a community, but rather as an ambassador and "bridge warden" empowered to represent and explain the semantic requirements of the community he represents. He is responsible for the construction of "his half" of the bridge when negotiating and forging relationships with other "islands."

Stability versus evolution of schemas

Schemas must adapt to and change with changing business needs. Therefore, the applications that use data elements within a community should be metadata-driven (i.e., actively use the schema) as far as possible. This will allow the application to adapt more quickly and easily to changes in the environment because a new, more suitable schema can be "plugged in" to the application. (The componentization and orchestration of Web services will also enhance the adaptability of the application.)

This adaptability is even more important when the construction of bridges is considered. Negotiating a bridge with another community often will necessitate rethinking and altering the local community's perspective on their schema, necessitating changes or tweaks. But the changes and tweaks are a positive behavior rather than negative: it is a self-corrective evolutionary action. It enables the local community to better align their applications for effective information interchange with the newly connected community, thus forging a large, better-integrated community.

Although almost all schemas will need to change over time, the stability of a schema is important for many applications (and some usage communities). There are conditions under which a schema is stable over time; empirically, a schema is most stable when it:

- Is small in size
- Has a small scope
- Services a small usage community (ideally just one person)
- Serves a single (or small number) of fine-grained purposes

Schemas that reflect these properties are also the least ambiguous schemas. Ironically, abstract schemas are also fairly stable over time because they have more "semantic play" – they are semantically more forgiving, looser, or more malleable with respect to semantics. However, semantic play means that such schemas are also ambiguous.

Ideally, schemas that reflect the stability properties above can be bridged to a larger, more widely scoped community schema, thus enabling more effective (i.e., less ambiguous) exchange of information because the links to the less ambiguous schemas are maintained. And although the local schemas must still be adaptable in the face of change, the use of bridges also isolates islands from the ripple effect of impacts of changes to other schemas and other bridges.

Standard data element semantics

Despite all this, it's still possible to envision "standard" data elements that everyone uses the same way regardless of context, application, or schema. These standard data elements are representations of concepts that are ubiquitous throughout human endeavors. The notions of a person and a person's name, a point in time, and a location on the planet are universal concepts (or at least universal within the sphere of human experience), and the adoption of standard data elements with conventional and widely understood semantics is not only feasible, but likely. The ebXML Core Components are targeting the definition of ubiquitous concepts like these.

It's also possible to define standard data elements for small, finely purposed domains. The Dublin Core Metadata Initiative (www.dublincore.org) has defined a set of 15 standard data elements for tagging Web resources for the purpose of cataloging and search (much like a global, searchable card catalog). These elements have been widely adopted and used by libraries to make their collection catalog information available on the World Wide Web.

It should be recognized, though, that these concepts and finely purposed domains are relatively few in number when compared to the infinite variety of information that humans might want to exchange between applications.

Conclusion

The purpose of this article hasn't been to argue that the problems and the challenges that face the SDV/registry development projects are unsolvable. Rather, it is to suggest that the solution vision must be more expansive. Faulty assumptions must be rooted out, and the problems that are thereby exposed must be explicitly and directly addressed. Despite their intuitive appeal, namespaces, SDVs, registries, and unique data element names will not solve the problem of interoperability. What's needed is the recognition that the semantics of a schema (or, more precisely, the semantics of data governed by a schema) must be explicitly bound to a known community that it serves, and that bridges between the communities will be an inevitable part of any comprehensive solution. When the semantics are left locally bound to a community, the whole issue of "standard" semantics becomes moot.

References

- GAO, "Challenges to the Effective Adoption of the Extensible Markup Language Report to the Chairman, Committee on Governmental Affairs, U.S. Senate," General Accounting Office, Washington, D.C. GAO-02-327, April 2002.
- Knox, R., "GAO Report on XML Adoption Challenges Breeds Confusion," Gartner, Inc., Research Note/Commentary COM-17-0045, July 11, 2002.
- Perry, W., "Standard Data Vocabularies Unquestionably Harmful," 2002. www.xml.com/lppt/a/2002/05/29/perry.html

Author Bio

William C. Burkett is a senior information systems engineer specializing in the design and development of data-level application interoperability solutions. He has led teams that have designed interoperability software solution components, such as an XML Registry/Repository, using established engineering and design engineering principles and practices. William has been a leader in the international data exchange standards development community, and his papers on XML semantics and PDML have been presented/published at XML conferences and in refereed technical journals.

WBURKETT@PDIT.COM

WRITTEN BY **CHANDRU BOLAKI**

Intelligent XML Switching

Streamlining global customer support

Many of the world's largest telecommunications service providers rely on CommWorks Corporation, a 3Com company, to build their Internet protocol (IP)-based multiservice networks. Providing responsive, reliable, and secure customer support across continents, languages, and time zones was a growing challenge, but one CommWorks needed to meet in order to exceed its strategic goal of differentiation through superior, personalized customer support.

CommWorks customers demand a high level of customer support and are accustomed to sophisticated technology integration. The company's success in penetrating this elite group of service providers – as well as a host of other service provider customers – placed ever-increasing demands on its customer support organization, which relied on many complex, manual and custom processes to provide the real-time, proactive communications needed.

When detecting a network problem, a CommWorks customer typically opened a trouble ticket either by calling a customer support engineer or via a secure extranet. A service engineer troubleshot the problem by collecting pertinent data and consulting with internal resources and customer support personnel. E-mail or phone updates were required for customers, sales and field service personnel, and CommWorks management, and updates were posted to the extranet. Depending on the severity of the network problem, such manually generated updates could consume a significant amount of the engineer's time.

Recognizing the potential of XML for information exchange in customer support and customer relationship management, CommWorks recently integrated Sarvega XPE Switches into its network to create a real-time information-aware network that can intelligently, securely, and reliably route critical customer and field service information to the right people.

Efficiency and Accuracy Through Automation

It was clear that manual processes and one-up custom responses would eventually deteriorate the company's ability to effectively service its customers. The ideal solution would unify voice and data content delivery on the same platform; deliver information to any system or device across the globe; package and deliver information as actionable bidirectional transactions; be based on open standards such as XML and leverage known and tested communication standards such as HTTP; and be easy to deploy and manage.

CommWorks' system, though reasonably reliable, could not guarantee real-time updates and was labor intensive for support engineers. We were convinced that the universality of XML would provide a strong basis upon which to build a viable, scalable solution.

CommWorks' technical staff considered the alternative: a combination software/hardware solution customized to the company's specific needs and working in conjunction with its Clarify system. While such a system could indeed provide much of the functionality needed, it would be costly not only to deploy initially, but to enhance and upgrade. And it would lock CommWorks into a specific software solution that might not be readily compatible with newer versions of software the company used or that might lead to prohibitive engineering costs in the future.

Another option was to turn this critical function over to an application service provider who would provide messaging services. While this may have been cost-effective, the nature of the communications with customers and within the organization was simply too sensitive to be entrusted to an outside party.

CommWorks eventually decided that intelligent XML switching could provide an efficient and cost-effective solution. The company selected Sarvega's XPE Switches, which provide a scalable and feature-rich platform that could meet short-term as well as longer-term requirements. (See sidebar on XML switching.)

XML switching was particularly appropriate for CommWorks' customer support application because it easily integrates with existing applications, intelligently routes information generated within it to designated recipients around the world in multiple formats to multiple applications in a secure manner, and requires minimal maintenance. Existing solutions left the "last-mile" of automation – getting information to the right people at the right time over the right delivery mechanism – to manual processes. An automated solution has to interface with multiple protocols with varying latency, failure recognition, and reliability issues in an integrated way. For example, if a field support employee could not be contacted, an escalation mechanism would automatically be triggered.

Now, using Sarvega's XPE Switches, critical customer and field service information, including top-priority cases, is quickly, intelligently, and reliably routed worldwide. Based on definable options, the XPE can intelligently route customer support information and updates to the right system, Web service, or device across wireless, telephony, or wired access networks. For example, personalized updates on high-severity customer support cases are routed automatically to field support people, to appropriate customer support personnel, to the support Web site, and to CommWorks' support managers as they occur – reducing response time and ensuring that the information reaches its intended recipient. Sarvega's differential security ensures that the most sensitive information is securely encrypted before being transmitted.

The ability of the solution to ensure that the information reaches the intended recipient in the way he or she wants to receive it is one of the greatest benefits. Some people require phone calls, and some get all information via e-mail – and we will be adding paging along with an audit feature in the future. Programming this system with those requirements is easy and it has worked virtually flawlessly.

Integrating the Solution

Designed and built as an appliance with easy-to-use interfaces based on XML and LDAP, Sarvega's XPE Switch was easy to deploy within CommWorks' existing environment. Leveraging existing application-level and network-level standards, the Sarvega XPE was nonintrusively deployed within weeks to provide an intelligent, information-aware network that works in tandem with CommWorks' existing infrastructure.

As customer support representatives receive information about field issues from around the world, the Clarify CRM System is updated, automatically triggering a series of events. The Sarvega XPE in the CommWorks' data center receives XML-based trouble tickets over HTTP, performs data integrity and authentication checks, applies classification rules, transforms the content as required, and routes it to any desired endpoint using a multitude of delivery protocols. It is also highly maintainable, providing an easy-to-use management console for configuration flexibility, upgrade, and maintenance.

Identifiable Business Results

CommWorks will realize a compelling return on investment through reduced manual processes, lowered operating costs, and support in meeting its primary goal of delivering exceptional customer service. This robust, real-time infrastructure streamlines the CommWorks' customer support organization, which in turn improves customer satisfaction.

The intelligent XML switches reduce operational costs by providing a foolproof way to automate manual processes. Using the XPE switches, CommWorks has greatly improved internal processing. Replacing e-mail and manual phone calls with customizable XML message delivery provides systematic reliability to help easily overcome geographic and time-delay issues. And more support engineering time is spent on higher-value work that directly benefits the company's customers.

CommWorks customers crave real-time information. With the new intelligent XML switching system, they not only get it, they get it precisely how they want it. And that's what superior customer service is all about.

For more information about Sarvega information-aware switching, visit www.sarvega.com. For more information on CommWorks, visit www.commworks.com

CHANDRU_BOLAKI@3COM.COM

AUTHOR BIO

Chandru Bolaki is the technical director of the Asia/Pacific Technology Center of CommWorks Corporation, a 3Com company (www.commworks.com). Chandru directs technical strategy and plans for projects in the center as well as the execution of these projects with the different teams in the CommWorks and 3Com organization. He holds an MBA from Oklahoma State University, an MS in computer science from Iowa State University, and a B.Com from St. Joseph's Commerce College in Bangalore.

Intelligent XML Switching

As enterprises increase their use of XML for applications such as Web services, it will be essential to have better infrastructure that can handle the complexities inherent in XML. The concept of XML switching addresses this problem – meeting the key requirements of transformation, processing, and security that enterprises need to make their infrastructure "information aware."
—Greg Howard, HTRC Group

While XML is rapidly becoming the lingua franca of business information exchange, the transfer, parsing, validation, transformation, and switching of XML data can severely tax enterprise systems and servers. And, while enterprise applications provide some measure of intelligence in prioritization of information based on its value, networks equalize it during transport because they are unable to differentiate between high-value business transaction traffic and other nonessential traffic. Intelligent XML switching is emerging as a powerful and cost-effective way to facilitate secure, meaningful information exchange in real time.

Business-critical information must be delivered from a variety of applications (CRM, supply chain, ERP, B2B, etc.) to a variety of applications, services, and devices/communication networks. Each of these endpoints differs in the underlying network protocols, capabilities in presenting and processing data, usage, and features – making it a costly and difficult task to create systems that can format, transact, track, and manage communication with the entire range of endpoints.

Intelligent XML switches such as the Sarvega XPE Switch used by CommWorks Corporation, a 3Com company, are designed to intelligently package and deliver information to a multitude of applications, services, networks, and devices across IP, telephony, and wireless networks. Such appliances typically reside in the enterprise network between the applications generating the information and the gateways to the applications, services, networks, and devices to which they need to deliver information. Originating applications simply connect to the switch using their own native version of XML, and Sarvega XPE uses business rules to intelligently package, secure, and deliver the information to multiple endpoints spanning application and communication networks:

- Taking data from different content sources, including databases, Web content sources, and enterprise applications
- Interfacing with enterprise directory systems to extract routing preferences to package the content (as part of the input XML stream or through configured subscriptions)
- Applying differential security, modulated based on the meaning and value of the information
- Translating and accelerating delivery of this information to the various endpoints accessible over HTTP (including SOAP-based systems), SMTP, voice, WAP, page and SMS
- Supporting bidirectional interaction regardless of application, service, or device
- Converting the response back into the appropriate form of XML and returning it to the originating content source.

Because it uses a flexible open standards–based platform, the switch can automatically adapt to various applications, protocols, device characteristics, and addressing schemes to provide information translation and routing that is highly reliable and scalable. Its XML schema–agnostic universal translator dynamically understands XML dialects, enabling intelligent XML transformation, while rich routing features ensure that critical information is routed and delivered reliably. Its hardware-based architecture drives ease of deployment; it deploys in 6–8 weeks to provide a uniform infrastructure that can deliver both data- and voice-based information. This provides smarter, faster, and secure infrastructure that leverages the true potential of XML while overcoming its limitations.

WRITTEN BY **ROY C. HOOBLER**

Working with Xindice

Moving to an XML Database

In my free time, I've been working on a CMS/Portal application using Java and XML. I was glad to discover some XML database tools that are now available – as more and more data is being stored and transmitted in XML format, XML databases are worth considering. Moving an XML application to Xindice (pronounced zin-dee-chay) is an interesting experience.

Xindice is a "new" open-source database engine, so a lot of issues must be resolved by brute force. However, getting started using the Java API was fairly easy, and getting a test class put together only takes an hour or two. Xindice actually has an HTTP server built in that runs on port 4080. However, the query mechanism over HTTP didn't work on my machine, so I started to build my own servlet that could search documents stored in Xindice.

Why Switch to Xindice?

After I spent some time building the portal, the documents were getting out of hand; going to the next level of functionality called for a more robust engine for delivering content. XML documents for Web content, posted articles, events, and catalogs would soon become unmanageable. Putting this information in a relational database would solve most of the problems, but a relational database would be a lot of overhead and would limit the user's platform (since everyone has their own favorite flavor of SQL relational database). In an earlier article, I mentioned searching XML with XSLT (***XML-J*** Vol. 3, issue 7). The methods described in that article could handle 30 or 50 documents, but what if a site needed to scale to hundreds or thousands of documents? Using a server such as Xindice facilitates the storage of documents and provides indexing as well as retrieval mechanisms. Xindice is written in Java, so it's platform independent as well.

Working with Xindice requires setting it up and building a Java package that includes a few classes to query the database. I also built a servlet class so the database could be queried over HTTP. This is new technology, so be sure to test and design well before deciding whether it's the right solution.

Setting Up Xindice

Setting up Xindice is pretty straightforward. Java 1.3 or later must be installed and the Xindice_Home variable must also be set to run the software.

Documents are stored in collections on the server. For my articles, I created an "articles" collection using the xindiceadmin tool: ./bin/xindiceadmin add_collection -c /db -n articles.

Collections are a repository for XML documents. The fact that collections can contain other collections separates them from relational models. The documents database itself contains a human-readable hierarchy. For now, all my articles are placed in an "articles" collection, but I could (and probably will) separate this into "articles/xml", "articles/msdotnet", and "articles/java" collections.

To begin with you may want to add a few documents using the command-line utility. Documents are stored in Xindice in a special format to which indexes can be added to increase performance. From the command line, adding a document is simple: xindice add_document-c/db/articles-ffx102.xml -n SSL1.

The -n parameter is the unique key for the document. Even though it's an optional parameter, it's a good idea to supply your own. Later, if you build a mechanism to retrieve a single document, looking it up by the generated key (i.e., 0625df60001a5d4000bc49d00060bf5) won't be very convenient. SSL1 or SSL09-2002 is a lot easier to type in and retrieve later. Beware: Xindice currently allows you to store documents with duplicate keys. Querying will return the results from both documents, but you'll only be allowed to retrieve the first document added.

Before diving into the Java classes, test to make sure everything is running well through the command-line interface. One note about the command line: double quotes need to be placed around XPath expressions. This is usually not the case, and it's not the case when using the Java classes. Use bin/xindice retrieve_document -c /db/articles -f fx102.xml -n SSL1 or bin/xindice xpath -c /db/articles -q /article["contains(title, 'SSL')"]/title.

The documentation that comes with Xindice explains everything. Check the User Guide, Developer Guide, and Administrator Guide – the information you need may be in any one of these guides (you can find the documentation at http://localhost:4080).

Creating a Java Class to Search DocBook Articles.

To implement an XML database, I divided the project into four classes. The xmlQuery class that handles calls to Xindice, the docBookSearch class that handles creating XPath queries specific to DocBook XML, a "parameters" class representing the search parameters needed for the DocBook searches, and finally, a dbSearchServlet class that takes the parameters from an HTTP request and passes them to the docBookSearch class.

The xmlQuery class shown in Listing 1 is based on the examples provided with Xindice. The terminology isn't the same as that of a relational database class, but there are enough similarities that this type of connecting and querying should be very familiar to most developers. Instead of connections, tables, and queries; managers, collections, and services provide the interface to get data from the system. The concept of a collection may be more familiar to developers who have previously worked with content management tools.

In the xmlQuery class, the getResults method builds a nonvalid XMLdocument as a string from the Results object's getContent() method. In the future these results may be appended to a valid XML document or sent to the client as a stream of results. Xindice returns a complete XML document with an XML declaration, so the first line (the XML declaration) is stripped from the results. Since Xindice is open source, the source code can also be modified to return result streams as one document. The JavaDoc documentation, which explains other classes and other methods to use, is also included with the distribution.

The second class, searchParams.java (see Listing 2), encapsulates the parameters needed to search for documents. At first the package used parameter list, but after it grew to about five parameters, switching to a class made more sense. To keep things simple, I created a public class with six public fields. For a real implementation, more than one class may be involved, using either a bean model or something different.

The third class, docBookSearch, does most of the work (see Listing 3), creating a valid XPath query for searching documents from the parameters passed in from the servlet. The best thing about this class is that it can be tested from a standard Java class with a "Main" method and, after testing, called from the servlet class below. The logic in this class needs to be refined and will become more complex to handle different types of queries and produce different results.

The fourth class (see Listing 4), dbSearchServlet, is the the servlet itself. The following code retrieves the results from the DocBookSearch class and passes them to the browser:

```
response.getOutputStream().println(DocBookSearch.getSearchResults(sp));
```

This class could function with less code, but some parameter checking has been included. After compiling the class, it should be in the ./WEB-INF/classes directory. To make things easier, I put everything into a package, added it to the ./WEB-INF/lib directory, and then added the code shown in Listing 5 to the ./WEB-INF/web.xml file:

Finishing Up

Using the new com.sonoma.xmldb package, the results can be accessed from the URL: http://localhost:8080/examples/xSearch?doc=article&searchString=SSL&searchNode=title&fullText=true&titlesOnly=true.

The goal, however, is to make the "searchString" parameter the only required parameter. Another servlet class will be "getArticle", which will take an article key as a parameter from the example given earlier, an entire article will be accessible by the URL: http://localhost:8080/examples/xArticle?SSL1.

On the portal, the search will be invoked in an XSLT document. This is another advantage of having an XML database incorporated with a servlet. In a previous article I wrote about searching through one XML document. With Xindice, implementing a variety of searches will be much simpler. The XSLT code looks something like Listing 6.

Of course, the search terms will be replaced by parameters or variables. The database can also be queried by other Web applications and programs. A few more parameters could produce results in RSS (Rich Site Summary) format as well; an XML-based application provides a lot of possibilities and a very open system. With XML architecture, the same type of system could be achieved. With Xindice as a datastore, the problem of storing and querying large amounts of XML documents is solved, enabling more XML-driven applications.

References

- *The Sonoma Project (XML-based portal)*:http://sonomap.sourceforge.net
- *Apache Xindice:* http://xml.apache.org/xindice

Other XML Database Products
- *XML DBMS:* www.rpbourret.com/xmldbms/index.htm
- *eXist:* http://exist.sourceforge.net

R_HOOBLER@HOTMAIL.COM

LISTING 1 XMLQuery class

```
package com.sonoma.xmldb;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
import org.apache.xindice.client.xmldb.services.*;
import org.apache.xindice.xml.dom.*;
import java.io.File;
import java.io.FileInputStream;
public class xmlQuery {
     /** Creates new xmlQuery */
     public xmlQuery(){
     }
     public String getResults(String DBCollection, String
        XPathQuery) throws Exception{
       Collection col = null;
       StringBuffer sb = new StringBuffer();
       try {
          String driver =
       "org.apache.xindice.client.xmldb.DatabaseImpl";
          Class c = Class.forName(driver);
          Database database = (Database) c.newInstance();
          DatabaseManager.registerDatabase(database);
          col = DatabaseManager.getCollection(DBCollection);
          String xpath;
          xpath = XPathQuery;
          System.out.println("query: " + xpath);
          XPathQueryService service =
             (XPathQueryService)
    col.getService("XPathQueryService", "1.0");
          ResourceSet resultSet = service.query(xpath);
          ResourceIterator results = resultSet.getIterator();
          if (results.hasMoreResources()){
             sb.append("<results>");
             String sResults = "";
              while (results.hasMoreResources()) {
                 Resource res = results.nextResource();
                 sResults = (String) res.getContent();
                 if(sResults.indexOf('\n')>-1){
                     sResults =sResults.substring(sResults.
          indexOf('\n')+1);
                 }
                 sb.append(sResults);
```

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

AUTHOR BIO

Roy Hoobler has been developing custom Web applications since 1996. After completing his MCSD certification, he spent the mid-'90s at a large consulting firm focused on intranet/extranet applications for Fortune 1000 companies. In 1998 Roy joined Net@Work (www.netatwork.com) as director of Internet technologies, specializing in systems architecture, project management, and research in emerging programming methods.

```
                }
                sb.append("</results>");
            }
        }
        catch (XMLDBException e) {
            System.err.println("XML:DB Exception occurred " +
e.getMessage());
        }
        finally {
            if (col != null) {
                col.close();
            }
           return sb.toString();
        }
      }

      public void addCollection(String collectionName) throws
       Exception{
        Collection col = null;
        try {
            String driver = "org.apache.xindice.client.
       xmldb.DatabaseImpl";
            Class c = Class.forName(driver);

            Database database = (Database) c.newInstance();
            DatabaseManager.registerDatabase(database);
            col =

DatabaseManager.getCollection("xmldb:xindice:///db/");

            //String collectionName = "mycollection";
            CollectionManager service =
                (CollectionManager) col.getService("Collection
       Manager", "1.0");

            // Build up the Collection XML configuration.
            String collectionConfig =
                "<collection compressed=\"true\" name=\"" +
       collectionName + "\">" +
                "    <filer
class=\"org.apache.xindice.core.filer.BTreeFiler\"
gzip=\"true\"/>" +
                "</collection>";

            service.createCollection(collectionName,
                  DOMParser.toDocument(collectionConfig));

            System.out.println("Collection " + collectionName
+ " created.");
        }
        catch (XMLDBException e) {
            System.err.println("XML:DB Exception occured " +
       e.getMessage());
        }
        finally {
            if (col != null) {
                col.close();
            }
        }

      }
      public void addDocument(String CollectionURI, String
       FileName)
          throws XMLDBException{
        Collection col = null;
        try {
            String driver =
"org.apache.xindice.client.xmldb.DatabaseImpl";
            Class c = Class.forName(driver);

            Database database = (Database) c.newInstance();
            DatabaseManager.registerDatabase(database);
            col =
                DatabaseManager.getCollection(CollectionURI);
            String data = readFileFromDisk(FileName);

            XMLResource document = (XMLResource) col.create
       Resource(null,
                  "XMLResource");
            document.setContent(data);
            col.storeResource(document);
            System.out.println("Document " + FileName + "
       inserted");
        }
        catch(Exception e){
            e.printStackTrace();
        }
        finally {
            if (col != null) {
                col.close();
            }
        }
      }
      private String readFileFromDisk(String fileName) throws
        Exception {
        File file = new File(fileName);
        FileInputStream insr = new FileInputStream(file);

        byte[] fileBuffer = new byte[(int)file.length()];

        insr.read(fileBuffer);
        insr.close();

        return new String(fileBuffer);
      }
}
```

LISTING 2 (searchParams.java)

```
package com.sonoma.xmldb;
public class searchParams {
      public String DocType = "article";
      public String SearchNode = "title";
      public String SearchText;
      public String Collection = "xmldb:xindice:///db/
       addressbook";
      public boolean FullText = false;
      public boolean TitlesOnly = true;

      public searchParams() {
      }
}
```

LISTING 3 (docBookSearch.java)

```
package com.sonoma.xmldb;
public final class docBookSearch {
      /** Creates new DocBookSearch */
      public static String getSearchResults(searchParams
       params){
          String sPath = "";
          sPath = "/" + params.DocType;
          if (params.Collection==null){
              params.Collection ="xmldb:xindice:///db/arti
       cles";
          }
          if (params.SearchText == null || params.Search
       Text.length()<1){
                return "parameter searchText cannot be empty";
```

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

```
        }
        if(params.FullText){
            sPath += "[contains("+ params.SearchNode +
      ",'"+params.SearchText+"')]";
        }else{
            sPath += "[decendant::"+ params.SearchNode +
      "='"+params.SearchText+"']";
        }
        if(params.TitlesOnly){
            sPath += "/title";
        }else{
            sPath += "/" + params.SearchNode;
        }
        try{
            xmlQuery xq = new xmlQuery();
            return xq.getResults(params.Collection,sPath);
        }catch(Exception e){
            return "<DBError>DocBookSearch: " +
e.getMessage() + "</DBERROR>";
        }
    }
}
```

LISTING 4 (dbSearchServlet.java)

```
package com.sonoma.xmldb;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
public class dbSearchServlet extends HttpServlet{

    /** Creates new docbookSearch */
    public dbSearchServlet() {
    }
    //request
searchString=someString&searchNode=node&fullText=false&titlesOn
     ly=true&doc=book|article
    public void doGet(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException{
        //basic errors
        if (checkParameters(request,response)){
        try{
            searchParams sp = new searchParams();
            String sQuery = "";
            sp.SearchText =
request.getParameter("searchString");
            sp.DocType = request.getParameter("doc");
            sp.SearchNode =
request.getParameter("searchNode");
            if (request.getParameter("fullText") != null &&
      request.getParameter("fullText").startsWith("t")){
                sp.FullText = true;
            }else{
                sp.FullText = false;
             }
             if (request.getParameter("titlesOnly") !=
null && request.getParameter("titlesOnly").startsWith("t")){
                sp.TitlesOnly = true;
             }else{
                sp.TitlesOnly = false;
             }

response.getOutputStream().println(DocBookSearch.getSearchRe
      sults(sp));

        }catch(Exception e){

response.getOutputStream().println("<DBError>" +
      e.getMessage() + "</DBError>");
        }
        }
    }


    private void returnError(HttpServletResponse response,
     String message){
        try{

response.getOutputStream().println("<DBError>" + message +
     "</DBError>");
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    private boolean checkParameters(HttpServletRequest
     request, HttpServletResponse response){
        if ((request.getParameter("doc") == null) ||
(!request.getParameter("doc").equals("book") &&
!request.getParameter("doc").equals("article"))){
            returnError(response,"parameter doc must be
     article or book");
            return false;
        }
        if (request.getParameter("searchString") == null
      || request.getParameter("searchString").length()<1){
            returnError(response,"parameter searchString
     cannot be empty");
            return false;
        }
        return true;
    }
}
```

LISTING 5 Add to ./WEB-INF/wdo-xml file

```
    <servlet>
        <servlet-name>
            xmlSearch
        </servlet-name>
        <servlet-class>
            com.sonoma.xmldb.dbSearchServlet
        </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>
            xmlSearch
        </servlet-name>
        <url-pattern>
            /xSearch
        </url-pattern>
    </servlet-mapping>
```

LISTING 6 XSLT Code

```
  <xsl:template name="showArticleSearch">
      <table width="100%">
       <tr>
 <td>Xindice Search Results<br/>
 <xsl:variable name="search"
select="document('http://localhost:8080/examples/xSearch?doc
     =article&amp;

searchString=SSL&amp;searchNode=title&amp;fullText=true&amp;
     titlesOnly=true')"/>
  <xsl:apply-templates select="search/results"/>
 </td>
 </tr>
      </table>
</xsl:template>
```

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

WRITTEN BY **MIKE JASNOWSKI**

Writing a Native XML Database Application

Using XSLT and XUpdate to modify data

Imagine this scenario: you've just been contracted to write a database application with a Web front end. The application is multitiered and uses a database, but the interesting thing is, it's a native XML database. Would you have an idea of where to start or what options you had to work with? The good news is that an XML database doesn't automatically mean you're restricted to working with XML directly.

This article, which assumes familiarity with XML, is designed to give you a starting point and information for writing database applications for native XML databases. I'll begin with a short comparison of writing applications for native XML databases versus relational databases. If you've worked with XML at all, you'll find many of these concepts familiar.

Writing database applications for a native XML database is not entirely different than writing for a relational database, but the tools and techniques are different.

Programming Relational vs Native XML

Any serious enterprise application that uses persistent data needs an industrial-strength DBMS to manage the data. There has been and continues to be a logical progression toward NXDBs that stems from this need. You can only do so much with data stored in the file system; data needs to be managed, secured, and available with near in-memory speeds. XML data currently stored in relational databases is often stored as either a BLOB or as a series of related tables. An NXDB, in short, is a database that stores data structured as XML in its native format – the Document Object Model (DOM). Unlike storing it in a flat file, which then must be parsed, or a relational database, which must also be parsed (possibly from several tables) and then constructed into a DOM object, an NXDB stores the document in a parsed object that can then be accessed using a number of techniques.

With database application development, several steps are generally followed to write the application. In a simplified description, these steps are:

1. Connect to the database
2. Read or update some data
3. Disconnect from the database

These steps are generally accomplished directly or through the use of some persistence framework, or by using another API. These steps also leave out many details about things such as use of connection pools and transactions, both of which are critical for use in enterprise database applications.

Depending on the platforms you're developing for, you may use Java, C++, Visual Basic, or even Perl for your programming language. You probably also use some standard API such as ODBC or JDBC, which define standard interfaces and classes for accessing relational databases. But what options do you have with XML databases? While a standard API for XML database access doesn't exist, there are numerous XML-related standards and specifications. An effort to provide a standard API for XML database access is underway at www.xmldb.org.

Connecting to and disconnecting from an XML database is generally done using some means specific to the XML database vendor, unlike ODBC or JDBC, which generally utilize some type of connection string. Reading and writing data from an XML database are conceptually the same, but XML databases use standards such as DOM or JDOM to provide a data model for XML documents. Unfortunately, DOM and JDOM don't provide a meaningful representation of the data represented by the XML document. You can use XML binding to work with XML documents in an object format. For example, if you have an XML document representing an employee, XML binding enables you to work with this XML document using an Employee object. XML binding provides the same functionality as persistence frameworks such as Toplink, which provides object-relational mapping capabilities for relational databases.

You've probably also utilized SQL for performing queries to read data and update and delete records from relational databases. As previously mentioned, XML databases have numerous standards for working with XML documents. You can use XSLT to query and update XML. You can use XPath in conjunction with the aforementioned to identify elements to work with in an XML document. You can use XQuery to perform queries as well.

Using XSLT to work with XML data does present some challenges; often, you must write a set of unrelated templates that are executed in succession to update the data. Another option for performing updates to XML data is to use the XUpdate update language.

XUpdate

The XUpdate specification defines an XML grammar for specifying modifications to an XML document. Not all XML databases support XUpdate natively, and there aren't many freely available reference implementations yet. This article will show you how to use XSLT to provide processing for XUpdate commands and instructions.

The basic processing of this application will involve generating XUpdate documents that will then be processed by an XSLT stylesheet. The XSLT stylesheet will then be used to transform the input document (the one containing the data). The transform can take the form of updating elements and attributes as well as inserting new elements. A sample XUpdate document might look like this:

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
xmlns:xupdate="http://www.xmldb.org/x
update">
  <xupdate:insert-after
select="/name[last()]">
    <xupdate:element name="sir-
name"/>
  </xupdate:insert-after>
</xupdate:modifications>
```

This particular example would insert an empty <sir-name> element after the last <name> element under the root node of the document. The basic idea behind XUpdate is to provide a way to perform changes to an XML document using meaningful syntax. While XSLT can provide the same functionality as XUpdate, it doesn't use element names or syntax that might be easier to understand.

The stylesheet that processes the XUpdate document works by generating <xsl:templates> that are then used to process elements of the input XML document. For example, take the following XUpdate document:

```
<?xml version="1.0" encoding="UTF-
8"?>
<xupdate:modifications version="1.0"
xmlns:xupdate="http://www.xmldb.org/x
update">
   <xupdate:update select="/address-
es/address[2]/fullname">Michael
Jasnowski</xupdate:update>
   <xupdate:update select="/address-
es/address[2]/town">Amherst</xup-
date:update>
</xupdate:modifications>
```

This XUpdate document indicates that the <fullname> and <town> elements should be updated to the specified values for the second <address> element. When run through the XSLT stylesheet it will produce another XSLT stylesheet that looks like Listing 1.

This XSLT stylesheet is then run in a transformation against the input XML document representing the data in our application, resulting in the updates. Each of the templates in the stylesheet is run in succession; the second and third templates are the ones that update the <fullname> and <town> elements. Figure 1 illustrates this process.

The entire XUpdate specification is not supported in this stylesheet; only the following tags are supported:

```
<xupdate:insert-after>
<xupdate:insert-before>
<xupdate:update>
<xupdate:remove>
```

There is also one caveat to the implementation of the <xupdate:update> tag: if you update an element with text, the attributes are lost. However, feel free to implement support for the other tags.

Filename	Description
testdoc.xml	Document containing application data
main.xsl	Stylesheet that presents list of entries in address book
add.xsl	Stylesheet that presents an HTML form for adding an entry
update.xsl	Stylesheet that presents an HTML form for updating an entry
xupdate.xsl	Stylesheet that transforms XUpdate document to XSLT
hello.jsp	JSP that presents entries in address book
add.jsp	JSP that adds an entry to the address book
update.jsp	JSP that updates an entry in the address book
delete.jsp	JSP that removes an entry in the address book

TABLE 1 | Files for sample application

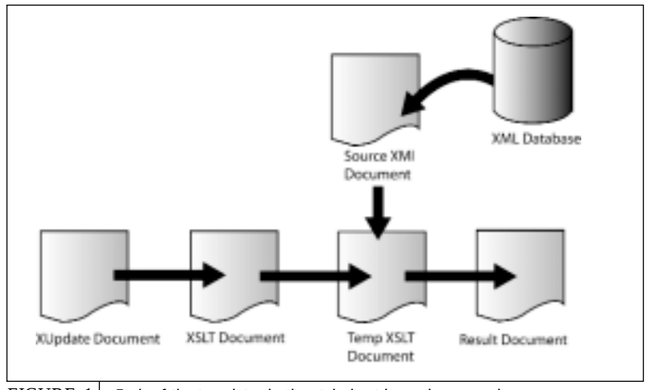


FIGURE 1 | Each of the templates in the stylesheet is run in succession

The Application: Address Book

The application we'll look at is a simple address book that enables you to add, update, and remove entries from it. The application will make use of the XUpdate specification to provide update capability to the application. What if the database you're using doesn't support XUpdate? No problem. We're going to write an XSLT stylesheet that can process the XML-based XUpdate documents and generate XSLT that can then be used to update an XML document. There are a number of implementations of XUpdate in Java that can be downloaded, and some NXDB vendors have gone as far as providing built-in support for XUpdate.

The application is run as a set of JavaServer Pages (JSP), which then access an XIS server located on the same physical machine. XIS can also be used in distributed environments for multitiered application architectures. The JSPs used to take input and invoke the Java API methods that will generate the XUpdate document run a transformation and produce the update. The JSPs mark the transaction boundaries, and the transformations occur within these boundaries. When the transaction commits, your XML data has been updated.

AUTHOR BIO
Mike Jasnowski, a Sun-certified Java programmer, has over 17 years of experience and over four years with Java. He's a senior software engineer at BEA Systems in Burlington, Massachusetts.

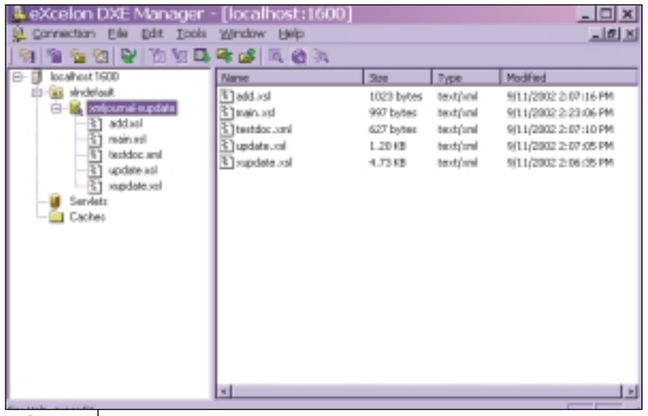


FIGURE 2 The DXE manager

Table 1 summarizes the files used in this application.

The XML database I've chosen for this application is XIS, which I've worked with for the past two years. You can obtain an evaluation copy of XIS from www.exln.com. I've also chosen to use the Java API that comes with XIS. You can also use the COM API from VC++ or VB, or even an ASP application. Working with XIS for this example entails the use of several interfaces, summarized in Table 2.

There are two basic modes of operation for this application. The first is running an XSLT transformation to present data, and the second is running an XUpdate against an XML document to modify it (which basically turns out to be another XSLT transform). If you've used XSLT to transform an XML document into HTML, then you're familiar with this step. To perform a transformation in XIS, use the Transform interface that is created using the createTransform() method of the Session interface. Pass a node or string representing an XSLT stylesheet; assuming no exceptions are thrown, you can then use one of the execute() methods of the Transform interface to perform the actual transformation.

The Transform interface can return a result representing the transformed document, or it can modify the document in place and return it. In this application I use both methods; the first is used when transforming the XML documents to HTML and the second when using XUpdate to transform the XML document.

Running the Sample Application

To run this sample application I used the J2EE SDK reference implementation as the servlet container and XIS as the database. In my particular installation, I had to change the default port of XIS (1050), which conflicts with the J2EE server. I changed the value of the com.exln.dxe.host property in the %XIS_INSTALL%/XIS/config/xlnserver.properties file to 1600 and restarted the server. You can do this from the NT Services applet. See the XIS documentation for further information on administering XIS.

I also needed to modify a couple of files in the J2EE reference implementation. I had to change the %J2EE_IN STALL%/bin/userconfig.bat script to include the dxeclient.jar and dxeserver. JAR library archives, which contain the API code mentioned in Table 2. Following is an excerpt of my userconfig.bat script:

```
rem set J2EE_CLASSPATH=
rem set JMX=f:/files/jmx-1_1-
ri/jmx_1.1_ri_bin/lib
set
J2EE_CLASSPATH=F:\progra~1\eXcelon\XI
S\classes\dxeclient.jar;F:\progra~1\e
Xcelon\XIS\classes\dxeserver.jar
```

Next, I had to modify the %J2EE_INSTALL%/lib/security/server.policy file so that the sample application would be able to load the necessary XIS runtime libraries, which happen to be native libraries. Below is an excerpt of my server.policy file:

```
// permissions for default domain
grant {

    permission
java.lang.RuntimePermission
"loadLibrary.*";
```

I packaged the application files hello, jsp, add.jsp, delete.jsp, and update. jsp into an application WAR and deployed it onto the server. I used the deploy tool that comes with the J2EE 1.3.1 SDK to perform the packaging. When packaging the application I associated an alias of /xis as well. You will also need to stage the XML documents and stylesheets that will be used by the application. To do this you can use the DXE Manager application that comes with XIS, which is an MFC windows application. Figure 2 shows the DXE Manager with files created for this application.

To load the data into XIS, you first need to create the XMLStore, which can be accomplished from the File > Create > XMLStore menu after selecting the xlndefault cache from the tree view. Name this XMLStore xmljournal-xupdate. You can then load the files from Table 1 (except the JSP files) by dragging them from Windows Explorer and dropping them on the newly created XMLStore node in the tree. When you add documents to an XMLStore, a menu appears that presents you with options about how you want to store the document. For this example, select "use server default setting," which means for XML, treat it as XML. After completing this step you're ready to run the application.

After starting the J2EE reference implementation server, run the application by requesting the following application in your Web browser: http://localhost:8000/xis/hello.jsp?document=testdoc.xml&stylesheet=main.xsl.

Upon running the application, you'll first be presented with a series of names; click on each name and update the fields of the entry. You can also delete entries by clicking on the "Delete" hyperlink after each name. New entries can be added by clicking on the Add link at the bottom of the list of names.

Interface or Class	Description
com.exln.dxe.Session	Used to represent a session with an XIS server
com.exln.dxe.XMLStore	Used to represent an XML database
com.exln.dxe.XlnTransaction	Used to represent a transaction
com.exln.dxe.engine.XlnLocalSessionFactory	Class used to create instances of a Session interface
com.exln.dxe.filesystem.XMLDocument	Used to represent an instance of an XML document
com.exln.dxe.Transform	Used to represent an object that can be used to transform an XML document

TABLE 2 Classes and interfaces used in simple application

Summary

You've seen how to write a native XML database application and use XUpdate and XSLT to modify an XML document stored in an XML database. XUpdate specifies an XML grammar for inserting, appending, and modifying elements, attributes, comments, and text contained in an XML document. XUpdate documents must be processed using either XSLT or some programming language component. Also note that the XSLT and XUpdate shown here should work with any XML database, whether it supports XUpdate or not, since ultimately the XUpdate is transformed into XSLT.

I've also included an EAR file of the application with WAR containing the source code for this example. This can be downloaded from www.sys-con.com/xml/sourcec.cfm.

Resources

- *XUpdate specification:* www.xmldb.org/xupdate
- *XSLT specification:* www.w3c.org/TR/xslt
- *XPath specification:* www.w3c.org/TR/xpath
- *Excelon XIS:* www.exln.com
- *J2EE SDK 1.3.1:* http://java.sun.com/j2ee

MJASNOWS@BEA.COM

LISTING 1

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
xmlns:xupdate="http://www.xmldb.org/xupdate"
xmlns:xsl="http://www.w3.org/1999/XSL/Transfor
m" version="1.0">

<xsl:template match="@*|node()">
   <xsl:copy>
     <xsl:apply-templates
select="@*|node()"/>
   </xsl:copy>
</xsl:template>

<xsl:template
match="/addresses/address[2]/fullname">
   <xsl:copy>Michael Jasnowski</xsl:copy>
</xsl:template>

<xsl:template match="/addresses/address
[2]/town">
   <xsl:copy>Amherst</xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

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

Using OAGIS for Integration

Enabling everywhere-to-everywhere integration

WRITTEN BY
MICHAEL ROWELL

The Open Applications Group Integration Specification (OAGIS) is implemented because it simply works – and it's available today. While it's not as "cool" as the Web services protocols, it does provide the missing piece for Web services.

Without OAGIS, a canonical business language that enables integration, these cool protocols would simply pass proprietary data. Yes, the data could be in XML, but there wouldn't be a common understanding of what the messages mean. OAGIS includes the most XML messages defined in any XML dialect, and more are being defined every day.

OAGIS Simply Works

OAGIS is the common content model needed to represent messages that enable communication between business applications, whether the messages are application-to-application (A2A), business-to-business (B2B), or vertical industry-to-vertical industry (V2V) in nature. In short, OAGIS enables everywhere-to-everywhere integration.

OAGIS Is the Content

To see how OAGIS works, let's use the mail analogy. If I send you a message using postal mail, I write the message in one of a few accepted formats. I then address an envelope with the sending and receiving addresses, put the message inside the envelope, seal the envelope, apply postage, and place it in the mailbox. My mail delivery person retrieves the package. It's routed through the mail system, and after some time it arrives at your address. You retrieve the package from your mailbox, open the envelope, and read the message. Assuming you and I understand the same format and speak the same language, you can understand the message.

If my application needs to send a message to you and your application(s), something similar takes place. My application creates a message in one of several formats. Next, the application or an agent for my applications wraps the message into a transport envelope (Web services, ebXML, RosettaNet Information Framework, or Message-Oriented Middleware [MOM]) and applies the sender's and receiver's addresses to the envelope. The package is placed on the transport mechanism, which routes the package to your application. Your application retrieves the package from the transport mechanism, opens the envelope, and reads the message. Again, assuming our applications understand the same format and speak the same language, our applications understand the message.

If both applications speak OAGIS, they can understand the message. The problem is that many applications use their own proprietary formats and languages for integration.

OAGIS provides a common data model that provides a common basis of understanding, allowing all the applications involved in the information exchange to understand the intent of the message. The messaging specifications are analogous to the envelope. The integration engine used to transport and route the message is analogous to the postal service. The message and message format are analogous to OAGIS, which enables the understanding of the information on each end.

Just as it doesn't matter what brand of envelope you use to send a message via postal mail, it doesn't matter what framework you use to carry your OAGIS messages. So, use the framework that makes the most sense for your business, your supply chain, or your integration. It doesn't matter if the exchange of information is A2A, B2B, or V2V in nature. It's always critical that both sides understand the message – if the receiver doesn't understand the purpose or the content of the message, it's worthless. In many integrations, messages have to be translated for each application, business, and vertical industry for which they're intended. Doing this in a point-to-point manner is messy, not to mention inefficient.

Let's assume that it takes one-tenth of a person's time to maintain each integration mapping. (By all estimates this is extremely low, but it makes the point.) The formula for point-to-point integrations is $n(n\text{-}1)$.

Using a canonical business language to translate to and from, it's possible to use a common format in the center. The formula for a canonical integration is $n * 2$. If we use the same one-tenth of a person's time to maintain each integration mapping, it's easy to see the savings possible. Figure 1 shows the number of people needed to support each integration model as the number of interfaces grows. The math speaks for itself.

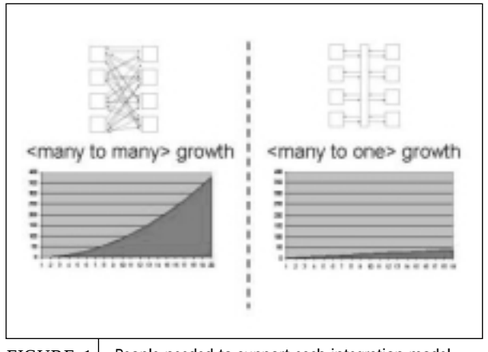


FIGURE 1 People needed to support each integration model

OAGIS can be used with any framework and integration engine to communicate information between applications, businesses, supply chains, and vertical industries. The natural next questions are, "Where do I get OAGIS?" and "How do I get started using OAGIS?".

Getting Started

Where to get OAGIS

OAGIS is freely available from the Open Applications Group, at www.openapplications.org. Follow these steps to download OAGIS:

1. Click on the Free Downloads button.
2. Click on the OAGIS or OAGIS Archives link, depending on which version of OAGIS you're interested in. The archives page gives you access to OAGIS releases all the way back to OAGIS 6.0, the first XML release based on DTDs.
3. Fill in the requested information on the form; click the Submit button at the bottom of the form.
4. For OAGIS 8.x, simply click on the OAGIS 8.x link. (OAGIS 8.x includes all the documentation schemas and examples of Business Object Document [BOD] instances in a single, self-contained zip file.) Prior to OAGIS 8.0, the OAGIS documentation and the different instantiations of OAGIS were distributed in separate zip files. Simply download the version and instantiation of OAGIS that you need along with its supporting documentation.

   If you're new to OAGIS and this is your first implementation, it's best to start with the most recent version of OAGIS. If you're supporting existing versions, you'll want to upgrade, just as you would with any software package. We'll assume OAGIS 8.0 as it is, at the time of this writing, the most recent release of OAGIS.
5. Unzip the zip file, making sure to maintain the directory structure contained in the zip file.

Congratulations, you've downloaded and installed OAGIS.

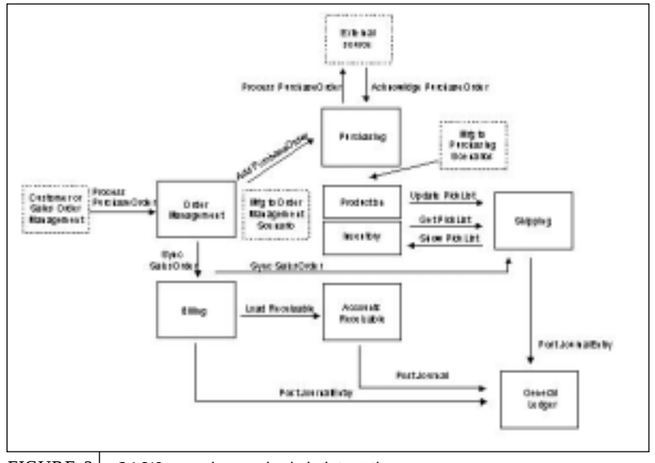


FIGURE 2 OAGIS scenario: supply-chain integration

Navigating OAGIS directories

Now that you've installed OAGIS, let's look around a bit at what is installed in the OAGIS 8.0 directory from the OAGIS 8.0 distribution. Inside the OAGIS 8.0 directory you'll find the following directories and files:

- ***Documentation folder:*** Contains all of the OAGIS documentation for OAGIS 8.0.
- ***OverlayExamples directory:*** Contains contrived examples of extending OAGIS.
- ***Utility directory:*** Contains several scripts used by the OAGI architects in the creation of OAGIS 8.0. They are provided here as is. These can be helpful in testing overlays of OAGIS. Utility also contains other tools that were used in the creation of OAGIS 8.0.
- ***index.html page:*** The starting point for the documentation. This is where we will go next.
- ***license.txt:*** Contains the licensing information.
- ***OAGIS8.0.spp:*** An XML Spy Project file to help navigate the OAGIS XML Schema files. Other XML Integration Development Environment (IDE) project files will be provided in future releases. They also may be available from the OAGIS 8.0 home site.
- ***Readme.txt file:*** To get started, read the Readme.txt file to see what updates have been made to any new release and see any known bugs. From here open the OAGIS directory, where the XML Schema for the specification is contained.
- ***OAGIS directory:*** Contains the XML Schema files and BOD example instances for OAGIS.
- ***BODConstraints directory:*** Where all of the cardinality constraints for the OAGIS BODs are kept. In a future release by co-occurrence constraints can be added. Inside of this directory are a Generated directory and a Rules directory.
  - -Generated is the generated XSL that tests for the existence of fields in the XML instances.
  - -Rules is the simple rules files that were used to capture the rules for the constraints that were then used to generate the generated files in the Generated directory.
- ***BODExamples directory:*** Where the generate XML instances of the BODs are kept. These files are intended to show what an instance of each BOD may look like.
- ***BODs directory:*** Where all of the BOD XML Schemas are located.
- ***Resources directory:*** Where all of thecomponents used in OAGIS are kept.Resources contains:
  - -Nouns directory, which contains all of the Nouns available in this release of OAGIS.
  - -Verbs directory, which contains all of the Verbs available in this release of OAGIS.
  - -Components.xsd file, which contains the components reused throughout OAGIS.
  - -Enums.xsd file, which contains all of the enumerated type lists that are reusable through out OAGIS.
  - -Fields.xsd file, which contains all of the types used for fields within OAGIS.
  - -Meta.xsd file,which contains the types used to define the meta data model used with in OAGIS.
  - -MfgComponents.xsd, which contains the manufacturing specific components.

Steps for Integration

Before you can start, you must know

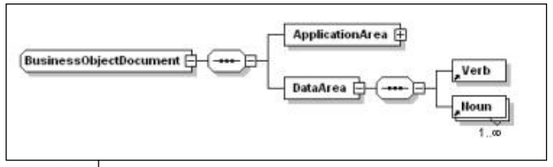


FIGURE 3 Generic structure of every BOD.

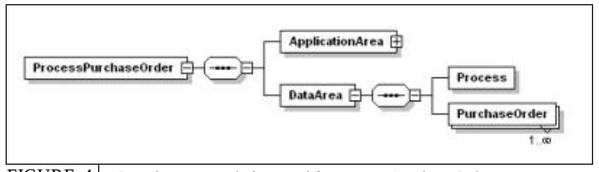


FIGURE 4 Generic structure being used for ProcessPurchaseOrder.

what you are integrating – identify the business applications and the components of each application to be integrated (the integration scenario). OAGIS includes 61 example business scenarios to get you started. Again, these are example scenarios, not a fixed set. OAGI does not assume that these 61 scenarios are the only integration scenarios that the BODs can be used within.

Second, identify the messages that need to flow between the applications, along with the intent of each message. The OAGIS scenarios reference the OAGIS BODs that have been identified to communicate the needed information. Once a scenario has been identified as a close match, drill down and look at the BODs.

Third, identify how you'll get access to the data. With knowledge of the business applications, you'll know how to best get access to the data.

Fourth, make it happen. Create the integration code (if it doesn't already exist) that accesses the application APIs and formats the information into OAGIS BODs, define the routes to transport the information to its destination, and create the code (if it doesn't already exist) to load the information into the target system.

Note: Instead of defining your own proprietary integration scenario and data formats, which won't work once you start trying to integrate with your extended supply chain, use a canonical business language that will allow you to reuse the same message definitions and scenarios whether you're inside your enterprise's four walls or integrating with your extended supply chain. You can take advantage of support for OAGIS. If you do have to create your own integration code, once you create a mapping for one BOD from one application, that same mapping can be reused for the same BOD coming from a different application. This allows you to leverage the canonical format in order to reduce the number of mapping points.

In either case, as you identify your scenario, it's important not to bite off too large a piece of the integration at one time.

Remember, the only way to eat an elephant is one bite at a time. So keep your pieces small and manageable. As you can see in Figure 2, OAGIS breaks the integration scenarios into small, manageable pieces. Viewed from end to end, the supply chain integration (manufacturing to purchasing, order management, billing, shipping, and financials) scenario is large; within OAGIS it's broken into several smaller scenarios that are easier the handle. This is often the case with OAGIS.

Not only does this make the scenarios easier to handle, the smaller modularized scenarios can also be reused in larger scenarios, which means that once the exchange is defined in your integration engine, it can simply be reused wherever needed and does not have to be recoded multiple times.

Note: OAGI understands that no matter how smart the developers of the specification are, there will always be additional scenarios in which the BODs may be used. This is especially true of a horizontal specification that enables industry vertical groups to layer their needs on top of it. While it may be possible to define a fixed set of integration scenarios with preset timeout limits, etc., in certain vertical industries this is not possible in a horizontal specification. OAGIS enables this capability for vertical industries so that these additional requirements can be defined in their overlay of OAGIS.

Using OAGIS scenarios

The scenarios identify the business applications and component modules being integrated, the messages, and the intended actions of those messages (in the BODs). The BODs define the data elements used to communicate the message. Once you know the business applications, the modules within each of these applications, and the information you want to communicate, simply look through the list of the OAGIS scenarios until you find one that matches or is similar to your particular needs. From this scenario you can drill down into each BOD to identify the content you need to communicate. Remember, the BODs have been defined with a base set of required information. Your applications and/or implementation may require additional fields to always be present. Address this by extending OAGIS via the UserArea or by using an overlay extension. Also, you may need to add additional messages (BODs) to a given scenario; these may already exist within OAGIS, or maybe in an area that OAGIS hasn't addressed. In these cases, you may need to define your own BODs or use BODs based upon other specifications.

Note: In the case of human resource messages, OAGI has agreed to reference the work of HR-XML instead of reinventing the HR specifications that have already been defined. Likewise, HR-XML reference OAGIS for non-HR messages (e.g., purchase orders, invoices, etc.).

In most cases, you'll find a scenario that is close to your particular integration needs. In cases in which you don't find a match in the OAGIS scenarios, take a look at the specifications from organizations that have partnered with OAGI for specific domains. If you still don't find a scenario that meets your needs, you'll need to define a new scenario. Often you can use existing scenarios as sub-scenarios. Similarly, it's often possible to reuse existing BODs within new scenarios. OAGI asks that anyone creating new scenarios or BODs bring them forward for possible inclusion in a future release of OAGIS.

To see the OAGIS scenarios, simply open the OAGIS documentation by opening the index.html page at the top level of the OAGIS8.0 directory. Select the Scenarios link. From here simply scroll through the list of scenarios. A table of contents provides links to each scenario.

Identifying the BODs

Once the scenario is identified, drill into it by looking at the BODs indicated in the scenario that need to be passed between the applications. Do these messages make sense for your integration? Are they needed? For example, a few of the BODs in the scenario shown in Figure 2 are:

- ***Process PurchaseOrder:*** From the external customer's order management system into the supplier's order management system.
- ***SalesOrder:*** If items are to come from inventory, the SalesOrder is synchronized with the Shipping module, which in turn picks the items from inventory to ship, per the SyncSalesOrder instructions.

By using the above BODs Figure 2 integrates the business models:

- ***Supplier's Order Management system:*** Determines whether to buy, build, or use existing inventory to meet the order. In some cases it may choose to do all three.
- ***Order Management system:*** If it decides to buy, a request is made to the Purchasing system to add a new PurchaseOrder (AddPurchaseOrder) for the items to be outsourced or purchased.
- ***Production system:*** If items are to be manufactured, a request to create a production order is sent to the Production system.

As you can see, there are many other messages that can be communicated in this scenario. In an integration it's necessary to review the intent of each BOD and compare it to what you need to do with your integration in order to verify the match.

Note: OAGIS is defined as a horizontal specification. Because of this a term that you're familiar with may be called something else within OAGIS or within another industry. If possible, we should look past this, to the meaning of the message, component, or field. Where possible, we should agree to use the same names looking at the definitions of these elements.

In some circumstances, it makes sense to use synonym names. We can do that, but we all must be aware of what we're doing – creating a message, component, or field that has context within a particular domain or industry vertical.

Only by working together will vertical industries be able to agree on a common dialect for business integration.

"Only by working together will vertical industries be able to agree on a common dialect for business integration"

Reviewing the BODs

After identifying the BODs to be used, drill into the BODs to ensure that the information you need to communicate is provided in them. Do this by reviewing the OAGIS documentation: select the *OAGIS Documentation* link from the home page. Next, find the BOD you're interested in. The OAGIS documentation includes a list of the BODs sorted alphabetically by Noun and by Verb.

For the scenario, review each BOD by drilling down into each of its layers to determine whether the information you need to pass is present. The documentation for each field, component, and the corresponding type is provided both in the documentation and in the XML Schemas themselves. The HTML documentation and the annotation within the XML Schema will always match because the HTML documentation is generated from the XML Schema.

This analysis can be done using spreadsheets or an XML analysis schema, anything that lets you see the fields, components, and structures and do a mapping from the application's format into the OAGIS BOD and from the OAGIS BOD into to the receiving application's structure. Once you've done your analysis for one Noun, it is done for the other BODs using that Noun, at least on the BOD side.

One of the key benefits in the modularization of OAGIS is that once you've created code (classes) to consume or produce an OAGIS field, compound, component, or type, you can reuse that code (class) any time the field, compound, component, or type occurs again. This is very powerful, because OAGIS reuses fields, components, and types throughout. Once you've coded a few BODs, the others become easier to implement because you're reusing your previous work.

BOD structure

Every BOD has an ApplicationArea and a DataArea (see Figure 3). The ApplicationArea serves to:

- Identify the sender of the message
- Identify when the document was created
- Provide authentication of the sender through the use of a digital signature, if applicable
- Uniquely identify a BOD instance (The BODId field is the Globally Unique Identifier for the BOD instance.)

The ApplicationArea includes the following elements: the sender of the message; the CreationDateTime; the signature; the BODId, the unique identifier for the BOD instance; and a UserArea.

The DataArea contains the instance(s) of the data values for the business transactions. For example, the ProcessPurchaseOrder from our scenario earlier includes one Verb (Process), which can be applied to one or more Nouns (PurchaseOrder), as shown in Figure 4.

As you can see in Figures 3 and 4, the data (the Noun) and the action (the Verb) have been separated in OAGIS 8.0. This means OAGIS has one definition for PurchaseOrder instead of one for each Verb that is used with PurchaseOrder, in this case eleven different copies of similar PurchaseOrders would have needed to be maintained. The action information for each BOD is contained in the Verb. The Verb uses XPath to reference the elements to be retrieved in the case of the Get and GetList BODs.

The cardinality constraints are provided by the BODConstraints, again by using XPath to point the elements and by applying rules to check whether the elements exist in the instance. This is done using the files in the BOD Constraints/Rules directory from which the OAGI architects generated the BODConstraints/Generated directory. The BODConstraints/Generated directory contains the constraints in XSL, which can then be applied to the XML BOD instances.

For more information on how the OAGIS BODs are structured, please see the OAGIS documentation *Architecture* link for a full description and definition of each element.

Accessing the data

Most business applications provide Application Programming Interfaces (APIs) that provide external systems access to their data. With the proliferation of XML, most APIs come in some form of XML. Some applications go as far as supporting the consumption and production of OAGIS BODs natively or through an add-on piece of software. For those applications that do not have APIs it is necessary to access the data through whatever means possible. This may include screen-scraping or accessing the data directly from the application's database.

Screen-scraping is the process of having an application that drives an application's graphical user interface (GUI) to provide the input and to capture the output of an application. While it's slow and error-prone and often specific to a version of the application's GUI, it does reuse the application's original logic and controls.

Accessing the data directly bypasses the logic of the applications, grabs the data, and inserts it in the database. This is always dangerous because inserting data in one area of an

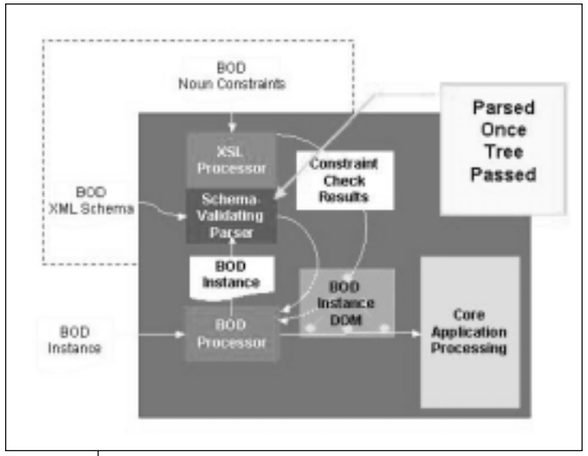


FIGURE 5 Using XSL processor to validate both XML Schema and XSL BOD constraints in a single pass.

application may generate additional data in another area, which must be replicated.

For more information on accessing data within your business applications, consult an expert for each application being integrated to identify how to obtain the data for that application in the safest, most efficient manner possible.

Implementing the Integration

The scenario and the BODs have been identified, along with where the information is coming from and going to, and the analysis has been done to identify the fields and structures to be used to communicate the information. It's now time to implement the integration. Do this using the tools that meet your integration needs. Your choices include any of the following:

- ***Enterprise Application Integration (EAI) tools and servers:*** These allow you to create a working graphical depiction of your integration, similar to that shown in Figure 5, where you can enable the flow of messages through the enterprise. Once you've identified the scenario and BOD, begin creating the scenario within the EAI tool – many include mapping tools that facilitate the mapping exercise. The maps produced are executable by the EAI engine either on the server side or by having adapters in front of the sending and receiving applications.
- ***Web services:*** Web service development applications aid in the development of Web services for communicationg BOD messages.
- ***Hand coding:*** While it's possible to code for yourself the functions of an EAI tool and a Web services application development kit, it's better for companies to focus on their core competencies. Unless your company is in the business, use off-the-shelf packages that meet your needs. This mapping code can take several forms: XSL translations for XML-to-XML mappings, scripts, or code written in any language (like Java). Most of the commercially available products give you options as to what this code can be.

Producing BODs

To produce a BOD, an application must provide the data in the format defined by the XML Schema; do this using an XML parser and driving the APIs that populate the XML DOM tree. The XML parsers provide a method to serialize the information out. Commercially available EAI, Web services application development kits, and the applications that support XML and OAGIS do this, translating the information from the internal formats into OAGIS.

It's recommended that the application producing BODs verify that the BOD is valid. While it is possible to turn validation off after the application has demonstrated that it can reliably produce a valid BOD, it is not recommended.

Eating BODs

Similarly, to consume a BOD, an application will make use of an XML parser to read in and validate a message using the XML Schema definitions to validate the schema-level structure. An application also needs to use an XSL processor to verify that the BOD instance meets the BOD constraints.

Remember that an XSL processor uses an XML parser underneath it. Because of this it's possible to do the XML Schema and BOD constraint validation in a single pass through an XSL processor (see Figure 5). Many commercially available off-the-self applications allow you to plug in your on validation capability to do this.

Similarly, you can do the same type of thing using an SAX parser to handle larger BODs.

Summary

OAGIS, available today, is enabling the integration of small, medium, and large organizations and vertical industries. OAGIS increases the efficiency of integration implementations by allowing companies to shorten the time required to realize the return on investment. It has reduced the maintenance cost of these implementations by improving the efficiency of the integrations.

In short, OAGIS is good for business, everyone's business! OAGIS is freely available from the Open Applications Group Web site at www.openapplications.org. If you'd like to learn more about OAGIS, take a look.

"OAGIS accelerates projects and reduces risk by enabling us to stabilize project requirements"

– Tim Thomasma, Ford Motor Company

Author Bio

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

MROWELL@OPENAPPLICATIONS.ORG



WRITTEN BY **RAHUL SINGH, KATIA SYCARA, TERRY PAYNE**

Distributed AI, Schedules, and the Semantic Web

RCal provides a glimpse of what's to come

AUTHOR BIOS

Rahul Singh is a graduate student at the Robotics Institute at Carnegie Mellon University and also works as a research programmer with the Advanced Agent Technology Laboratory at Carnegie Mellon University. He received his B. Eng. in computer systems engineering from the University of South Australia, Adelaide, and is currently pursuing a master's degree in robotics.

Katia Sycara is a research professor at the Robotics Institute at Carnegie Mellon University and director of the Advanced Agent Technology Laboratory. She received a BS in applied mathematics from Brown University, an MS in electrical engineering from the University of Wisconsin, and a PhD in computer science from the Georgia Institute of Technology. She is the founding editor-in-chief of Autonomous Agents and Multi-Agent Systems.

Terry R. Payne is a lecturer at the University of Southampton, UK. He received a B.Sc. in computer systems engineering from the University of Kent at Canterbury, UK, and his M.Sc. and PhD in artificial intelligence from the University of Aberdeen, Scotland. He is currently involved in the DAML program, and is a coauthor of the DAML-S Service Description Language.

Meetings are an integral part of modern life, regardless of whether they're formally established at the workplace or casual agreements made over the phone. Many factors affect meeting scheduling, some of which are explicit (What existing meetings do I have?), implicit (I prefer to avoid meetings before 10:00 am), or cultural (social events should be scheduled for Friday evenings and weekends, but not Sunday mornings).

In addition, humans typically reason about where the meeting will take place, its duration, purpose, and so on. All these factors contribute to making meeting scheduling a difficult problem. However, recent developments in automated preference acquisition, multi-agent negotiation, and reasoning techniques for semantic content on the Web are slowly making automated scheduling a reality.

An important factor in automated meeting scheduling is being able to share, understand, and reason about all the details of an event or meeting request. The Semantic Web facilitates the representation and distribution of knowledge as structured data with meaning, thus allowing agents to reason about concepts in the real world. We've developed several intelligent agents that negotiate with each other to organize meetings on behalf of their users, using published knowledge to make appropriate decisions. The RETSINA Semantic Web Calendar Agent (RCal) is one such agent built using the RETSINA AI infrastructure, which augments a widely used Personal Information Manager (PIM) – MS Outlook 2000. RCal combines knowledge about its user's current schedule, information about colleagues and friends (using MS Outlook 2000's Contact entries), and knowledge gathered from the Semantic Web to better automate meeting scheduling and management. In this scenario, users running MS Outlook 2000 on their desktops also have an instance of RCal running in the background, acting on their behalf.

RCal (see Figure 1) schedules meetings for its user, updates the user's calendar with schedules from the Semantic Web, interacts with Web services that may provide additional relevant information pertaining to scheduled meetings, and provides alerts based on occurring events.

Distributed Meeting Scheduling

RCal negotiates with other agents to find mutually agreeable times based on the user's availability and preferences. Traditionally, the burden of maintaining an up-to-date calendar has fallen on the user – a task that is time consuming and error prone. To address this, RCal can reason about events and schedules published on the Semantic Web, and automatically incorporate them directly into the user's schedule. This reduces the burden on the user, and maintains an up-to-date calendar that can be consulted by the agent when scheduling meetings.

RCal currently supports two types of distributed meeting negotiation – multi-party negotiation and appointment-request negotiation. Multiparty negotiation occurs when several agents try to identify a mutually agreeable meeting slot based on their users' current schedules and preferences, whereas appointment-request negotiation identifies possible meeting times for one party based on a meeting request. This latter form of negotiation is used by the Web-based E-Secretary to allow people to request meetings or appointments via a Web-based interface.

RCal's multiparty negotiation occurs when someone desires a meeting with one or more individuals, each of which employ their own RCal agents to manage their schedules. RCal goes through several rounds of automated negotia-

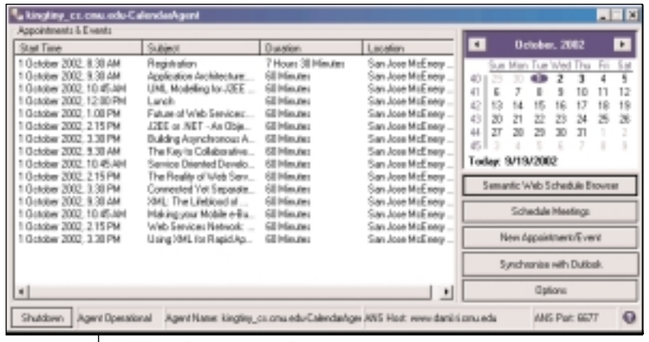


FIGURE 1 RETSINA Semantic Web Calendar Agent

tion with other RCal agents until all the agents agree upon a common time. Users can instruct RCal to schedule a meeting by specifying a particular time they would prefer or a window within which an appropriate slot of the specified duration should be found. For example, a user may request "an hour-long meeting with all the members of the project group within the next two days." The negotiation uses the Contract Net Protocol, which starts with an initial document (a contract) specifying details of the event, such as its attributes (subject, body location, etc.) and its constraints (start time, end time, and duration). The contract is broadcast to the other RCal agents, which consult their users' calendars and reply with one of three types of bid:

- Accept the contract for the meeting at the specified time
- Reject the contract outright (i.e., do not wish to meet)
- Reject the time specified by the contract and propose alternative time slots

The host evaluates all bids upon receipt and looks for a time that is acceptable to all the attendees. If a common time is found, the host sends out awards corresponding to the received bids and the RCal agents update their calendars before sending out confirmations that terminate the negotiation. If no common time can be found, the negotiation restarts with a new contract generated within the original constraints specified by the host. This iteration continues until a meeting time is identified or no new contract can be generated due to the original constraints set by the host of the meeting.

The RETSINA E-Secretary

There may be occasions when users don't have access to their RCal agent, or may not use a PIM to manage their calendar and hence need some alternative mechanism for requesting meetings. Alternatively, individuals or organizations (such as clinics or dental offices) may want to publish a Web-based interface through which appointments can be requested. The RETSINA E-Secretary agent (see Figure 2) is a Web-based agent that facilitates appointment requests, without the need for both parties to use RCal. The design is based on the concept of scheduling meetings via a human secretary – the secretary interacts with meeting requesters and manages meeting requests according to when time slots are available. In such cases, the negotiation is limited to only two parties: a human requesting a meeting or appointment at a preferred time, and RCal responding to this request (via the E-Secretary) with the appropriate meeting time based on the requestee's calendar. Typically, users who manage their calendars using RCal also have E-Secretary agents running on their behalf on the Web.

Anyone desiring a meeting enters details such as name, e-mail address, location for the meeting, and desired time via a form presented by the E-Secretary. The E-Secretary then sends the meeting request to RCal, which looks for an appropriate time slot. If one is found, it's presented to the human requesting the meeting, who can either accept or reject the proposed meeting time. If the meeting is accepted, the E-Secretary sends a confirmation to RCal, which updates the calendar and notifies both parties of the scheduled meeting via e-mail.

Maintaining Calendars

One advantage that RCal has over many other agent-based meeting scheduling systems is the ability to gather relevant information from the Semantic Web. Traditionally, calendar managers relied on humans to enter meetings that were not automatically entered. While this approach works for occasional events, it breaks down when large-scale schedules (such as conference schedules) need to be added. In many cases, users simply enter single events to represent the whole schedule, and mark the time as busy, an approach that prohibits further negotiation during this time. For example, "Bob" might plan to attend the three-day Web Services Edge Conference, and hence would want to update his calendar to reflect this. If this conference is entered as a single event, RCal won't schedule any meetings during this time. However, this doesn't accurately reflect Bob's actual schedule, since it doesn't take into account coffee and lunch breaks, and talks or presentations he may choose not to attend. In addition, Bob won't benefit from being able to consult his calendar to find out when individual events occur, or get reminders sent to his PDA or mobile phone. More important, it's often desirable to schedule meetings with other delegates at a conference, yet at such events, access to PIMs, schedules, and the delegates themselves can be difficult.

RCal and the E-Secretary overcome this problem by importing schedules directly from the Semantic Web. Traditionally, extracting schedules from the World Wide Web has been a problem since HTML (currently used to publish schedules) requires custom-built software tools, such as screen scrapers, to elicit the relevant information from the Web pages. Although XML representations can be used to simplify this, such an approach requires the standardized use of a single DTD or XML Schema. The Semantic Web relaxes this constraint by providing a framework (built upon XML) within which ontologies (formal specifications on how to represent concepts) can be built to describe concepts in the real world. Additionally, AI-based reasoning techniques can be used within Web services to search, translate, merge, or navigate across markup in these ontologies.

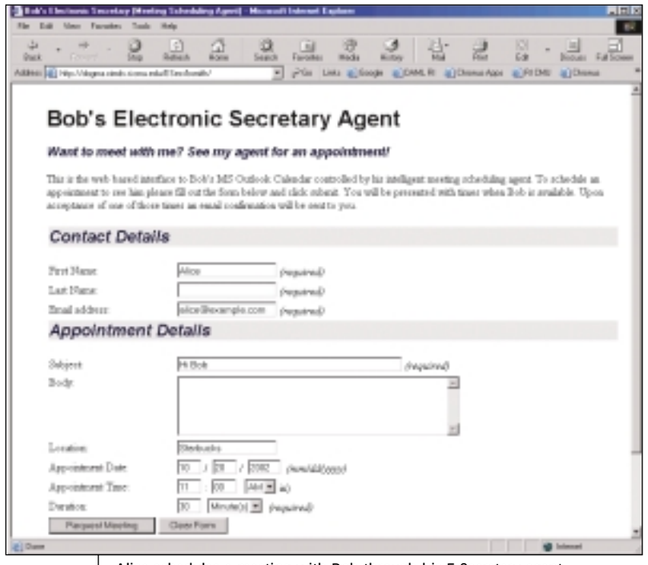


FIGURE 2 Alice schedules a meeting with Bob through his E-Secretary agent

The Semantic Web initiative utilizes RDF to mark up knowledge and publish it to the World Wide Web, where software agents can access it. Moreover, information can be made available in a structured form with links to other pieces of information, much in the way Web pages are currently linked together. The Hybrid iCal ontology, derived from the iCalendar specification, is one of several ontologies that provide a framework for schedules to be marked up in RDF and published on the Web. Used by several applications (including RCal), it allows sharing and reasoning of schedules. Listings 1-3 illustrate a schedule marked up in RDF (the namespace declaration and <rdf:RDF> tags have been removed for brevity) with three events for the Web Services Edge Conference.

In this example the iCal onthology is used to present events in the schedule, whereas the Dublin Core ontology (xmlns:dc) is used to mark up meta-information about the document itself, such as the source, title, and description. Fields in RDF can either be populated with simple text strings or with references to URIs (Uniform Resource Identifiers) that point to other RDF concepts. A URI reference such as this allows for more information to be extracted about the concept in question, which could be located elsewhere on the Web. This framework leads directly from the present architecture of the World Wide Web, except that now it links knowledge rather than plain text.

A schedule may contain multiple calendars (or schedules) represented by instances of the RDF class <ical:VCALENDAR>. Each calendar contains properties (<ical:VEVENT-PROP>) that link multiple events (<ical:VEVENT>) in the calendar. The schedule in Listing 1 contains four events of which the "Registration" event is inline, while the others reference a URI to resources in another document (e.g., the http://www.daml.ri.cmu.edu/Schedules/WSE2002-JavaTrack.rdf#J1). Each event in the file WSE2002-JavaTrack.rdf contains information about the event, such as its start time, duration, etc., exactly in the way the "Registration" event is marked up. Here the events related to each track of the conference are in another document, and the URIs allow an agent to navigate across this web of knowledge and extract information about them.

The formats for the fields (such as <ical:DATE-TIME> and <ical:DURATION>) depend upon the designer of the ontology and could very well be explicitly broken up into individual fields of day, month, year, and so on, but comprehensive markup should reference URIs to resources that represent time using an ontology that allows temporal reasoning.

RCal can import a schedule such as this using the Semantic Web Schedule Browser (see Figure 3) and present the information to the user in an organized manner, allowing more information to be retrieved by right-clicking on the concepts in question. Events can also be selected and imported into Outlook, thus allowing a user to update his/her calendar without having to type out the details of each event.

Ontologies: The Building Blocks

Marking up schedules in RDF ensures that they can be shared between different applications without assuming a tightly defined standard, thus allowing calendars to be kept up to date. An application that understands schedules defined by one ontology may be able to reason about a schedule defined by another ontology through AI-based reasoning techniques and articulations (rules that map concepts from one ontology to another). This is useful, as new ontologies are continually being developed on the Semantic Web. However, on-the-fly resolution of semantic mismatches may not be possible. In such situations, it may be desirable to delegate the task of translating a schedule from an unknown representation into one that is familiar, another agent or service provider. The DMA2ICal Service is one such service-oriented application that can be located through a semantic-based discovery service.

The DMA2ICal Service converts markup based on the DAML Meeting Agenda (DMA) ontology to that based on the Hybrid ICal ontology, and demonstrates several technologies:

- How simple translation services can convert markup based on one ontology to that of another ontology
- How agents can utilize translation-based Web services when encountering unknown markup
- How agents can utilize the semantic-based DAML-S discovery service

When RCal encounters schedules marked up based on an unknown ontology, it attempts to identify the top-level concepts, and uses these to generate service requests that can be submitted to a discovery service. Though infrastructures for discovery, such as UDDI, are slowly being deployed, they typically provide white-page (name lookup), or yellow-page (capability-based) lookup. The DAML-S Matchmaker is a lookup service that uses a DAML (DARPA Agent Markup Language)-based logical reasoning engine and utilizes resources on the Semantic Web. DAML-S is the DARPA Agent Markup Language for Services ontology, which provides a framework and a set of resources for performing semantic-based service discovery.

The DAML-S Matchmaker attempts to match the request submitted by RCal with previously advertised capability descriptions, and returns a list of the names of agents (or services) that provide the desired service. The DAML-S service profile that advertises the DMA2ICal translation service is shown in Listing 4. This profile represents the service named "DMA2ICal," which takes a "Meeting" object as input and returns a "VCALENDAR" object as output. The <profile:restrictedTo> tags indicate that the input (Meeting) and output (VCALENDAR) objects lie in the domain of the ontology referred to. A profile such as this describes a service in details that allow an agent to reason about concepts such as the Meeting concept in the domain of events that occur in time rather than, say, a meeting in the sense of a merger of two physical objects.

Listing 5 shows one of the events of the Web Services Edge Conference in the DMA ontology, and Listing 6 shows its corresponding translated schedule document in the iCal ontology. The DMA ontology contains concepts analogous to those within the iCal ontology, although the ontologies themselves aren't logically equivalent. While this lack of logical equivalence means that some schedules defined in one ontology may not be representable in another, there is a subset of schedules that can be represented by both ontologies. It's therefore possible to define an agent or service that algorithmically translates between the two representations, and once advertised, any agent may be

"Marking up schedules in RDF ensures that they can be shared between different applications without assuming a tightly defined standard"

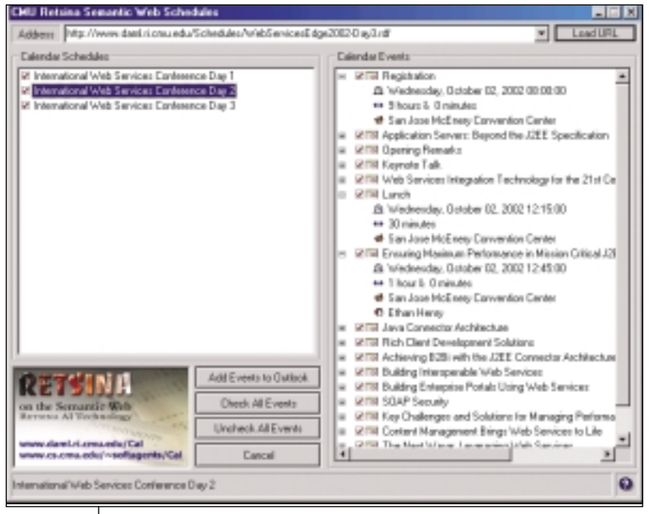


FIGURE 3 Schedules in the Semantic Web Schedule Browser

requested to perform this task. For example, the DMA2ICal translation service ensures that the start time (represented by a "DataType" entry such as a string, and linked to the <dma:Day> concept by the property <dma:time>) is translated into the concept <ical:DATE-TIME>, the value of which is equivalent (although formatted differently) to the DataType string.

Future Work

The Semantic Web allows RCal to gather and reason about information in the scheduling domain, and use this to facilitate automated distributed meeting scheduling. By adding Web-based and PDA-based agents, access to and management of one's calendar in a ubiquitous environment is becoming a reality. The Semantic Web aims to publish knowledge that will be machine understandable. Given this, distributed communities of agents and Web services will collaborate to provide intelligent assistance to the user, replacing the need for single, monolithic systems. These communities, driven by a distributed knowledge base such as the Semantic Web, will give us access to information currently unattainable without tedious searches and manual navigation.

To achieve these goals, improved methods for service discovery and negotiation will be required. The DAML-S initiative is currently developing ontologies and tools to facilitate service discovery. However, more work is required to explore issues related to semantic interoperability within service communication. The current version of RCal has prior knowledge of the DMA2ICal translation service – once the service has been located, it is invoked as a simple cgi-bin script. Further work is required to extend it to fully exploit the DAML-S interface definition to relax assumptions on message formats for invocation.

Another problem is that of generally describing the capabilities of services semantically, as in the case of the DMA2 ICal service with the DAML-S capability description language. The DMA2ICal service here is relatively simple, but more complex services involving many interactions with other supporting services will require more complex descriptions that will then need to be found and reasoned about. The development of a framework to allow such reasoning in open environments is currently under way.

As a first step however, RCal gives us a glimpse of things to come, as intelligent agents begin to navigate the large knowledge base the Web promises to evolve into. RCal is one of the first killer applications that demonstrate the utility of the Semantic Web, using knowledge to provide assistance at a level higher than that available from current applications.

References

- Smith, R.G. "The Contract Net Protocol: High-Level Communications and Control in a Distributed Problem Solver." *IEEE Transactions on Computers*, December 1980.
- Hendler, J. and McGuinness, D.L., "DARPA Agent Markup Language." IEEE Intelligent Systems. Vol. 15, issue 6.
- *Resource Description Framework (RDF) Model and Syntax Specification*: www.w3.org/TR/REC-rdf-syntax
- Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., and Zeng, H. "DAML-S: Semantic Markup for Web Services." International Semantic Web Working Symposium, 2001.
- Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., Martin, D., McIlraith, S., Narayanan, Paolucci, M., Payne, T., and Sycara, K. "DAML-S: Web Service Description for the Semantic Web." Proceedings of the 1st International Semantic Web Conference (ISWC), 2002.
- Paolucci, M., Kawamura, T., Payne, T., and Sycara, K. "Semantic Matching of Web Services Capabilities." First International Semantic Web Conference, 2002.
- Hendler, J. "Agents and the Semantic Web."*IEEE Intelligent Systems.* March/April 2001.
- Berners-Lee, T., Hendler, J., and Lassila, O. "The Semantic Web." *Scientific American*. May 2001.
- Sycara, K., Paolucci, M., Van Velsen, M., and Giampapa, J.A., "The RETSINA MAS Infrastructure" technical report CMU-RI-TR-01-05. Robotics Institute, Carnegie Mellon University. March 2001.
- *The iCalendar Specification*: www.ietf.org/rfc/rfc2445.txt
- Payne, T.R., Singh, R., and Sycara, K., "Calendar Agents on the Semantic Web." *IEEE Intelligent Systems.* IEEE Computer Society. May/June 2002.
- *The Resource Description Framework (RDF)*: www.w3c.org/RDF
- *The UDDI Technical white paper*: www.uddi.org/whitepapers.html
- Payne, T.R., Singh, R., and Sycara, K., "RCAL: A Case Study on Semantic Web Agents," First International Conference on Autonomous Agents and Multiagent Systems, 2002.

KINGTINY@CS.CMU.EDU

KATIA@CS.CMU.EDU

TRP@ECS.SOTON.AC.UK

LISTING 1

```
<!-WebServicesEdge2002-Day1.rdf -->
<ical:VCALENDAR
rdf:ID="WebServicesEdge2002-Day1">
 <dc:source rdf:resource="http://www.sys-
con.com/WebServicesEdge2002West/sched.cfm"
/>
  <dc:description>International Web
Services Conference Day 1</dc:description>
 <dc:contributor>
  <foaf:Person>
    <foaf:name>Rahul Singh</foaf:name>
     <foaf:mbox
rdf:resource="mailto:kingtiny@cs.cmu.edu"
/>
  </foaf:Person>
 </dc:contributor>
  <dc:date>2002-08-25</dc:date>
 <ical:VEVENT-PROP>
  <ical:VEVENT>
   <ical:DTSTART>
    <ical:DATE-TIME>
      <ical:TZID rdf:resource="#PDT" />

<rdf:value>20021001T083000</rdf:value>
     </ical:DATE-TIME>
    </ical:DTSTART>
   <ical:DTEND>
    <ical:DATE-TIME>
      <ical:TZID rdf:resource="#PDT" />

<rdf:value>20021001T160000</rdf:value>
     </ical:DATE-TIME>
    </ical:DTEND>
    <ical:LOCATION
rdf:resource="#SJMcEneryConvCenter" />

<ical:DESCRIPTION>Registration</ical:DESCR
IPTION>
   </ical:VEVENT>
  </ical:VEVENT-PROP>
  <ical:VEVENT-PROP
rdf:resource="http://www.daml.ri.cmu.edu/S
chedules/WSE2002-JavaTrack.rdf#J1" />
  <ical:VEVENT-PROP
rdf:resource="http://www.daml.ri.cmu.edu/S
chedules/WSE2002-JavaTrack.rdf#J2" />
</ical:VCALENDAR>
```

LISTING 2

```
 <!-WSE2002-JavaTrack.rdf -->
<ical:VEVENT rdf:ID="J2">
 <ical:DTSTART>
  <ical:DATE-TIME>
   <ical:TZID rdf:resource="#PDT" />
   <rdf:value>20021001T104500</rdf:value>
  </ical:DATE-TIME>
 </ical:DTSTART>
 <ical:DTEND>
  <ical:DATE-TIME>
   <ical:TZID rdf:resource="#PDT" />
   <rdf:value>20021001T114500</rdf:value>
  </ical:DATE-TIME>
 </ical:DTEND>
 <ical:LOCATION rdf:resource="
#SJMcEneryConvCenter" />
 <ical:DESCRIPTION>
 UML Modelling for J2EE and Web Services
 </ical:DESCRIPTION>
 <ical:ATTENDEE>
  <ical:CAL-ADDRESS>
   <rdf:type rdf:resource="foaf:Person" />
   <foaf:name>Oliver Le
Diouris</foaf:name>
   <foaf:fundedBy>Oracle</foaf:fundedBy>
  </ical:CAL-ADDRESS>
 </ical:ATTENDEE>
</ical:VEVENT>
```

LISTING 3

```
<!-WSE2002-JavaTrack.rdf -->
<ical:VEVENT rdf:ID="J1">
 <ical:DTSTART>
  <ical:DATE-TIME>
   <ical:TZID rdf:resource="#PDT" />
   <rdf:value>20021001T093000</rdf:value>
  </ical:DATE-TIME>
 </ical:DTSTART>
 <ical:DTEND>
  <ical:DATE-TIME>
   <ical:TZID rdf:resource="#PDT" />
   <rdf:value>20021001T103000</rdf:value>
  </ical:DATE-TIME>
 </ical:DTEND>
 <ical:LOCATION
rdf:resource="http://www.daml.ri.cmu.edu/S
chedules/WebServicesEdge2002-
Day1.rdf#SJMcEneryConvCenter" />
 <ical:DESCRIPTION>
 Application Architecture: A necessary
foundation for WebServices
 </ical:DESCRIPTION>
 <ical:ATTENDEE>
  <ical:CAL-ADDRESS>
   <rdf:type rdf:resource="foaf:Person" />
   <foaf:name>Walter Hurst</foaf:name>
   <foaf:fundedBy>Wakesoft</foaf:fundedBy>
  </ical:CAL-ADDRESS>
 </ical:ATTENDEE>
</ical:VEVENT>
```

LISTING 4

```
translation service
<service:serviceProfile
rdf:ID="ontology_translation_service" />
<profile:Profile>

<profile:serviceName>DMA2ICAL</profile:ser
viceName>
 <profile:textDescription>Service trans-
lates schedules in RDF from the     iCal
ontology to the DMA Ontology
 </profile:textDescription>
 <profile:contactInformation>
  <profile:Actor
rdf:resource="http://www.cs.cmu.edu/~kingt
iny/
kingtiny.rdf" />
 </profile:contactInformation>
 <profile:input>
  <profile:ParameterDescription
rdf:ID="DMA_Meeting_Object">
   <profile:parameterName>Meeting</pro-
file:parameterName>
   <profile:restrictedTo
rdf:resource="http://http://www.daml.org/
2001/10/agenda/agenda-ont.daml#Meeting" />
  </profile:ParameterDescription>
 </profile:input>
 <profile:output>
  <profile:ParameterDescription
rdf:ID="iCal_VCALENDAR_Object">

<profile:parameterName>vcal</profile:param
eterName>
   <profile:restrictedTo
rdf:resource="http://ilrt.org/
     discovery/2001/06/schemas/ical-
full/hybrid.rdf#VCALENDAR" />
  </profile:ParameterDescription>
 </profile:output>
</profile:Profile>
```

L isting5:

LISTING 5

```
<!—WSESchedule-DMA.rdf -->
<dma:Meeting ID="WebServicesEdge2002-DMA">
 <dma:name>
  Web Services Edge 2002 West
 </dma:name>
 <dma:location>
  San Jose McEnery Convention Center
 </dma:location>
 <dma:day>
<dma:Day>
 <dma:start>
  2002-10-01T09:30:00
 </dma:start>
  <dma:items rdf:parseType="daml:collec-
tion">
   <dma:Talk>
    <dma:title>
     Application Architecture: A Necessary
Foundation for Web Services
    </dma:title>
    <dma:speaker>
     <dma:Speaker rdf:ID="baron">
      <dma:name>
       Walter Hurst
      </dma:name>
      <dma:organisation>
       Wakesoft
      </dma:organisation>
     </dma:Speaker>
    </dma:speaker>
    <dma:duration>
     PT60M
    </dma:duration>
   </dma:Talk>
  </dma:items>
</dma:Day>
</dma:day>
</dma:Meeting>
```

LISTING 6

```
<!—WSE2002-JavaTrack.rdf -->
<ical:VEVENT rdf:ID="J1">
 <ical:DTSTART>
  <ical:DATE-TIME>
   <ical:TZID rdf:resource="#PDT" />
   <rdf:value>20021001T093000</rdf:value>
  </ical:DATE-TIME>
 </ical:DTSTART>
 <ical:DTEND>
  <ical:DATE-TIME>
   <ical:TZID rdf:resource="#PDT" />
   <rdf:value>20021001T103000</rdf:value>
  </ical:DATE-TIME>
 </ical:DTEND>
 <ical:LOCATION>
  <ical:GEO rdf:ID="SJMcEneryConvCenter">
   <ical:GEO-NAME>
    <ical:TEXT
     rdf:value="San Jose McEnery
Convention Center" />
     </ical:GEO-NAME>
    </ical:GEO>
 </ical:LOCATION>
 <ical:DESCRIPTION>
 Application Architecture: A necessary
foundation for WebServices
 </ical:DESCRIPTION>
 <ical:ATTENDEE>
  <ical:CAL-ADDRESS>
   <rdf:type rdf:resource="foaf:Person" />
   <foaf:name>Walter Hurst</foaf:name>
   <foaf:fundedBy>Wakesoft</foaf:fundedBy>
  </ical:CAL-ADDRESS>
 </ical:ATTENDEE>
</ical:VEVENT>
```

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

2002 READERS' CHOICE AWARDS

AND THE WINNERS ARE:

Congratulations to the winners and finalists of the 2002 *XML-Journal* Readers' Choice Awards! We commend the hard work and dedication it has taken to earn your award, and it is our pleasure to recognize and reward your efforts.

The Readers' Choice Awards program, coordinated by SYS-CON Media, identifies and publicly recognizes excellence in the solutions and services provided by the top XML vendors in the market. The purpose of this program is to raise awareness of and support for XML technologies within the business community and the public at large.

Most Innovative Application of XML

–Winner–

Tamino XML Server

–1st Runner Up–
AgileBlox Chart 1.0 (Elansoft)

–2nd Runner Up–
Sarvega XPE Switch (Sarvega)

–3rd Runner Up–
OpenOffice 1.0/StarOffice 6.0 (OpenOffice.org/Sun Microsystems)

Best XML Book

–Winner–

System Architecture with XML

–1st Runner Up–
XML Bible, 2nd Edition (Wiley)

–2nd Runner Up–
Essential XML Quick Reference: A Programmer's Reference to XML (Addison-Wesley)

–3rd Runner Up–
Java, XML, and Web Servics Bible (Wiley)

Best XML Content Management Tool

–Winner–

XML Spy Suite 4.2

–1st Runner Up–
Content Manager v8.1 (IBM)

–2nd Runner Up–
Stellent Content Management System (Stellent, Inc.)

–3rd Runner Up–
Oracle XML DB (Oracle)

Best XML Database Product

–Winner–

DB2 Universal Database v7.2

–1st Runner Up–
Tamino XML Server (Software AG)

–2nd Runner Up–
XML Spy Suite 4.2 (Altova)

–3rd Runner Up–
Oracle9*i* Database (Oracle)

Best XML Development Tool

–Winner–

WebSphere Studio Application Developer 4.0

–1st Runner Up–
Tamino XML Server (Software AG)

–2nd Runner Up–
XML Spy Suite 4.2 (Altova)

–3rd Runner Up–
Credible XML 1.0 (Java Edition) (Crediware LLC)

Best XML Editor

–Winner–

XML Spy Suite 4.2

–1st Runner Up–
WebSphere Studio Application Developer 4.0 (IBM)

–2nd Runner Up–
jEdit (Open Source)

–3rd Runner Up–
Adobe FrameMaker 7.0 (Adobe Systems)

Best Educational XML Web Site

–Winner–

Tamino Developer Community

–1st Runner Up–
XML Spy (Altova)

–2nd Runner Up–
xml.com (O'Reilly)

–3rd Runner Up–
PerfectXML.com (PerfectXML.com)

Note: XML-Journal Web site at www.XML-Journal.com was not included in this category.

Most Innovative XML Product

–Winner–

WebSphere Studio Application Developer 4.0

–1st Runner Up–
Tamino XML Server (Software AG)

–2nd Runner Up–
XML Spy Suite 4.2 (Altova)

–3rd Runner Up–
AgileBlox Chart 1.0 (Elansoft)

Best XML Integration Tool

–Winner–

WebSphere Studio Application Developer 4.0

–1st Runner Up–
EntireX (Software AG)

–2nd Runner Up–
XML Spy Suite 4.2 (Altova)

–3rd Runner Up–
Credible XML 1.0 (Java Edition) (Crediware LLC)

Most Overlooked XML-Related Tool or Technology

–Winner–

AgileBlox Chart 1.0

–1st Runner Up–
OpenOffice 1.0/StarOffice 6.0 (OpenOffice.org/Sun Microsystems)

–2nd Runner Up–
Sarvega XPE Switch (Sarvega)

–3rd Runner Up–
Scalable Vector Graphics (W3C)

Best XML Parsers and Processors

–Winner–

XML Parser for Java and Xalan

–1st Runner Up–
Xerces and Xalan (Apache XML Project)

–2nd Runner Up–
XML Spy Suite 4.2 (Altova)

–3rd Runner Up–
Sarvega XPE Switch (Sarvega)

Best XML Portal Product

–Winner–

WebSphere Portal Version 4.1

–1st Runner Up–
Tamino XML Server (Software AG)

–2nd Runner Up–
BEA WebLogic Portal (BEA Systems)

–3rd Runner Up–
Sybase Enterprise Portal (Sybase)

Best XML Schema Tool

–Winner–

WebSphere Studio Application Developer 4.0

–1st Runner Up–
XML Spy Suite 4.2 (Altova)

–2nd Runner Up–
Tamino XML Server (Software AG)

–3rd Runner Up–
Stylus Studio (eXcelon)

Best XML Transformation Tool

–Winner–

EntireX XML Mediator

–1st Runner Up–
WebSphere Studio Application Developer 4.0 (IBM)

–2nd Runner Up–
XML Spy Suite 4.2 (Altova)

–3rd Runner Up–
GE Global eXchange Service Application Integrator (GE Global eXchange Services)

Most Valuable XML Vendor

–Winner–

–1st Runner Up–
Software AG

–2nd Runner Up–
Altova

–3rd Runner Up–
Sarvega

Best XSLT Editor

–Winner–

XML Spy Suite 4.2

–1st Runner Up–
WebSphere Studio Application Developer 4.0 (IBM)

–2nd Runner Up–
jEdit (Open Source)

–3rd Runner Up–
Stylus Studio (eXcelon)

For more details

on approved quotes, press releases and logo usage to announce your achievement,

please contact

Carrie Gebert

at carrieg@sys-con.com

www.XML-JOURNAL.com

–continued from page 12

bodies and provides the opportunity to establish business relationships with other organizations involved in the standards initiative and those who are using the standards.

Conclusion

XML allows the specification of an arbitrary set of nonbinary tags for representing information components. This offers an open, independent, and low-cost platform for representing information for a multitude of purposes. However, left uncontrolled, the introduction of XML results in the proliferation of many diverse data formats that represent the same piece of information. This would create new islands of information and result in a new version of the expensive one-off formatting problems that most enterprises are looking to solve.

Information components can be shared, exchanged, and reused through the deployment of an XML Clearinghouse for the enterprise that provides a single, common, sharable location to store and manage the XML standards and vocabularies used in the enterprise. However, the technical solutions used to implement an effective XML strategy need to be managed properly. It is therefore important for an enterprise to establish an XML Standards Governance Model for the XML Clearinghouse to oversee the effective development and deployment of an integrated set of XML standards to represent, share, and exchange enterprise information components.

References

- Ferguson, R. (2002). "XML: Plugging into 'standard' hybrids." *eWEEK*. January.
- Kumar, R. (2002). "XML Standards for Global Customer Information Management." *DM Review*. Volume 12, Number 5.
- *ebXML:* www.ebxml.org
- Challenges of effective adoption of the Extensible Markup Language. (2002). GAO Report to the U.S. Government. April.
- *OASIS Customer Information Quality Technical Committee:* www.oasis-open.org/committees/ciq

Author Bios

Ram Kumar is the chief IT strategist and architect of MSI Business Solutions Pty. Ltd, Sydney, Australia, a solutions and consulting company focusing on maximizing the value of information assets through appropriately tailored strategies, services, and enabling technologies. Ram is the founding chairman of the Customer Information Quality Technical Committee of OASIS.

George Langley is a founding director of MSI Business Solutions Pty. Ltd and also heads the strategic development team. Since founding MSI in 1989, George has developed a comprehensive knowledge of business information views, integration, interoperability, and management and the information quality requirements and business processes that underpin them. The past decade has provided George with broad experience in the banking, finance, insurance, government, and telecommunication sectors.

RKUMAR@MSI.COM.AU

GLANGLEY@MSI.COM.AU

–continued from page 7

The Business Document

A business document is simply a document filled with data that is passed from service to service. It is similar to a manila folder (in the nondigital world) that gets passed from one in-basket to another. Throughout the process, services may add new records to the business document (like adding another sheet of paper to the manila folder) and may extend the schema of the data (like scribbling in the margins of a sheet of paper in the folder).

XML is the clear choice for business documents, as XML is standard, portable, simple, flexible, and most important, extensible. Throughout the life of the process, the document is stored, updated, transformed, and delivered to the appropriate service. Because of the extensibility of XML, the business document is not confined by a predefined schema and is free to extend to accommodate changing information.

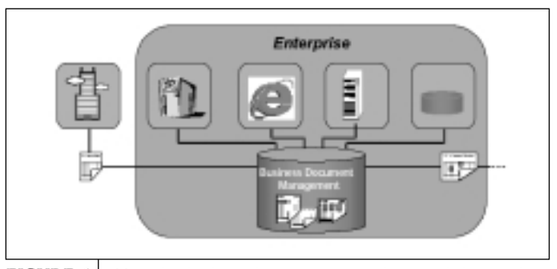


FIGURE 1 Managing business documents across the extended enterprise

Architecting for Business Documents

When architecting for business documents, there are two things to keep in mind that will simplify development. First, assume that services are truly loosely coupled, and do not plan on any special software at each service beyond a network connection and the ability to read XML. Second, fully decouple the data from the application so changes in the information don't bring about changes in the infrastructure.

This separation can be accomplished by creating an architectural component dedicated to managing business documents across the extended enterprise (see Figure 1).

Management of XML business documents allows documents to be stored, updated, transformed, and delivered between remote and diverse services. This permits services to participate in a common business transaction within a long-running process. Business document management solves the data-sharing problems created by moving away from monolithic, tightly coupled applications to disparate services.

Traditionally, applications have had to be slaves to the data model, but XML changes all that. XML business documents provide a container, or a buffer, that removes the dependencies so the data model can change freely without disrupting applications. So the next time you're investing time and money in defining an information model…don't!

XML NEWS

Rapidly Growing SYS-CON Media Garners Awards

(Montvale, NJ) – SYS-CON Media has been named one of the fastest-growing 500 technology companies in North America by Deloitte & Touche in its 2002 Technology Fast 500. The announcement came one week after SYS-CON was named one of the nation's fastest-growing private companies by Inc 500 for the third time. The 2002 Technology Fast 500 listing will be available exclusively at Forbes.com in November.

SYS-CON Media is widely recognized in the *i*-technology and magazine publishing industries as the world's leading publisher of print magazines, electronic newsletters, and accompanying Web portals. The company further solidified its dominant role in the *i*-technology space with the 2000 launch of an events business with trade shows, conferences, and education seminars.

SYS-CON Media achieved a record 752% growth in the past five years. From 1998 to 2001, revenue grew at a compounded annual growth rate of 72.9%. In 2003, the company projects its gross margin to increase 51.9%, and the contribution is projected to increase 70.4%.
www.sys-con.com

DataDirect Technologies Introduces DataDirect jXTransformer for Transforming Data Between Relational and XML Formats

(Rockville, MD) – DataDirect Technologies, the leading provider of components for connecting data and applications, has announced DataDirect jXTransformer, a new software component designed for Internet application architects and developers who need to transform data between XML and relational formats in Java programs.

DataDirect jXTransformer is an XML software component that saves time and removes complexity by providing a simple, consistent approach to creating XML data from any relational database or updating any relational database from an XML input. While database vendors' proprietary tools require specialized knowledge about complex XML extensions that work only on a single database, DataDirect jXTransformer enables architects and developers to write XML code once that works across multiple relational databases and with older versions of individual databases.
www.datadirect-technologies.com

Rogue Wave Software Expands Web Integration Technologies with XML Object Link

(Boulder, CO) – Rogue Wave Software, Inc., has announced the availability of a new product, Rogue Wave XML Object Link, the latest release supporting the Company's Web services integration strategy. XML Object Link addresses the growing need for businesses to smoothly integrate XML and Web services with their existing applications in order to extend their mission-critical systems over a network.

XML Object Link enables development teams to focus on building and deploying critical business solutions faster and more robustly by providing the ability to easily handle XML documents that can readily be exchanged with other systems, including those based on Java or Microsoft .NET systems using XML or Web services.
www.roguewave.com

Corel Ventura 10 Now Available: Extending the Power of XML Content to Professional Business Publishing

(Ottawa, Canada) – Corel Corporation has announced that Corel Ventura 10 is now available. A comprehensive page-layout and publishing application, Corel Ventura 10 provides exceptional tools designed specifically for the creation of highly formatted and visually rich business documents. Including new features such as XML import, Publish to PDF, and enhanced graphics support, Corel Ventura 10 is the ideal tool for business document creation.
www.corel.com

XML Encryption, Decryption Become W3C Proposed Recommendations

(Cambridge, MA) – The W3C is pleased to announce the advancement of XML Encryption Syntax and Processing and Decryption Transform for XML Signature to Proposed Recommendations. Encryption makes sensitive data confidential for storage or transmission, and the W3C's proposed recommendations will allow the encryption of selected sections or elements of a document.
www.w3.org

PolarLake Releases PolarLake v2.0, the Enterprise XML and Web Services Platform

(Dublin, Ireland) – PolarLake has announced the release of PolarLake v2.0, a major new version of its Enterprise XML and Web services platform for Java technology.

PolarLake v2.0 delivers a high performance, scalable, and easy-to-use solution for enterprises wishing to integrate and extend existing systems with XML and Web services.

PolarLake v2.0 includes significant new functionality in the areas of XML schema support, management capabilities, and support for enterprise messaging middleware standards. PolarLake v2.0 also adds Sun ONE Studio as a supported component development environment.
www.polarlake.com

Altova, Inc., Extends Market Leadership in Web-Based XML

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