Virtual XML: A toolbox and use cases for the XML world view

K. H. Rose
S. Malaika
R. J. Schloss

Although the Extensible Markup Language (XML) has gained in popularity and has resulted in the creation of powerful software for authoring, transforming, and querying XML-based business data, much information remains in non-XML form. In this paper we introduce an approach to virtualize data resources and thus enable applications to access both XML and non-XML sources. We describe the architectural components that enable *virtual XML*—a toolbox that includes a cursor model, an XML-view mechanism such as the view created with the Data Format Description Language (DFDL), and XML processing languages. We illustrate the applicability of virtual XML through a number of use cases in various environments. We discuss the products that we expect from vendors and the open-source community and the way enterprises can plan to take advantage of virtual XML developments. Finally, we outline future research directions that include a vision of virtual XML that covers large-scale structures such as entire file systems, databases, or even the World Wide Web.

INTRODUCTION

Enterprises seeking ways to make their business processes more integrated, more nimble, and more flexible are evolving their data centers and software platforms by using concepts that IBM has called the *On Demand Operating Environment*. Increased integration can be achieved through IT simplification, and a major strategy for simplification is virtualization. Although much work has been done on virtualization of physical servers and provisioning of applications over a grid of such servers, less attention has been given to the idea of virtualization of data resources and the use of fewer languages for querying, reporting, and manipulating stored data. An example of virtualization of data resources is provided by WebSphere Information Integrator,[1] which offers a relational interface to heterogeneous data. This paper discusses the idea of information virtualization by using the Extensible Markup Language (XML) data model as the framework.

Despite predictions to the contrary, XML has become a very successful notation for information exchange between disparate systems. The technologies, tools, and knowledge accumulated over XML's long ancestry starting with SGML (Standard Generalized Markup Language)[2] have contributed to its success. XML's support of different platforms and

diverse encodings, and now its use in various industries and technologies, have also been a factor. Due to its success, XML has evolved further and is no longer constrained to being just an exchange format. These are some examples of its evolution:

- XML has become a storage format for data held in files and databases. Indeed database systems focused on managing XML have been developed.
- XML is the underlying notation for Web services, which are now used in many distributed systems. Web services standardization often relies on the XPath[3,4] interface to manipulate pieces of XML associated with Web services.
- XML is supported by a variety of widely deployed tools and interfaces, such as XMLSpy**,[5] and open source software, such as XML parsers from the Apache Software Foundation.[6]
- XML has popular generic standards such as XPath[3,4] for manipulating XML, XSLT[7,8] for transforming XML, and XQuery[9] for querying XML, as well as industry-specific standards such as HL7[10] and ACORD[11] for exchanging information in the health-care and insurance industries.

With XML's success, technologies have naturally evolved to convert non-XML data into XML, and to convert XML data into other forms. In the world of relational databases, Structured Query Language (SQL) has been extended to SQL 2003 to incorporate SQL/XML.[12] One of SQL/XML's features is the availability of functions that assist in the generation of XML documents from relational data (e.g., XMLELEMENT, XMLATTRIBUTES, and XMLAGGREGATE). SQL/XML also describes a default XML format for relational data. See Reference 13 for the W3C** XML Schema (an XML document describing allowable structure and data types)[14] defined in the ISO:SQL/XML—Part 14 (SQL 2003) specification. An alternative XML schema, which is common to the result of every possible SQL query, is provided through JCP (Java Community Process**) JSR (Java Specification Request) 114.[15]

In addition there are many types of data that are not represented as XML for practical reasons, such as data generated by low-level sensors, data residing in archives, data not used for exchange with unknown partners, and data that are not tree structured (e.g., image metadata, video metadata). The inability to format data as XML may be caused by the lack of computing power, bandwidth, or storage capacity, and sometimes, it may even be necessary to transmit the same data in both XML and non-XML formats as legacy requirements dictate.

In this paper we propose virtual XML, an approach to an XML-based virtualization of data resources. As part of this approach, we separate the representation of the data from the processing model; that is, we describe how to support processing of non-XML data as though it is XML and without explicit conversion to XML. The benefits associated with the virtual XML approach include leveraging existing interfaces, tools, knowledge, and communities of interest and also the ability to process both non-XML and XML data in a uniform way by using XML processing languages.

Through virtual XML, a *default view* can be constructed for a well-known non-XML format such as EXIF JPEG (Exchangeable Image File Joint Photographic Experts Group, an international specification to encode information into the headers or application segments of a JPEG file that includes shutter speed, aperture, and the date and time the image was captured).[16] Another example of a well-known non-XML format for which a default view could be created, is ASN.1 (Abstract Syntax Notation 1), an international data standard used in communication protocols such as mobile phone systems.[17] Alternatively, a *specific view* can be created for a non-XML format using a mechanism such as DFDL (Data Format Description Language), which is described later.

To achieve a reasonable notion of virtual XML we need an XML abstraction, that is, a data model. Fortunately, this is provided by the W3C XQuery 1.0 and XPath 2.0 Data Model,[18] which extends the XML Infoset[19] with new features to meet the requirements for the new XML processing languages, XPath 2.0,[4] XSLT 2.0,[8] and XQuery 1.0.[9] The extensions include the following:

- Supporting W3C XML Schema types, both structures[20] and simple data types,[21] that extend the XML Infoset with precise type information
- Representing collections of documents and not just single documents
- Describing the formats of arguments and the results of XML processing
- Supporting intermediate results during XML processing, such as typed atomic values arising from

an arithmetic or logical expression, and ordered heterogeneous sequences arising from a path expression

Because the concern of virtual XML is processing rather than representation, adopting the W3C XQuery 1.0 and XPath 2.0 Data Model[18] is a good fit. The data model specifies the accessible information in documents, but because W3C has not specified the programming-language interfaces or bindings used to represent or access the data, we propose an application programming interface (API) for accessing virtual XML.

We would consider using virtual XML when we have some non-XML data that we need to process with agreed-on and well-known XML-based interfaces. We may also consider a virtual XML approach when we need to make some data visible, which for some reason we cannot represent natively as XML but would like to offer also through some standard XML-based interface, for example, those used by Web services.

Virtual XML could benefit a variety of general-purpose software products and tools and applications dealing with stationary data and data streams; for example, the following:

1. *Data publishers*—These are producers of non-XML data that have to be made available later, without reformating, in an XML format or through XML interfaces. There is much demand for such capabilities for data producers that cannot afford the cost of publishing as XML (e.g., resource-constrained sensor devices or mobile phone systems that publish through ASN.1).[17]
2. *Data brokers*—These are processors that interpret data in messages and events, such as for mapping and transformation purposes. Not all data that brokers process is in XML. Virtual XML enables brokers to process data through a common XML model and interfaces, and to offer their users plug points with standard XML interfaces even for heterogeneous data. For example, an ESB (Enterprise Service Bus)[22] contains brokers that offer service transformation support. These are points where data transformations and data filtering can take place and where users can supply their own transformation scripts. With a virtual XML approach, an ESB can provide a uniform transformation environment based around XML, even though the data being transformed may not be XML.
3. *Data consumers*—These are software components that interpret existing non-XML data in files, databases (and general archives), and messages as though it were XML, a capability provided through tools that offer XML interfaces. Software processing ASN.1[17] data as though it were XML through an XML interface such as XPath[3] is an example in this category.
4. *General purpose data-oriented software and virtual XML applications*—These are higher levels of general-purpose software or applications that are constructed with virtual XML technology. Software that offers an XML interface to any ASN.1[17] data or to archived non-XML data is an example in this category.

The rest of the paper is organized as follows. In the next section we describe the virtual XML toolbox, which includes a cursor model, an XML-view mechanism such as the view created with DFDL, and XML processing languages. In the following section, we present several use cases for the preceding patterns, and we point to a worked-out example of a virtual XML application that we published elsewhere. We summarize our results in the last section. Along with the list of cited references and notes, we provide an annotated list of general references for work related to XML views.

■ Virtual XML is a powerful technology that enables processing of non-XML data as though it were converted to XML ■

THE VIRTUAL XML TOOLBOX

In this section we present three technologies and tools that together enable virtual XML solutions: a cursor model, an XML-view mechanism such as the view created with DFDL, and XML processing languages. But first, we start by explaining the important role that XPath[3] plays.

The ubiquitous XPath

One of the most common ways of processing XML data today is to select subsets of that data by using

the XML Path language, XPath.[3] XPath is provided as an addressing mechanism in many programming-language modules used to read or filter XML content, such as LibXML2.[23] The DOM (Document Object Model) Version 3 standard[24] even includes a description of how to support XPath in a generic way.[25] XPath expressions are also fundamental to higher-level languages for operating on XML, such as XSLT[7,8] and XQuery 1.0.[9] Work is now being completed on a second version of XPath, XPath 2.0.[4]

To illustrate XPath, consider the following XML document:

```
<flight-updates>
  <country>
    <origin>CPH</origin>
    <destination>LAX</destination>
    <lift-off-time>2005-07-11T08:00:00+01:00
    </lift-off-time>
    <passenger-count>140</passenger-count>
    <company-name>SAS</company-name>
  </country>
  <country>
    <origin>LHR</origin>
    <destination>LAX</destination>
    <lift-off-time>2005-07-11T08:00:00+00:00
    </lift-off-time>
    <passenger-count>160</passenger-count>
    <company-name>British Airways</company-name>
  </country>
</flight-updates>
```

A very simple XPath expression to select information from this document is

```
/flight-updates/country/origin
```

The expression reads as "from the document root select the flight-updates child; for each of those, select the country children; for each of those, select the origin children." Evaluating the expression returns a sequence of two nodes, similar to the following:

```
(<origin>CPH</origin>, <origin>LHR</origin>)
```

(where we have adopted the use of "(. . .,. . .)" notation for sequence construction from XPath 2.0), except that the members of the sequence are not actually subtrees but *references* into the original tree.

In most cases, the entity that creates an XPath expression uses names of elements or attributes and their structural relationship to each other in the XPath. This is so because all documents processed are part of a class described by an XML Schema, using the W3C XML Schema Definition language,[14] although DTDs (Document Type Definitions), RELAXNG Schemas,[26] and other schema representations may also describe the document class.

A cursor model

A *cursor* is a control mechanism for managing interaction with a potentially large data structure. Cursors were introduced for relational databases[27] to encapsulate the entire state of an interaction with a relational database, using the database query language to change the state of the interaction (or *cursor state*) with special controls to allow iteration over the data currently captured by the query.

It is natural to wish to carry this notion over to XML data with an XML query language such as XQuery. So far, such efforts have been focused on two approaches:

1. "Upgrading" a database cursor interface such as JDBC** (Java Database Connectivity)[28] to allow XML access to XQuery using relational patterns.[29,30]
2. Wrapping access to an existing in-memory representation such as the DOM in a JDBC-like interface.[31,32,33]

For a survey of APIs for XML, see Reference 34.

None of these approaches, however, permit the diversity of XML data access that virtual XML requires: when using virtual XML, we can make very few assumptions about the native organization of the data. To allow for the minimal XML processing capabilities as expressed by the XPath and XQuery Data Model,[18] a *universal XML cursor* should support the following:

1. A notion of *current node* through
   - support for *node-local* access to the value of the data model accessors that are *not* of node type, and
   - methods that *navigate* to the nodes returned by accessors of node type, i.e., of type document, element, attribute, text, processing-instruction, and comment.

   For more information about accessors and their types, see Reference 18.

2. A notion of *context sequence* containing the current node, as well as access to the *context position* of the current node in the sequence
3. Easy navigation to a different node in the context sequence
4. Primitives to ensure that all functions and operators[35] are available
5. Cursor management (cursor creation, cloning, destruction, etc.)

We have explicitly separated everything having to do with nodes (item 1 above) from the other four functions to avoid a client program or path expression from having any direct access to nodes. For virtual XML, where we do not know the native representation of the data, this is key: if we permitted the client program to hold references to individual nodes, then we would have to deal with intricate interdependencies between nodes held by cursors and client programs (indeed this is the problem with those APIs that are based on the DOM[24]); therefore, we shall enforce the use of the cursor for all access to nodes.

Requiring that the cursor model implement everything needed to ensure that all functions and operators can be implemented raises an issue with the W3C XQuery and XPath Data Model: the functions and operators that deal with *node identity* and *document order* cannot be expressed directly in terms of the data model accessors; thus, we shall include primitives for such operations.

Patterns of access to diverse data

The cursor technology described in the previous section provides different patterns of access to diverse data. Supporting the processing of a variety of data poses a new set of challenges, however, because different real data instances reveal different native access patterns. This leaves us in a generalized instance of the common "SAX DOM dilemma," where the choice is between simple efficient interfaces or more advanced less efficient interfaces. Should we

- design the access API with the least common denominator that will allow efficient access to all XML data, traditionally captured by the intentionally "Simple API for XML" (SAX),[36] or
- assume a rich implementation of XML optimized for every kind of access, such as is standardized by the elaborate DOM[24] or another object model such as SDO (Service Data Objects)?[37]

The answer is, of course, that none of these will fit every application: the "SAX-like" approach breaks down when XML data is used in complex ways involving reorganization, such as sorting, whereas the "DOM-like" approach invariably leads to excessive storage requirements. Our solution to the dilemma is a compromise: we slice the aspects of the data model into feature sets that can be combined into *profiles* corresponding to the use patterns.

The SAX pattern

SAX was one of the earliest interfaces for XML processing.[36] It is a `push` model based on the XML producer calling the XML consumer once for each part, or `event`, that is encountered during a left-to-right traversal of the XML tree. The key interface of SAX is thus "ContentHandler," implemented by the consumer. (In practice one encounters the notion of "SAX-like" for all XML solutions that involve some kind of streaming of the XML source into something similar to a SAX event sequence.)

Table 1 shows how the basic SAX events translate to cursor operations. The translation of a `startElement` SAX event into cursor operations involves visiting every attribute individually, which is conveyed by iteration (denoted, as usual, with '*') where the cursor operations are as follows:

- `free`—Releases this cursor as it will no longer be used
- `node-properties`—Denotes a generic way to access the truly local properties of the current item: name, value, and type
- `toAttributes`—Sets up the cursor to iterate over the attributes of the current element node, if possible
- `toChildren`—Sets up the cursor to iterate over the children (immediate descendants) of the current node, if possible
- `toNext`—Advances the cursor to the next item
- `push`—Shorthand for pushing the cursor state onto a stack
- `pop`—Shorthand for replacing the cursor state with the cursor state popped from the hidden stack

This shows that a single cursor object can mimic the behavior of the SAX protocol. Because a cursor is a `pull` protocol, this means that no node properties

Table 1 Translation of basic SAX events to cursor operations

SAX Event	Cursor Operation Sequence
`startDocument`	`to Children`
`endDocument`	`free`
`startElement`	`node-properties; push; toAttributes; node-properties; (toNext; node-properties;) pop; push; toChildren`
`endElement`	`pop`
`text`	`node-properties`

Table 2 Mapping the DOM pattern to the cursor model

Part 1

DOM Pattern	Cursor Model
`Node`	`Cursor that will never be moved`
`Return neighbor node`	`Navigation of a duplicate cursor to the neighbor`

Part 2

Property	DOM	Cursor
`Node identity`	`Fast (object identity)`	`Slow`
`Node release`	`Only when a node is explicitly deleted (all nodes are interlinked)`	`Normal (cursors are not interlinked)`

have to be generated unless explicitly requested, allowing for high-speed skipping of encoded character strings, for example.

Most important, streaming XML processing using "the SAX pattern" has essentially the same efficiency with real SAX and the cursor protocol. Further details, including the relationship between `push` and `pull`, are given in Reference 38.

The DOM pattern

DOM[24] was originally developed to allow programmatic management of HTML pages but has been generalized since to XML. It allows complete control over the in-memory representation of the XML data as follows:

- It is an *object model* where the user gets individual separate references to objects representing "nodes." Consequently, a DOM implementation uses at least as many objects per document instance as there are nodes that the application has touched in that document.
- It allows full navigation of the XML tree, in the downward (parent to child), upward (child to parent), and sideways (sibling to sibling), directions by having a rich set of methods on nodes that return other related nodes.
- It allows changes to be made to the tree.

The DOM pattern can thus be mapped to the cursor model as shown in ***Table 2***, Part 1. As for efficiency, the comparison in Table 2, Part 2 is telling. Thus, using a cursor model instead of the DOM essentially trades the DOM's fast node-identity checking for the cursor's better automatic memory management.

Data Format Description Language

The second virtual XML tool allows the XML representation of data in all forms. The W3C XML Schema Definition language[14] includes special `xs:annotation` declaration elements containing (among other things) one or more `xs:appinfo` elements, called *application information annotations*, and also permits attributes to appear on all the elements used in the XML representation of schema that come from a supplementary, or "foreign," namespace.

A number of communities have been active in areas related to virtual XML through defining XML schema annotations to handle non-XML data. A scientific project of this kind is the BFD (Binary Format Description) project at the Pacific Northwest National Laboratory.[39] BizTalk**, on the other hand, is a commercial project from Microsoft that includes a flat-file extension feature to allow the definition of annotations to control how a flat-file business document is translated to and from its equivalent XML business document.[40,41]

We provide here examples using DFDL.[42] DFDL, which is being developed by the Global Grid Forum (GGF)[43] Data Format Description Language Working Group,[44] exploits application information annotations to define the precise mapping between non-

```
00572 ******************************************
00572 *  COBOL COPYBOOK - CUSTOMERS
00572 *  DATA FOR CUSTOMER TABLE
00572 ******************************************
00572  01 CUSTOMER-RECORD.
00573     05 CUSTOMER-LAST-NAME       PIC X(20).
00574     05 CUSTOMER-FIRST-NAME      PIC X(15).
00575     05 CUSTOMER-AGE             PIC 999.
00576     05 CUSTOMER-PHONE           PIC 9(10).
```

Figure 1
Example of a COBOL copybook

XML formatted data (in byte strings) and the XML data structure declared by the schema. By design, DFDL is limited to fairly "direct" mappings to ensure that implementations can map both ways (to and from XML).

One of the most common formats for simple data exchange is the COBOL (common business-oriented language) "copybook" record storage format. (Of course, data formats described in any other language, not just COBOL, can be mapped with DFDL.) We examine the COBOL copybook in ***Figure 1***.

The unit (at level "01") is a record named CUSTOMER-RECORD. Each record has four fields (at level "05"). The first field contains 20 characters, and the second field has 15 characters ("X(20)" and "X(15)", respectively). The third and fourth fields contain three and 10 digits ("999" and "9(10)", respectively).

The following record corresponds to the COBOL structure in Figure 1:

```
Callas    Maria    0254084444444
```

The W3C XML Schema in ***Figure 2*** corresponds to a collection of the records described by the copybook. The schema contains the information in the copybook. The unit is an element named CUSTOMER-RECORD. Each of these elements has four child elements of type "last-name," "first-name," "age," and "phone." The first two types are strings restricted to 20 and 15 characters, respectively (encoded with the "length" facet). The third type is an integer restricted to three digits (expressed by restricting the "totalDigits"). The fourth type is a string restricted to ten digits (expressed by setting the `pattern` facet to `\d{10}`).

Note that the elements are in the same order as the data in the copybook and that we have translated the last field as a string rather than as an integer (because leading zeros are not ignored, etc.).

The XML Schema in Figure 2 becomes a DFDL specification by changing the beginning to the following:

```
<xs:schema xmlns:xs =
              "http://www.w3.org/2001/XMLSchema"
           xmlns:data="http://dataformat.org/">
  <xs:annotation>
    <xs:appinfo source="http://dataformat.org/">
      <data:defaults>
      <data:format data:encoding="ebcdic-cp-us"/>
      </data:defaults>
    </xs:appinfo>
  </xs:annotation>
...
```

This makes the mapping from a byte stream to XML precise. Here's how it works:

- The `data:` prefix is defined to be bound to the DFDL namespace.
- The `data:defaults` directive is a container for default DFDL format properties.
- The `data:format` directive is the main DFDL declaration containing *property values*.
- The `data:encoding` property, in particular, sets the character encoding to the EBCDIC character set used by COBOL copybook.

That's all. Using this DFDL format, the data with two copybook records in ***Figure 3A*** (as one contiguous stream) is viewed as if it were the XML document in ***Figure 3B***. The DFDL engine generates this view by making some simple assumptions:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="last-name">
    <xs:restriction base="xs:string"><xs:length value=”20”/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="first-name">
    <xs:restriction base="xs:string"><xs:length value="15"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="age">
    <xs:restriction base="xs:int"><xs:totalDigits value="3"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="phone">
    <xs:restriction base="xs:string"><xs:pattern value="\d{10}"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="copybook">
    <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="9999">
        <xs:element name="CUSTOMER-RECORD">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CUSTOMER-LAST-NAME" type="last-name"/>
              <xs:element name="CUSTOMER-FIRST-NAME"
                          type="first-name"/>
              <xs:element name="CUSTOMER-AGE" type="age"/>
              <xs:element name="CUSTOMER-PHONE" type="phone"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 2
W3C XML Schema corresponding to the collection of records described by the copybook in Figure 1

- Only the top-level element is assumed to be the root element.
- The data is expected to contain only the simply typed values mandated by the W3C XML Schema, appearing in the same order as in the XML view.
- The W3C XML Schema textual form is used as the default textual form for each value.

The `data:format` directive allows a large number of properties to be specified directly on element/attribute declarations, on type declarations, or as defaults, including for

- basic byte size and order and character-set encoding used for text,
- binary numeric formats (integer, decimal, and floating-point representation details),
- textual numeric representation properties (base, decimal-point convention, negation mark, etc.),
- identifying leading/trailing tags for specific structures as well as separating sequences (both binary and textual), and
- computation to select between choices or compute property values from the data.

If some business information is stored as real XML documents and some is stored in a representation that DFDL can map as virtual XML, the same XML Schema can be used to describe the XML structure of

```
A
ROSE                KRISTOFFER      0402025555555
ROSE                SOFUS           0060000000000
```

```
B
<copybooks>
  <CUSTOMER-RECORD>
    <CUSTOMER-LAST-NAME>ROSE                </CUSTOMER-LAST-NAME>
    <CUSTOMER-FIRST-NAME>KRISTOFFER     </CUSTOMER-FILE-FIRST-NAME>
    <CUSTOMER-AGE>40</CUSTOMER-FILE-AGE>
    <CUSTOMER-PHONE>2025555555</CUSTOMER-FILE-PHONE>
  </CUSTOMER-RECORD>
    <CUSTOMER-RECORD>
    <CUSTOMER-LAST-NAME>ROSE                </CUSTOMER-LAST-NAME>
    <CUSTOMER-FIRST-NAME>SOFUS          </CUSTOMER-FILE-FIRST-NAME>
    <CUSTOMER-AGE>6</CUSTOMER-FILE-AGE>
    <CUSTOMER-PHONE>0000000000</CUSTOMER-FILE-PHONE>
  </CUSTOMER-RECORD>
</copybooks>
```

Figure 3
Cobol copybook records transformed into DFDL format; (A) copybook records; (B) DFDL format

both. For the real XML, the `dfdl:foo` attributes are simply ignored.

XML processing languages

XML processing languages that run efficiently over the diverse virtual XML data sources discussed earlier represent the third and last tool needed for virtual XML. The most pervasive tool is XPath, of course, but a set of tools also includes XSLT,[8] XQuery,[9] and BPEL4WS (Business Process Execution Language for Web Services),[45] which supports copying and moving XML. The challenge is to implement these efficiently.

The approach that we have adopted in our prototype release, the virtual XML Garden,[46] is to (1) implement the processing languages lazily such that processing is driven in an "on demand" fashion by the consumer accessing the processing result, and (2) analyze expressions and queries to understand the data profile properties of each component. The technical details of how this is achieved are beyond the scope of this paper.[46]

SOLVING BUSINESS PROBLEMS WITH VIRTUAL XML

In this section we describe a number of use cases that illustrate business solutions implemented with virtual XML. These involve a sensor-based computer system, a commercial broker system, an archival system, a file access method, and a data aggregator application.

Use case 1—A sensor-based computer system

The use of sensors in computer systems is increasing, and the integration of diverse sensor systems into a single system infrastructure is becoming desirable. Although sensors do not usually generate XML data, the data is often converted to XML in order to take advantage of common infrastructure and interfaces and to conform to standards. Virtual XML provides the ability to view and process sensor data as XML without performing any conversion, and thus, it helps speed up the operation of the system. In other words, without virtual XML, the emitted sensor data would first be converted into XML in advance of being processed and consumed. With virtual XML technology, the emitted sensor data is transformed to XML as required, in a just-in-time manner.

This use case is an example of applying virtual XML to the *data publisher* pattern. Other data publisher examples include relational-database event publishers, where data is emitted as a result of changes made to a database system, or publishers of system infrastructure events; that is, events such as the failure of a software component or a change in the utilization of a software component.

Publishing such data as XML can be costly, but having a way of processing binary data through general-purpose XML interfaces facilitates the production and distribution of non-XML.

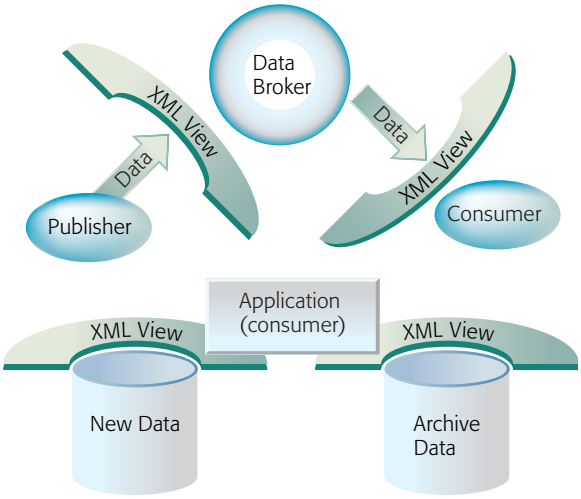

Figure 4
Putting the use cases together

Use Case 2—A commercial broker system

Brokers are software systems that mediate between entities such as service requestors and service providers. They can be viewed as an example of applying virtual XML to the *data broker* pattern. Messaging brokers are used to implement the event-driven and XML-based messaging engine (the bus) of the Enterprise Service Bus.[22] Commercial brokers and messaging systems are becoming more versatile in the types of data they process, transform, and aggregate. Virtual XML provides the ability for brokers and their users to view the data processed or produced as XML, regardless of its format.

Without virtual XML, a broker would have to first convert a non-XML message into XML, before applying any transformations or filtering, to reduce or remove messages from the stream flowing through the broker. With virtual XML, the broker could apply transformations and filters without converting the messages in advance.

Use case 3—An archival system

Many businesses and scientific institutions preserve their data for long periods. They do so for a variety of reasons, such as complying with governmental regulations and the need to perform data mining in order to ascertain trends. The preservation periods can exceed 20 years. Data formats in common use change over time, and the software tools to process the formats become less readily available. When dealing with archival data we can either convert the data in advance, or convert it when there is a need to process it. Virtual XML makes it possible to create XML views on the archival data with currently used tools. The Scientific Annotation Middleware (SAM) system,[47] for example, which makes it possible to view all of the recorded information through a single interface or protocol, follows an approach similar to virtual XML. This use case is an example of applying virtual XML to the *data consumer* pattern.

Use case 4—A file access method

Just as it is desirable to access archival data in a uniform way, it is also desirable to access heterogeneous data stored in a file system in a uniform way. The GEDDM: Grid Based Conversion of Unstructured Data using a Common Semantic Model[48] program is an example of a project that attempts to access a variety of file formats through XML interfaces and is similar in concept to our virtual XML approach. Another possibility that has been discussed at the GGF,[43] is the use of the WS-DAIX (Web Services Data Access and Integration for XML)[49] interface to access heterogeneous data, which can be viewed as an extension to virtual XML. This interface supports XPath and XQuery through Web services. These use cases are examples of applying virtual XML to the *data consumer* pattern as well as using general-purpose virtual XML tools.

Use case 5—A data aggregator application

Often data collection, especially when done from a wide range of sources, is done with some small easy-to-distribute tool that was put together in the past beyond recall. Such ad hoc legacy programs often use standard tools to do their job. By making it easy for programmers to access legacy formats such as the ZIP[50] format in XML, the processing of aggregated data can be moved to use the XML tool stack. This use case is an example of applying virtual XML to the *data aggregator* pattern.

Putting the use cases together

Figure 4 illustrates two scenarios that combine elements of our five use cases in which all data is non-XML data viewed as XML. In the top part of the figure a publisher, such as a sensor device, sends non-XML data to a data broker. Through the use of virtual XML technologies, the data broker views the published data as XML. The data may be processed by the data broker and then sent as non-XML data to a consumer, which views the same data as XML. At the bottom of the figure an application accesses non-

XML data from two sources: a source of new data and a source of existing (archived) data.

The Chartered Arrivals System example[51] demonstrates how a collection of non-XML files can be zipped into one file and transmitted to interested parties who can process the zipped file and its contents by using virtual XML tools and technologies. The example illustrates how particular data items in the individual files within the zip archive can be accessed directly by using XPath or XQuery and combining a specific XML view of the individual files with a default XML view of the zip archive.

CONCLUSION

In this paper we introduce the concept of virtual XML as a way of representing and processing non-XML data as XML. We describe the architectural components needed to enable applications to work with both XML and virtual XML without change: an XML cursor concept that supports various patterns of access to diverse data, a way of describing the data as XML (DFDL), and XML processing languages.

We show that the virtual XML concept can be applied in two ways. We can either construct a *default view* for a well-known non-XML format such as EXIF JPEG[16] or ASN.1,[17] or we can construct a *specific view* for a non-XML format as illustrated in the Chartered Aircraft Arrivals use case, in which we describe a zip archive via DFDL notation.

It is likely that DFDL 1.0 will be published sometime in 2006. Alpha (experimental) software suitable for learning DFDL is becoming available[46] and more examples are expected in 2006. Soon thereafter vendors of message brokering and mapping software, Enterprise Service Bus transformation facilities, and ETL (Extract, Transform and Load) software will be able to support virtual XML by using DFDL. Because additional powerful XML facilities, such as XQuery 1.0, will also be widely available, organizations that wish to correlate non-XML with XML information are likely to adopt virtual XML whenever messaging or query-infrastructure middleware updates are installed.

Many well-known formats, such as the EXIF JPEG format,[16] can be described with DFDL once, and it would be reasonable to expect many of these one-time mapping specifications (default views) will be published. For meta-formats, such as HL7 (earlier than Version 2.4)[10] or ASC X12 (American National Standards Institute [ANSI] Accredited Standards Committee X12 for electronic data interchange),[52] a translator from the data format description to DFDL can be developed. As previously mentioned, researchers have already standardized XML Schema views of relational schemas so that virtual XML will apply to both structured and tabular information. As additional frameworks for analysis of free-form text, such as the IBM Unstructured Information Management Architecture,[53] mature, much free-form text will also be viewable after augmentation in some form of XML, involving items such as resumes, contracts, and clinical notes. For applications in which minimizing the traffic between devices is crucial but general-purpose compression and decompression circuitry is too expensive or draws too much power, DFDL-mapped structured information may be interchanged (although this is logically XML, it does not physically involve instances of XML documents).

■ Virtual XML enables the processing of ZIP files with standard XML tools ■

XML interfaces for accessing information are likely to become a significant part of all enterprise operations because of the trend toward including XML data model services and content transformation and building on top of the virtual XML technologies described in this paper. The Chartered Aircraft System[51] is an illustration of this trend, by demonstrating how an XPath- or XQuery-based query can operate on an aggregate of XML or non-XML documents or records.

We observe that most file systems have a hierarchical arrangement of folders or directories, which corresponds nicely to the XML structure. Therefore, writing XPath expressions that a virtual XML engine interprets (with some navigation occurring in the file system and some navigation occurring within XML documents or a virtual XML document view of structured information in the files) is an appealing next step. Our prototype release[46] already contains examples of such functions.

Similarly, databases that include a large collection of documents could also be addressed as if they were

represented as a single document with an enriched XPath. Using the concept of profiles, an engine accepting XPaths against an entire database would reject queries that could lead to non-terminating searches (for example, looping through documents that are considered siblings). Thus knowledge of scale can be used to avoid accepting impractical or non-terminating queries.

Finally, all the information on the Web that is addressable with URIs might be addressable with a single query as proposed by one of the authors.[54] New metadata (different than DFDL) may be needed to support these applications, and these are some of the long-term research directions that the virtual XML concept opens.

ACKNOWLEDGMENTS

We thank the anonymous reviewers whose feedback was most helpful. We are grateful to the members of the Virtual XML Garden: Lionel Villard, Rajeshwari Rajendra, Paul Castro, Christopher Holtz, William Li, and Stefan Schmidt for their contribution to virtual XML. We thank Manfred Oevers for his comments on an earlier version of this paper.

CITED REFERENCES AND NOTES

1. S. Bourbonnais, V. M. Gogate, L. M. Haas, R. W. Horman, S. Malaika, I. Narang, and V. Raman, "Towards an Information Infrastructure for the Grid," *IBM Systems Journal* **43**, No. 4, 665–688 (December 2004).
2. SGML is described at Cover Pages, http://xml.coverpages.org/sgml.html.
3. *XML Path Language (XPath), Version 1.0, W3C Recommendation*, J. Clark and S. DeRose (Editors), World Wide Web Consortium (November 1999), http://www.w3.org/TR/1999/REC-xpath-19991116.
4. *XML Path Language (XPath) 2.0, W3C Candidate Recommendation*, A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon (Editors), World Wide Web Consortium (April 2005), http://www.w3.org/TR/2005/CR-xpath20-20051103.
5. XMLSpy from Altova, Inc., http://www.altova.com/.
6. The Apache XML Project, Apache Software Foundation, http://xml.apache.org/.
7. *XSL Transformations (XSLT), Version 1.0, W3C Recommendation*, J. Clark (Editor), World Wide Web Consortium (November 1999), http://www.w3.org/TR/1999/REC-xslt-19991116.
8. *XSL Transformations (XSLT), Version 2.0, W3C Candidate Recommendation*, M. Kay (Editor), World Wide Web Consortium (November 2005), http://www.w3.org/TR/2005/CR-xslt20-20051103/.
9. *XQuery 1.0: An XML Query Language, W3C Candidate*, S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon (Editors), World Wide Web Consortium (November 2005), http://www.w3.org/TR/2005/CR-xquery-20051103.
10. Health Level Seven, Inc., http://www.hl7.org/.
11. ACORD Global Insurance Standards, Association for Cooperative Operations Resarch and Development, http://www.acord.org/.
12. The ISO: SQL/XML—Part 14 (SQL 2003) specification, ISO/IEC 9075-14:2003, International Organization for Standardization, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=35341&scopelist=.
13. The ISO: SQL/XML—Part 14 (SQL 2003) specification, XML schema, defined at the International Organization for Standardization, http://standards.iso.org/iso/9075/2002/12/.
14. D. C. Fallside and P. Walmsley: *XML Schema Part 0: Primer (Second Edition), W3C Recommendation*, World Wide Web Consortium (October 2004), http://www.w3.org/TR/2004/REC-xmlschema-0-20041028.
15. JCP JSR 114 JDBC Rowset Implementations XML schema, defined at http://java.sun.com/xml/ns/jdbc/webrowset.xsd.
16. EXIF JPEG format, defined at Japan Electronics and Information Technology Industries Association, http://www.jeita.or.jp/english/standard/html/1_4.htm.
17. ASN.1 (Abstract Syntax Notation 1) is described at http://asn1.elibel.tm.fr/en/.
18. *XQuery 1.0 and XPath 2.0 Data Model, November 2005, W3C Candidate Recommendation*, M. F. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh (Editors), World Wide Web Consortium (November 2005), http://www.w3.org/TR/2005/CR-xpath-datamodel-20051103/.
19. *XML Information Set (Second Edition), W3C Recommendation*, J. Cowan and R. Tobin (Editors), World Wide Web Consortium (February 2004), http://www.w3.org/TR/xml-infoset.
20. *XML Schema Part 1: Structures (2nd Edition), W3C Recommendation*, H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn (Editors), World Wide Web Consortium (October 2004), http://www.w3.org/TR/xmlschema-1.
21. *XML Schema Part 2: Datatypes (2nd Edition), W3C Recommendation*, P. V. Biron and A. Malhotra (Editors), World Wide Web Consortium (October 2004), http://www.w3.org/TR/xmlschema-2.
22. M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen, "The Enterprise Service Bus: Making Service-Oriented Architecture Real," *IBM Systems Journal* **44**, No. 4, 781–797 (December 2005).
23. XMLSoft.org: LibXML2: The XML C Parser and Toolkit for Gnome and Other Systems, with a Variety of Language Bindings (2005), http://xmlsoft.org/.
24. *Document Object Model (DOM) Level 3 Core Specification, Version 1.0, W3C Recommendation*, A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion,

and S. Byrne (Editors), World Wide Web Consortium (April 2004), http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407.

25. *Document Object Model (DOM) Level 3 XPath Specification (Version 1.0), W3C Working Group Note*, R. Whitmer (Editor), World Wide Web Consortium (February 2004), http://www.w3.org/TR/2004/NOTE-DOM-Level-3-XPath-20040226.
26. *RELAX NG Specification*, J. Clark and M. Murata (Editors), OASIS (December 2001), http://www.oasis-open.org/committees/relax-ng/spec-20011203.html.
27. D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade, "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control," *IBM Journal of Research and Development* **20**, No. 6, 560–575 (1976).
28. JDBC Technology, Sun Microsystems, Inc., http://java.sun.com/products/jdbc/.
29. N. Li, J. Hui-I Hsiao, and P. Tijare, "Cursor Management for XML Data," in *Proceedings of XML Database Symposium (XSym 2003), Lecture Notes in Computer Science* **2824**, Springer-Verlag, Berlin, Germany (September 2003), pp. 52–69.
30. .NET Framework Class Library, XPathNavigator Class, Microsoft Corporation, http://msdn.Microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemxmlxpathxpathnavigatorclasstopic.asp.
31. Introduction to OJXQI—The Oracle Java XQuery API, Oracle Corporation, http://www.oracle.com/technology/sample_code/tech/xml/xmldb/jxqi.html.
32. K. Inaba, *Purely Applicative XML Cursor*, http://www.kmonos.net/pub/Slit/index.en.html.
33. XmlCursor Interface, BEA Systems, Inc., http://e-docs.bea.com/workshop/docs81/doc/en/core/index.html.
34. D. Obasanjo, *A Survey of APIs and Techniques for Processing XML*, O'Reilly Media, Inc. (2003), http://www.xml.com/pub/a/2003/07/09/xmlapis.html.
35. *XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Candidate Recommendation*, A. Malhotra, J. Melton, and N. Walsh (Editors), World Wide Web Consortium (November 2005), http://www.w3.org/TR/2005/CR-xpath-functions-20051103/.
36. *Simple API for XML, Version 2.0.2* (April 2004), http://www.saxproject.org/.
37. J. Beatty, S. Brodsky, M. Carey, R. Ellersick, M. Nally, and R. Preotiuc-Pietro, *Service Data Objects, Version 2.0*, IBM Corp. and BEA Systems, Inc. (June 2005), ftp://www6.software.ibm.com/software/developer/library/j-commonj-sdowmt/Commonj-SDO-Specification-v2.0.pdf.
38. K. Rose and L. Villard, "Phantom XML," *Proceedings of the XML 2005 Conference*, November 14–18, Atlanta, Georgia (2005), http://www.idealliance.org/proceedings/xml05/abstracts/paper80.HTML.
39. Binary Format Description Language (BFD) project at the Pacific Northwest National Laboratory, U.S. Department of Energy, http://collaboratory.emsl.pnl.gov/sam/bfd/.
40. BizTalk project from Microsoft Corporation, http://www.microsoft.com/biztalk/default.mspx.
41. T. Restrepo, *BizTalk 2004 Flat File Schema Tutorial (Parts 1 and 2)*, http://www.winterdom.com/dev/bts/.
42. M. Beckerle, *Data Format Description Language (DFDL), A Proposal: Data Format Description Language Working Group Working Draft* (2005-08-29), https://forge.gridforum.org/projects/dfdl-wg.
43. Global Grid Forum, http://www.ggf.org/.
44. M. Westhead, M. Beckerle, and J. Myers, *Data Format Description Language Working Group*, Global Grid Forum, 2004, http://forge.gridforum.org/projects/dfdl-wg.
45. BPEL4WS (Business Process Execution Language for Web Services), described at http://www.ibm.com/developerworks/library/specification/ws-bpel/.
46. Virtual XML Garden, described at http://www.alphaworks.ibm.com/tech/virtualxml.
47. Scientific Annotation Middleware (SAM) described at http://www.scidac.org/SAM.
48. K. Loughran, P. Donachy, T. J. Harmer, R. H. Perrott, M. Prentice, S. Bearder, and J. Rasch, *GEDDM: Grid Based Conversion of Unstructured Data Using a Common Semantic Model*, http://www.allhands.org.uk/2004/proceedings/papers/166.pdf.
49. M. Antonioletti, S. Hastings, A. Krause, S. Langella, S. Laws, S. Malaika, and N. W. Paton, *Web Services Data Access and Integration—The XML Realization (DAIS-WG)* (December 2005), https://forge.gridforum.org/projects/dais-wg.
50. Zip Archive, described at http://www.info-zip.org/.
51. Chartered Aircraft System, IBM Corporation, http://domino.research.ibm.com/comm/research_projects.nsf/pages/virtualxml.examples.html.
52. ASC X12 (American National Standards Institute (ANSI) Accredited Standards Committee X12 for Electronic Data Interchange), described at http://www.x12.org/.
53. D. Ferrucci and A. Lally, "Building an Example Application with the Unstructured Information Management Architecture," *IBM Systems Journal* (Special Issue on Unstructured Information Management) **43**, No. 3, 455–475 (2004).
54. *The XML World View*, ACM Symposium on Document Engineering, University of Wisconsin-Milwaukee, USA (October 2004), http://www.sdml.info/doceng2004.

GENERAL REFERENCES ON XML VIEWS

These references are organized by location or subject as follows:

IBM Almaden Research Center (the first work on XML views was done here):

M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. Subramanian, "XPERANTO: Publishing Object-Relational Data as XML," *Proceedings of the Third International Workshop on the Web and Databases (WebDB 2000)*, May 18–19, 2000, Dallas, Texas, ACM, New York (2000), pp. 105–110.

J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald, "Efficiently Publishing Relational Data as XML documents," *Proceedings of the Third International Workshop on the Web and Databases (WebDB 2000)*, May 18–19, 2000, Dallas, Texas, ACM, New York (2000), pp. 65–76.

INRIA:

S. Abiteboul, "On Views and XML," *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1999)*, May 31–June 2,

1999, Philadelphia, Pennsylvania, ACM, New York (1999), pp. 30–38.

University of Washington:

Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld, "An Adaptive Query Execution System for Data Integration," *Proceedings of the ACM SIGMOD International Conference on Management of Data,* June 1–3, 1999, Philadelphia, ACM, New York (1999), pp. 299–310.

University of Pennsylvania and AT&T Research:

M. F. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima, and W-C. Tan, SilkRoute," A Framework for Publishing Relational Data in XML," *ACM Transactions on Database Systems* **27**, No. 4, 438–493 (2002).

UCSD Mix project—The idea of virtual views of data as XML was explored with early strategies documented in the following references:

B. Ludascher, Y. Papakonstantinou, and P. Velikhov, "Navigation-Driven Evaluation of Virtual Mediated Views," *Proceedings of the 7th International Conference on Extending Database Technology (EDBT 2000),* Konstanz, Germany, March 27–31, 2000, *Lecture Notes in Computer Science* **1777**, Springer, Berlin (2000), pp. 150–165.

C. Baru, V. Chu, A. Gupta, B. Ludascher, R. Marciano, Y. Papakonstantinou, and P. Velikhov, "XML-Based Information Mediation for Digital Libraries," *Proceedings of the Fourth ACM Conference on Digital Libraries*, August 11–14, 1999, Berkeley, CA, ACM, New York (1999), pp. 214–215.

A large body of work on adapting relational data into XML has been published, including the following references:

P. Bohannon, H. F. Korth, and P. P. S. Narayan, "The Table and the Tree: On-Line Access to Relational Data through Virtual XML Documents," *Proceedings of the Fourth International Workshop on the Web and Databases (WebDB 2001)*, Santa Barbara, California, USA, May 24–25, 2001, ACM, New York (2001), pp. 55–60.

M. L. Lo, S-K Chen, S. Padmanabhan, and J-Y Chung, "XAS: A System for Accessing Componentized, Virtual XML Documents," *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, May 12–19, 2001, Toronto, Ontario, Canada, IEEE, New York (2001), pp. 493–502.

P. Bohannon, S. Ganguly, H. F. Korth, P. P. S. Narayan, and P. Shenoy, "Optimizing View Queries in ROLEX to Support Navigable Result Trees," *Proceedings of the 28th International Conference on Very Large Databases (VLDB 2002)*, Morgan Kaufmann Publishers, San Francisco, CA (2002), pp. 119–130.

A large body of work on adapting HTML pages into XML has been published, including the following references:

A. Sahuguet and F. Azavant, "Web Ecology: Recycling HTML Pages as XML Documents Using W4F," *Proceedings of the ACM SIGMOD Workshop on The Web and Databases (WebDB'99)*, June 3–4, 1999, Philadelphia, Pennsylvania, ACM, New York (1999), pp. 31–36.

J. Naughton et al., "The Niagara Internet Query System," *IEEE Data Engineering Bulletin* **24**, No. 2, 27–33 (2001).

Accepted for publication November 29, 2005.
Published online May 16, 2006.

Kristoffer H. Rose
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (krisrose@us.ibm.com).
Dr. Rose received a Ph.D. degree in computer science from the University of Copenhagen in 1996, doing research in tree and graph rewriting systems. After four years in academia, the last as an associate professor at Ecole Normale Superieur in Lyon, France, he joined the Watson Research Center in 2000. At IBM, he is working on XML technology with a special interest in how XML and the XML processing languages (XSLT, XQuery, etc.) can be implemented so that they can be used efficiently over large data structures even when those are not in XML. Most recently he has been experimenting with (and has implemented) Data Format Description Language (DFDL) to this end.

Susan Malaika
IBM, 294 Route 100, Somers, New York 10589 (malaika@us.ibm.com).
Susan Malaika is a Senior Technical Staff Member in IBM's Information Management Group. She develops standards that support data for grid environments at the Global Grid Forum. Her specialties include XML, the Web, and databases. In addition to working as an IBM product software developer, she has also worked as an Internet specialist, a data analyst, and an application designer and developer. She has also co-authored a book on the Web and published articles on transaction processing and XML. She is a member of the IBM Academy of Technology.

Robert J. Schloss
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (rschloss@us.ibm.com).
Robert Schloss, a Senior Technical Staff Member, is working on runtime tools for XML and Web Services middleware at the Watson Research Center in the Next Generation Web Group. He received an A.B. degree in mathematics and computer science from Yale University. Mr. Schloss is a founding member of the XML Research group at IBM, and he has a long involvement with Web data interchange standards. ■