

# XML, Databases, and Interoperability<sup>1</sup>

Arnon Rosenthal<sup>1</sup>, Len Seligman<sup>2</sup>, Roger Costello<sup>1</sup>

<sup>1</sup>The MITRE Corporation  
202 Burlington Road  
Bedford, MA 01730

<sup>2</sup>The MITRE Corporation  
1820 Dolley Madison Boulevard  
McLean, VA 22102

{arnie, seligman, costello}@mitre.org

## Abstract

*The eXtensible Markup Language (XML) and related technologies offer promise for (among other things) applying data management technology to documents, and also for providing a neutral syntax for interoperability among disparate systems. But like many new technologies, it has raised unrealistic expectations. We give an overview of XML and related standards, and offer opinions to help separate vaporware (with a chance of solidifying) from hype. In some areas, XML technologies may offer revolutionary improvements, such as in processing databases' outputs and extending data management to semi-structured data. For some goals, either a new class of DBMSs is required, or new standards must be built. For such tasks, progress will occur, but may be measured in ordinary years rather than Web time. For hierarchical formatted messages that do not need maximum compression (e.g., many military messages), XML may have considerable benefit. For interoperability among enterprise systems, XML's impact may be moderate as an improved basis for software, but great in generating enthusiasm for standardizing concepts and schemas.*

Key words – XML, data interchange, interoperability, semi-structured data, database management systems, internet data management, web databases

## 1. Introduction

The eXtensible Markup Language (XML) and related technologies were originally created to improve document processing, by separating presentation from content and by revealing documents' semantic structure. But XML is expected to become more widespread, as the default way organizations will publish information to the Web. Powerful, easy to use, inexpensive tools have already begun to emerge. So it is natural that other disciplines wish to adopt the standard, and to extend its use to their purposes. This is already happening.

Database and object environment vendors have already committed to supporting XML as one way of outputting their data, particularly for publication purposes, but also for interoperability with other software. Another arena of use is for building large scale information resources out of millions of *interrelated* documents or multimedia objects. This kind of information resource requires both large scale read/write data management, and (mostly outside our scope) relevance-based searches by users unfamiliar with exactly how each document is structured. Finally, XML is seen as facilitating data sharing among disparate organizations, for both databases and business objects. In this sense, it is expected to help the

---

<sup>1</sup> *Federal Database Colloquium*, AFCEA, San Diego, 1999

heterogeneous database community meet its integration goals, as well as facilitating use of formatted messages.

The paper will:

1. Review the web-based motivations for XML, and for other emerging WWW standards that provide critical extensions.
2. Examine how XML will affect relational databases directly, in limited ways.
3. Explain XML's use as a knowledge-representation, and describe the "semi-structured" databases intended for documents and web content.
4. Sketch a reference model for interoperability support, and examine the degree to which XML enhances each capability in the model.
5. Consider what government programs ought to do now.

## 2. Motivation for XML

In today's Internet and intranets, HTML is the predominant format for publishing information. HTML is a simple markup language, which has contributed to its widespread adoption. Unfortunately, HTML suffers from the following shortcomings:

- In an HTML document, the content (data) is married to the presentation. The data is marked-up with tags that tell the browser how to render the data in a browser. Because of this close coupling of content and presentation, HTML does not effectively support alternate presentations of the same underlying content (e.g., for different user groups or media).
- The set of markup tags provided by HTML is fixed. There is no support for creating new tags that are useful for a particular application (e.g., Patient\_ID for medical applications or BE\_Number for targeting).
- HTML provides no support to applications that need to validate data that is being imported.
- HTML provides little assistance with improving the precision of searching the Internet and intranets. For example, suppose one wanted to find articles authored by Henry Kissinger. Currently, one can search for "Kissinger" using a search engine (e.g., AltaVista), which would return many thousands of hits. Because of HTML's non-extensibility, there is no way to specify that "Kissinger" is not just an arbitrary string but that it designates the document's author. As a result, search engines cannot offer fielded search.

Because of these limitations, it is currently cumbersome to build and maintain applications on top of multiple HTML documents (e.g., a comparison shopping agent). Such applications require a great deal of hand-crafted (and brittle) 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"). This is especially frustrating, since much of the information on the web today is actually stored in structured databases, but the structure is thrown away as the information is published to the web as HTML. In addition, HTML provides little help to applications that need to share data with other applications.

The Standardized Generalized Markup Language (SGML) addresses these shortcomings, and is used to create some large information corpuses (e.g., in the Intelligence community). However, its complexity has discouraged widespread adoption, at web scale. To address these problems, the World Wide Web

Consortium (W3C) developed a simple subset, adapted to the web. Extensible Markup Language (XML) <<http://w3.org/XML>>, offers the following:

1. Independence of content and presentation. XML addresses content only. The presentation of XML is handled by Cascading Style Sheets (CSS) or the Extensible Stylesheet Language (XSL), both W3C standards.
2. Extensibility. Information publishers are free to invent their own tags that are useful for their applications.
3. Validation. XML documents can be associated with a Document Type Description (DTD), which defines a structure for conforming applications. Then, importing applications can validate that data conforms to the DTD.

Benefits of XML include:

- *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.
- *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.

Two caveats should be mentioned. First, the selective search depends on the use of widely understood tags (e.g., Author). Second, legacy data will not enjoy the above benefits, unless it can be automatically tagged. However, with web content growing explosively, the legacy will soon be a small fraction. We therefore expect that within a few years, most documents published for wide dissemination (and many other resources, e.g., images) will have XML tags or something equivalent.

As 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. Products are already emerging. But not surprisingly, users wanted more. In particular, many users wanted to take it beyond document markup, to be useful for broader information processing uses. As a result, W3C spawned several efforts to extend XML functionality, described in the next section.

### 3. XML Technologies

XML itself is a modest standard, but it is important for two reasons. First, XML has the support of key vendors (e.g., Microsoft, IBM, Sun, Oracle, and Netscape). As a result, there is a large commercial marketplace for XML tools, from which the Department of Defense (DoD) will clearly benefit. Second,

there are several related and sometimes overlapping standards that will greatly increase XML's impact. We call them *XML technologies*.

We describe some of the proposals below, focusing on their aims and possible benefits. First, we discuss standards that allow a document to be manipulated at a higher level of abstraction. Then we discuss some of the other standards, which identify additional structure or provide additional manipulation tools. Since publication delays soon render tool surveys obsolescent, we refer the reader to <http://www.w3.org> and <http://www.xmlinfo.com> for current information on these standards and the tools that support them.

### 3.1 Abstract Models for XML Data

Brackets and backslashes are uninteresting. We hope (and predict) that most XML-related work will be expressed in terms of abstractions that hide these details. The next subsection discusses several proposals that define interfaces for manipulating XML in terms of node, tree and graph abstractions.

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. For programmatic access to XML documents, standard APIs are needed. The treatment resembles the ISO 10303 STEP/EXPRESS standards for product design data [HARD99]. The base level describes a format as a string of characters or bits (i.e., pure XML). The next level is an object Application Program Interface (API), that provides access to elements, preferably within a graphical structure. The API defines operations to work on a single node, or on a node and its children. Next, STEP/EXPRESS has a mapping language for information aggregates.

*APIs for XML.* W3C has approved two APIs. SAX (Simple API for XML) is an event-based API; a SAX parser does a single scan through an XML document and generates events as it encounters items (e.g., an event is generated when an Author tag is encountered). The application program must "catch" these events and process them accordingly. DOM (Document Object Model) is a more powerful, tree-based API; a DOM parser presents an XML document as a tree data structure. (Some DOM parsers store the entire document in memory, which might be inappropriate for huge, multimedia documents). The DOM API provides methods for traversing the tree, such as `getParentNode()`, `getChildNodes()`, etc.

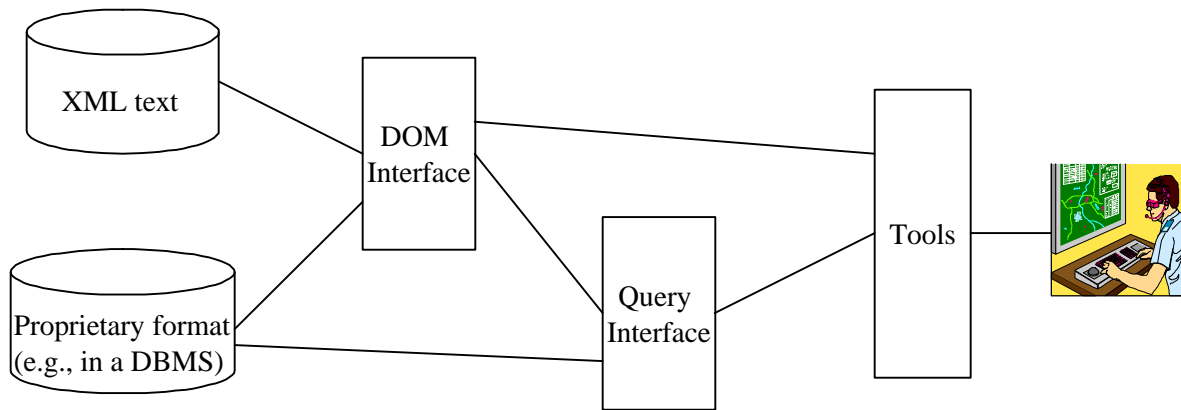
*Query Languages.* Within the W3C, there is widespread recognition that XML needs a query language, however, there are different perspectives on which communities' requirements should initially be addressed. Should the language aim at document transformation, or information retrieval from the Web and intranets, or at general purpose data query and interchange? Currently, there are several candidates, including XQL [ROBI98], XML-QL [DEUT98], and LOREL [MCHU96]. XQL is an extension of XSL; XQL queries return a subset of a single XML document. In contrast, XML-QL and LOREL are more powerful (and complex) semi-structured query languages with features like joins, graph constructors, and declarative views. [SELI99] discusses XML query language requirements for the DoD.

*Abstract Resource Descriptions:* XML is primarily hierarchical and is biased toward physical structure. The *Resource Description Framework* is a W3C Recommendation that provides an object/properties model for describing metadata. RDF can be implemented using XML, and is currently supported in the Netscape browser.

**3.1.2 The Paradoxical Disappearance of XML.** The above object and query APIs will have a paradoxical effect – they will reduce the importance of the textual standard for XML. DOM defines a standard wrapper for XML text, above which most services can work. (XSL, constraint checking, and even linking care about elements, not bracket symbols.) So the first step that DOM enables is that these services can use a single parser that produces DOM.

The next step is for services to use the DOM abstraction rather than raw XML. Then most XML benefits will be available to any data source that supports a DOM interface. A giant XML text string is a poor representation for a large, complex, updatable structure, which needs indexes, clustering, free-space management), transactions, etc.

Tools will access the abstractions of document sets through the standard interfaces, as seen in Figure 1 below. (Perhaps the same could be done for document preparation, e.g., DOM over a word processor internal form or transfer syntax, but that is outside our scope.) In the end, we might find XML text being used mainly for document management applications and at the interfaces between loosely-coupled systems.



**Figure 1: Tools Using the DOM and Query Interface Abstractions**

### 3.2 Greater Expressiveness for Each Document

*Formatting and Transformation.* The Extensible Stylesheet Language (XSL) is both a formatting language and a limited but useful query language. An XSL document contains a set of rules that indicate how to transform an XML document. This transformation could be to a presentation format (e.g., HTML, PDF, Microsoft Word), 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. XSL's document transformations provide a hierarchical query language.

*Enhanced Links.* HTML's hyperlinking model is very simple, allowing only in-line, unidirectional links (i.e., "click here and I'll take you to the resource described in this URL"). XLink and XPointer are emerging standards that will enable richer hyperlinking capabilities, including

- Both in-line or out-of-line links
- Bi-directional links (with bidirectionality enforced)
- One-to-many-links
- Links to substructures and to spans of characters
- Embedded links that enable the dynamic creation of documents composed of fragments from a variety of other documents

These features will enable the creation of value-added link databases in which one can attach descriptions from afar, create links for documents you can't modify, and provide new ways of associating information. These features will be valuable for intelligence analysis and other domains that require so-called "knowledge management."

*Types and Schemas.* XML DTDs were intended originally for document management and provide inadequate support for modeling data. In response to this, the W3C has formed an XML Schema working group which is adding data types, relationships, constraints and possibly class hierarchies to what currently exists in DTDs.

### **3.3 Standard Data Elements and DTDs**

A third category of results has less to do with actual capabilities of XML, and more to do with the enthusiasm it has evoked. Many communities that previously did not try to create a shared information model are now trying to define appropriate standards. These efforts may produce element definitions, or entire DTDs.

The Dublin Core (<http://purl.oclc.org/dc/>) is a set of element definitions that has emerged from the library, digital library, image, and web communities (among others). These standard metadata elements may be used to describe any resource on the Web; the hope is that they will be widely used, especially for documents intended for wide dissemination.

Other communities (e.g., electronic commerce, healthcare, data warehousing vendors) are specifying whole structures, i.e., standard DTDs, for exchanging information of interest to them. Such DTDs will reduce diversity of interfaces, and ease data sharing. For example, BizTalk is an XML repository effort being spearheaded by Microsoft (but supported by numerous companies) that facilitates interoperability by a one-time mapping among XML elements and models.

These initiatives will significantly reduce barriers to interoperability, but do not eliminate them. Competing standards, overlapping problem domains, and unforeseen interchange requirements, and differing semantic interpretations of the same DTD will not disappear. Element-to-element mappings can be costly to determine across communities, and context free mappings among atomic elements do not seem a complete solution.

## **4. XML and Databases**

DBMSs are the antithesis of web-style tools, which emphasize simplicity, portability, and direct interaction with humans. XML and database technology are more complementary than competitive. XML technologies reveal structure in data that is not organized into relational tables. XML thus opens opportunities for database techniques that can exploit the revealed structure. Conversely, database techniques are essential for adding integrity and greater semantic integration to sets of XML resources, and supply many ideas to XML schema and linking technologies. These same requirements for integrity, rigor, and integration imply that enterprise information processing will continue to be based on relational (or object-relational) databases.

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

### **4.1 How Ordinary Relational Databases Can Benefit from XML Technologies**

Today's dominant DBMSs 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., Aircraft, Mission) rather than information structures used to describe that world (Document, Title). The regularity of relational tables (all tuples 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. 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 not really sufficient 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), 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 (i.e., both the DTD and tags used that are not mentioned in the DTD), the information is self-describing.

For structures that are more complex than flat tuples (e.g., in object-relational systems), XML offers a way to serialize objects – verbosely. Where efficiency is needed, other structures (e.g., OLE/DB rowsets [BLAK96]) may prove more attractive.

## 4.2 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 (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, versioning) over this data. Object database vendors have begun addressing this need by offering XML data managers [POET99, OBJE99] layered on top of their object databases. Object-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 semistructured data. Some have developed "wrappers" that mine data with implicit structure and make the structure explicit and machine readable (e.g., [ASHI97], [ADEL98]). Other projects (e.g., [PAPA96]) are investigating the use of XML and similar data models as a common representation for heterogeneous information sources, including both structured and semi-structured sources.

Finally, several groups are developing prototype full-scale DBMSs for semi-structured data. For example, they aim to provide full-featured queries, optimization, and triggers [FERN98, MCHU97, BUNE96]. These researchers have converged on the use of graph-structured data models (e.g., 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 operators that understand the hierarchy.

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

- *Hierarchical model.* Relational systems are moving in this direction, by allowing tables as values in databases. The query language can refer to data in such tables, to some degree. The case where the resulting tree is of fixed structure (including fixed depth) is likely to receive the best support. (Since the 1970s, IBM's IMS hierarchical database has offered such fixed hierarchies, with cross-links and very high performance, but without a full-fledged query language).

Hierarchical languages simplify tasks that suit the data's hierarchy, but at a price. Pure hierarchical languages can combine information only in limited ways. If one adds joins, outerjoins, and references, one obtains a complex hybrid. Also, it can be very awkward to work with a hierarchy that is antithetical to how your application views the world.

- *Path operations:* Semi-structured databases provide 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 lists. 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. (CCA's Model 204 has offered this feature since the 1970s, however its query language is more procedural than SQL.)
- *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, or sections merged. For resources pulled off the web, even if the resource has a 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. Researchers have devoted considerable attention to this issue, e.g., treating the structural information as a materialized view.
- *Advanced features: sequences and graph-structured data model.* Unlike tables, document sections are ordered, so sequence must be represented. When one goes beyond XSL to include join queries and update, sequence introduces significant complexity. Also, for representing database structures or documents with references, one needs a general graph model rather than just a tree.

## 5 XML and Data Sharing

Some industry observers have heralded XML as the solution to data sharing problems. For example, [BOSA98] 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 are no silver bullet.

This section examines the likely impact of XML on data sharing. First, we survey data sharing architectures (Section 5.1), and then list six functional tasks that data sharing must accomplish (Section 5.2), regardless of architecture and formalism. Section 5.3 uses this list of data sharing functions as a framework for answering the question: where can XML help? Section 5.4 analyzes two additional issues. First, most XML-based interoperability approaches use XML DTDs to define an interchange format for bulk transfers; we discuss the limitations of bulk-transfers for interoperability. Second, some have advocated the use of XML DTDs for developing standard information models; we close the section with a discussion of issues that should be considered in picking a formalism for representing information models.

## 5.1 Architectures for Data Sharing

Users wish to have access to all relevant information about the real world objects in their domain. They want information about the same object gathered together, and to have a variety of policies for handling discrepancies. The literature provides several general architectures (plus many variations). The architectures are presented below roughly in order of increasing demands on the “global” data management.

- *Local database.* Each application works with its own database. As an extreme case, all applications use the same database. No reconciliation is needed, but the system is a monolith. This approach does not scale or cross organizations.
- *Applications work directly with source databases.* Each application does its own reconciliation. (This is sometimes called the *multi-database* approach).
- *Data warehouses.* For a data warehouse, 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. One typically extracts and loads big chunks of data; this can be fast, does not require that sources support selective access, and allows intermittent connectivity. (In a variation of data warehousing, customized views of the global warehouse—called data marts—are maintained for particular communities.)
- *Federated databases (virtual warehouses).* Again, central administrators provide the logic to reconcile differences, and derive a database conforming to the global schema. Applications run against a global schema or one chosen for the mart. Here, the source systems retain the physical data, and a middleware layer translates all requests to run against the source systems. This approach requires that the sources be able to execute the selective translated queries.
- *Application-specific databases, with some data supplied from outside.* This case is a hybrid of the previous three. An application sees a database that holds both data generated locally and data imported (either physically or virtually) from other systems. Unlike the warehouse and federation, though, here the schema matches local needs.
- *World model.* One begins by defining a model of the entire world, all types and all instances of each type (e.g., all properties for all registered vehicles and licensed drivers in the world, throughout history). We temporarily ignore the awkward fact that much of the data is unavailable on-line. Each

source advertises the information that it has, described as a view derived from a fully-populated world model (e.g., name, birthdate, vehicles, and violations for Utah licensed drivers since 1993). Applications pose requests against this same world model (e.g., drivers' names and traffic violations for drivers born since 1980.)

The "world model" query processor then rewrites the user's request to be the union of two parts: a query expressible over the views that the sources' can actually provide, plus a residue it cannot satisfy (explained in English). This technology allows us to exploit descriptions of what instances each source has (Utah, since 1993), so we can identify what additional data we need. Several research prototypes exist, mostly for read-only nonredundant data.

## 5.2 Generic Tasks for Data Sharing

Regardless of which distributed architecture is chosen, one must perform certain generic tasks to derive what the recipients see. We now describe these tasks in two areas: data reconciliation and control functions. The following section considers the contribution of XML toward performing these tasks.

**5.2.1 Data Reconciliation.** In all cases above, someone (a standard-setter, an application programmer, or a 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 or domain experts 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, Enterprise Java Beans, http (for session-less processing).
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., trigger definition) may not always 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 atomic 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. Data sharing is easier if programmers need not examine all representational details, e.g., datatype, units of measure. In the future, interfaces may be defined in terms of abstract attributes with self-description, e.g., Altitude(datatype=integer, units=miles). Mediators will insert the appropriate translators [RENN96] into the derivation view. Units are easy to translate, while other transformations (e.g., coordinate systems) can be difficult.

More challenging than representation heterogeneity is differences in semantics (i.e., meaning). For example, two personnel systems could have an attribute Employee.Compensation. One might be gross salary plus annual bonus, while the other is net salary (after taxes) without bonuses, but including the value of certain benefits (e.g., life insurance premiums paid by the employer). In some cases, semantic differences can be resolved through transformations (e.g., deriving net salary from gross salary). However, in other cases, 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 can be split up many ways. Federation views construct the global objects from multiple sources. Various application communities are attempting to define standard schemas, e.g., as UML or IDEF1x 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 cannot be generic and must be application dependent; the researcher or designer's task is to make it easy for administrators to specify the desired policies. System designers should allow the reconciliation rules to be flexible and modular, not buried in code.

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 9/9/29), 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 four sources report John Public Sr. to be respectively: 5', 5'1", 6', and 0. What value or values should be returned to the application?

**5.2.2 Control Functions.** In addition to specifying how data should be reconciled, integrators need an execution environment in which reconciliation can take place. Today, skilled personnel are needed to provide:

- An execution infrastructure for invoking functions (e.g., CORBA, JAVA RMI, DCOM) and for shipping data (message oriented middleware, e.g. IBM's MQ-Series). Creating this infrastructure, often from multiple commercial middleware tools, is nontrivial.
- Functions that do the actual work, e.g., creating, sending, reading interchange files. (For database-oriented data sharing, one may use submittal protocols like ODBC, and standard database operations like query.)

These control functions are not specific to a problem domain such as C4I. Instead, they control a general data sharing process, and are appropriate for middleware vendors to provide. The vendors are moving in such a direction, although somewhat slowly. The end goal is to have a framework that provides the execution infrastructure and the implementations of generic data sharing functions that invoke the transformations defined for a problem domain.

### 5.3 Where Can XML Help?

The reference model of the previous section enables us to discuss specific uses of XML for data sharing. This section examines where XML is reasonable to use. The final subsection examines tradeoffs of using XML versus using another model.

First we dismiss control issues and distribution. These seem to be in the domain of the middleware vendors, and XML technologies seem to have little functionality to contribute. Other systems will need to determine how they reveal their data as XML, but such interfaces are overhead for introducing XML, not new functionality. We recognize that other advances in web technology improve collaboration processes for

defining standards, but these again are not XML issues. The remainder of this discussion thus focuses on data reconciliation. XML can contribute at several stages. We discuss them in turn.

*Heterogeneous data structures and language (level 2).* XML provides a neutral syntax for expressing tree-like data structures, with tagged elements. The APIs (DOM, SAX) and query language proposals will be natural choices to implement over the heterogeneous servers. The APIs compete with Microsoft's OLE/DB (which is a proprietary interface, but currently offers a richer abstraction of server capabilities, e.g., for search interfaces and cursor manipulation). When the query language standard emerges, it will compete with SQL and OQL.

For sharing information 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). Some MITRE prototypes explored the use of XML in place of the government's hierarchical message formats (e.g., USMTF, TADIL/J). Free and low cost XML tools were able to replace costly government tools, even today; one can expect improved XML tools to cut development costs. However, message lengths and processing times increased several-fold (J. Schneider, A. Kazura, personal communication). Compression techniques can ameliorate the problem, but there is not currently a standard for compressed XML.

In cases where there is a fundamental disagreement in business practice, there may not be a basis for data sharing. When different suborganizations use different definitions, they may simply be unable to collaborate closely. Part of the pain (and benefit) of adopting enterprise resource planning systems (e.g., SAP, Oracle Financials) is that they may force business units to adopt consistent definitions (e.g., of "price" or "return on investment").

*Heterogeneous attribute representations and semantics (level 3).* One can imagine commercial, web-enabled dictionaries that contain the elements defined by different communities, thereby meeting some of the original goals of DISA's Defense Data Repository System. (XML namespaces already provide some means for decentralized administration of the element set and DTDs). As discussed in Section 3.3, many content communities are capturing their basic concepts as "standard" sets of XML data elements.

Sometimes a source and receiver will agree on the concept (e.g., Altitude) but not on its representation. A natural solution is for the source and recipient to each prepare an elaboration of their DTD, e.g., "<Altitude>500<units>miles</units></Altitude>", showing their further assumptions as subsidiary elements. One would then want standards to make this subsidiary information sharable, e.g., what are "units" and "MPH". But there is still considerable work that the standards do not support. The extra elements should typically be virtual, derived by a formula rather than included with each value [ROSE94]; also mediation is necessary to compare and insert appropriate translations. Neither of these tasks is directly supported in the standard.

Beyond RDBMS capabilities, XML contributes an annotation mechanism, a good means of disambiguating names provided by different authorities (XML Name Spaces), and most important, the ubiquity and toolsets that attract interest from domain communities. Other XML standards (e.g., the schema and constraint facilities) help XML approximate capabilities that are familiar in databases.

*Heterogeneous schemas (level 4).* XML DTDs defined by various communities provide a neutral model for describing their data structures. With current technology, one can use this format to define an interchange file. To support more powerful, tightly coupled forms data sharing, one will need to translate requests rather than just data, as discussed in section 5.4.1.

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 integration (i.e., schema integration) tools. As a bonus, XML is friendly to partial solutions, that let some data be interpreted and other not.

In the developer community, there is increasing awareness that schema diversity will be a serious problem even if XML DTDs are widely used [GONS99]. [PAUL99] raises this issue for e-commerce, and describes a model with roughly the same layers as ours. However, rather than one standard at each layer, we expect several to be viable. (Communities' interests overlap, and within the overlap, each community will follow its own path.) The standards 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 standards. [HARD99] notes that the product data exchange community's language Express was designed for inter-schema mappings. SQL is also a reasonable choice, now that user-defined functions are supported. 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-estimate" 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 (assuming that the recipient is prepared to use such output).

Frequently there is sufficient basis for communication, but it is hidden by all the forms of diversity at levels 3 and 4, in attribute names and data structures. In such cases, one needs analysts to identify and verify the similarities, tools to help the analysts do the identification. Furthermore, as advocated in [SELI96], we need tools (such as BizTalk) to remember the similarities and make them available for reuse.

## 5.4 Further Analysis

This section looks further at two issues: the limitations of interoperability based on bulk data transfers, and choosing formalism(s) in which one captures community information models.

**5.4.1 Limitations of Bulk Data Transfers.** Proposals to use XML DTDs as a basis for interoperability among diverse systems generally rely on shipping a file containing data that conforms to the DTD. This "bulk transfer" approach makes minimum requirements on the systems involved, and is often used when little coupling is possible. The comments below apply whether the transfer format is expressed in XML, in USMTF, TADIL, or in commercial EDI syntax.

The basic idea is that the transfer format acts as a derived database, derived on demand from the sources. Similarly, each recipient restructures the data from the transfer format, to match the recipient's view [RENN96]. Both XML tools and data warehouse tools can help express these restructurings. In current systems, the derivations are frequently by a mixture of SQL and C, Java, Ada, or C++.

However, bulk transfers are unresponsive to specific needs. For example, many distributed planning systems fit section 5.1's category "*application-specific databases, with some data supplied from outside*". One such system broadcasts the current plan to stakeholders every 24 hours, as an Air Tasking

Order. A more responsive system would allow recipients to formulate additional kinds of requests, in terms of the recipient schema, and have the requests automatically executed against the source. The following three scenarios are not supported by bulk transfer mechanisms.

- A recipient queries the virtual data (in terms of the recipient's virtual schema), to obtain the latest information from the source. For example, they could ask if there are now any planned missions for their unit that require weapon X.
- A recipient sees that the planners are using obsolete information about their unit. They issue an update to their virtual database, and the changes are automatically translated and propagated to the source database.
- A recipient has identified certain virtual information to be critical, and *subscribes* to it, i.e., asks to be notified of any change to those values.

One way to support the above scenarios is to manually implement each as a method. However, programmers are costly, and this process cannot cope with ad hoc requests. (As an extreme, we know of one prototype logistical monitoring system whose subscriptions change several times an hour.) Instead, we want request translation to be automated. That is, based on an understanding of how the recipient's data is derived, the query processor should automatically translate requests against the virtual schema into requests that access the sources. Request translation can coexist with bulk transfer mechanisms, using derivations to and from the same "standard" schema or DTD.

For request translation to succeed, the derivations must be understandable to the query processor, i.e., they must be in the processor's query language. (In contrast, programs that mix SQL into a full programming language tend to be opaque, beyond the reasoning power of the query processor or other automated tools). Today, SQL processors have the power to optimize queries (the first bullet), many updates, and some subscriptions. The power will increase, slowly. In principle, an XML query language processor can have the same power; however, for implementations aimed at document rendering, such features (e.g., update, query optimization) may be low priority. A fuller semi-structured DBMS will be needed.

**5.4.2 Choosing a Formalism for Community Information Models.** One role that XML could play in data sharing is as a modeling formalism for expressing community information models. Such information models describe common concepts to which all systems within the community must map. This section considers the issue of what formalism(s) should be used to express such information models. Should they be expressed as XML DTDs? As SQL schemas? Perhaps in IDEF1X or UML? And also in CORBA, COM+ and Java?

Our answer, for the long term (when translation tools are easily available) is that one can express the model in multiple formalisms simultaneously, to serve different uses (e.g., for object designers, query access, document output, and so forth). Furthermore, though there are generally alternative ways that a model can be represented in each formalism, it seems preferable to declare one of them the standard. Thus, relational users might use a standard SQL schema for C4I, object developers a standard UML representation of the same model, and so forth. Standards will probably emerge more rapidly in XML than in the other formalisms, due to the enthusiasm factor and low cost of entry.

Perhaps the overriding advice is that if you have a widely-accepted standard in *any* formalism, exploit it. Use automated translators to generate the standard in each other formalism where a standard interface is needed. A DISA Semi-Structured Data TWG was an early adopter of this philosophy. A prototype repository for XML is described at <http://diides.ncr.disa.mil/xmlreg/index.cfm>.

If one needs to choose a formalism for capturing a standard for extensive interchange, XML would not be our first choice on pure technical grounds. Relational and object systems have richer type, relationship, and constraint features (though XML is improving fast). Additionally, object models describe method interfaces, allowing one to model behavior as well as data structure. Also, with XML, one must

choose an anchor point for a hierarchy, a decision that may help within a community but create disagreement across communities. The RDF standard might be an excellent candidate, but is not yet widely supported; in particular, DBMS support for it may lag. IDEF1X may be the lowest priority, since it is neither used at run time (lacking DBMS support) nor does it accommodate methods.

There is one circumstance where XML looks very attractive, even today. Sometimes there is a requirement to interchange between a DBMS (or application) and a formatted message, e.g., USMTF, or commercial electronic data interchange. Here, one may wish to use an XML model (i.e., DTD) that reflects the message hierarchy. However, in environments where message contents overlap, one will get overlapping “standards”.

## 6. Conclusion/Recommendations

XML is an important technology that can provide a default format for much database input and output. It allows creation and transmission of self-describing information (if the recipient understands the semantics of the tags). It can capture irregular structures that have defeated relational products. It is gaining a strong set of commercial tools.

Within a system, one can manipulate abstract interfaces— today SAX and DOM, someday perhaps also a standard query interface. Such abstraction layers can greatly increase the applicability of other XML technologies. Because the abstractions insulate most services from the physical representation, one is free to replace verbose text by other representations, and to apply those services to any data that is so wrapped. All of the above conform to standards, and will have many good and inexpensive tools.

The tremendous enthusiasm generated by XML technologies has had a beneficial political effect, causing many groups to attempt to produce “standard” DTDs. The web helps negotiations proceed, and makes the results easy to distribute. In addition, XML is vendor-independent, and its ability to represent hierarchies helps communities who share a world view to organize their models.

But there are also downsides. Compared with relational formalisms, the introduction of hierarchies adds another design decision – which type is at the root. Extra decisions can mean extra diversity. Also, XML is an inefficient encoding, unsuitable for transmitting or storing huge quantities of data, e.g., for loading a large data warehouse. Compression is possible, but introduces its own problems. Finally, XML does not yet have a standard query language, or industrial-strength DBMSs.

For homogeneous DBMS environments, XML will not help directly, in the short term. Semantic integration problems that were difficult among Oracle systems will still be difficult among XML systems. (There is hope, however, that semantic integration tools will improve more rapidly, with a larger market).

The biggest immediate wins concern interoperability where at least one party wants data that eludes relational systems. Formatted messages (for electronic data interchange, military message formats) are best described as semi-structured. If one is forced to create such messages, XML tools can make it much easier to create message parsers and manipulators. One reason why the hierarchical format works better than relational is that one can *sometimes* grab a complex object in one operation, rather than pulling appropriate tuples out of multiple relations.

If a community is planning to do interchange based on capturing a standard model, they should choose a formalism with which they are comfortable, rather than quibble about which is optimal. Capture standard elements and models in some formalism supported by a good repository, with a rich set of translators. Attributes should be described abstractly, as in section 5.3’s Altitude example. Then for every other formalism in which you need to operate, generate an equivalent of your standard (by automated translation, ideally). Among the many possible representations in the other formalism, declare one to be the standard for users in that formalism. For example, if one captures the standard as an XML DTD and needs to develop objects, generate a standard UML form. Also, if possible, capture how your standard elements and models map to other communities’ standards.

Some organizations may have more than one formalism from which to choose for expressing standard elements and models. If so, they should consider models with richer modeling capabilities than XML, especially for modeling relationships and constraints. These models also have the benefit of not forcing all information into a hierarchy; hierarchies only work well when all players agree about what is on top.

Some organizations may choose to use XML as their primary model. In that case, it seems best for a development programs to begin now to:

- Capture content in an XML-compatible form (e.g., XML, or database) that retains semantic structure for use by applications and databases.
- Express applications and tools in terms of abstractions of XML (e.g., DOM, query) rather than raw XML.
- Migrate government message and interchange formats to XML. MITRE is currently prototyping an XML-based approach to disseminating Air Tasking Orders.
- Capture derivations declaratively, rather than in code that is difficult for automated tools to manipulate.

To sum up, XML does not come close to eliminating the need for database management systems or solving the Government's data sharing problems. But it does offer significant benefits: 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.

Of course, 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.

## References

- [ADEL98] B. Adelberg, "Nodose: A Tool for Semi-automatically Extracting Structured and Semistructured Data from Text Documents," in *ACM-SIGMOD International Conference on the Management of Data*, 1998.
- [ASHI97] N. Ashish and C. Knoblock, "Wrapper Generation for Semi-structured Internet Sources," *SIGMOD Record*, 26(4), December 1997.
- [BLAK96] J.A. Blakeley. "Data Access for the Masses through OLE DB," *ACM-SIGMOD International Conference on the Management of Data*, 1996, <http://www.microsoft.com/data/oledb>.
- [BOSA98] J. Bosak, "Media-independent Publishing: Four Myths about XML," *IEEE Computer*, October 1998
- [BUNE96] P. Buneman, S. Davidson, G. Hillebrand, D. Suciu, "A Query Language and Optimization Techniques for Unstructured Data," in *ACM-SIGMOD International Conference on the Management of Data*, 1996.
- [DEUT98] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, Dan Suciu, "XML-QL," *World Wide Web Consortium Query Languages Workshop*, Boston, MA, December 1998.

- [FERN98] M. Fernandez, D. Florescu, J. Kang, A. Levy, D. Suciu, "Catching the Boat with Strudel: Experiences with a Web-site Management System," in *ACM-SIGMOD International Conference on the Management of Data*, 1998.
- [GONS99] Gonsalves, A., Pender, L., "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>.
- [HARD99] Martin Hardwick, David L. Spooner, Tom Rando, K.C. Morris, Peter Denno, "Lessons Learned Developing Protocols for the Industrial Virtual Enterprise," in *CAD Journal*, Sept. 1999.
- [MCHU97] 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.
- [OBJE99] Object Design Inc., "eXcelon: The eBusiness Information Server," <http://www.odi.com/excelon/main.htm>, 1999.
- [PAPA96] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, "Object Fusion in Mediator Systems," *International Conference on Very Large Databases*, September, 1996.
- [PAUL99] Paul, L., "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>.
- [POET99] POET Software, "POET Content Management Suite," [http://www.poet.com/products/cms\\_solutions/cms.html](http://www.poet.com/products/cms_solutions/cms.html), 1999.
- [RENN96] S. Renner, A. Rosenthal, J. Scarano, "Data Interoperability: Standardization or Mediation" (poster presentation), *IEEE Metadata Workshop*, Silver Spring MD., April 1996. <http://www.computer.org/conferen/meta96/renner/data-interop.html>.
- [ROBI98] Jonathan Robie, Joe Lapp, David Schach, "XML Query Language (XQL)" *World Wide Web Consortium Query Languages Workshop*, Boston, MA, December 1998.
- [ROSE97] Arnon Rosenthal, Edward Sciore, Scott 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>.
- [SCIO94] E. Sciore, M. Siegel, A. Rosenthal, "Using Semantic Values to Facilitate Interoperability among Heterogeneous Information Systems", *ACM Transactions on Database Systems*, June 1994
- [SELI96] L. Seligman, A. Rosenthal, "A Metadata Resource to Promote Data Integration", *IEEE Metadata Workshop*, Silver Spring MD., April 1996. At <http://www.computer.org/conferen/meta96/seligman/seligman.html>
- [SELI98] Len Seligman and Arnon Rosenthal, "XML Query Language Requirements of Large, Heterogeneous Organizations," *World Wide Web Consortium Query Languages Workshop*, Boston, MA, December 1998, <http://www.w3.org/TandS/QL/QL98/pp/seligman.html>

## Acknowledgments

The authors thank Dave Lehman, Steve Huffman, and Ed Lafferty for their support of research in semi-structured data and XML, and Marie Spadano for her help in production. We also thank Scott Renner, John Schneider, Amy Kazura, Tom Lee, and Ken Smith for many interesting conversations on XML and interoperability.

## Author Biographies

Arnon Rosenthal has worked in recent years on data administration, interoperability, distributed object management, migration of legacy systems, and database security, at the MITRE Corporation, Bedford MA. He is Vice Chair for “System Admin, Ease of Use & DB Design” for the International Conference for Data Engineering. 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.

Len Seligman works 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.

Roger L. Costello has been active in the XML field for two years. He has created an XML course and taught it at numerous locations, most recently in Virginia to members of the USMTF community. In addition to XML, Dr. Costello's interests include: OO design, component technologies, and Java. He holds a Ph.D. in Computer Science from Ohio State University.