A Practical Introduction to the XML, Extensible Markup Language, By Way of Some Useful Examples

Robin Snyder
Savannah State University
126 Jordan, P. O. Box 20359
Savannah, GA 31404
(912) 356-2716
snyderr@savstate.edu
http://www.RobinSnyder.com

Abstract

XML, Extensible Markup Language, is important as a way to represent and encapsulate the structure of underlying data in a portable way that supports data exchange regardless of the physical storage of the data. This paper (and session) introduces some useful and practical aspects of XML technology for sharing information in a educational setting (e.g., class rosters). Such ideas can be useful for both end-users requesting information from the computer support staff and for the computer support staff in providing information needed and/or requested by end-users in a useful manner.

Introduction

Anyone who has looked at any book on XML will probably go away wondering what XML is all about. This paper introduces some useful and practical aspects of XML technology for sharing information in a educational setting by way of some, hopefully, useful examples.

Officially, XML stands for Extensible Markup Language.

XML, Extensible Markup Language, has become increasingly important in recent years as a way to represent and encapsulate the underlying data structures of a problem in a portable way that supports data exchange of the relevant data, regardless of the actual physical storage of the data, while supporting data presentation at the client browser using HTML. This paper presents a way to use XML to represent the critical path project management problem using, as an example, project management software written by the author for classroom use. As such, the paper serves as both an introduction to XML in general and as a specific application of XML.

XML, Extensible Markup Language, is a metalanguage that has become increasingly important in recent years as a way to represent and encapsulate the underlying data structures of a problem in a portable way that supports data exchange of the relevant data, regardless of the actual physical storage of the data, while supporting data presentation at the client browser using HTML [10]. Like HTML, XML can be used to format text, but, in addition, XML can be used to structure the underlying data.

When using XML, the first step is to design an XML data format for exchange. There are many ways to do this. In practice, the preferred way is the way in which everyone has agreed to repre-

sent a particular problem so that all of the systems are interoperable. In XML, customized tags are created to represent the data. The meaning of each tag is determined by mutual agreement of those using the XML format. XML was designed so that data in XML format can be self-describing. Here is one way to represent a sample of virtual red and blue M&M's using XML.

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE sample SYSTEM "mm1.dtd">
<?xml-stylesheet type="text/xsl" href="mm1.xsl" ?>
<sample type="M&amp;M" id="0">
<item value="red"  />
<item value="blue" />
<item value="red"  />
<item value="red"  />
<item value="blue" />
<item value="red"  />
<item value="blue" />
<item value="red"  />
</sample>
```

The first three lines are header information that specify the XML version, the syntax requirements of the XML (described below), and the stylesheet to be used to visually depict the XML (described below). The rest of the XML, for this example, represents the structured data. XML looks superficially similar to HTML except that XML has a much stricter syntax. In particular, every tag needs and ending tag. The following are equivalent.

```
<item value="red" />
<item value="red"></item>
```

Every valid XML document must have a root element. In an HTML file, the `html` element is the root element. In a document representing virtual M&M's, the `sample` element is the root element. In this XML format, the `type` attribute for the `sample` tag is the type of sample and the `id` attribute is the database identifier for the sample, or `0` if there is no database behind the scenes (i.e., the virtual M&M sample is dynamically generated but not stored persistently).

One way to make the data format more robust for exchanging data in XML format is to specify some syntactic restrictions on the data format using a DTD, Data Type Definition.

Here is the (external) DTD for the previous XML that, for reference purposes, is stored in file `mm1.dtd`.

```
<!ELEMENT sample (item*)>
<!ELEMENT item EMPTY>
<!ATTLIST sample
   type CDATA #REQUIRED
   id   CDATA #REQUIRED
   >
<!ATTLIST item
   value CDATA #REQUIRED
   >
```

Without getting into too many technical details, here is a short explanation of the above DTD.

- The root element of the document is `sample`, so `<sample>` and `</sample>` tags are needed.
- The tag element `sample` contains zero or more `item` tags.
- The `sample` tag has the required attributes `id`, the database id, if any, of the sample, and `type`, the type of the sample.
- The `item` tag is empty and has the required attribute `value` which is unevaluated text (i.e., `CDATA`).

The primary use of the DTD is to insure that interacting processes developed by different people, organizations, etc., conform to an agreed-on data format. Thus, the use of the DTD is not strictly necessary and can be omitted when first starting to use XML until one has gained some experience with XML.

There are many ways to visually depict the underlying data. An XML-compliant browser, such as Microsoft Internet Explorer 6.0, will, in the absence of a style sheet, display the XML tree.

Another way to visually depict the underlying data is to use a stylesheet. One stylesheet format is CSS, Cascading Style Sheets. A problem with using CSS is that attributes of tags cannot be displayed. Another way is to use XSLT, Extensible Stylesheet Language with Transformations, which is essentially a system that transforms tree structures into trees (or other output).

Another common method to visually depict the underlying data is XSLT, which allows access to the attributes of each tag. An XSLT stylesheet file is an XML file so it must adhere to the rules for an XML file. By changing the stylesheet used, the HTML output can be easily changed.

Another way is to use program code that explicitly loads the data and displays it. A simple way to do this is with SAX, a Simplified API for XML. A more powerful, complicated, and more computationally expensive way to do this is with the XML DOM, Document Object Model. The Microsoft MSXML component makes this fairly easy to do.

Usefulness

An example to justify the usefulness of XML is as follows. Suppose that ***n*** groups wish to share related data, but each has a specific way of storing that data (e.g. transcript information by a Registrar at a University). In order to share data, each of the ***n*** groups must write customized program code to export to ***n-1*** data formats and to import from ***n-1*** data formats. This could be a nontrivial effort, as in general $2*(n-1)^2$ data format conversions are required to make each groups format interoperable with every other groups format. If there were a common intermediate language for data transfer, then, if each group created ***1*** conversion of their format into that intermediate language and ***1*** conversion of their format from that intermediate language, only ***2*n*** data format conversions would be needed for all the groups, rather than $2*(n-1)^2$. XML is such an intermediate format. And, in the case of XML, each of the ***2*n*** data format conversions can use standard software specifications and software to work with XML and XML-related technologies. Finding qualified people to do the transformations should become easier as XML achieves more widespread usage. These are some of the motivating factors behind XML. Note that the previous

analysis is similar to using an intermediate language between ***n*** high level source languages and ***m*** low level target languages whereby, instead of ***n*m*** complete compilers, only ***n*** front end (i.e., source code handling) compilers and ***m*** back-end (i.e., target code generation) compilers are needed.

For example, one might want to make student transcript information available to approved users (e.g., faculty advisers). Allowing faculty access to the actual database might be either difficult or place a burden on the database management system (i.e., the institutional database system). A common approach, used in business for many years, is a three tier client-server model whereby the data is replicated and synchronized to/from an intermediate server so that end-users can easily access the data but not bog down the mainframe system. Since transcripts do not change very often, and, in most cases, an adviser is interested only in the transcript for a one student at any one time (i.e., not in a query that asks which students have taken a given course), then an XML transcript record could be very useful.

The primary problem is coming up with in XML format to which everyone agrees. Then both sides, the user interface side (i.e., boundary object) and database side (i.e., entity object) are separated from each other, allowing independent changes on either side so long as the XML format is maintained.

Even if data is not to be shared among groups, it might be shared between one web page. One useful example is that of a text database.

This process can be simplified to three steps.

Step 0. Decide on an XML format.

Step 1. Convert the data to XML.

Step 2. As needed, convert the XML to a display format (e.g., HTML).

Some examples will now be briefly discussed.

Text databases

XML can often be used to facilitate database access for data for which a database management system is not needed or desired. Using a data management system, such as SQL Server, introduces an overhead and, in some cases, a licensing fee, as well as support costs. The advantage of a relational database system is that data can be accessed in a relational manner. That is, the data can be accessed in many related ways that are not dependent on the fixed structure of the database. However, there are many applications where a full relational model is not needed. In such cases, a hierarchical database structure can be used in the form of XML. Note that it may still be useful to keep the data in a relational database, but generate the XML to be accessed in a hierarchical manner.

The author was looking for an application of XML for storing a large text database. The large text database used was a public domain version of the 1611 King James Bible that consisted of several megabytes of text organized into books (66), chapters (over 1000 total), and verses. Other text databases might include the works of Shakespeare, etc.

The approach used was as follows.

Step 1. A (Delphi/Turbo Pascal) program was written to take the (huge) text file and partition it into XML consisting of books containing chapters containing verses. The XML directory consists of over 5MB of data in 1190 files. This is fairly easy once the format of the XML is decided.

The XML directory format used the number of the book (i.e., 01 to 66) as a prefix, followed by the number of the chapter of the book (i.e., 001 to 999). So, the first chapter of Genesis is in file 01-001.xml while the last chapter of Revelation is in file 66-022.xml. The file 00-000.xml is the index file and has the following format.

```
<?xml version="1.0" standalone="yes"?>
<rmsBible>
   <rmsBook index="01" chapters="50" book="Genesis"/>
   ... books 02 to 65 omitted ...
   <rmsBook index="66" chapters="22" book="Revelation"/>
</rmsBible>
```

A chapter in a book, such as Genesis 1, in file 01-001.xml, has the following format.

```
<?xml version="1.0" standalone="yes"?>
<rmsChapter book="Genesis" chapter="1" verses="31">
<rmsVerse index="1">
In the beginning God created the heaven and the earth.
</rmsVerse>
... verses 3 to 31 omitted ...
<rmsVerse index="31">
And God saw every thing that he had made,
and, behold, it was very good.
And the evening and the morning were the sixth day.
</rmsVerse>
</rmsChapter>
```

Thus, starting at the index file 00-000.xml, any verse in any chapter can be accessed.

Step 2. A web page using ASP, Active Server Pages, was written to provide access to the text. Thus, one web page provides access to over 1000 chapters. Creating thousands of similar web pages to access the data would not be a good idea.

For completeness, here is the HTML/ASP that could be used to load the first chapter of the first book, 01-001.xml, and generate the HTML.

```
<%@ language="VBSCRIPT" %>
<% option explicit %>
<% Response.Buffer = True %>
<html>
```

```
<head>
<title>King James Bible</title>
<body>

<%
Sub loadChapter(fs)
Dim xml1, doc1
Dim book1, chapter1
Dim nodes1, i, n

   Set xml1 = Server.CreateObject("MSXML2.DOMDocument.4.0")
   xml1.async = False
   xml1.resolveExternals = False
   xml1.load fs

   Set doc1 = xml1.documentElement
   book1 = doc1.getAttribute("book")
   chapter1 = doc1.getAttribute("chapter")
   Response.Write book1 & " " & chapter1 & ":"
   Response.Write "<ol>"
   Set nodes1 = doc1.selectNodes("rmsVerse")
   n = nodes1.length
   For i = 1 To n
      Response.Write "<li>" & nodes1.item(i-1).text & "</li>"
      Next
   Response.Write "</ol>"
   Set doc1 = Nothing

   Set xml1 = Nothing
   End Sub
%>

<% loadChapter("D:\X\KJV.XML\01-001.xml") %>
</body>
</html>
```

The output would appear similar to the following.

```
Genesis 1:

1. In the beginning God created the heaven and the earth.
... verses 3 to 31 omitted ...
31. And God saw every thing that he had made,
and, behold, it was very good. And the evening
and the morning were the sixth day.
```

A more complete page would allow the user to select from any book and, within that book, select any chapter. The author's has written an HTML/ASP page that does this.

Generated versions of questions

Another useful application of XML that the author has used is a system to generate multiple instances of problems such that students can access and try the questions, knowing that some of them will be on the exam. During the 2004 Spring semester, the author created over 30 such problems for a course in business statistics. The details of the creation of these problems, involving object-oriented programming techniques, is beyond the scope of this paper, but the XML output can help in understanding the use of XML.

Here is the format of the index file 000.xml for one of the problems.

```
<?xml version="1.0" standalone="yes"?>
<rmsQuestions type="regress1" fd="REGRESS1" fn="000"
   topic="Slope of a line" index="0" count="24">
   <rmsRandoms count="4">
      <rmsRandom value="15"/>
      <rmsRandom value="111"/>
      <rmsRandom value="141"/>
      <rmsRandom value="201"/>
   </rmsRandoms>
</rmsQuestions>
```

In this case, the problem type involves determining the "Slope of a line". There are 24 such problems, in files 001.xml to 024.xml. In general, hundreds of instances could be created. Note that, for generating random numbers in a certain way, some random number seeds are provided in this file.

Here is the format of the first question for this question type.

```
<?xml version="1.0" standalone="yes"?>
<rmsQuestion type="regress1" fd="REGRESS1" fn="001"
   topic="Slope of a line" index="1" count="24">
   <rmsRandoms count="4">
      <rmsRandom value="7"/>
      <rmsRandom value="155"/>
      <rmsRandom value="53"/>
      <rmsRandom value="183"/>
   </rmsRandoms>
   <rmsSteps>
<rmsStep>
What is the slope of the line going through the points
(11, 26) and (49, 68)?
</rmsStep>
<rmsStep>
The slope of a line with points (x1, y1) and (x2, y2) is
m = rise/run = (x2-x1)/(y2-y1) = (49-11)/(68-26) = 42/38 = 1.11.
</rmsStep>
<rmsStep>
The answer, in the required format, is 1.11.
</rmsStep>
</rmsSteps>
<rmsChoices type="1" answer="1.11" format="x.xx"/>
</rmsQuestion>
```

Again, random number seeds are provided so that a problem formatting system can generate random numbers in the same way every time that a question is generated. Four values are provided to allow for four versions of the random formatting.

By convention (of the author's format), the first step is the problem statement. The last step is the answer. The in-between steps cover how to solve the problem. The choices could consist of multiple choice letters. But, in this case, the choice is a numeric answer with format "x.xx". The interface program would allow the user to specify all three digits in multiple choice format, but selecting digits 0 to 9, in some convenient way (depending on the interface).

The web page allows access to any of the generated questions. The interface might appear as follows.

```
Example multiple choice questions for "Slope of a line".

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

A student can select a question, enter an answer, see if it is correct, and select the "Show me how" to see how the question might be answered. Many students use the "Show me how" first, and then learn how to do the questions that way, but at least most of them learn how to answer the questions.

Web server considerations

From the web server point of view, the ASP pages need access to the XML data. Typically, the XML data is not to be directly available on the Internet. Thus, the web server needs access to a directory containing the XML, but that directory is not directly available on the Internet. The webmaster must allow this access for XML to be used in this way, which could be a problem if the webmaster will not cooperate in granting the access.

Summary

This paper has introduced some useful and practical aspects of XML technology for sharing information in a educational setting by way of some useful examples.

References

[1] Aitken, P. (2002). XML the Microsoft way. Boston, MA: Addison-Wesley.

[2] Bradley, N. (2002). The XML companion, 3rd ed. Boston, MA: Addison-Wesley.

[3] Carlson, D. (2001). Modeling XML applications with UML: Practical e-business applications. Boston, MA: Addison-Wesley.

[4] Snyder, R. (1996). The M&M problem: a tasteful example of summarizing and analyzing data for decision making. Proceedings of the 8th Annual Conference of the International Academy of Business Disciplines. Rockville, MD.

[5] Snyder, R. (1998). Using JavaScript and simulation techniques to create virtual M&M's. The Journal of Computing in Small Colleges. Vol. 13, No. 3.

[6] Snyder, R. (2002). XML for the critical path problem in project management. Proceedings of the 38th Annual Meeting of the Southeastern Chapter of the Institute for Operations Research and the Management Sciences. Myrtle Beach, SC.

[7] Snyder, R. (2003). XML for a spreadsheet modeling system. Proceedings of the 32nd Annual Meeting of the Southeastern Region of the Decision Sciences Institute. Williamsburg, VA.

[8] Snyder, R. (2003). HTML form automation and web page scraping using MSXML and ASP. Proceedings of the 31st Annual Conference of the International Business Schools Computing Association. Daytona Beach, FL.

[9] Snyder, R. (2003). Making sense of XML and background server-to-Server processing. Proceedings of the 39th Annual Meeting of the Southeastern Chapter of the Institute for Operations Research and the Management Sciences. Myrtle Beach, SC.

[10] Von See, C., & Keskar, N. (2002). XSLT developers guide. New York: McGraw-Hill/Osborne.