Technical context and cultural consequences of XML

S. Adler
R. Cochrane
J. F. Morar
A. Spector

The Extensible Markup Language (XML) is an open standard for creating domain- and industry-specific markup vocabularies. XML has become the predominant mechanism for electronic data interchange between information systems and can be described as a universally applicable, durable "Code of Integration." As we celebrate its tenth anniversary, it is appropriate to reflect on the role XML has played and the technical ecosystem in which it functions. In this paper, we discuss both the environment from which XML arose and its technical underpinnings, and we relate these topics to companion papers in this issue of the *IBM Systems Journal*. We discuss the broad consequences of XML and argue that XML will take its place among the technical standards having the greatest impact on the world in which we live. We conclude with some reflections on the significant technical, economic, and societal consequences that XML is likely to have in the future.

INTRODUCTION

In 1996, a committee of the World Wide Web Consortium (W3C**) began work on what became the Extensible Markup Language (XML).[1] Based on SGML[2] (Standard Generalized Markup Language), XML is a general-purpose markup language that creates domain- and industry-specific markup vocabularies which share certain semantic and syntactic characteristics, facilitating interoperability of tools, techniques, and even programs. Although it is most commonly seen to be a standard format for delineating textual data, XML is more accurately a technology for labeling information with descriptive names that can be consistently used and accessed in a multitude of applications. The original motivation for SGML, subsequently passed on to XML, was to ensure that the content or data residing in documents survived long after the application that processed it became obsolete or unusable; thus no processing or procedural information is embedded within the content; instead, content is encoded as clear text and available everywhere. The goal was to bring SGML concepts to the Web. XML was designed to be simpler than SGML and more universally applicable, with a lower barrier to entry: users are able to start simply and build their applications incrementally. Any of the world's languages can be used for XML markup or content, due to its incorporation of Unicode as a key component. All of these features have resulted in the broad availability

of a basic set of techniques and tools for processing content. What started out as a simple standard for electronic publishing has matured into one of the most important and widely used paradigms in distributed computing. The impact and influence of the standard and the conceptual base are visible in every area of computing today.

As we celebrate XML's tenth anniversary, it is appropriate to reflect on the role XML has played and the technical ecosystem in which it functions. XML was not conceived nor did it evolve in a vacuum. The phenomenon of the Web and the proliferation of HyperText Markup Language (HTML)[3] allowed XML to emerge as the lingua franca of the Internet. Its rapid adaptation in conjunction with a wealth of existing technology quickly transformed XML into a standard distributed-computing protocol that is helping to enable many things, including a new generation of service-oriented architectures (SOAs). In this paper, we discuss both the environment from which XML arose and its technical underpinnings. We introduce the papers in this issue of the *IBM Systems Journal*[4] and describe and speculate on the broad consequences of XML, which are significantly changing the world in which we live.

THE BIRTH OF XML

A clear vision and a sound technical approach do not ensure an established technology. Rather, the establishment of a broadly accepted new technology generally depends on many factors coming together. In this section, we review the business environment whose technical requirements are addressed by XML and discuss the ecosystem in which XML was born. This includes the role that the open-source/open-standards movement had in defining XML, the significant impact that the World Wide Web had on its rapid adoption, and the technical foundations that provided a fertile breeding ground for its conception.

Expanding needs of business and scientific communities

The ever-growing need of industry and government to integrate disparate IT (information technology) systems has resulted in interconnected networks that can no longer be constructed, managed, or enhanced centrally. The IT industry's response to this situation has been to embrace the distributed nature of today's computational world by allowing for the creation and execution of products and services that can perform effectively as components of larger, dynamically assembled, or federated applications. The overall stability of such systems requires that the individual components be loosely coupled so that they can be managed, modified, and replaced without threatening the operational integrity of the entire system. This requirement has become equally important for both internal operations of large enterprises and business-to-business interactions between enterprises of differing size. The number and diversity of distributed applications adds to the complexity of this problem. The need for a common, self-describing, and highly dynamic framework capable of supporting diverse domains arose from these expanding computing requirements.

System architects realized early that many specialized languages suited to specific domains were required to represent the numerous bodies of data used in those domains. Though the specifics of such languages needed to be responsive to the needs of particular domains of discourse, many common requirements were immediately apparent:

- Each language needed well-defined and self-consistent mechanisms for describing data, including the naming and description of constituent elements of the data, the expression of relationships between those elements, and the capability of verifying that the overall data was expressed in a well-formed fashion.
- Data elements needed to be easy to transport, transform, search, combine, extract, filter, and view in different forms.
- The definitions of the languages themselves needed to be easily shared and maintained by a large body of users.

XML provided a very general approach for satisfying these common requirements. It allowed the definition of languages in which information is encoded as tagged text and in which different encodings and tags support different domains of discourse.

Role of open source/open standards

To address these common requirements, a committee (more commonly referred to as a working group) of the World Wide Web Consortium was formed in 1996. Initial discussions on developing XML, even before the formation of the working group, focused

on the need to involve both large- and small-scale commercial interests to create greater market value and a base for growth. Involvement of universities and government organizations was nurtured as the standards evolved.

XML is rarely used alone. A body of standards emerged around XML to support its use. The original XML initiative included three aspects: the syntax of the language itself, a linking component, and a style component. As its use grew, XML extended into other areas: graphics formats, forms, data models, APIs for accessing XML structures, linking, query languages, transport protocols, and finally business processes and workflow that use XML.

The standardization of XML stimulated the establishment of a community of interests around it and the Web, involving a large and diverse group of individuals and organizations. Companies participated in and adopted the XML open standards and worked on open-source technology for many reasons: to gain the benefit of an open community to supplement their own development resources, to take advantage of the positive marketing perceptions surrounding the participation in nonproprietary solutions, and to benefit from the vast market opportunities created. Others simply saw this as a tremendous intellectual and technical challenge—the next great thing.

The availability of open-source implementations had a profound impact on the development of open standards and on their widespread adoption and popularity. It quickly became economically feasible to explore the use of the new standards in production environments. Prototypes were simpler and faster to develop with technology readily available from open-source efforts such as Apache[5] and SourceForge.[6] Companies adopted open standards for similar reasons. The desire not to be left behind the competition, customer requirements for interoperable solutions, and the simple economics of sharing in a common pool and community of interests all led to the rapid development and adoption of open standards.

The success of this community stimulated the creation of new domains. Academic communities began to participate as the ideas around XML took root, realizing the potential for fruitful and innovative research in areas such as algorithms, analytics, models, and optimization. Government agencies also increased their role over time and demanded more functions than those achievable with SGML.

One of the most compelling aspects of this evolution was the intense and spirited collaboration of communities from different disciplines, each having its own ideas of what was important and often dismissing the requirements of other communities. This dissension might have destroyed the entire experience, but perseverance defeated it. Discounting the small but persistent set of detractors, the diverse communities realized that there was much to learn from each other and much value in considering the broad range of requirements. The communities began to understand that XML could allow the integration of data-centric, document-centric, forms-centric, protocol-centric, and process-centric views of information and the processing they undergo. For the first time, XML and its related standards enabled data interoperability, content manipulation, content sharing and reuse, document assembly, document security and access control, document filtering, and document formatting across all disciplines and for all types of devices and applications. This collaboration and discussion continues today as work progresses on the next tier of standards, taking into account the benefits and risks brought by XML's growing complexity and diversity.

■ XML has become one of the most important and widely used paradigms in distributed computing ■

Technical ecosystem and foundations

In addition to the requirements that provided fertile ground for its invention and the standards and open-source communities that converged to bring XML to fruition, XML was born into a very rich and exciting technical ecosystem that contributed to its rapid adoption and provided a foundation upon which XML tooling and technology could be built and extended.

XML could not have happened without the World Wide Web. The Web has become a universal mechanism to deliver information to consumers and

increasingly, to applications as well. XML gained notoriety in the heyday of the "dot-com" frenzy, but it continues today because it worked and created vast new opportunities. It enabled information reuse by integrating text and data from different sources and by searching and linking across these sources, thereby breaking down traditional silos, which were barriers to information sharing. The Web became a vortex for this confluence of forces and allowed people to get a glimpse of the tremendous potential of universal access to information. It simply and easily provided the proof-of-concept, the business case, and the funds to enable development. XML started simply, matured, endured after the dot-com bubble, and spread.

XML also could not have had such impact without a diverse collection of tremendous advances in computer science made over approximately a 50-year period. XML quickly found a home with many of these technologies, which thus contributed to its widespread adoption. Of particular applicability to XML were seven categories in which there were significant advances and substantial technical agreement:

1. *The value of information hiding, generalization, encapsulation, and reuse in programming languages and methodologies*—This work began in the early 1960s with the advent of such languages as Algol and Pascal, followed by the object-oriented approaches of Simula, Smalltalk, C++, Modula, Java**, and C#. XML is a superb, common approach for defining interfaces to encapsulate abstractions.
2. *The value of search and information extraction on relatively unstructured information, even when it crosses multiple abstraction layers to access low-level data*—In contrast to information-hiding approaches, some problems benefit from direct use of information, despite operating across multiple levels of an abstraction hierarchy. XML provides a rich data representation, with significant opportunities for high-value semantic tagging, which can provide superior support for information retrieval and related activities. This approach is not entirely new. The Lisp language introduced the concept of an attribute list (known as an "a-list") that allowed data structures to have auxiliary lists of pairs of attributes and values. These lists could represent a variety of concepts, from uninterpreted data, in which the list contained arbitrary content, to conventional lists in which the order in the list had some meaning (i.e., the third element was always the maximum size), to self-defining structures.
3. *A standardized nonprocedural, high-performance approach to storage and retrieval of structured information*—Relational databases[7] possessing a powerful data model, underlying concurrency control, integrity and performance benefits, and a consistent Structured Query Language (SQL)[8] interface were a great advance of the 1970s and 1980s. Relational structures provided a rich set of storage and manipulation mechanisms, but their true value across applications was realized when database administrators could define agreed-to data dictionaries. There have been several proposals for semantic data models[9] and standards developed for object-oriented databases.[10–12] None of these models and systems predominated over relational systems. Those that work in conjunction with the relational model are still not ubiquitous because they add yet another layer of programming language to the already cumbersome task of getting data from the application layer into and out of the persistence framework. XML provides an opportunity to augment structured relational systems with increased capabilities, at the same time permitting them to become more integrated with the surrounding technical ecosystem.
4. *The utility of simple key-value pair tagging and its application to providing metadata through annotations*—With GML (Generalized Markup Language)[13] in the late 1970s and then SGML, which was standardized in 1986, there was a clear understanding of the importance of applying tags to documents and using these tags to provide semantic information. XML is a tagging architecture which grew from SGML. Another example (from the distributed computing/protocol domain) is the Multipurpose Internet Extensions (MIME)[14] tagged architecture for delineating objects in e-mail, invented in 1992. MIME set a precedent for the utility of an extensible tagged, object-definition capability for the Internet.
5. *A consensus on a layered-protocol stack for network communication, which standardizes not only layered protocols but also the interfaces to those layers*—ISO (International Organization for Standardization) OSI (Open Systems Interconnect) provided a well-regarded conceptual seven-layer reference document. TCP (Transport Con-

trol Protocol) introduced reliable streams with commonly used socket interfaces. This was another great advance, as were the various efforts to make remote procedure invocation both efficient and common.[15–18] The importance of the presentation layer was also understood. XML is clearly an advanced presentation layer protocol, but as stated in item 1, it is also a valuable mechanism for defining interfaces for supporting method invocation.

6. *The necessity and practicality of sophisticated user interfaces programmed with very high-level techniques*—Metaphor-based interfaces (i.e., those in which the target audience interacts with aesthetic concepts familiar to their area of expertise) became significantly better understood as did both component-based and nonprocedural specification of those interfaces. XML is a natural tool for specifying an extremely wide class of human interfaces to systems, even ones that are multimodal in nature.
7. *Performance and bandwidth*—Relative to optimized binary formats, XML is expensive to process (parse) and transmit and would not have been practical without the many decades during which processor performance and network bandwidth have been accelerating according to Moore's law. While it remains the case that XML's processing requirements limit its use in low-power computers, there is a vast domain today in which XML's use is feasible. Additionally, the W3C is looking at this problem to determine if further work is necessary in this context.[19]

The Web's coming of age ushered in new technologies that were born of necessity and continue to fulfill enduring needs of large distributed environments. One of the hallmarks of the Web is that the parts must work together, even though they were not designed to do so. Nowhere is this more evident than in the handling of documents and data. On the Web, this information is transported, transformed, searched, combined, and subdivided in any desired manner. Once it has been suitably transformed, it is displayed, packaged, streamed, archived, and used by people and processes. XML provides a very general approach for defining rich vocabularies to support this semantic interoperability. The vocabularies defined in XML can share many common engines, tools, and practices, such as tools for language creation, data manipulation, and personalization.

By virtue of using the common framework that XML provides, many aspects of programming, unstructured and structured information storage and retrieval, networking and integration, and user-interface design and implementation can be advanced. XML solutions to these diverse problems emerge as coherent and consistent techniques, with many resultant benefits. It is remarkable that one technology, based on a fairly straightforward concept, is having such a big impact.

KEY XML TECHNOLOGY AND JOURNAL PREVIEW

Although it started simply, an extensive and complex set of standards emerged as XML evolved. Just as Ted Codd's invention of the relational database proved to be only the tip of the iceberg, which required "hundreds of thousands (sic) of algorithmic innovations to make it work,"[20] the success of XML has required several standards bodies (as previously discussed) and countless tools, algorithmic innovations, and clever add-ons to make it a success.

This section describes many of the key XML technologies[21] that are instrumental in facilitating its further development and use and contributing to its ubiquity as its evolution progresses. This section also serves as a preview of the papers in this issue, which present advances in some of these key technologies. The categories into which the papers of this issue are grouped are: core XML technologies, connecting to business data, connecting data to applications, mapping technologies, and connecting business to business with Web Services.

Core XML technologies

This section describes a number of significant technologies central to XML and how they are used for information modeling, processing, and transformation.

Modeling and creating data in XML

XML provides a means to impose structure on content by bracketing it with "begin" and "end" tags that create named constructs called elements. Element names (e.g., "part number") are not limited by XML, but are chosen by the content provider and normally provide some indication of the meaning of the content. Elements can be nested to form

arbitrarily complex trees. Attributes are optionally associated with elements to carry further information about the content, such as access control, security levels, data type, revision dates, and usage properties. These simple constructs form a foundation for exposing content—previously available only as a "blob" of text—as data. The refined granularity that the markup provides makes the content more accessible. Furthermore, this structure provides a means for easily associating metadata with the content, which can be used for a variety of purposes, including interpretation, authorization, and personalized rendering. XML documents support flexible and late-bound reuse and error checking. This greatly enhances the use of XML in every domain.

Data processing using SAX and DOM

Simple syntax and structure are not the only factors that have contributed to the widespread adoption of XML. They are accompanied by two widely supported APIs for working with XML documents: the Document Object Model (DOM)[22] and the Simple API for XML (SAX).[23] These APIs provide the ability to programmatically process the tree structure of XML. DOM is a W3C recommendation which provides an in-memory tree-structured view of an XML document. Using DOM interfaces, the application processes XML by "walking" the nodes of the DOM tree. SAX, on the other hand, provides a means for the application to process XML as a stream. It offers events to the application as the SAX processor parses the document. In contrast with DOM, SAX is not a standard, but was developed by a community of early XML developers. It provides a compact parser, making minimal demands on system resources. Both SAX and DOM APIs were widely supported and readily available to run on multiple platforms early in the evolution of XML, greatly enabling its rapid adoption.

■ An extensive and complex set of standards emerged as XML evolved ■

Transforming XML with XPath and XSLT

Although SAX and DOM provide procedural mechanisms to work with XML documents, developers also needed declarative mechanisms to provide granular access to content. Extensible Stylesheet Language Transformation (XSLT)[24] began life primarily as a styling language, but evolved to provide an XML native method for XML-to-HTML and XML-to-XML conversion. Because XSLT is itself XML, all XML tooling (editors, parsers, XPath) can be used in the creation and manipulation of XSLT style sheets and transforms.

XPath[25,26] is the predominant language used to navigate the tree structure of XML content and was designed as an integral part of XSLT. As such, it forms the foundation for XML query languages. XPath is a language on its own and is used by several W3C recommendations, including the W3C Schema language and XForms[27]; it is also interwoven into XSLT 2.0,[28] XQuery,[29] and hence, SQL/XML.[30] XPath is declarative in that its results are insensitive to how the XML tree is materialized. Using a very small subset of XPath, an application can accomplish most of what it needs in querying the contents of documents; XSLT, XQuery and SQL/XML were introduced to provide more intuitive query capabilities for their respective communities.

Schemas and namespaces

XML relies on user communities creating domain- and industry-specific vocabularies, which were initially expressed using DTDs (Document Type Definitions). As XML grew in popularity, communities defined sets of element names, types, and interpretations for XML structures, which greatly enhanced interoperability between the applications used by those communities. Two related XML technologies, XML Schema and XML Namespaces, are used extensively to allow industries to define their semantics.

XML Schema[31] is the standard that emerged for defining XML structure and type information. XML Namespaces[32] is the standard that was created to facilitate the reuse of documents and document fragments. The basic objective of Namespaces is to resolve the inevitable naming conflicts that arise in the sharing of documents and fragments among various communities. More sophisticated use of this standard allows the same data format for a structure, such as a book description, to be used in several other structures, such as purchase orders, publisher's inventories, and card catalogs. The common data format enables the modular repurposing of individual data instances and the creation

of modular software components that work on such reusable document fragments.

Paper preview

As XML has matured, many techniques have been developed for processing it; early techniques included open-source XML parsers (e.g., Xerces) and XPath/XSLT processors (e.g., Xalan and Saxon). The beauty of XML is that it is ubiquitous and, to a significant degree, self-describing. This self-describing nature supports flexible and late-bound reuse and error checking, but, with this flexibility there came inherent performance issues.

XML parsing is a critical aspect of XML processing performance. The ubiquity of XML allows it to be parsed anywhere by anything, but to increase performance, parser techniques need to be extended. Perkins et al. in "Generation of efficient parsers through direct compilation of XML schema" present the next advance in parsing technology, one which makes industry-strength enterprise deployment possible.

Connecting to business data

The papers in this section discuss the ways in which query languages for XML have developed and how businesses can use these languages with structured and unstructured XML data.

Evolution of XML query languages

As the popularity of HTML grew, users wanted first to be able to render their business data, stored in relational databases, into HTML. They were not satisfied with simply presenting and viewing static HTML; they wanted to be able to connect their databases to the Web so that people could query them. The Common Gateway Interface (CGI)[33] is a standard interface that provides exactly this functionality within HTML, and although this is a relatively simple idea, it is frequently difficult to implement.

With the advent of XML and XML's close relationship to HTML, it became apparent that an important usage scenario was to make data from databases available on the Web. One technique relied on translating relational data into XML for rendering as HTML. In response to this need, the SQL language was extended to provide a bridge between the SQL and XML worlds. Initial versions of these extensions primarily focused on providing a set of functions for publishing relational data in an XML format.

In contrast, as the use of XML matured, it became evident that a query language intended from the beginning to work with databases was needed. Although very expressive for navigating within an XML document and isolating elements and attributes that satisfy a given search criterion, XPath was not designed originally as a stand-alone language. For example, it relies on XSLT to construct new XML documents. Similarly, XSLT was designed primarily for style transformations. Although it is very powerful, its recursive pattern-matching paradigm (which is needed for document processing) is difficult to optimize, particularly in a database context. Clearly, relational database concepts were needed for commerce and transactions. These evolving requirements gave rise to the formulation of the W3C XQuery working group in 1999, which was chartered to work closely with the XSLT working group to jointly design the next version of XPath and other supporting specifications, which formed the basis for XQuery. XQuery and SQL/XML emerged to provide full query capabilities for XML, while maintaining compatibility with XPath and XSLT.

Querying structured and unstructured content

As XML became a de facto standard for representing business artifacts and as investment increased in XML technology and products and in the deployment of relational systems, there was an obvious need for interoperability between XML data, relational-database systems, and text search. Businesses require the ability to seamlessly query a collection of documents that include structured and unstructured data, using query functionality traditionally available only in relational systems or information-retrieval systems.

When XML arrived on the scene, structured and unstructured documents were typically processed by two independent systems. Consequently, early deployments of XML treated data either as structured data, represented by an inexact, parsed representation in a relational database, or as unstructured data, stored as a "blob" in a file system or content management system. Although the former representation enabled leveraging the full support of data mining and business-intelligence reporting tools, it fell short for applications with complex schemas such those in the Health Care and Life Sciences (HCLS) industry. The latter representation allowed the association of valuable metadata

with the content, but the naïve search did not distinguish between markup and content. Additionally, there was a growing need to leverage interoperability with relational data.

Three things have happened to rectify this isolation of technologies. First, relational systems began to adopt more features of full-text, such as SQL/MM.[34] Second, technology was developed to merge XML with relational systems. This integration of XML with SQL provides the power of navigation with a set-oriented query language, and leverages the full power of the widespread adoption of search engines such as AltaVista** and Google**. As this trend continues, users have higher expectations for fuzzy searching, and language features supporting such semantics are being integrated into the XML query languages. Third, there has also been progress in architectural support in systems that allows combinations of semantic annotators to process documents and generate metadata that makes the documents more searchable and more useful. This automatically generated metadata can populate XML documents or database, search engine, or knowledge-base indexes. IBM has proposed the unstructured information management architecture[24] (UIMA)[35] as a standard structure for supporting componentized, reusable annotators; it has also contributed the UIMA toolkit to the open-source community[36] to facilitate the broad adoption of UIMA. These efforts can help bridge the gap between semantically enriched XML and otherwise semantically impoverished unstructured data.

In effect, UIMA facilitates a fluidity which enables documents that contain significant amounts of raw text, audio, image, and video to be semantically enriched with tags through automatic or semi-automatic annotation. This information can then be represented as XML and manipulated with the vast array of XML tools and methodologies. In all, there is the prospect of having access to record-level relational data, XML data, and unstructured data by using a combination of "best of breed" technologies, with XML providing a structure for semantic encoding. Academic, government, and commercial sources have shown interest in this approach by developing many solutions with compliant annotators that generate semantic content.

Early attempts to merge XML and relational technology provided XML views of relational data, so that XML tooling could access relational data, or provided relational storage of XML data, so that relational techniques could be used to process XML data. These early convergences inspired and drove yet more demands for further integration. XML views of relational data fell short in their ability to fully and efficiently translate XML queries into SQL queries. Various attempts to store XML data in relational tables resulted in the inability to store and retrieve the XML data in its original form as well as the inability to adapt to the dynamic schema changes expected in XML. As a result, many relational vendors have been active in the XQuery and SQL/XML efforts and have produced systems that integrate XML and relational data and support the ability to query both simultaneously.[37]

Paper previews

Several of the papers in this issue describe XML native extensions that will be available in the upcoming Version 9.1 release of DB2* Universal Database* for Linux**, Unix**, and Windows**. These extensions are commonly referred to as DB2 XML. "Integration of SQL and XQuery in DB2 XML" by Özcan et al. is a recommended starting point, as its focus is primarily on the externals for defining and querying XML data in DB2. It contains a brief history of the evolution from XPath to XQuery and a detailed comparison between the foundations of SQL and XQuery. The main part of the paper gives a comprehensive description of the SQL/XML features and functions from a standards perspective, details how these features manifest themselves in DB2 XML, and overviews DB2's approach to integrating XQuery and SQL/XML, which has enabled interoperability between the languages. This integration differentiates IBM from other major vendors in this field. The paper concludes with a discussion of the syntactic and semantic challenges of integrating SQL and XQuery.

In "DB2 goes hybrid: Integrating native XML and XQuery with relational data and SQL," Beyer et al. describe an architecture for fully integrating native XML support in an industrial-strength database engine such as DB2. The rationale behind the tight integration of these systems is given, details of the architecture are provided, and the decision points for each of the system components are discussed. DB2 XML's support for XML schema, how applications interface with DB2 to query and process XML query results, and the extensions to the DBMS

(database management system) utilities and tools to support native XML data are also described. Optimizations are critical to the success of any XML repository and query processing system. In "Cost-based optimization in DB2 XML," Balmin et al. describe a type of optimization that has yielded significant performance gains. This paper presents the extensions made to the DB2 UDB (DB2 Universal Database) compiler and its cost-based query optimizer to support XQuery and SQL/XML queries.

A native XML data store system, Natix, is the underlying system for the paper "The importance of sibling clustering for efficient bulkload of XML document trees" by Kanne and Moerkotte. As the title suggests, this paper discusses requirements for a bulkload component. It derives new algorithms for use in the bulkload operation specific to XML and presents the design of this component in the context of Natix.

To address the need to seamlessly query over both the structure and the text content of XML documents, the W3C is specifying full-text search querying in XML by adding extensions to XQuery and XPath. These extensions supplement the structured search inherent in XQuery with a wide range of full-text search primitives, such as phrase matching, keyword proximity, stemming, thesaurus, ranking, and scoring. The emerging Recommendation, XQuery 1.0 and XPath 2.0 Full-Text (XQFT),[38] adds full-text extensions to XQuery and XPath. As its title suggests, the paper "XQuery Full-Text extensions explained" by Amer-Yahia et al. explains the evolution and design principles behind this emerging Recommendation and illustrates its core features. The paper "Enhancing XML search with XQuery 1.0 and XPath 2.0 Full-Text" by Case provides further support for XQuery Full-Text from an end-user point of view. It provides the motivation for the XQFT extensions and describes how these extensions apply to a search system at the Library of Congress.

Connecting data to applications

In this category, we discuss the XML infrastructure that enables data from one application to be used or manipulated by another application.

The use of XML for industry-specific languages

Applications need semantic interoperability, that is, the ability for one application to operate on another application's data as if the data were its own. Although XML does not have semantics per se, it does provide the common infrastructure on which semantics are easily standardized and conveyed in a real application. Using this infrastructure and building on the core XML standards, a second tier of standards has emerged, which defines industry-specific languages consisting of sets of common data types. The XML Schema and Namespaces standards were instrumental in enabling the definition of semantics for the various industries. XML standards are being developed for almost all vertical industries, such as banking, biology, defense, insurance, and retail. XML.org lists a variety of vertical industries that use XML,[39] and more than 400 vertical and horizontal XML standards were listed on ZapThink's XML Standards Watch.[40]

Paper previews

As the title implies, the paper "Revolutionary impact of XML on biomedical information interoperability" by Shabo et al. describes the considerable impact that XML is having on the HCLS industry. It describes how XML is used to represent clinical data, clinical-trial data, and genomic data and how the use of XML is enabling integration across these three domains.

In "Emerging patterns in the use of XML for information modeling in vertical industries," Hinkelman et al. discuss the impact of XML on vertical industries and describe a set of XML usage patterns that has emerged based on the history of the industry in data interchange. They explore the use of XML for a sample set of industry-level standards: the Open Application Group incorporated (OAGi),

■ One of the most compelling aspects of XML's evolution was the intense and spirited collaboration of communities from different disciplines ■

which defines a Business Object Document; the Association for Cooperative Operations Research and Development (ACORD),[41] which is a leader in global insurance standards; MedBiquitous (MedBiq), which is an distinguished organization in professional medicine; the Open Travel Alliance (OTA),[42] an organization in the travel industry; and Health Level 7 (HL7)[43] for the HCLS.

Mapping technologies

The attempts to provide interoperability through integrated systems and industry standards notwithstanding, there is still a need for mapping representations between XML formats and between legacy data and XML. XML is the de facto standard for heterogeneous data exchange, representing diverse kinds of information, but it will not always be possible, or even desirable, to modify applications so that they work on the same format. There will always be autonomous sources of data for which governing standards bodies do not exist, and new applications will be developed with a unique view of the data. In Enterprise Information Integration (EII) and service-oriented architecture (SOA), these data sources and applications will evolve to interoperate with other data sources and applications, exchanging and operating on the same data. Furthermore, XML data and tooling must interoperate with legacy data.

Paper previews

Two papers in this issue address the issues of mapping data formats to XML and connecting to non-XML data. "XML mapping technology: making connections in an XML-centric world" by Roth et al. defines an extensible, model-driven architecture for mapping technology that enables the capturing, recording, and reuse of integration activity, while providing a rich platform for further research challenges in this area. This architecture is the foundation for the IBM Rational* Data Architect and satisfies requirements derived from EII and SOA examples, which are also presented in the paper. "Virtual XML: A toolbox and use cases for the XML world view" by Rose et al. supports the querying of non-XML data from a variety of formats as if they were XML. Virtual XML separates the concerns of representation from those pertaining to a common processing model. This paper gives an overview of how virtual XML can be realized. It describes the architectural components that enable applications to work with either XML or virtual XML and provides use cases demonstrating the applicability of virtual XML.

Connecting business to business with Web Services

As the use of IT in business has matured over the years, there has been a desire and a need to connect disparate systems and to take advantage of the functionality of legacy systems in modern-day applications. The ubiquity of XML and its ability to be used as an underlying specification language enabled a new generation of application-to-application communication, supporting flexible integration of heterogeneous systems in a variety of domains. This new generation of XML-centric interactions led to the birth of the Web Services platform, whose goal is to better take advantage of existing component frameworks, distributed services, and platform and network engineering resources.[44]

The Web Services technology suite is also an important enabler of the SOAs that are now being embraced by the entire IT industry. SOA is an abstract architectural concept founded on the idea of building software systems with uniformly described, discoverable services that interact in a loosely coupled way and can be composed.

The success of Web Services in this arena can be attributed to the nonproprietary nature of the underlying technologies as well as the loose coupling supported by the technology. Web Services specifications are being developed through industry partnerships and broad consortia such as W3C and OASIS (Organization for the Advancement of Structured Information Standards), and are thus based on standards and technology that are the foundation of the Internet, such as XML and HTTP (HyperText Transport Protocol). Furthermore, the participants in Web Services communications are loosely coupled and need only agree on the format of messages and their semantics. In contrast, prior technology (such as CORBA [Common Object Request Broker Architecture], DCOM [Distributed Component Object Model], and RMI [Remote Method Invocation]) required that communicating partners agree on an object model and significant aspects of an object management runtime.

Web Services had their beginnings in mid to late 2000 with the introduction of the Simple Object Access Protocol (SOAP),[45] Web Service Description Language (WSDL),[46,47] and Universal Description, Discovery and Integration (UDDI).[48] XML and HTTP are two basic technologies supporting the Web Services framework of specifications. In addition to its intrinsic relevance, XML is also the underlying specification language for all Web Services standards: XML provides the interoperable format to describe message content between Web Services, and is the basic language in which Web Services

specifications are defined.[49] SOAP, WSDL and UDDI form the initial set of specifications.

SOAP is an XML messaging protocol for basic service interoperability. It provides a uniform mechanism for exchanging structural and typed information encoded as XML. Hence, SOAP inherits the self-descriptive properties of XML. As such, XML supports communication between parties that have imperfect agreement on the format of messages and documents. Software can discover that the document being processed is almost what was expected and can adjust. Indeed, generic processing may be possible even on unexpected parts of the data. Web Services support communication on a global scale among parties that cannot always simultaneously revise their software to adjust to the evolution of data formats. XML's explicit tagging provides robustness in the face of changing versions, although this is an area where the technology is still maturing.

Although HTTP provides a commonly used interoperable protocol for SOAP, it works with any underlying communication protocol. Sending messages as plain XML ensures interoperability, requiring only that the processing middleware have basic abilities to parse and serialize XML. SOAP also provides much richer and more robust protocol extensibility based on XML and namespaces; that extensibility has been the basis for much of the rich function provided by other layers of the WS* stack (e.g., those enabling application-level security, reliable delivery, and long-running transactions). The result has been far greater interoperability of middleware platforms and the ability to scale to the wider networks enabled by the World Wide Web.

SOAP provides only the protocol for exchanging self-describing messages between services, but by itself does not provide any information about the services. WSDL is a common grammar for providing design-time description of services and messages. It defines a template to encode the information required by service clients to access and interact with the service. It describes what a Web service does, where it resides, and how it should be invoked.

UDDI provides a mechanism for clients to dynamically find other Web services, allowing businesses to dynamically connect to services provided by external business partners. UDDI assumes that requests and responses are UDDI objects that are communicated as SOAP messages.

SOAP, together with WSDL and UDDI, addressed many fundamental challenges of distributed computing by providing a uniform way of describing, locating, and accessing components or services within a network. The difference between Web Services and traditional approaches is primarily in the use of self-describing, platform-independent messages to enable loose coupling of aspects of the architecture, making the approach more dynamic and adaptable to change. However, these core technologies are only the beginning, and there are high expectations for the maturity of Web Services beyond basic message exchange, service description, and discovery. The interoperability of Web Services is being raised to a higher level of infrastructure services by the introduction of several other horizontal standards. Some of these new standards such as WS-Policy,[50] which adds to WSDL, extend the WS description specifications. Others, such as WS-Addressing,[51] WS-Security,[52] WS-Transactions,[53] and WS-ReliableMessaging[54] are protocol extensions built on SOAP.

■ Early attempts to merge XML and relational technology provided XML views of relational data or relational storage of XML data ■

Paper preview

As mentioned previously, one important feature of Web Services is their ability to be composed. BPEL4WS (Business Process Execution Language for Web Services), or BPEL[55,56] for short, provides a language for specifying how Web Services can be composed following a business-process-centric approach. BPEL supports two types of processes. *Executable processes* provide a full implementation of a service composition, which can be executed by any compliant process engine. *Abstract processes* use the same notation to specify only the mutually visible message-exchange behavior of the services without revealing their internal implementation. BPEL extends the Web Services interaction model, building on top of the WSDL service-interface

model, but does not specify the use of a particular protocol or discovery mechanism. BPEL is layered on top of several XML specifications, including WSDL 1.1, XML Schema 1.0, XPath 1.0, and WS-Addressing.

The paper "Business processes for Web Services: Principles and applications" by Khalaf et al. describes an example of what XML has enabled. It contains a brief overview of BPEL, focusing on the architectural drivers, usage, and kinds of applications (beyond traditional workflow) that XML has enabled. It discusses the potential of using abstract BPEL processes and presents case studies where these have been used, including "people-facing" workflows, grid computing, and automatic computing, specifically for dynamic provisioning.

In this section we have taken a very brief tour of the underlying technologies that are being developed to support XML processing as presented in the papers composing this issue. In the next section of the paper, we will look at the impact that XML and its enabling technologies are having and promise to have on society.

TECHNICAL AND CULTURAL CONSEQUENCES

The world in which we live is strongly affected, if not dominated, by a collection of amazingly varied and powerful technical, economic, political, and cultural norms and standards. Although it is easy to forget the impact of technological standards, their importance is recalled merely by contemplating the significance of agreements to drive on the same side of the road, standardized weights and measures, standards for a common electrical power grid, or TCP/IP. If we eliminated even a small number of technology standards, the world would be a very different place.

With respect to XML, we hope this paper, the others in this issue of the *Systems Journal*, and voluminous additional literature and experience establish that (1) XML is itself the product of a long history, (2) XML has very broad applicability, and (3) XML is achieving its potential through broad usage. In light of this, we contend that XML will take its place among the technical standards having the greatest import to the world. The authors believe that many computer scientists would agree with this observation.

Why do we think XML is so important? Perhaps, this is because we can describe XML as a universally applicable, durable "Code of Integration"; that is, a broadly applicable language for creating, storing, transmitting, accessing, and transforming information from a multitude of sources. It also naturally leads to a set of extensions which support semantically rich, tagged interchange and storage standards. Even though we would postulate that the von Neumann computing architecture, the techniques for analyzing algorithms, and the elegant structures that fuse complexity theory, formal language theory, compilers, and programming languages may be more important to computer science, and are in some sense considerably deeper accomplishments, the Code of Integration may be of comparable importance. This is because a Code of Integration can be applied coherently to a wide range of technical problems with a number of benefits, the most significant of which are the following:

1. *A consistent programming paradigm*—As programming involves the expression of rich interfaces and the techniques for manipulating information, the XML Code of Integration can be a basis for significant consistency, automation, and reuse in expressing software processes. Although XML does not purport to solve all problems, it does provide the language in which solutions can be expressed. This will increasingly improve the economics of IT-based automation.
2. *Simplicity of integration*—A Code of Integration can greatly reduce the cost of integrating and processing information. Just as common languages and vocabulary are among the most important cultural bases of civilization, agreement on a standardized form for defining information is exceedingly valuable to enable knowledge synthesis and systems integration. With a common way to express semantic information, there will be more (albeit incomplete) standardization of semantic information, paving the way to numerous benefits: information fusion, totally automated or semiautomated assembly of systems, greatly increased use of machine learning, computer-based reasoning, and more.
3. *Economies of scale*—Because of the universality of the Code of Integration, skills related to its use are widely useful. Significant investment can wisely be made in its implementations, leading to a high degree of optimization. Examples of this

include significant investments in high-performance XML software[57] and hardware (e.g., IBM's recent DataPower acquisition).[58]

Even beyond the probable importance of the XML Code of Integration as a primary technical standard, XML will become a defining element (albeit one that is behind the scenes) of economics, politics, and culture.

Economic impact

Because XML will become the key enabler of an economic system having vastly reduced transaction costs and economic rigidities, an ever more complete integration of markets and productive capabilities should be enabled. Distance, time, language and communication barriers will be vastly reduced. Although IT has served to integrate production capabilities previously, the cost has been high enough that only very large firms with great scale (e.g., Wal-Mart) could initially afford it. The development of HTTP and HTML over TCP/IP began the democratization process, yet many integration problems in business and government (e.g., health-care-related technology) are still exceedingly expensive to solve. XML and standards built on XML should make these problems tractable at a far more reasonable cost. To name just two effects, XML will accelerate the creation of new global markets, such as that created by eBay, and make the operation of worldwide supply chains far more comprehensive and efficient.

With IT becoming a very significant part of the world's capital stock (approaching 15 percent in the United States), it now has enormous leverage over the world's productive systems. Hence, integrating computerized processes and information will permit the creation of hybrid products and services that yield new innovations, higher quality, and lower costs. As one example of this, health-care systems are approaching a universal strategy for integrating information, resulting in the decline of duplicative care and medical errors.[59] The most significant factor in addressing this problem is the Code of Integration, due to the immense breadth and dynamism of the health-care challenge.

Some real-life implementations illustrate how the health-care industry is using XML. HL7, the most important standard for representing both clinical and administrative health-care data, uses XML. The Center for Information Technology Leadership at Partners Healthcare System has defined a model of the economic value of increasing interoperability by use of the health-care information exchange and interoperability (HIEI) model. The model predicts the value of different levels of information integration and defines "Level 4 HIEI" as machine-interpretable data that "uses the same messaging, format, and content standards, removing the need for customized interfaces."[60] The detailed, bottom-up analysis performed by the Center shows direct economic benefits of nearly $80 billion per year in the steady state, when the system has been operational long enough to recover from transient start-up costs. Although many of the standards referenced use XML as a basis, the medical community must do even more to create and deploy the higher-level semantic standards built on XML.

As a second example, we believe that more and more types of economic systems, whether large or small, will be able to adapt the principle of "continual optimization"; that is, systems will be able to gather input parameters needed for excellent decision making ever more easily, effect change to optimize behavior ever more cheaply, and close the feedback loop. High-performance computing is needed to achieve this, as is excellent mathematics, but the greatest challenge is the cost-effective ability to integrate and fuse information. As information is ever more consistently represented in XML and standardized XML schemas are created, the cost of information integration will drop greatly, facilitating optimization to the point where it can be performed ever more universally and continually.

In their book *Let Go To Grow*,[61] Sanford and Taylor look at the business implications of IT's ability to support the componentization of business. They make strong arguments that the ability of a business to disassemble itself into a collection of services that can be flexibly applied to numerous problems is a significant "game changer" for business.

Political impact

If economics and business are impacted as we have described, we believe this inevitably will have a political impact. Changes in wealth, the global distribution of wealth, the operation of markets, and the makeup of goods and services have political impacts. One can see just a few of these impacts in

the rapidly developing worlds of India and China (and the impacts of that development worldwide), on current thoughts relating to free trade in the services economy, in the tax system's ability to handle taxation in a global economy, and elsewhere. These are clearly political impacts, and XML is likely to accelerate them.

Cultural impact

Culture is defined as "the concepts, habits, skills, arts, instruments, institutions, and so forth, of a given people in a given period."[62] It is arguable that the aforementioned economic and political impacts constitute cultural impact. Change of this magnitude in issues related to wealth and globalization significantly impacts the habits and institutions of the many. The Code of Integration is likely to have many other societal impacts as well.

The codification of information will allow for much greater and more effective computer- and network-based instruction. For example, many believe there will be modularized educational modules, known as *learning objects*. The expression of these objects in the Code of Integration will permit each object to be well-integrated with other objects and customized to a student's needs.[63] The opportunity to provide highly configurable, customized education to everyone is world-changing.

Many problems traditionally in the sphere of artificial intelligence seem to require the creation and association of semantics with information. It is becoming clearer that semantics will come from multiple sources and that there will be many ontologies that provide useful meanings. As mentioned earlier, the Code of Integration provides a basis for handling large amounts of semantic tagging, with the strong possibility that the combined use of all the semantics may lead to breakthroughs in document understanding and reasoning. This is sometimes called the "combination hypothesis."[64] The Code of Integration can lead to vast increases in the domains to which computers are applied, making them an even more integral part of human discourse.

With the right language definitions, policies (within and among organizations) and even societal regulations and laws might come to be expressed formally, providing for much greater efficiency throughout society. If the tax code[65] or traffic regulations were formally expressed in XML, many cumbersome processes would be amenable to automation. Although this may seem unlikely, it is safe to predict that there will be a progression of automated expression of policy, and ultimately some aspects of law, over time. XML will almost certainly be at the center of this progression, as has already been demonstrated for defining policies[66] and regulations.[67]

The technical, economic, and societal consequences (as exemplified in the preceding discussion) will engender significant change in society. These consequences have sufficient substance and potential impact for us to conclude that XML is a technology and standard that will engender significant cultural impact.

CONCLUSIONS

XML is a technology with a profound opportunity to affect the world in which we live. It provides a catalyst for achieving semantic interoperability between systems and businesses, which could lead to a new generation of products, services, and information dissemination. Together with its associated tools, XML provides an infrastructure in which the standards and open-source communities, industry, and academia can define the semantics needed to provide this interoperability. If XML achieves its promise, it will take its place among the great technical milestones of computer science.

XML's potential, however, has only begun to be realized, and there are many issues yet to be resolved and risks to be avoided in order for XML to fulfill its promise. First, although standards are vital to the success of XML, any technology can collapse under the weight of too many standards being developed too soon. There must be a balance between simplicity and functionality and between hardening the standards and letting the usage of a technology dictate its priorities. As XML matures, the standards bodies must selectively focus their efforts on identifying the right set of standards, both for the core technology and for the industry-specific and domain-specific vocabularies. There are key issues in XML core technology, such as versioning, that must be solved in order to provide long-term scalability and flexibility. Furthermore, XML usage communities must define a cogent and architecturally sensible collection of worthwhile schemas that define the vocabularies of interchange to achieve the

promise of semantic interoperability. As work progresses on the next tier of standards, the benefits and risks brought by XML's growing complexity and diversity must be considered.

Second, systems that implement the tooling required to process and query XML must address inherent performance problems without destroying the requirement for ubiquity. Ubiquity provides a low barrier to entry, enabling implementors to rapidly prototype ideas within days. However, building implementations that are scalable, robust, and flexible enough to work across all types of applications still requires significant invention and development.

Finally, the deployment of XML must be done sensibly and realistically. XML is not meant to replace rich application modeling, nor should applications immediately convert their legacy data to XML when real-time performance is required and interoperability is not important.

Despite these issues, XML and its related standards and tools are already for the first time significantly enabling data interoperability, content manipulation, content sharing and reuse, document assembly, document security and access control, document filtering, and document formatting for all types of devices and applications. As such, XML is already having significant technical, economic, and societal consequences. We are pleased to present this special issue of the *IBM Systems Journal*, highlighting this important and fascinating technology, and we are grateful to have had some impact on its creation and growth.

ACKNOWLEDGMENTS

We gratefully acknowledge the assistance of the following people in IBM in reviewing this paper (listed in alphabetical order): Anders Berglund, Paco Curbera, Andrew Eisenberg, Dave Ferrucci, Brent Hailpern, Noah Mendelsohn, and Yael Ravin. Thanks are also due to Michael Sperberg-McQueen of the W3C.

CITED REFERENCES

1. *Extensible Markup Language (XML) 1.0* (Third Edition), W3C Recommendation (February 4, 2004), http://www.w3.org/TR/REC-xml/.
2. *Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML)*, ISO 8879:1986 (August 13, 2001), http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387&ICS1=35&ICS2=240&ICS3=30.
3. HyperText Markup Language (HTML) Home Page, W3C Interaction Domain, http://www.w3.org/MarkUp/.
4. *IBM Systems Journal* **45**, No. 2 (2006, this issue).
5. Apache.com—Providing Web Server and Network Security Resources, http://www.apache.com.
6. SourceForge.net, http://www.sourceforge.net.
7. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM* **13**, No. 6, 377–387 (July 1970).
8. JCC's SQL Standards Page, JCC Consulting, Inc., http://www.jcc.com/sql.htm.
9. J. Peckham and F. Maryanski, Semantic Data Models, *ACM Computing Surveys*, **20**, No. 3, 153–189 (September 1988).
10. R. G. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez, *The Object Data Standard: ODMG 3.0* Morgan Kaufmann Publishers, San Francisco, CA (January 2000).
11. V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl, "From Structured Documents to Novel Query Facilities," *Proceedings of the ACM SIGMOD International Conference on Management of Data* (May 1994), pp. 313–324.
12. T. Yan and J. Annevelink, "Integrating a Structured Text Retrieval System with an Object-Oriented Database System," *Proceedings of the 20th International Conference on Very Large Data Bases* (September 1994), pp. 740–749.
13. Charles F. Goldfarb, "A Generalized Approach to Document Markup," *SIGPLAN Notices* **16**, No. 6, 68–73 (June 1981).
14. *RFC 2045–Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, N. Freed and N. Borenstein, Editors, Network Working Group, Internet Engineering Task Force (November 1996), http://rfc.sunsite.dk/rfc/rfc2045.html.
15. B. J. Nelson, *Remote Procedure Call*, Ph.D. Dissertation, Report CMU-CS-81-119, Carnegie-Mellon University, Pittsburgh, PA (1981).
16. *Common Object Request Broker: Architecture and Specification, Revision 1.2*, OMG TC Document 93-12-43, The Object Management Group, Framingham, MA (1993).
17. W. Rosenberry, D. Kenney, and G. Fisher, *Understanding DCE*, O'Reilly & Associates (1992).
18. D. Reilly, "Introduction to Java RMI," Online publication (October 1998), http://www.javacoffeebreak.com/articles/javarmi/javarmi.html.
19. *Report From the W3C Workshop on Binary Interchange of XML Information Item Sets*, W3C Architecture Domain (September 2003), http://www.w3.org/2003/08/binary-interchange-workshop/Report.html/.
20. A. Orlowski, "Bruce Lindsay on Codd's Relational Legacy," *The Register* (April 25, 2003), http://www.theregister.co.uk/2003/04/25/bruce_lindsay_on_codds_relational/.

21. E. R. Harold and W. S. Means, *XML in a Nutshell*, Third Edition, O'Reilly Media, Sebastopol, CA (September 2004).
22. *Document Object Model (DOM) Level 1 Specification*, W3C Recommendation (October 1, 1998), http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/.
23. SAX official website, http://www.saxproject.org/.
24. *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation (November 16, 1999), http://www.w3.org/TR/xslt.
25. *XML Path Language (XPath) Version 1.0*, W3C Recommendation (November 16, 1999), http://www.w3.org/TR/xpath.
26. *XML Path Language (XPath) Version 2.0*, W3C Candidate Recommendation (November 3, 2005), http://www.w3.org/TR/xpath20/.
27. *XForms 1.0* (Second Edition) W3C Recommendation (March 14, 2006), http://www.w3.org/TR/2006/REC-xforms-20060314/.
28. *XSL Transformations (XSLT) Version 2.0*, W3C Candidate Recommendation (November 3, 2005), http://www.w3.org/TR/xslt20/.
29. *XQuery 1.0—An XML Query Language*, W3C Candidate Recommendation (November 3, 2005), http://www.w3.org/TR/xquery/.
30. *Information Technology—Database Languages—SQL—Part 14: XML-Related Specifications (SQL/XML)*, International Organization for Standarization (December 15, 2003), http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=35341.
31. XML Schema, World Wide Web Architecture Domain, http://www.w3.org/XML/Schema.
32. *Namespaces in XML*, World Wide Web Consortium Recommendation (January 14, 1999), http://www.w3.org/TR/1999/REC-xml-names-19990114/.
33. The Common Gateway Interface (CGI), http://hoohoo.ncsa.uiuc.edu/cgi.
34. Database Language SQL, http://www.itl.nist.gov/div897/ctg/dm/sql_info.html.
35. D. Ferrucci and A. Lally, "Building an Example Application with the Unstructured Information Management Architecture," *IBM Systems Journal* **43**, No. 3, 455–475 (2004).
36. T. R. Weiss, "IBM Releases Unstructured Data Framework Code As Open Source," *Computerworld* (January 23, 2006).
37. M. Rys, D. Chamberlin, D. Florescu, N. Agarwal, V. Arora, K. Beyer, S. Chandrasekar, D. Kossmann, S. Kotsovolos, V. Krishnamurthy, M. Krishnaprasad, Z. H. Liu, R. Murthy, F. Özcan, S. Saiprasad, E. Sedlar, A.-T. Tran, and B. Van der Linden, "XML and Relational Database Management Systems: The Inside Story," *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (2005), pp. 945–947, http://portal.acm.org/citation.cfm?id=1066157.1066298.
38. *XQuery 1.0 and XPath 2.0 Full-Text*, W3C Working Draft (April 4, 2005), http://www.w3.org/TR/2005/WD-xquery-full-text-20050404.
39. XML.org Focus Areas, http://www.xml.org/xml/focus_areas.shtml.
40. "BizTalk Talks Up Vertical Standards," eBizQ.net (June 11, 2001), http://www.zapthink.com/news.html?id=18.
41. ACORD Global Insurance Standards, Association for Cooperative Operations Research and Development, http://www.acord.org/home.aspx.
42. Open Travel Alliance, http://opentravel.org.
43. Health Level Seven, http://www.hl7.org.
44. R. Lai, "J2EE Platform Web Services," in *Web Services Architecture and Best Practices*, Addison Wesley, Reading, MA (2003).
45. Latest SOAP Versions, http://www.w3.org/TR/soap12-part1/.
46. *Web Services Description Language (WSDL) 1.1*, W3C Note (March 15, 2001), http://www.w3.org/TR/2001/NOTE-wsdl-20010315.
47. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Candidate Recommendation (March 27, 2006), http://www.w3.org/TR/wsdl20/.
48. OASIS UDDI, http://www.uddi.org/.
49. S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*, Prentice Hall PTR, Upper Saddle River, NJ (2005).
50. *Web Services Policy Framework* (March 2006), http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polfram/.
51. M. Gudgin, M. Hadley, and T. Rogers, *Web Services Addressing 1.0—Core*, W3C Proposed Redommendation (March 21, 2006), http://www.w3.org/TR/2006/PR-ws-addr-core-20060321/.
52. *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, OASIS Standard Specification (February 1, 2006), http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.
53. Web Services Transactions specifications (August 16, 2005), http://www-128.ibm.com/developerworks/webservices/library/specification/ws-tx/.
54. *Web Services Reliable Messaging* (February 2005), http://www-128.ibm.com/developerworks/webservices/library/specification/ws-rm/.
55. OASIS Web Services Business Process Execution Language (WSBPEL) Technical Comittee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.
56. *Business Process Execution Language for Web Services Version 1.1*, http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel/.
57. *XML Software Guide: Specialized XML Software*, http://www.wdvl.com/Software/XML/special.html.
58. Datapower—SOA Appliances, http://www.datapower.com/.
59. *Crossing the Quality Chasm: A New Health System for the 21st Century*, National Academy Press, Washington, D.C. (2000), http://www.iom.edu/Object.File/Master/27/184/Chasm-8pager.pdf.
60. J. Walker, E. Pan, D. Johnston, J. Adler-Milstein, D. W. Bates, and B. Middleton, "The Value of Health Care Information Exchange and Interoperability," *Health Affairs*, Web Exclusive (January 19, 2005), http://content.healthaffairs.org/cgi/content/abstract/hlthaff.w5.10.
61. L. S. Sanford and D. Taylor, *Let Go to Grow*, Prentice Hall, Upper Saddle River, NJ (2005).

62. *Webster's New World Dictionary,* College Edition, World Publishing, NY (1966).
63. "The Instructional Use of Learning Objects," David Wiley (Editor), Online Version (2001), http://www.reusability.org/read/.
64. A. Z. Spector, "Architecting Knowledge Middleware," WWW2002 Keynote Address (2002), http://www2002.org/spector.pdf.
65. US Internal Revenue Service and SGML/XML for Tax Filing (May 5, 2003), http://xml.coverpages.org/irs.html.
66. D. Agrawal, K.-W. Lee, and J. Lobo, "Policy-Based Management of Networked Computing Systems," *IEEE Communications* **43**, No. 10, 69–75 (October 2005).
67. C. Giblin, A. Y. Liu, S. Müller, B. Pfitzmann, and X. Zhou, "Regulations Expressed as Logical Models (REALM)," *Proceedings of the 18th Annual Conference on Legal Knowledge and Information Systems (JURIX 2005),* IOS Press, Amsterdam (2005), pp. 37–48.

Accepted for publication February 13, 2006.
Published online June 1, 2006.

Sharon Adler
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (sca@us.ibm.com). Ms. Adler is a senior manager at IBM Research in Hawthorne, New York. Her teams focus on research topics related to XML standards and Web Services. Before she rejoined IBM in 1999, she was a director of product management for publishing tools for Inso Corporation in Providence, Rhode Island. From 1985 to 1992, Ms. Adler held several key positions with IBM in Boulder, Colorado, where she was involved with the development of standards-based authoring and document management tools. Prior to that, she was a senior manager for Boeing Computer Services in Vienna, Virginia. Ms. Adler has been instrumental in the development of international computer standards for more than 25 years. She served on multiple ANSI/ISO standards committees, producing specifications such as ISO 8879 SGML and ISO/IEC 10179 DSSSL. From 1997 to the present, she has been chair of the XSL Working Group of the W3C, which produced the XSLT/XPath and related specifications as well as the XSL Formatting Objects specification. She also sits on the XML Coordination Group of the W3C and is a member of the board of directors of Idealliance, an industry association responsible for notable XML conferences held each year internationally.

Roberta Cochrane
IBM Software Group, 294 Route 100, Somers, New York 10589-0100 (bobbiec@almaden.ibm.com). Dr. Cochrane is a Senior Technical Staff Member in IBM's Software Group Strategy division. She is a leader in the delivery of advanced query technology to IBM's database products, providing many new advanced features over the last 15 years, including materialized views, triggers and constraints. She has conducted extensive research in active database systems and played a major role in the definition of the SQL3 standard for triggers and constraints. Dr. Cochrane is a member of the IBM Academy of Technology, a Master Inventor, and was one of IBM's 2002 YWCA TWIN awardees, honoring women in industry. She received a B.S. degree in computer science and mathematics from James Madison University in Virginia and a Ph.D. degree in computer science from the University of Maryland at College Park.

John F. Morar
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (morar@watson.ibm.com). Dr. Morar received a Ph.D. degree in experimental solid-state physics from the University of Maryland in 1982. After joining IBM, he spent two years in residence at the National Synchrotron Light Source project at Brookhaven National Laboratory, where he used soft X-ray spectroscopy to probe the outer few atomic layers of semiconductors. Over the following eight years, he did research on metastable semiconductors using molecular beam epitaxy. Dr. Morar spent seven years in computer virus research, managing the Anti-Virus Technology and Systems group. He contributed to numerous releases of the IBM Anti-Virus and Digital Immune System software, which was built to find, analyze, and automatically distribute cures for new computer viruses faster than the virus itself could spread. He has written 70 articles in peer-reviewed scientific journals and has contributed to IBM's patent portfolio in the areas of device processing, computer virus detection, Web services, and economic systems. Dr. Morar currently manages a group that focuses on the application of service-oriented architectures and the use of Web services both within and between enterprises.

Alfred Spector
IBM Software Group, 294 Route 100, Somers, New York 10589 (aspector@us.ibm.com). Dr. Spector is Vice President of Strategy and Technology for the IBM Software Group, where he is responsible for such diverse activities as standards, software-development methodologies, advanced technology, leading-edge technical engagements, and strategy. Previously, he was a vice president in the Research Division, where he was responsible for setting IBM's worldwide services and software research strategy and overseeing the work of more than 1300 researchers worldwide. In previous assignments within IBM, he was the general manager of marketing and strategy for IBM's middleware business and the general manager of IBM's transaction software business. Dr. Spector was also founder and CEO of Transarc Corporation, a pioneer in distributed transaction processing and wide-area file systems, and a tenured faculty member of the Carnegie Mellon University computer science department. He received a Ph.D. degree in computer science from Stanford University and an A.B. degree in Applied Mathematics from Harvard University. He is a member of the National Academy of Engineering, and he is recognized for his contributions to the design, implementation, and commercialization of reliable, scalable architectures for distributed file systems, transaction systems, and other applications. Dr. Spector is also an IEEE Fellow and the recipient of the IEEE Kanai Award in distributed computing. ■