The Impact of XML on Databases and Data Sharing

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

Abstract

The Extensible Markup Language (XML) is receiving much attention as a likely successor to HTML for expressing much of the Web's content. In addition, XML can benefit databases and data sharing by providing a common format in which to express data structure and content. But like many new technologies, XML has raised unrealistic expectations. We give a brief overview of XML and offer opinions to help separate the benefits from the hype. 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.

This paper will first describe the limitations of current Web technologies for data sharing, and how XML addresses them. Next, it assesses the impact of XML on data management for both well structured and more loosely structured data. The longest section outlines the challenges of data interoperability and then describes which of these challenges XML does (and does not) address. 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, data interchange

1. Background: The HTML Data-Sharing Dilemma

The World Wide Web has been a great boon to businesses that need to share data across distributed, heterogeneous hardware and software environments. Rather than looking for data from one location, businesses increasingly use a 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, 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 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"). HTML focuses on the computer-to-human interface and has limited value as a data format for computer-to-computer applications such as transferring information between databases. In addition, because of the tight coupling of content and presentation, 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 (or via dynamic scripting languages such as ColdFusion and ASP) on the Web.

Below, Section 2 gives a brief survey of XML and related technologies. Section 3 discusses how these developments are likely to affect databases. The heart of this paper is Section 4, which describes six issues that must be resolved to achieve interoperability, and examines how XML affects each issue.

2. Enter XML

To address the limitations of HTML for representing Web content, the same World Wide Web Consortium (W3C) that is responsible for HTML standards decided to develop a new standard. The goal was to create a language similar to HTML in format, but more extensible and capable of clearly separating content and presentation. The result of this effort was the Extensible Markup Language (XML). XML is a simplified subset of the Standard Generalized Markup Language (SGML), an earlier document-structuring language. SGML has been used to create some large information collections such as encyclopedias and multi-volume case law books, but its complexity has discouraged widespread adoption.

XML eliminates many of the limitations of HTML. First, it cleanly separates content and presentation. XML itself addresses content only, while presentation is handled separately—e.g., by Cascading Style Sheets, Extensible Stylesheet Language (XSL), or XHTML. Second, XML is extensible. This means that with XML, information publishers are free to invent their own tags as needed for particular applications, or to work together with other organizations to define shared sets of tags that promote interoperability. Finally, XML supports validation. XML documents can be associated with a Document Type Description (DTD) or an XML Schema, either of which defines a structure for conforming applications. This allows applications that import data to validate that data for conformation to the DTD or schema.[1]

Although it is a young standard, XML is already having a significant impact in intranets and on the Web. Businesses appreciate XML, because it eliminates many of the costly and fragile workarounds needed to represent rapidly changing data in HTML. Application developers like the extensibility of XML, and communities that must share common data (e.g., the chemical industry) like XML's support for 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. XML should continue to have strong support over the next decade.

As a result of fixing the known limitations of 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

[1] XML DTDs were intended originally for document management, with inadequate supportfor modeling data. To address this, the W3C has developed XML Schemas, which add data types, relationships, and constraints. In the rest of the paper, we use "DTD" as a shorthand for "DTD or XML Schema."

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.[2]
- *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:[3]

- **Extensible Stylesheet Language (XSL).** 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 initial 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 predict that most XML-related work will be expressed in terms of tree and graph abstractions that hide these details. DOM provides a tree-based application programming interface to XML with 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 to encapsulate non-XML data via a mapping mechanism.

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.

3. 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.

3.1 XML for Well Structured Data

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

[3] For current information on these standards and the tools that support them, see http://w3.org/xml and http://www.xml.org.

Today's dominant database management systems (DBMS) are broad, mature products that will continue to dominate management of critical enterprise data. They are rapidly widening their scope to serve newer areas—e.g., electronic commerce has become a major revenue and development focus. DBMSs offer high-integrity, read, write, and (increasingly) subscribe-to-changes processing of large amounts of regularly structured data. For the data that supports critical but routine processing, these requirements will continue. Facilities include highly tuned query and transaction processing, recovery, indexes, integrity, and triggers. The simplicity of the relational data model (regular tables, no queries over element tags, weak support for paths) are exploited to simplify semantics and improve performance. Even load and dump utilities are optimized for performance. It will require even more complexity to provide similar functionality over XML's more complex structures.

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. 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.

There are several benefits of publishing database contents as XML. First, the XML output includes its own schema information; for anyone who understands the tags used, the information is self-describing. Also, XML reduces the need for multi-site systems to migrate to a new interface all at once. With XML Schema, one can keep part of the format open. If new tags are inserted, those sites equipped to use the new information can do so, while parsers for other sites will ignore it.

3.2 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 and to conform to a prespecified schema, which promotes integrity but discourages rapid development and change for irregular data or data whose structure evolves rapidly. Frequently RDBMS purchase prices are high. For document data, *semi-structured* data models 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. 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. Object-oriented database vendors have begun addressing this need by extending their capabilities to support XML.[4] Relational systems are also moving in this direction. For efficiency, these products will often use highly tuned indexed structures, rather than just store XML as text.

The long-term direction for database support of XML and other semi-structured data, though, may be best seen in the database research community. Many researchers are addressing the challenges [16] of interfacing to semistructured data, and of managing that data. For interfacing, "wrappers" can mine data with implicit structure and make the structure explicit and machine-readable (e.g., [1, 10]). Other projects (e.g., [11]) are investigating the use of graph-structured data models (such as that which underlies XML)

[4] 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.

as a common representation for heterogeneous information sources, including both structured and semi-structured sources.

Finally, several groups are developing prototype DBMSs that manage semi-structured data, with new query languages and optimization techniques [4, 6, 9]. These researchers have converged on the use of graph-structured data models (especially XML), in which all data is represented in labeled directed graphs.[5] In some cases, 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:

- *Irregular structure*. For example, for one location, an attribute Weather may be described by a short string—Good, Bad, or Ugly—while for another location it may consist of a collection of <date, time, temperature, humidity, wind-speed, and remarks> tuples. Document data is often quite variable in structure, especially if assembled from multiple sources. The structure may also change, e.g., when figures are added. Relational systems can model some irregularity by having missing attributes as nulls. However, SQL is awkward with null values, and current storage structures can have excessive overhead.
- *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.
- *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, its interface will be more awkward than a flat, neutral structure.)
- *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 (e.g., [7]), or (3) directly over some new data manager.

4. XML and Data Sharing

Some industry observers have heralded XML as the solution to data sharing problems. For example, Jon Bosak [3] wrote 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 some of the best known architectures for data sharing. Second, we list six challenges that data sharing must overcome,

[5]While XML was originally defined as a text markup language and not a formal data model, if one abstracts from the text representation, the underlying model is a labeled directed graph. It is in fact very similar to the models used in some research prototypes (e.g., [4, 6, 9]).

regardless of architecture and formalism. We then use this list of challenges as a framework for answering the questions: Where can XML help? What portion remains unsolved?

4.1 Architectures for Data Sharing

Users want seamless access to all relevant information about the real world objects in their domain. There are several general architectures (plus hybrids), including the following [5]:

- *Integration within the application.* An application (or web portal) communicates directly with source databases. The application uses each source's native interface and then reconciles the data it receives.
- *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. As a variation, *data marts* give individual communities their own subsets of the global data.
- *Federated databases.* This can be thought of as a virtual data warehouse—i.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.
- *Messaging.* One application or database passes data to others via structured messages.
- *Parameter passing.* One application calls another and passes data as parameters.

4.2 Data Sharing Challenges

Regardless of what 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, described below. Typically, one must address the levels in order—e.g., one must solve the problem at levels 1 and 2 before one addresses level 3.

1. *Distribution.* Data may be widely distributed geographically. Off-the-shelf middleware products handle most of the challenges at this level, often supporting standard protocols—e.g., CORBA, DCOM, SOAP, and HTTP.

2. *Heterogeneous DBMSs.* Diversity here includes different data structuring primitives (e.g., tables vs. objects) and data manipulation languages (e.g., SQL vs. proprietary vs. file systems with no query language). Standards like ODBC and middleware products increasingly handle this difficulty. However, the middleware may be costly, may not offer advanced features (e.g., triggers), and may be inefficient compared to using native interfaces.

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 [14].

   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). 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. For example, one system might store all customer account information in one denormalized table, while two others split it among several tables, in different ways. Many application communities are addressing this challenge by defining standard interface schemas, expressed as Unified Modeling Language models, XML DTDs and Schemas, or SQL tables. 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 9/9/39), should a police automobile-check view consider the tuples to refer to the same person and return (John Public, 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's height to be respectively: 180, 187, and 0 centimeters. What value or values should be returned to the application?

4.3 Where Can XML Help?

Given the challenges described in the previous section, we now consider where XML can help improve data sharing.

Distribution (level 1). XML indirectly provides some assistance with distribution, by supporting mechanisms for remote function invocation across the web. For example, Simple Object Access Protocol (SOAP) uses XML and HTTP as a method invocation mechanism.[6] SOAP 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 DBMSs (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

[6] http://www.w3.org/TR/SOAP

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 can be used for relational data, but really shines 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. For example, the U.S. military and its coalition partners are transitioning their Message Text Format (MTF) to an XML-based infrastructure. XML's strong base of tools (freeware and commercial) gives great flexibility and has greatly reduced development costs. In another example, our research group found XML to provide a useful common representation for integrating two semistructured text data sources, Periscope and the CIA's unclassified World Factbook [10].[7]

When observers say "XML provides interoperability," they usually are referring to this level. For the purpose of representing data structures, XML—right out of the box— provides both a representation and (with its current and future query languages) a manipulate/transform capability. It enables a recipient to reassemble the same data structure that was sent, i.e., to obtain the same labeled graph. Many organizations and data exchange standards (e.g., [15]) are employing XML in this way. In such settings, this level's obstacles to establishing interoperability are removed. Interpreting the meaning of this graph is the (substantial) task for the subsequent levels.

Heterogeneous attribute representations and semantics (level 3).

This level deals with atomic concepts, while the next level will consider structures. To transmit a fact between systems, one must relate the semantics that each system uses, and also their representations. One does not require that the computer "understand" either the source or target concept; rather, one needs to know whether they are identical, or how they might be converted. XML helps by providing a convenient mechanism for attaching descriptive metadata (literally data about data) to attributes of both the source and target schemas.

For semantics, the key metadata is to know whether the source concept is good enough for the target (not necessarily that two concepts mean the same thing). For example, an instrument landing system might measure Altitude from the current lowest point of the aircraft; for air traffic control, any part of the aircraft may suffice.

When a source and target database disagree about representation, each should describe representation details explicitly, as subsidiary elements, e.g.,

```
<Altitude> 500
      <LengthUnit>miles</LengthUnit>
      <MeasurePoint>lowest</MeasurePoint>
</Altitude>
```

One would then want standards to make this subsidiary information sharable, e.g., what is a "LengthUnit"? The descriptions determine what transformations are needed. Increasingly, integration tools come with libraries of such conversions, and insert them automatically.

However, it is not enough to have a mechanism to collect metadata. These have existed before (e.g., repositories) and have not brought us the expected interoperability benefits. Without effective tools for

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

finding and exploiting this metadata, there are insufficient incentives for accurate metadata to be collected and kept up to date. As a result, interoperability will not be greatly eased.

Fortunately, typical XML environments have characteristics—namely universal connectivity and rich toolsets—that provide wide accessibility and ease construction of interoperability tools. The universal connectivity of web environments means that one can easily point to standard element definitions, conversion function libraries, and other resources that promote interoperability. Also, because of the ubiquity of XML, tool builders benefit from a large marketplace of high quality, inexpensive commercial development tools that simplify the construction of interoperability tools.

Heterogeneous schemas (level 4).

To support interoperability at this level, one needs a way of describing and sharing community schemas and a way of expressing mappings across schemas.

XML DTDs and XML Schemas 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. Oasis and BizTalk are examples of XML repositories that facilitate 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 XML 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 [8]. [12] 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. Options for expressing these mappings include XSL and SQL. The XML Query Language will offer another alternative, though it may be some time before XML query processors execute complex mappings efficiently.

Currently, inter-schema mappings are too often handled by code that is tied to particular proprietary tools (especially Extract-Transform-Load tools offered by data warehousing vendors). A better approach is to represent mappings declaratively and to generate "glue" code from reusable and vendor-independent mappings, an approach being pursued by the Metadata Coalition and the Object Management Group's Common Warehouse Metadata (CWM) [15]. CWM supports relational, XML, and several legacy data sources. It uses SQL-99 to express mappings and XML for data interchange.

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).

Level	Challenge	XML Contribution
6	Data value reconciliation	Provides a convenient mechanism for attaching metadata.
5	Object identification	Provides a convenient mechanism for attaching metadata.
4	Heterogeneous schemas	Rich environment eases building tools that encourage resource reuse. Makes it easier to map to *some* well understood schema. Also, mechanisms for expressing inter-schema mappings (e.g., CWM) can leverage XML. However, standardization is not intrinsically easier in XML (vs. relational or object schemas).
3	Heterogeneous attribute representations & semantics	Provides convenient way to attach and reference metadata to describingdata representation and semantics. Ubiquitous web infrastructure eases compliance with data standards. However, semantic heterogeneity will be a continuing pain in the budget.
2	Heterogeneous data structures & languages	*Very substantial help*. It provides convenient, neutral, self-describing syntax for heterogeneous data structures.
1	Distribution	Assistance with remote function invocation (e.g., via SOAP). System integrators will be mostly insulated from this use of XML, because it will be provided by middleware vendors.

Table 1: XML Contributions to Data Sharing

Table 1 summarizes XML's contributions to data sharing. XML and related tools provide enormous help at level 2 (often making the problem disappear. In addition, XML enables data administrators to express results that help at other levels, especially at levels 3 and 4.

However, a key issue is not resolved by XML: how to best specify intersystem mappings. First, analysts need tools to help them identify candidate relationships across systems. Second, mappings should be specified declaratively and not in procedural or vendor-dependant 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. As noted above, two efforts offer promise for standardizing the expression of intersystem mappings—OIM and CWM; currently, of the two, only CWM directly supports XML data sources. Finally, we need tools to record intersystem relationships and mappings to any community standard DTDs or schemas, and to make them available for reuse [13].

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 limited 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.

5. 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, Frank Manola, Roger Costello, John Schneider, Kit Lueder, Lisa Harper, and the anonymous reviewers for their helpful comments.

References

[1] B. Adelberg, "NoDoSE: A Tool for Semi-automatically Extracting Structured and Semistructured Data from Text Documents," *Proceedings of ACM-SIGMOD,* 1998.

[2] J. Blakeley, "Data Access for the Masses through OLE DB," *Proceedings of ACM-SIGMOD,* 1996.

[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," *Proceedings of ACM-SIGMOD,* 1996.

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

[6] M. Fernandez, D. Florescu, J. Kang, A. Levy, D. Suciu, "Catching the Boat with Strudel: Experiences with a Web-site Management System," *Proceedings of ACM-SIGMOD,* 1998.

[7] D. Florescu and D. Kossmann, "Storing and Querying XML Data using an RDMBS," *IEEE Data Engineering*, 22(3), September 1999.

[8] 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.

[9] 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.

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

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

[12] 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.

[13] A. Rosenthal, E. Sciore, S. Renner, "Toward Integrated Metadata for the Department of Defense", *IEEE Metadata Workshop,* Silver Spring, MD, 1997.

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

[15] T. Vetterli, A. Vaduva, M. Staudt, “Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metadata”, *SIGMOD Record*, September 2000.

[16] J. Widom, “Data Management for XML: Research Directions,” *IEEE Data Engineering*, 22(3), September 1999.

Biographical Sketches

Len Seligman is a Principal Scientist at The MITRE Corporation. He holds a Ph.D. from George Mason University and is an Associate Editor of *ACM SIGMOD Record*. His interests include heterogeneous databases, semi-structured data, and large-scale information dissemination.

Arnon Rosenthal is a Principal Scientist at The MITRE Corporation. Recent interests include data administration, interoperability, distributed object management, legacy system migration, and database security. At Computer Corporation of America, ETH Zurich, and earlier, his interests included query processing, database design, and active databases. He holds a Ph.D. from the University of California, Berkeley.