The Impact of XML on Databases and Data Sharing

Len Seligman and Arnon Rosenthal
The MITRE Corporation
{seligman, arnie}@mitre.org

Abstract

XML (eXtensible Markup Language) has received a great deal of attention as the likely successor to HTML for expressing much of the content of the Web. However, XML also has the potential to benefit databases and data sharing by providing a common format in which to express data structure and content. By describing the broad challenges presented by data access and (especially) data interoperability, we highlight both the potential contributions of XML technologies to databases and data sharing, and the problems that remain to be solved. In some areas, XML promises to provide significant and revolutionary improvements, such as by increasing the availability of database outputs across diverse types of systems, and by extending data management to include semi-structured data. While some of the benefits of XML are already becoming apparent, others will require years of development of new database technologies and associated standards.

Key words – XML, Internet data management, Web databases, semi-structured data, interoperability, database management systems, middleware

Background: The HTML Data-Sharing Dilemma

The World Wide Web has been a great boon to businesses needing to share data across distributed, heterogeneous hardware and software environments. Rather than looking for data from a single controlled location, businesses increasingly use a single browser interface to access data from sources around the world.

However, there is a problem with the current Web model of diverse data access. At present, both the Web and most intranets use HTML (HyperText Markup Language) as the predominant format for publishing information. HTML is an easy-to-understand language for presenting and displaying data, and this ease of use was fundamental to its rapid spread as the basis of the Web. Unfortunately, as a universal way to represent information from diverse sources, HTML suffers from several serious shortcomings, including:

- **Presentation (not content) orientation**. HTML uses presentation-oriented mark-up tags (e.g., "<H2>" for a second level heading) that tell a browser how the to display data to human users. HTML gives no information about the *meaning* of the data (e.g., "this is a warranty description for a retail product"). Because HTML focuses on the computer-to-human interface, it has limited value as a data format for computer-to-computer applications such as transferring information between databases.
- **Tight coupling of content and presentation.** In HTML, presentation-oriented tags are intermingled with data content. Because content and presentation are tightly coupled, HTML does not effectively support alternate presentations of the same underlying content (e.g., for different audiences or media).
- **No extensibility.** HTML has a fixed set of markup tags. There is no support for creating new, application-specific tags (e.g., Patient_ID for medical applications) that help applications communicate about the data content.
- **No data validation capabilities.** HTML does not help applications validate data as it is entered or imported.

While HTML was instrumental in creating a new perception in businesses that data can and should come from many diverse sources, it is poorly suited for building systems in which applications (not users) interpret the data. Because of these limitations, it is cumbersome to build and maintain a complex data access application (e.g., a comparison-shopping agent) based on HTML documents. Such HTML-based applications require brittle, handcrafted code to "screen scrape" information from web pages (e.g., "find the third column and second row of the fourth HTML table in this page; that's usually the price"). The need for such convoluted techniques is especially frustrating given that much information on the Web is derived from structured databases. The structural information contained in those databases would vastly simplify the extraction of data by applications, but most of it is thrown away when the information is published in HTML on the Web.

Enter XML

To address the limitations of HTML for representing Web content, the same World Wide Web Consortium (W3C) (see www.w3.org) that is responsible for HTML standards chose in the late 1990s to develop a new standard that would eliminate these shortcomings. The goal was to create a language that was similar to HTML in format, but more extensible and capable of clearly separating content and presentation. The result of this effort was called *XML*, or *eXtensible Markup Language*. The W3C defined XML as a simplified subset of an earlier document-structuring language called *SGML*, or *Standardized Generalized Markup Language*. SGML had been used to create some large information collections such as encyclopedias and multi-volume case law books, but its complexity had discouraged widespread adoption.

As a standard, XML eliminates many of the data representation problems that plague HTML. The advantages of XML include:

- **Independence of content and presentation.** XML addresses content only. With XML, the presentation of data is handled separately with either Cascading Style Sheets (CSS) or the Extensible Stylesheet Language (XSL), both of which are also W3C standards.
- **Extensibility.** XML allows the creation of an indefinitely large number of new data labels or tags. This means that with XML, information publishers are free to invent their own tags as needed for their particular applications, or to work together with other organizations to define shared sets of tags that promote interoperability.
- **Validation.** XML documents can be associated with a Document Type Description (DTD), which defines a structure for conforming applications. This allows applications that import data to validate that data for conformation to the DTD.

Although it is a young standard, XML is already having a significant impact in intranets and on the Web. Businesses like XML, because it eliminates many of the costly and fragile workarounds needed to represent rapidly changing data in HTML. Developers of new applications appreciate the extensibility of XML, which makes it easy to adapt to new uses, and communities that must share common data (e.g., the chemical industry) like XML for its ability to support well-defined, common data representations. As a consequence of widespread acceptance, there is a vibrant XML marketplace providing inexpensive tools for preparing, validating, and parsing XML data. It is likely that XML will continue to have strong support over the next decade in terms of tools, standards, and users.

As a result of fixing the known problems with HTML, XML offers several benefits:

- *Support for multiple views of the same content for different user groups and media.* As the chairman of Adobe said in his keynote at XML '98, "To date we have had a separate workflow for each output format…We are switching to XML because it will allow us to have a single workflow with multi output."

- *Selective (field-sensitive) query over the Internet and intranets.* For example, one could search for documents with an Author field that contains “Kissinger”, and the search would not return documents that merely mention Kissinger unless it was within an Author tag.[1]
- *The semantic structure of web information becomes more visible.* There will be less need for brittle screen-scraping parsers.
- *A standard infrastructure for data and document interchange.* This includes freely available parsers that can validate conformance with a DTD.

Several related standards will greatly increase the utility of XML for data sharing and management. These include:[2]

- **eXtensible Stylesheet Language.** XSL allows one to specify rules that indicate how to transform an XML document. This transformation could be to a presentation format (e.g., HTML or PDF), or to an alternate representation of the content (e.g., an XML document with a different DTD). As a result, one can manage content independently of its presentation and use different XSL stylesheets to produce alternate views of that content.
- **Document Object Model (DOM).** Brackets and backslashes are uninteresting. The XML standard gives enough information to drive a parser, but does not specify the form of the parser’s output, either as a data structure or in terms of operations. We hope (and predict) that most XML-related work will be expressed in terms of tree and graph abstractions that hide these details. In particular, DOM provides a tree-based application programming interface to XML. The DOM API provides methods for traversing the tree, such as getParentNode(), getChildNodes(), etc.
- **XML Query Language.** The W3C has formed a query language working group whose goal is to provide flexible query facilities to extract data from collections of XML documents as well as non-XML data viewed as XML via a mapping mechanism.
- **XML Schema.** XML DTDs were intended originally for document management and provide inadequate support for modeling data. To address this, the W3C XML Schema working group is adding data types, relationships, and constraints.

To sum up, XML and related technologies offer an excellent combination of benefits and simplicity. XML has quickly achieved wide vendor acceptance, in browsers, document preparation tools, and as a database input/output format, and XML based products are already emerging. We now consider the likely impact of XML on database management systems.

XML and Databases

XML and database technology are more complementary than competitive. By revealing structure in non-tabular data, XML enables that structure to be exploited by database technologies. Conversely, database techniques can improve the integrity and semantic integration of sets of XML resources. This synergy has been recognized by W3C working groups, which are adapting database ideas to provide XML schema and query technologies.

Below, first we look at how XML impacts information systems built around relational databases. Then we examine prospects for applying database technology to semi-structured data, including XML.

XML for Well Structured Data

[1] Note that this capability depends on agreements (at least within a particular community) on the meaning of certain widely used tags (e.g., Author).

[2] For current information on these standards and the tools that support them, we refer the reader to http://w3.org/xml and http://www.xmlinfo.com.

Today’s dominant database management systems (DBMS) are broad, mature products that will be hard to displace, and are likely to dominate management of critical enterprise data, even in newer areas such as electronic commerce.

The database world today emphasizes high-integrity, read-write processing of regularly structured data. Table definitions or object types model the external world (e.g., Customer, Account) rather than information artifacts (e.g., Document, Title). The regularity of relational tables (all rows in a table conform to a structure that is known in advance) makes interfaces simpler and more static. Modern databases provide efficient update and retrieval of huge amounts of data, highly tuned transaction processing, recovery, indexes, integrity, and triggers. They also support complex queries with good performance and explicit semantics (as opposed to less formal “relevance” criteria). Even load and dump utilities are optimized for performance. For the data that supports critical but routine processing, these requirements will continue.

For applications involving regularly structured data, XML tools will not replace such DBMSs. There is simply too much functionality to implement rapidly, and migration would be too traumatic. Also, the need for efficiency argues against a verbose format like XML for internal storage. XML data can be stored directly in relational systems (e.g., by encoding its graph), but relational operators are insufficient for the manipulations users want. Still, XML is rapidly gaining a role for even highly structured data—as an interface format.

Whenever required (e.g., to publish the information on the Web, or send it over the wire for e-commerce), XML versions of appropriate views of the data can be created. Sometimes, these will be created on the fly, in response to user queries, while for other applications (especially where the data is nonvolatile or users don’t need completely current information) the XML may be created in advance (e.g., daily). Already, major vendors (e.g., Oracle and IBM) have released tools for creating XML extracts of databases, and these tools will become more powerful. Import utilities are also being customized to accept XML. An advantage of XML output is that it includes its own schema information. For anyone who understands the tags used, the information is self-describing.

A second advantage is that XML can naturally represent structures that are more complex than flat tuples. XML offers a way to serialize objects (verbosely), e.g., for e-commerce artifacts.

XML for Document and Other Semi-structured Data

Relational DBMSs hold a small fraction of the world’s data, for several good reasons. They tend to require professional administration. They require data to be tabular (i.e., flat) and to conform to a prespecified schema, which promotes integrity but discourages rapid development and change. Frequently their purchase prices are high. For document data, *semi-structured* data models (including but not limited to XML) offer promise of addressing all but the first objection. But for now, they lack the features needed for robust systems.

As semi-structured data becomes more widely shared, and is processed more automatically, organizations will need data management capabilities (e.g., powerful queries, integrity, updates, and versioning) over this data. Object-oriented database vendors have begun addressing this need by offering XML data managers layered on top of their tools.[3] Relational systems are also moving in this direction.

The long-term direction for database support of XML and other structured media, though, may be best seen in the database research community. Many researchers are improving the ability to interface with semi-structured data. Some have developed “wrappers” that mine data with implicit structure and make the structure explicit and machine-readable (e.g., [1, 9]). Other projects (e.g., [10]) are investigating the use of graph-structured data models (including XML) as a common representation for heterogeneous information sources, including both structured and semi-structured sources.

[3] E.g., Poet <http://www.poet.com> and eXcelon <http://www.exceloncorp.com>. In fact, the latter company (formerly Object Design) has redefined its identity to focus on XML data management rather than object databases.

Finally, several groups are developing prototype DBMSs for semi-structured data, with new query languages and optimization techniques [4, 6, 8]. These researchers have converged on the use of graph-structured data models (especially XML), in which all data is represented in labeled directed graphs. DBMSs for semi-structured data are intended to handle data from, e.g., ordinary documents, web sites, and biochemical structures. In these arenas, it is often infeasible to impose any schema (even an object schema) in advance. Data may be irregular, or the structure may evolve rapidly, lack an explicit machine-processable description, or be unknown to the user. Even when the structure is known, it is frequently seen as hierarchical, and it is advantageous to have operations that understand the hierarchy.

Compared with an ordinary relational or object database, semi-structured databases offer several capabilities:

- *Hierarchical model*. Some data is most naturally modeled as a hierarchy. For this data, hierarchical languages simplify data manipulation. (However, if the hierarchy is different from the way your application views the world, it can be very awkward to work with.)
- *Tag and path operations*: Conventional database languages allow manipulation of element values, but not element names. Semi-structured databases provide operators that test tag names (e.g., “find all documents that have a ReferenceList or Bibliography element”). They also include operators that manipulate paths. For example, one can have path expressions with wild-cards, to ask for a Subject element at any depth within a Book element.
- *Irregular structure*. The contents of a node need not conform to a type description, i.e., the node need not have a predetermined attribute list. Relational systems could model this by having missing attributes as nulls. However, SQL is awkward with null values, and current storage structures can have excessive overhead.
- *Structure that varies, or is not known in advance*. For example, documents differ in their structure, especially if assembled from multiple sources. The structure may also change, e.g., when figures are added. For web resources, even those with rigid structure, that structure might not be explicitly recorded on-line. This is a major challenge, in terms of providing a user interface, and efficient processing.
- *Sequence*. Unlike tables, document sections are ordered, so sequence must be represented. In query processing—especially joins and updates—sequence introduces significant complexity.

These features have been demonstrated in research prototypes and are likely to appear in commercial products in the next few years. It is unclear how the market will be split among the three approaches: (1) layered over an object database, (2) layered over a relational database, or (3) directly over some new data manager.

XML and Data Sharing

Some industry observers have heralded XML as the solution to data sharing problems. For example, Jon Bosak [3] indicates that XML (together with XSL) will bring “complete interoperability of both content and style across applications and platforms.” In reality, data sharing will continue to be a significant challenge. XML technologies will make a positive impact, but they give only partial or indirect help for many of the toughest issues.

This section examines the likely impact of XML on data sharing. First, we list six functional tasks that data sharing must accomplish, regardless of architecture and formalism. We then use this list of data sharing functions as a framework for answering the questions: Where can XML help? What portion remains unsolved?

Generic Tasks for Data Sharing

Users want seamless access to all relevant information about the real world objects in their domain. The literature provides several general architectures including the following [5]:

- *Multi-databases.* Applications work directly with source databases, and each application does its own reconciliation.
- *Data warehouses.* Administrators define a “global” schema for the data to be shared. They provide the derivation logic to reconcile data and pump it into one system; often the warehouse is read-only, and updates are made directly on the source systems.
- *Data marts.* This is like data warehousing, except that customized views of the global warehouse called “data marts” are maintained for particular communities.
- *Federated databases.* This can be thought of as a virtual data warehouse¾i.e., the global schema is not populated. The source systems retain the physical data, and a middleware layer translates all requests to run against the source systems.

Regardless of which distributed architecture is chosen, someone (standard-setter, application programmer, or warehouse builder) must reconcile the differences between data sources and the consumer’s view of that data, in order for data to be shared. Applications need to be insulated from several forms of diversity, in order to make their requests. (We assume the insulation mechanisms also provide an interface for programmers to look under the hood.) *Data reconciliation* must overcome challenges at multiple *levels.*

1. *Distribution.* Many protocols are available, and they may be hidden within middleware products. Options include CORBA, DCOM, and HTTP.

2. *Heterogeneous data structures and language* (i.e., heterogeneous DBMSs). Standards like ODBC and middleware products increasingly handle this difficulty. However, advanced features (e.g., triggers) may not be visible at the middleware tier, and there may be an efficiency loss.

3. *Heterogeneous attribute representations and semantics*. Integrators often must reconcile different representations of the same concept. For example, one system might measure Altitude in meters from the earth’s surface while another measures it in miles from the center of the Earth. In the future, interfaces may be defined in terms of abstract attributes with self-description, e.g., Altitude(datatype=integer, units=miles). Such descriptions enable mediators to shield users from these representational details [13].

   Differences in semantics (i.e., meaning) are more challenging than representation heterogeneity. For example, two personnel systems include an Employee.Compensation attribute. One might be gross salary plus annual bonus, while the other is net salary (after taxes) without bonuses, but including the value of insurance premiums paid by the employer. Semantic differences can sometimes be resolved through transformations (e.g., rederiving gross salary). However, often no automated transformation is possible, and the integrator must simply indicate whether he can use a particular attribute for a particular purpose.

4. *Heterogeneous schemas*. The same information elements can be assembled into many different structures. Various application communities are therefore attempting to define standard interface schemas, e.g., as Unified Modeling Language models, SQL tables, or XML DTDs. Such standards reduce the number of external interfaces that a system must support. Sometimes they are also suitable for use internal to an application; however, one should not force an application’s functions to use an unnatural schema just to avoid occasional translation when passing data outside.

As the previous problems become better solved, researchers are turning to the next two types of reconciliation. The treatments are generally application dependent; the tool developer’s task is to make it easy for administrators to specify the desired policies. System designers should allow the reconciliation rules to be flexible, modular, and displayable to domain experts who are not programmers.

5. *Object identification.* This type of reconciliation determines if two objects (usually from different data sources) refer to the same object in the real world. For example, if CriminalRecords has (John Public, armed robber, born 1/1/70) and MotorVehicleRegistry has (John Public Sr., license plate "JP-1", born 9/9/69), should a police automobile-check view consider the tuples to refer to the same person and return (John Public Sr, armed robber)?

6. *Data value reconciliation:* Once object identification has been performed, the different sources may disagree about particular facts. Suppose three sources report John Public Sr. to be respectively: 180, 187, and 0 centimeters. What value or values should be returned to the application?

Where Can XML Help?

Given the reference model of the previous section, we now consider where XML can help improve data sharing.

Distribution (level 1). XML provides some assistance with distribution by supporting mechanisms for remote function invocation across the web. For example, Simple Object Access Method (SOAP) uses XML and HTTP as a method invocation mechanism.[4] SOAP mandates a small number of HTTP headers that facilitate firewall/proxy filtering and specifies an XML vocabulary for representing method parameters, return values, and exceptions.

In addition to mechanisms for sending method invocations and results across the distributed networks, data sharing requires functions that do the actual work of creating, sending, and reading interchange files. For example, players must know the syntax and exact semantics for "Send." (For database-oriented data sharing, one may use submittal protocols like ODBC.) XML does not provide these functions, but presumably they will be provided by middleware vendors, often layered on top of XML-based invocation mechanisms such as SOAP.

Heterogeneous data structures and language (level 2). XML provides a neutral syntax for describing graph-structured data as nested, tagged elements with links. Since one can transform diverse data structures into such graphs, XML offers a way to represent heterogeneous data structures. Adding DOM (and perhaps a query language) provides the needed operations for accessing these data structures.

Microsoft's ODBC and OLE/DB make available analogous functionality for accessing flat and nested data sources (plus a model for describing servers' search capabilities). Among relational databases, ODBC, OLE/DB, and native interfaces seem to offer extra power and fairly low cost. (OLE/DB seems relevant mostly when the target is on a Microsoft platform).

XML will shine in other settings. When a source or recipient sees the world hierarchically (e.g., for formatted messages), XML technologies can help in restructuring the information between relational and hierarchical formalisms (once one understands the mappings needed for levels 3 and 4). For example, the U.S. military and its coalition partners are beginning to transition their Message Text Format (MTF) to an XML-based infrastructure. XML's strong base of tools (freeware and COTS) gives great flexibility. and has greatly reduced development costs. In another example, our research group found XML to provide a useful common data model for integrating two semistructured text data sources, Periscope and the CIA's unclassified World Factbook [9].[5]

Heterogeneous attribute representations and semantics (level 3).

One approach to this challenge is to develop standards within a community of interest. Past standardization efforts have often suffered from a lack of tools for informing stakeholders about

[4] http://search.ietf.org/internet-drafts/draft-box-http-soap-01.txt

[5] http://www.periscope.usni.com/demo/demoinfo.html and http://www.odci.gov/cia/publications/factbook

relevant standardized concepts. XML and the web-based infrastructure on which it is built provide opportunities for addressing this concern.

Sometimes a source and receiver will agree on the semantics of a concept (e.g., Altitude) but not on its representation. A natural solution is for the source and recipient to describe representation details explicitly, e.g., "<Altitude>500<LengthUnit>miles</LengthUnit></Altitude>", showing their assumptions as subsidiary elements. One would then want standards to make this subsidiary information sharable, e.g., what is "LengthUnit". But there is still considerable work that the standards do not support. The extra elements should typically be virtual, derived from profiles of sources rather than included with each value; also mediation is necessary to compare and insert appropriate translations [13]. Neither of these tasks is directly supported in the standard.

XML contributes toward making the above descriptions more manageable (compared with ordinary relational DBMSs). It provides an easy way to attach subsidiary elements, a good means of disambiguating names provided by different authorities (i.e., XML Name Spaces), and most important, the ubiquity and toolsets that attract interest from domain communities.

Heterogeneous schemas (level 4). XML DTDs defined by various communities provide a neutral model for describing their data structures. Communities developing standard DTDs include electronic commerce, healthcare, and data warehousing vendors. Such DTDs will reduce diversity of interfaces and ease data sharing. One effort of interest is BizTalk, an XML repository effort being spearheaded by Microsoft (but supported by numerous companies) that facilitates interoperability by mapping among XML elements and models.

The model standardization problem is *not* intrinsically simpler in XML, compared to object systems. However, XML's likely ubiquity and cheap tools have sparked enthusiasm for defining standards in many communities. Organizations will need to map their schemas (and non-DBMS data) to the standards; this market may spur creation of a new generation of DTD (i.e., schema) integration tools.

In the developer community, there is increasing awareness that schema diversity will be a serious problem even if XML DTDs are widely used [7]. [11] raises this issue for e-commerce, and describes a model with roughly the same layers as ours. However, rather than one standard schema at each layer, we expect several to be viable. (Communities' interests overlap, and within the overlap, each community will follow its own path.) The schemas will also be incomplete, so one will need an easy way to supplement them, to meet a pair of organizations' particular needs.

Hence there will be a great need to create mappings between schemas. SQL is a reasonable choice to express those mappings, especially since it now supports user-defined functions. It will be some time before XML query processors become strong enough to handle such mappings.

Object identification (level 5). Improvements in identifying data elements (at level 3) can remove one source of misidentification of objects (e.g., in a Payment, is the date in US or European format?). Also, XML makes it easy to attach uncertainty estimates (as subsidiary elements) to any output (if the recipient is prepared to interpret them).

Data value reconciliation (level 6). Many strategies for data value reconciliation depend on having metadata (e.g., timestamp, source-quality) attached to the data . XML helps in attaching such annotations. XML also makes it easy to return a set of alternative elements for an uncertain value (again assuming the recipient is prepared to use such output).

To sum up, XML and related tools provide considerable help with data interoperability, especially at levels 1 - 4. However, a key issue is not resolved by XML: how to best record the intersystem mappings. First, analysts need tools to help them identify candidate relationships across systems. Second, mappings should be specified declaratively and not in procedural code about which optimizers and other automated tools cannot reason. SQL is an example of such a declarative language and it is hoped that the upcoming XML Query Language will be also. Finally, we need tools (such as

BizTalk) to record intersystem relationships and mappings to any community standard DTDs or schemas, and to make them available for reuse [12].

Another issue is that XML-based interoperability often means using XML as a message syntax for bulk transfers among systems. However, bulk transfer is a weak form of interoperability that supports one kind of request: “Generate the standard message, please”. There is no built-in support for ad hoc query, update, or a subscription to be notified of specific changes. These capabilities are offered by commercial relational databases, but it will be some time until XML tools offer similar features.

Conclusion

As with any hot new technology, XML has generated exaggerated claims. In reality, XML does not come close to eliminating the need for database management systems or solving large organizations’ data sharing problems. Nevertheless, XML is an important technology with significant benefits in three areas:

- as an input/output format (particularly in web environments)
- for managing and sharing semi-structured data that is difficult to describe with a prespecified schema, and
- as a ubiquitous and inexpensive infrastructure for exchanging self-describing messages.

In addition, the enthusiasm surrounding XML is motivating some communities to agree on standards where previously they could not. Also, the Web makes it easy to share information about community standards, thereby dramatically increasing their impact.

To realize XML’s full potential, further progress is needed. We need better tools, particularly for managing semi-structured data, for reconciling heterogeneous attributes and schemas, and for performing object identification and data value reconciliation. Researchers and vendors have made significant strides, but much more remains to be done.

Acknowledgments

The authors thank Terry Bollinger and Frank Manola for their helpful comments and Roger Costello for assistance with an earlier version of this paper. Any remaining problems are the responsibility of the authors.

References

[1] B. Adelberg, “NoDoSE: A Tool for Semi-automatically Extracting Structured and Semistructured Data from Text Documents,” *SIGMOD International Conference on Management of Data,* 1998.

[2] J. Blakeley. “Data Access for the Masses through OLE DB,” *SIGMOD International Conference on Management of Data*, 1996, http://www.microsoft.com/data/oledb.

[3] J. Bosak, “Media-independent Publishing: Four Myths about XML,” *IEEE Computer*, October 1998.

[4] P. Buneman, S. Davidson, G. Hillebrand, D. Suciu, “A Query Language and Optimization Techniques for Unstructured Data,” *SIGMOD International Conference on Management of Data,* 1996.

[5] A. Elmagarmid, M. Rusinkiewicz, A. Sheth, Management of Heterogeneous and Autonomous Database Systems, San Francisco: Morgan Kaufmann Publishers, 1999.

[6] M. Fernandez, D. Florescu, J. Kang, A. Levy, D. Suciu, “Catching the Boat with Strudel: Experiences with a Web-site Management System,” *SIGMOD International Conference on Management of Data,* 1998.

[7] A. Gonsalves, L. Pender, “Schema Fragmentation Takes a Bite out of XML”, *in PC Week Online, May 3, 1999.* http://www.zdnet.com/pcweek/stories/news/0,4153,401355,00.html.

[8] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, “Lore: A Database Management System for Semistructured Data,” *SIGMOD Record*, 26(3), September 1997.

[9] D. Mattox, L. Seligman, K. Smith, “Rapper: A Wrapper Generator with Linguistic Knowledge,” *Workshop on Web Information and Data Management*, Kansas City, 1999.

[10] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, “Object Fusion in Mediator Systems,” *International Conference on Very Large Databases*, September, 1996.

[11] L. Paul, “Are XML Standards Too Much of a Good Thing” , *in PC Week Online,* Apr 12, 1999, http://www.zdnet.com/pcweek/stories/news/0,4153,398134,00.html.

[12] A. Rosenthal, E. Sciore, S. Renner, “Toward Integrated Metadata for the Department of Defense”, *IEEE Metadata Workshop,* Silver Spring, MD, 1997. At http://computer.org/conferen/proceed/meta97/papers/arosenthal/arosenthal.html.

[13] E. Sciore, M. Siegel, A. Rosenthal, “Using Semantic Values to Facilitate Interoperability among Heterogeneous Information Systems”, *ACM Transactions on Database Systems*, June 1994.

Biographical Sketches

Len Seligman is a Principal Scientist in MITRE’s Intelligent Information Management and Exploitation group in Reston, Virginia. His experience includes developing, researching, and managing advanced information systems. He has a Ph.D. from George Mason University and is an Associate Editor of *SIGMOD Record*, the quarterly bulletin of the ACM Special Interest Group on the Management of Data. Dr. Seligman’s interests include heterogeneous databases, semi-structured data, and large-scale information dissemination.

Arnon Rosenthal is a Principal Scientist at The MITRE Corporation in Bedford, Massachusetts. He has worked in recent years on data administration, interoperability, distributed object management, migration of legacy systems, and database security. He is Vice Chair for “System Administration, Ease of Use & Database Design” for the International Conference for Data Engineering. 2000. At Computer Corporation of America (Xerox AIT), ETH Zurich, and earlier, his interests included query processing, database design, active databases, and discrete algorithms. He holds a Ph.D. from the University of California, Berkeley.