The ITB Journal

Volume 1 | Issue 1 Article 7

2000

XML for Business to Business Data Exchange

Geraldine Gray

David Kerwick

XML for Business to Business Data Exchange

By Geraldine Gray & David Kerwick,
School of Informatics and Engineering, ITB Blanchardstown, Dublin, Ireland

Abstract

This paper examines to use of XML for business to business data exchange. Starting with creating an XML document from an existing data source and transmitting that document, we explain some of the supporting standards for XML which facilitate automated processing and transformation of an XML document. Finally we look at the advantages of using XML, and why it is expected to revolutionise electronic data interchange.

Introduction to XML

The Extensible Markup Language (XML), which is a subset of SGML[7], was developed by the World Wide Web Consortium (W3C) to provide a freely available, widely transportable methodology for controlled data interchange. XML was designed principally for the exchange of information in the form of computer documents over the Internet.[4], [7]. An XML document contains data, and tags which describe that data. The tags are syntactically similar to HTML tags, however unlike HTML, XML tags are not pre-defined. When creating a document, you decide what tags are required in a document, and what each tag will be called. This allows an XML document structure to mirror the equivalent business documents. Figure 1: XML Document.

```
<? XML version="1.0"  ?>
<!DOCTYPE prescription SYSTEM "prescription.dtd">
 <prescription>
        <date> 14th May 1999 </date>
        <doctor>
                <name> Joe Bloggs </name>
                <address> Dublin 1 </address>
        </doctor>
        <medication>
                <product> Elicon 250g</ product >
                <quantity>  1 </quantity>
        </medication>
</prescription>
```

Figure 1: XML Document - medical prescription

[7] HTML, the language currently used to create web pages, is also a subset of SGML.

XML is very strict in regards to how it is formed, The following rules dictate the composition of a syntactically correct XML document [3] :

- The document must be surrounded by a single outermost root element or document.
- As with HTML, a start tag can have attributes, however no attribute can appear more than once in the same start-tag.
- Tags must be properly nested and all tags must have matching pairs.
- An XML document can make reference to constant declarations, called parameter entities. These must be declared before they can be used.
- You can also use binary entities to reference images or files, provided you have declared them as **entity** or **entities**. They cannot be referenced in the content.
- All entities, elements, and attributes are case sensitive.
- String attribute values cannot contain references to external resources.

The XML family of standards

The XML standard itself is quite simple. There is however a family of standards supporting XML, some not yet available, which are complex, and cover a range of IT disciplines [1] . This paper will cover some of these standards, and where they fit in a business to business data interchange solution.

Creating an XML document

The first step in data interchange is to extract the data from an existing data source, be it RDBMS, legacy system or flat files, and use it to populate an XML document. This step can be implemented in a range of development environments. Each data item would be read from source, wrapped in the appropriate tag and outputted to a text file creating the XML document.

Alternatively, this step may involve converting an existing EDI message into an XML document. XML Solutions Corporation have developed a suit of software to convert from the X12 standard to XML and are currently developing software to convert EDIFACT messages to XML [10]

Defining the document content

To automate the processing of an XML document, it must conform to some pre-determined structure. There are two methods for defining the content of an XML document, using a DTD (Document Type Declaration) or using an XML schema.

A DTD, which can be embedded in the XML document, or created separately and referenced in the XML document, imposes a structure on an XML document by defining:

- the tags that are to be included in the document,
- whether a tag is mandatory or optional,
- which tags will be embedded in other tags,
- the tags that can be repeated in the document, and
- what data elements will be enclosed in each tag.

Trading partners would agree on the structure of the DTD. All transmitted XML documents would then conform to the DTD structure. An XML parser (discussed in section 2.5.1 below) can validate the XML document against the DTD to ensure it contains the correct information. Figure 2 DTD.

```
<?xml version="1.0"?>
<!DOCTYPE prescription [
<!ELEMENT date(#PCDATA)>
<!ELEMENT doctor(name,address)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address(#PCDATA)>
<!ELEMENT medication(product,quantity)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT quantity(#PCDATA)>
]>
```

Figure 2 DTD.

This DTD specifies that a prescription tag must contain a date, doctor and medication tag. A doctor tag must enclose a name and address tag. The medication tag must contain product and quantity tags. Name, address, product and quantity tags contain character data.

DTD can also inherit a definition from another other DTD. So you can define each entity required by an application, and then import these definitions (external entities) into the Document Type Declaration. This allows for code reuse, and easy maintenance of the entity definitions.

DTDs are the official W3C standard for enforcing the structure of an XML document, however DTDs have the following limitations:

- A limited set of pre-definined data types
- DTD is a separate language from XML, and so involves an additional learning curve.

An alternative to using a DTD is to use an XML schema, which is written in XML syntax. A schema encompases the same functionality as a DTD, but has a richer set of data types, and has additional features including the ability to represent relationships between data elements. The W3C standard for schemas is not yet complete, however they are gaining increasing support in the industry [2], [6]. Figure 3 below gives an example of an XML schema for the medical prescription in Figure 1.

```
<?xml version="1.0"?>
<s:schema>
      <elementType id=”name”>
            <string/>
      </elementType>

      <elementType id=”address”>
            <string/>
      </elementType>

      <elementType id=”doctor”>
            <element type=”#name”/>
            <element type=”#address”/>
      </elementType>

      <elementType id=”product”>
            <string/>
      </elementType>

      <elementType id=”quantity”>
            <string/>
      </elementType>

      <elementType id=”medication”>
            <element type=”#product”/>
            <element type=”#quantity”/>
      </elementType>

      <elementType id=”prescription”>
            <element type=”#doctor”/>
            <element type=”#medication”/>
      </elementType>

</s:schema >
```

Figure 3 Schema for the prescription

Standardising XML document content for business documents

One of the initial problems with EDI was that everybody was using different message formats meaning that separate messages had to be produced for each trading partner. To overcome this the EDIFACT and X12 standards were developed to standardise message content. As yet there is no equivalent standard for XML document content. However there are repositories of XML schemas for a number of business documents which in time could form the basis for a standard. The purpose of the repository is to generate a consensus on document content for business transactions. The two main repositories of XML schemas are xml.org from a consortium of companies, and Biztalk from Microsoft. See XML.com have a more complete list of repositories. [5].

Document transmission

An XML document can be transmitted using the HTTP[8] protocol. Documents can be pushed from source, or pulled from their destination using standard HTTP methods.

For a more robust, secure and fault tolerant solution, a number of middleware products now support XML including BEA's WEBLOGIC[11] and webMethods B2B[12].

Automated processing of XML documents

There are a family of standards to support the automatic processing of XML documents. DOM and SAX are the two main programming interfaces to process XML documents, while XSLT which is at recommendation stage, covers how to transform data in an XML document to other formats. In this section we shall explain these three standards, and also look at XML parsers, which process the XML document. Most XML parsers include implementations of the DOM and SAX standards.

XML parsers

An XML parser (or processor) reads through an XML document to check that it is both syntactically correct according to the rules described in section 1 above, and that it's content conforms to the corresponding DTD if one exists.

XML parsers are available free. Tests done in September 1999 rated the following as the top for parsers: [10]

Sun ``Java Project X"
IBM XML4j
Oracle XML Parser
Microsoft MSXML in Java

Document Object Model (DOM)

The Document Object Model is an official W3C standard, originally used with Dynamic HTML, but now adapted for XML[2]. Like all official standards DOM is quite large and powerful. A DOM parser converts the XML document into a tree structure, called a DOM tree as shown in Figure 4 below, and provides a

[8] Hyper Text Transfer Protocol used to transmit data over the Internet. It consists of a set of rules that define how web servers and browsers communicate with each other over a TCP/IP connection.

range of methods for processing the DOM tree. Methods include traversing the tree, retrieving particular data items, and adding, updating and deleting data items.

The following example will shows two alternative methods of processing the data in a DOM tree. These examples were written in Java. Both are based on the prescription document in **Figure 1: XML Document**

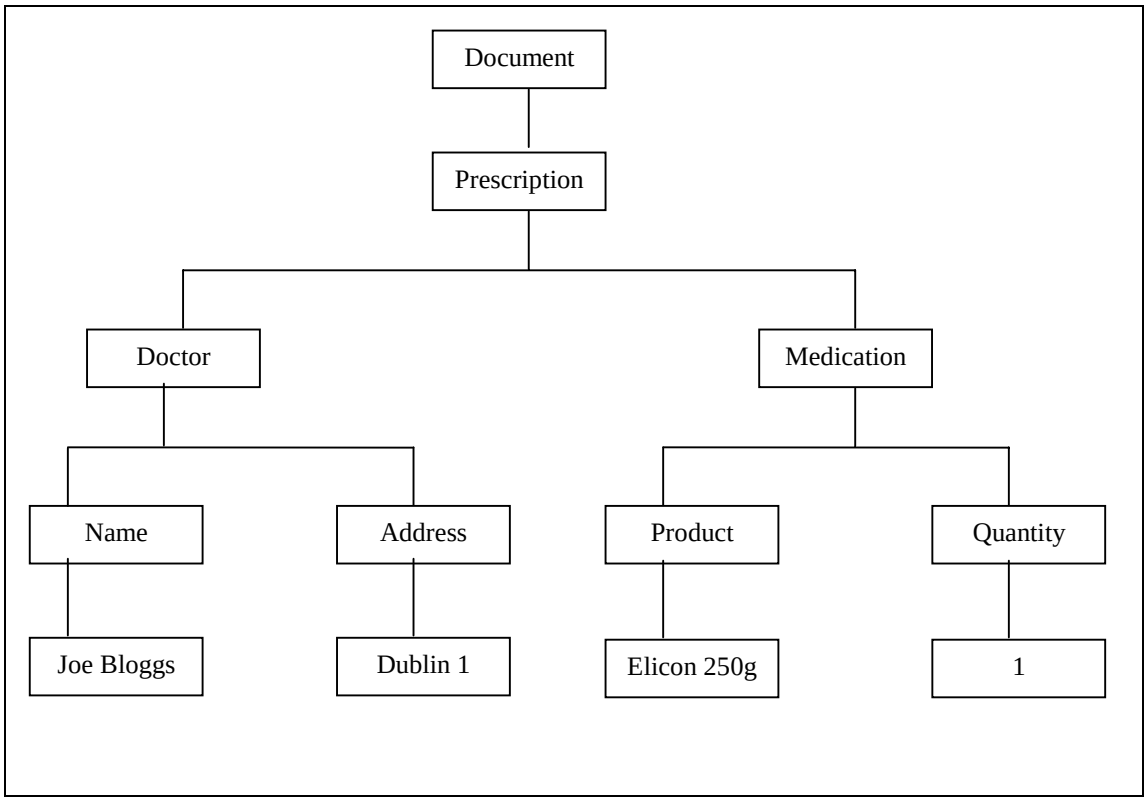

Figure 4 DOM tree for a prescription

Example1: Traversing the tree

This example illustrates methods which allow you to move between nodes on the tree. Starting at the root note (prescription), the code moves to the product being subscribed, and prints it[9].

[9] Note: This code assumes no carriage return or white spaces in the XML document. The DOM standard does not make allowances for carriage return characters and spacing used to format an XML document. If the XML was written as per Figure 1, the prescription node would actually have 5 children:

1. Carriage return and spaces
2. <doctor>
3. Carriage return and spaces
4. <medication>
5. Carriage return and spaces

```
root = ((Document)node).getDocumentElement();
prescription = root.getFirstChild();
doctor = prescription.getFirstChild();
medication = doctor.getNextSibling();
product = doctor.getFirstChild();
productText = product.getFirstChild();
System.out.print("Product is: " + "\"" +
productText.getNodeValue() + "\"");
```

Example2: Searching the tree

An alternative method of processing data in a DOM tree is to search for tags by name. A fragment of this code is given below.

```
Node medication, product, productText;
root = ((Document)node).getDocumentElement();

      //converts the node to an Element
Element rootElement = (Element)root;

      //return a list of the children of this element which
      //have the tag name medication
NodeList findMedication =
rootElement.getElementsByTagName("medication");

      //the medication node will be the first one in the list
      // as it is the only one
medication = findMedication.item(0);

      //converts the node to an Element
Element medicationElement = (Element) medication;

      //return a list of the children of this element which
      //have the tag name “product”
NodeList findProduct =
      medicationElement.getElementsByTagName("product");

product = findProduct.item(0);

      //moves into the text portion of the product node
productText = product.getFirstChild();
      //prints out the value stored in the product tag
System.out.print("Product is: " + "\"" +
            productText.getNodeValue() + "\"");
```

This code is more flexible as it will work regardless of the white spaces used or if other elements are added into the XML document. Is does require that the tag names of the two elements used, medication and product, are not changed.

One of the drawbacks of DOM is the amount of space required to create the DOM tree for large XML documents. For example searching an XML document of 100,000 element for just one element still requires the overhead of creating a tree for the 100,000 elements.

SAX:

The Simple API for XML is a defacto standard created by the XML-dev mailing list. SAX is an event driven model developed to overcome the DOM overhead for simple processing.

As an XML document is parsed a number of events are triggered. An application can be coded to listens for particular events and then execute the corresponding code for that event. This has a very low memory overhead and is much more efficient than DOM for the search example above. But as the name suggests this is a simple API and so does not support the same level of functionality provided by DOM.

Continuing from the DOM examples above, the approach to finding the product name in the medical prescription document is quite different with SAX. SAX parsers work on the principle of 'listeners', which are activated when the action they are listening for gets triggered.

The example below listens for two events, the start of elements and character data. The listeners for this are

startElement(String name, AttributeList attrs) and

characters(char ch[], int start, int length)

The startElement method will be fired when the parser reads the

1. prescription tag
2. doctor tag
3. medication tag
4. name tag
5. address tag
6. product tag
7. quantity tag

and the characters method will be fired when the parser reads

1. John Doe
2. Dublin 1
3. Elicon 250g
4. 1

Below is the code for the two listeners, the code is written in Java.

```
//listener that's fired everytime a element is started
public void startElement(String name, AttributeList attrs){
            //checks to see if this element if the product
            //element
      if (name.equals("product")){
            //if it is print out the name of the element
            System.out.print("product = ");
            found = true;
      }
}


//listener that's fired everytime the parser comes
//across character data
public void characters(char ch[], int start, int length){
      //Checks the flag to see if this is the data we are
      //interested in
  if(found == true){
    System.out.print(new String(ch, start, length));
    found = false;
  }
}
```

As the program parses the XML document the above code compares the start elements to "product". When a match is found the startElement method will print out '`product` = ' and set the value of found to true. At this stage the next event triggered by the parser is for the character data 'Elicon 250g' that is picked up by the characters method, because the flag found is true it will print out this data. The found flag is set back to false to prevent the program printing out the rest of the character data in the document.

Transforming the XML document.

XSLT (XSL (extensible stylesheet language transformation) which is an extension of the W3C standard for XSL, is a standard which covers how to convert an XML document into a range of other formats including a flat file or a word document. The standard is currently at recommendation stage, but will facilitate updating existing data sources from an XML document [8].

Displaying an XML document in a browser

XSLT can also be used to display an XML document in a browser. Using XSLT you can dictate the tags to include / omit, and how to display the information, for example in a table or frame, colours, font sizes, backgrounds etc. There is also limited processing logic for loops and conditional statements.

As the standard is yet to be finalised, browser do not yet support the standard, However at the time of writing three browsers support XML documents:

- Microsofts Internet Explorer
- Netscape Navigator
- Opera

Internet Explorer 5 uses a default style sheet for displaying XML documents. You can specify your own format using an implementation of a draft of the XSLT standard, or alternatively you can use a Cascading Style Sheet. Netscape does not yet support XSLT, but does support the use of a CSS to format the XML document. However the implementation of the CSS standard used by IE5 and Netscape are not the same. Opera also supports XML formatting using CSS's.

Benefits of Using XML

One of the primary advantages of XML is the low running and implementation costs of an XML solution.[9]. XML data is transmitted over the Internet. While there is an associated cost, it is considerably cheaper than using a VAN. Also as it is standard practice to encrypt sensitive data transmitted over the Internet, the security and integrity of the data during transmission is guaranteed.

Many implementations of XML related standards such as DOM and SAX's are available free, and the wide spread industry support for XML has resulted in a wealth of products coming to market which support XML.

A second significant advantage of XML is the flexibility of an XML document format. An XML document can be changed, without changing the software that sends, receives and processes the document. So the document content can be updated to meet the criteria of a new trading partner without impacting on existing trading partners solutions.

For example, using XML to describe medication on a prescription as shown in the following code:

```
<medication>
        <product> Elicon </ product >
        <quantity>  250g </quantity>
</medication>
```

an application can locate a particular tag and extract its value, regardless of the order of the tags within the document. Code to locate the product tag, as in examples 2 and 3 in section 2, would still work if , for example, a price tag was added to the document.

Finally there is a range of W3C standards and recommendations supporting XML. We have looked at some of these in section 2 above including DTDs, XML schemas, DOM, SAXs and XSLT. There are a

number of others including a query language of XML. This technology is in its infancy as yet, but the volume of support given to XML, and the rate at which standards are emerging and developing, will guarantee is widespread adoption in the industry.

Conclusion

XML is expected to revolutionise business to business trading, and become the standard for all e-commerce applications. As with EDI, XML can transmit message to a pre-determined format. It is however significantly cheaper to implement and is a more flexible solution than EDI. It is also supported by a powerful family of standards including standards to process, update, query and transform XML documents. There is a range of good quality software and tools that are available free and as open source code. Industry leaders have all adopted XML, so there is a vast array of XML compatible software on the market place. Therefore small and medium size enterprises, for whom the cost of an EDI solution proved prohibitive, can now benefit from the advantages of business to business electronic data exchange.

References

[1] XML Marks Up Application Integration, Gartner Group Symposium, Itxpo99, 1999

[2] www.w3c.org

[3] XML Black Book – Natanya Pitts-Moultis and Cheryl Kirk

[4] XML for Dummies – Ed Tittel and Norbert Mikula

[5] www.xml.com

[6] www.schema.net

[7] www.xml-zone.com

[8] www.ibm.com/developer/xml/

[9] Jeffrey Ricker, Drew Munro, Doug Hopeman, XML and EDI, peaceful co-existence, www.xmls.com – XML Solutions Corporation.

[10] Jeffrey Ricker, Drew Munro, Doug Hopeman, XML representation of X12 EDI, www.xmls.com – XML Solutions Corporation.

[11] www.bea.com/products/index.html

[12] www.webmethods.com/content/0,1107,SolutionsIndex,FF.html