Creating and Maintaining Data-Driven Course Web Sites

Jesse M. Heines
Department of Computer Science
University of Massachusetts Lowell
One University Ave., Lowell, MA 01854, USA
heines@cs.uml.edu

Abstract

This paper deals with techniques for reducing the amount of work that needs to be redone each semester when one prepares an existing course Web site for a new class. The key concept is algorithmic generation of common page elements while still allowing full control over page content via WYSIWYG tools like Microsoft FrontPage and Macromedia Dreamweaver. The paper explores both client- and server-side techniques and discusses their advantages and disadvantages. The most advanced techniques are those that encode control information in XML rather than HTML or JavaScript and apply that information on the server side using XSL and JavaServer Pages. The paper also touches on course organization techniques that might involve students in the creation and maintenance of course Web sites, thereby fostering more student-centered learning.

This paper was presented on October 18, 2002, as an invited address at the E-Learn 2002 Conference in Montreal, Canada, sponsored by the Association for the Advancement of Computing in Education.

1 It'sa Lotta Work!

Numerous student and faculty surveys have shown that parties on both sides of the podium consistently laud the instructional benefits of course Web sites. For example, when 62 students in a Web site-supported course were asked to react to the statement "I wish other professors maintained course Web sites like this one," 82.3% "strongly agreed" and another 16.1% "agreed" (Heines 2000). On the faculty side, 68.0% of the 25 respondents to a survey on course Web site development who did not have a course Web site (out of 150 total responses) indicated that they would like to have one (Grankovska & Heines 2003). Gehringer (2002) found that while only 7.6% of 250 survey respondents reported that their universities have a policy on making course materials publicly available on the Web, 85.2% at least posted their syllabi, 68.8% posted their assignments, and 53.2% posted their lecture notes.

There is also ample evidence that the amount of work it takes to post material on the Web is considerable. David Jordan (1997) estimated that "creating and maintaining the site roughly doubles the work involved in teaching the course." Even with the availability of commercial tools, 81.3% of the 150 professors who responded to the Grankovska and Heines survey (2003) cited time as the most "serious obstacle to making their course Web site everything they want it to be" or the most "significant part of the reason why they did not have a course Web site."

Even with commercial tools – general-purpose as well as those designed for instructional use – course Web site development is time-consuming at best and daunting at worst. But as difficult as *creating* a rich course Web site may be, *maintaining* it – keeping its content and links current and updating the site when the course is revised – is even more arduous. Even for those of us who are highly dedicated and have the best of intentions, the amount of work that seems to need to be redone each semester is so frustrating that we often give up and leave things the way they were. The focus of this research is therefore to explore techniques for reducing the amount of work needed not only to create course Web sites, but to maintain them as well.

2 Moving to Data-Driven Course Web Sites

There are two basic ways to build most computer applications. First, you can embed data within program code, a technique referred to as "hard-coding." On a Web site, this is analogous to creating pages in HTML, either directly

using a text editor and entering HTML tags and attributes yourself, or indirectly using a WYSIWYG editor like Microsoft FrontPage or Macromedia Dreamweaver that creates those tags and attributes for you. Pages can be built from templates to achieve consistency in their look and feel, but, for the most part, once a hard-coded page is created it remains independent of all other pages on the site.

The second approach is to separate content data from code logic and thereby make the application "data driven." On a Web site, this is analogous to creating pages where what is displayed is stored in some sort of data file, while how that content is displayed is controlled by some sort of scripting technique such as JavaScript on the client side or JavaServer Pages on the server side. Overall look-and-feel or deployment of a new feature across the entire site can then be modified or added by manipulating a small set of control files.

Data-driven techniques are used by virtually all commercial course Web site "engines" such as Blackboard (www.blackboard.com), WebCT (www.webct.com), and IntraLearn (www.intralearn.com). Such tools are excellent for courses that fit their models and instructors who are satisfied with their facilities, but it is typically difficult to break out of their paradigms to add new course models or interaction capabilities. While this restriction is frustrating to experienced Web developers, the data-driven characteristics of these engines use techniques that all serious developers embrace.

Consider, for example, the elements in the typical lecture notes posting for shown in Figure 1. Focusing solely on the notes themselves inside the navigation frames, you can see that there are numerous common elements here that will be formatted identically on all lecture notes pages: the course number, class number, and date in the header, as well as the eight control buttons to the right of the "Handouts and Materials" heading. From left to right, the four arrow buttons jump to the first, previous, next, and last posted lecture notes. Likewise, the four control buttons jump to the course home page, print and reload the current frame, and remove (or restore) the navigation frames. It is probably obvious that you wouldn't want to include the code needed for each of these elements on every lecture notes page. It is more subtle, however, to appreciate that the "previous" and "next" page is different for every page and therefore requires that link to be generated algorithmically rather than hard-coded. If you have the dates of each class stored in some sort of data structure, you can not only write code to format the header information consistently across all similar pages, but you can also automatically generate the links from one page to the next.

3 Using JavaScript on the Client Side

There are two basic choices for implementing data-driven course Web sites on the client side: Java applets and Dynamic HTML. The use of Java applets and similar plug-ins (like Macromedia Flash) involves a dramatic shift away from the basic programming paradigm based on HTML that is used in most WYSIWYG editors. This is not inherently bad, and many would argue that it's high time we retired HTML anyway. However, use of Java applets obviously requires knowledge of those languages and their facilities that is considerably beyond the scope of this paper.

Dynamic HTML, on the other hand, is easily integrated into standard Web pages in the form of JavaScript or VBScript. There is no need to throw away HTML that you've already generated. Instead, you can simply augment it by inserting scripting code into the same file. VBScript is a Microsoft product and is more closely integrated with Internet Explorer than JavaScript (or even Jscript, Microsoft's version of JavaScript), but I prefer to work in JavaScript because it is more universal. There are differences in the way JavaScript works on the major browsers, but those differences tend to be small and easy to rectify.

The common elements in Figure 1 pointed out in the previous paragraph are all implemented on my Web site using JavaScript. I use a couple of simple conventions to make the job easier, which help me organize the site as well as make it data driven. First, I name all related files sequentially, like class01.htm, class 02.htm, etc. Thus the file name indicates which class it is, and I can move content between files at will without affecting the links between them. (I do this all the time, because no class ever goes exactly as planned.) This simple convention makes it possible to determine the class number dynamically using the JavaScript code shown in Listing 1. When this code is included in an HTML page – or, as I implement it, in a common file that is included via a link in almost every HTML page on my site – it sets variables that can be used to generate the page header and to control navigation between related pages "on the fly." For example, the page shown in Figure 1 has a URL of http://www.cs.uml.edu/~heines/91.353/91.353-2001-02s/353-lecs/class09.htm. Executing the routine in Listing 1 from this page sets variable `intClassNo` to 9 and `strPrefix` to "`class`."

Next, I use a simple JavaScript array data structure to hold the dates of all classes. The second convention involves the format in which those dates are stored: YYYY-MM-DD, known as "ISO 8601" format because it is an official standard of the International Organization for Standardization. The advantage of ISO 8601 over the more common M/D/YY format used in the United States is not only that it is Y2K compliant, but, more important, that it allows dates to be sorted alphanumerically. Thus my array looks like this:

```
// class days for the Spring 2002 semester
var arrClassDate = new Array(
        "2002-01-23" ,  //  1
        "2002-01-30" ,  //  2
        "2002-02-06" ,  //  3
           ... and so on ...
        null ) ;
```

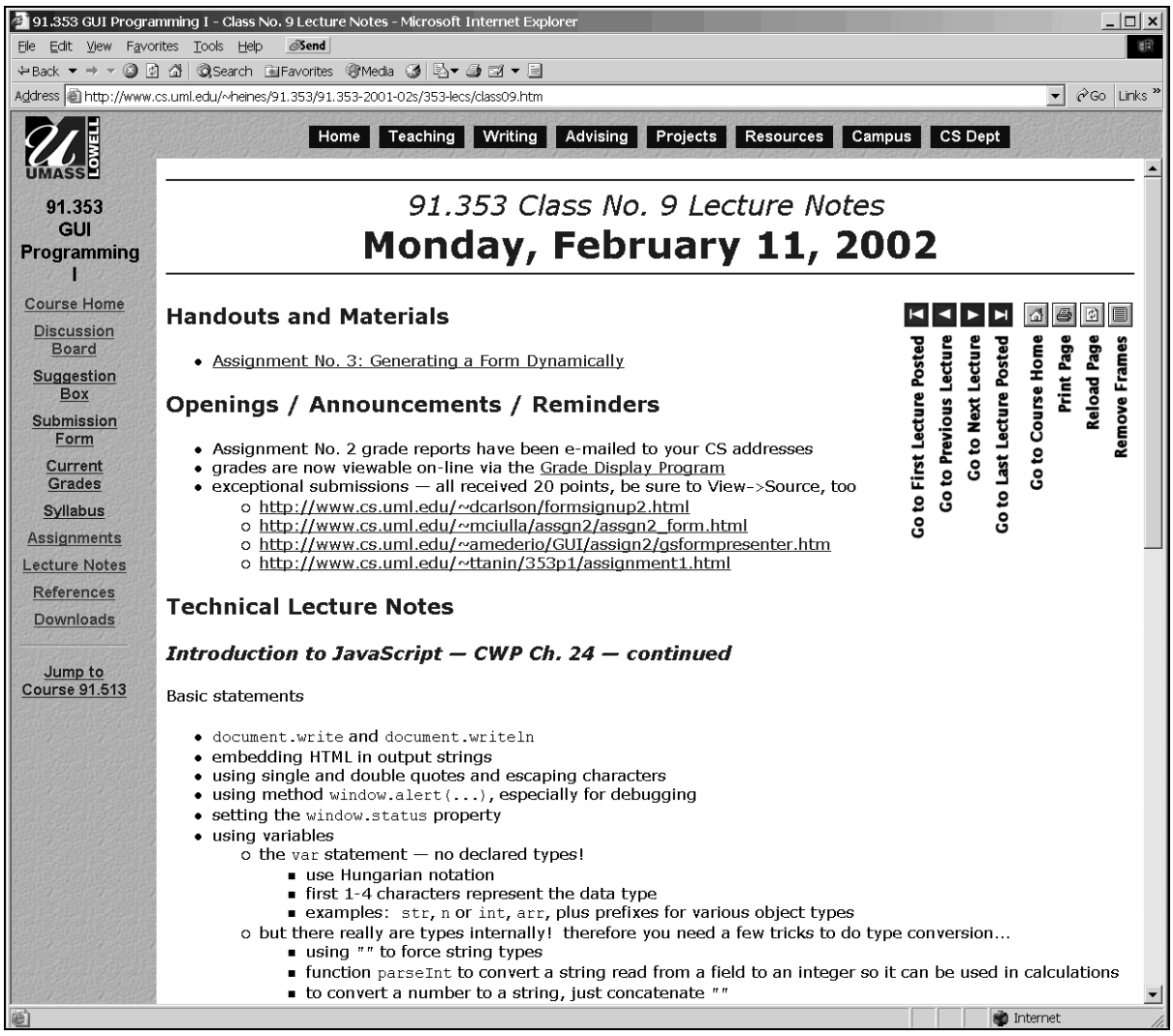

Figure 1: A typical Web posting of lecture notes with numerous common page elements.
The vertical button labels were added for clarity; they do not appear on the actual Web page.

Listing 1: JavaScript code to extract the 2-digit lecture number and its prefix from a file name.

```
// determine the file name from the current page's URL
var strURL = "" + document.location ;          // full URL of current page as a string
var intSlashPos = strURL.lastIndexOf( "/" ) ;  // position of last forward slash in URL
var strFile =                                  // name of lecture notes file
        strURL.substr( intSlashPos+1, strURL.length ) ;

// extract the class number and its string prefix
var intDotPos = strFile.lastIndexOf( "." ) ;   // position of dot in filename.extension
var strClassNo =                               // 2-digit class number as a string
        strFile.substring( intDotPos-2, intDotPos ) ;
if ( strClassNo.substring( 0, 1 ) == "0" )     // remove leading 0 if it exists
  strClassNo = strClassNo.substring( 1, 2 ) ;
var intClassNo = parseInt( strClassNo ) ;      // class number as an integer
var strPrefix =                                // characters in file name that precede
        strFile.substring( 0, intDotPos-2 ) ;  //    the class number
```

If the lecture notes for a certain date have not yet been posted, or if I simply don't want to make them available to students yet, I comment out the corresponding dates in the array as you see for May 1 and 8 above.

Given a series of lecture notes files that follow the naming convention discussed above and their corresponding dates in an array data structure, I use the built-in JavaScript Date object along with a few small routines of my own convert the ISO 8601 string to the "Weekday, Month Day, Year" format you saw in Figure 1. (All of these routines are available from the author on request.) In addition, I use the same data to dynamically create the links for the four arrow buttons. Part of the code accomplishes this is shown in Listing 2 to give you a feel for what's involved. Data-driven creation of these links insures that they will never be "broken," and that, for example, the "Next" and "Last" buttons are automatically disabled when students are viewing the last page in the sequence.

Listing 1: JavaScript code to generate the previous and next page buttons.

```
 1  /** output links to first, previous, next, and last pages
 2   *  @param intClassNo an integer specifying the current page number
 3   */
 4  function setPrevNextLinks( intClassNo )
 5  {
 6    // last and next buttons
 7    if ( arrClassDate[intClassNo] != null )
 8    {   // enabled versions
 9      document.writeln(      // strImagesLoc is the relative path to the images directory
10        '<a href="' + strPrefix +
11        ( arrClassDate.length-1 < 10 ? '0' + (arrClassDate.length-1)
12                                     : (arrClassDate.length-1) ) + '.htm">' +
13        '<img src="' + strImagesLoc + '/blue_last.gif" alt="Go to Last Lecture Posted" '+
14        'align="right" border="1" hspace="2" width="24" height="24" ' +
15        'onMouseDown="this.src=\'' + strImagesLoc + '/blue_last_down.gif\'" ' +
16        'onMouseOut="this.src=\'' + strImagesLoc + '/blue_last.gif\'"></a>' ) ;
17      document.writeln(
18        '<a href="' + strPrefix +
19        ( intClassNo+1 < 10 ? '0' + (intClassNo+1) : (intClassNo+1) ) + '.htm">' +
20        '<img src="' + strImagesLoc + '/blue_next.gif" alt="Go to Next Lecture" ' +
21        'align="right" border="1" hspace="2" width="24" height="24" ' +
22        'onMouseDown="this.src=\'' + strImagesLoc + '/blue_next_down.gif\'" ' +
23        'onMouseOut="this.src=\'' + strImagesLoc + '/blue_next.gif\'"></a>' ) ;
24    }
25    else
26    {   // disabled versions
27      document.writeln(
28        '<img src="' + strImagesLoc + '/blue_last_disabled.gif" ' +
29        'align="right" border="1" hspace="2" width="24" height="24">' ) ;
30      document.writeln(
31        '<img src="' + strImagesLoc + '/blue_next_disabled.gif" ' +
32        'align="right" border="1" hspace="2" width="24" height="24">' ) ;
33    }
      ... code for the first and previous buttons is analogous to what appears above ...
60  }
```

4 Moving to XML

The use of JavaScript and its built-in structures work very well for simple data-driven routines that use Dynamic HTML. Development is straightforward, code size is small, testing is easy, and client-side execution is fast. When the data gets more complex, however, arrays and JavaScript's simplified class structure (some would call it a "pseudo class structure") are insufficient to encapsulate the relationships between various data elements and make them available throughout the Web site. A better alternative is to encode data in XML, the eXtensible Mark-up Language, and access it using one of the various techniques specifically developed to allow XML data to be used on Web pages.

For readers unfamiliar with XML, one can describe it in a nutshell as a data-encoding scheme in which HTML-like tags are used to label individual data items. Those items can then be arranged into a hierarchy that structures the entire data set and adds further meaning (semantic content) to its components. The simple, self-explanatory example shown in Listing 3 should suffice to give a feel for how data is encoded in XML. Given such data, it is not difficult to imagine using this information in a variety of displays, such as:

Listing 3: Sample XML file to show basic syntax and structure.

```
 1  <?xml version="1.0" ?>
 2  <conferences>
 3    <conference name="E-Learn 2002" location="Montreal, Canada" startdate="2002-10-15">
 4      <session number="1234">
 5        <title>Creating and Maintaining Data-Driven Course Web Sites</title>
 6        <presenter>
 7          <name>
 8            <first>Jesse</first>
 9            <middle>M</middle>
10            <last>Heines</last>
11          </name>
12          <affiliation>University of Massachusetts Lowell</affiliation>
13          <e-mail>heines@cs.uml.edu</e-mail>
14        </presenter>
15        <categorization>
16          <application-domain>Higher Education</application-domain>
17          <technology>Web Technologies</technology>
18          <strategic-focus>Blended Learning</strategic-focus>
19        </categorization>
20      </session>
21    </conference>
22  </conferences>
```

- a listing of all sessions sorted by the primary presenter's last name
- a listing grouped by the various application domains, technologies, or strategic-focuses
- an APA-formatted bibliographic reference or one in any other standard format

A full discussion of XML is, of course, beyond the scope of this paper, but many books are available that address its syntax, capabilities, and related technologies (of which there are many). My favorite remains Martin (2000), even though it is now two years old (ancient in this field!).

There are several ways to work with XML data, the easiest of which may be via client-side extensions to HTML provided by Microsoft in Internet Explorer 6.0 and available as a plug-in to Internet Exploror 5.5. This client-side technique is unique to Internet Explorer, however, and therefore is not truly usable in course Web sites that may be accessed via other browsers or even versions of Internet Explorer older than 5.5. Thus the preferred way to use XML is on the server side.

XML parsers come in various flavors, many of which are freely available (a list of reasonable size is available at http://www.w3.org/XML/#software, W3C 2002a). Most XML parsers can be seamlessly integrated with other server-side techniques, although the effort to do so can be substantial due to inadequate and sometimes even incorrect documentation. Parsers exist that work with CGI programs written in C++ or Perl, as well as Java programs implemented as JavaServer Pages or Java Servlets. Some parsers, such as *Xerces* from the Apache Software Foundation (2002), load entire XML documents into memory and allow them to be manipulated using a Document Object Model (DOM). Others, such as the *Simple Application Programmer Interface (API) for XML* (SAX), read XML data as a stream and process each data node in turn (Meggison 2002). I prefer to process XML data using XSL – the *XML Stylesheet Language* – which has a built-in parser and capabilities specifically designed to translate XML data into a variety of formats, including HTML (W3C 2002b, Heines 2003). Listing 4 shows an XSL file that generates the output shown in Figure 2 from the XML data in Listing 3.

The advantage of using these techniques on the server side is that it is possible to know exactly what software is installed on the server and thus not have to deal with browser differences on the client side. XSL stylesheets (as well as programs that process XML using the DOM or SAX) use the XML data to generate straight HTML with is then "sent down the pipe" to the client. That HTML can be as simple or as sophisticated as you like, including embedded code that detects and compensates for the minor differences between browsers at the HTML level. Luckily, as of this writing those differences are small.

Note that all of these techniques require some degree of server-side programming support, something that is not always enabled on university servers. In the UMass Lowell Computer Science Department, for example, we do not allow server-side scripts of any sort on our main server that hosts student and faculty Web pages due to the high probability that such scripts will crash the server. As an alternative, we make space available to all students and faculty who request it on lower-capacity, non-production, CGI- and Java-enabled servers.

Listing 4: XSL file to generate the output in Figure 2 from the XML data in Listing 3.

```
 1  <?xml version="1.0" ?>
 2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 3    <!-- processing begins here -->
 4    <xsl:template match="/">
 5      <html>
 6      <body>
 7        <xsl:apply-templates select="conferences/conference" />
 8      </body>
 9      </html>
10    </xsl:template>
11
12    <!-- this template is executed for each conference -->
13    <xsl:template match="conference">
14      <xsl:apply-templates select="session" />
15    </xsl:template>
16
17    <!-- this template is executed for each session -->
18    <xsl:template match="session">
19      <xsl:apply-templates select="presenter/name" mode="last-name-first"/>. 
20      <i><xsl:value-of select="title" /></i>.
21    </xsl:template>
22
23    <!-- this template generates a presenter name in last-name-first format -->
24    <xsl:template match="name" mode="last-name-first">
25      <xsl:value-of select="last" />, <null />
26      <xsl:value-of select="first" /> <null />
27      <xsl:value-of select="middle" />
28    </xsl:template>
29  </xsl:stylesheet>
```

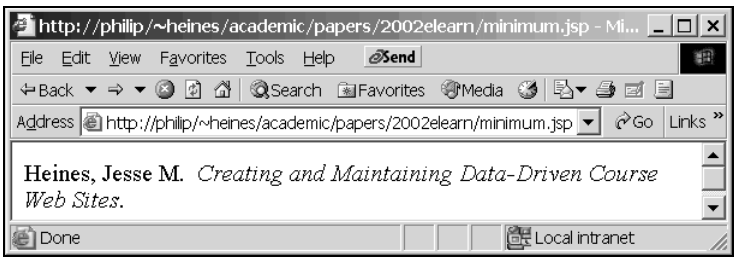

Figure 2: Output generated by applying the XSL file in Listing 4 to the XML data in Listing 3.
minimum.jsp is a JavaServer Page that applies XSL to XML and is available from the author on request.

5 Using XML on the Server Side

To understand the contribution XML can make to data-driven course Web sites, consider the list of assignments and lecture notes on the course home page in Figure 3. Setting up and maintaining the assignments table with a standard WYSIWYG editor is not too bad, but doing likewise for the lecture notes table with the number and date of each class, the related reading, the lecture topics to be covered, complementary lab activity, and correct URL for each linked element is error-prone even with FrontPage or Dreamweaver. In addition, after you've gone to the trouble to encode the information in `<table>`, `<tr>`, and `<td>` tags, there is nothing else you can do with it. That is, you can't include it on the lecture notes pages themselves and you can't use it to control navigation between related pages as discussed at the beginning of this paper.

Storing the data in XML as shown in Listing 5, however, yields not only the ability to use the data elsewhere, but also the ability to manipulate it easily and use it in a variety of other ways. For example, instead of cutting and pasting table rows as each class moves from "upcoming" to current or passed – a tricky maneuver even in WYSIWYG editors – you simply delete the `status="upcoming"` attribute of the XML file's corresponding `class` element or changes its value to an empty string (`""`). Once the lecture notes and activity for a class are finalized, all you have to do is delete the `suppresslinks="true"` attribute or change its value to `"false"` and the appropriate links will be generated automatically. As before, automatic link generation assures that those links will never be "broken" as long as you follow the appropriate file naming conventions.

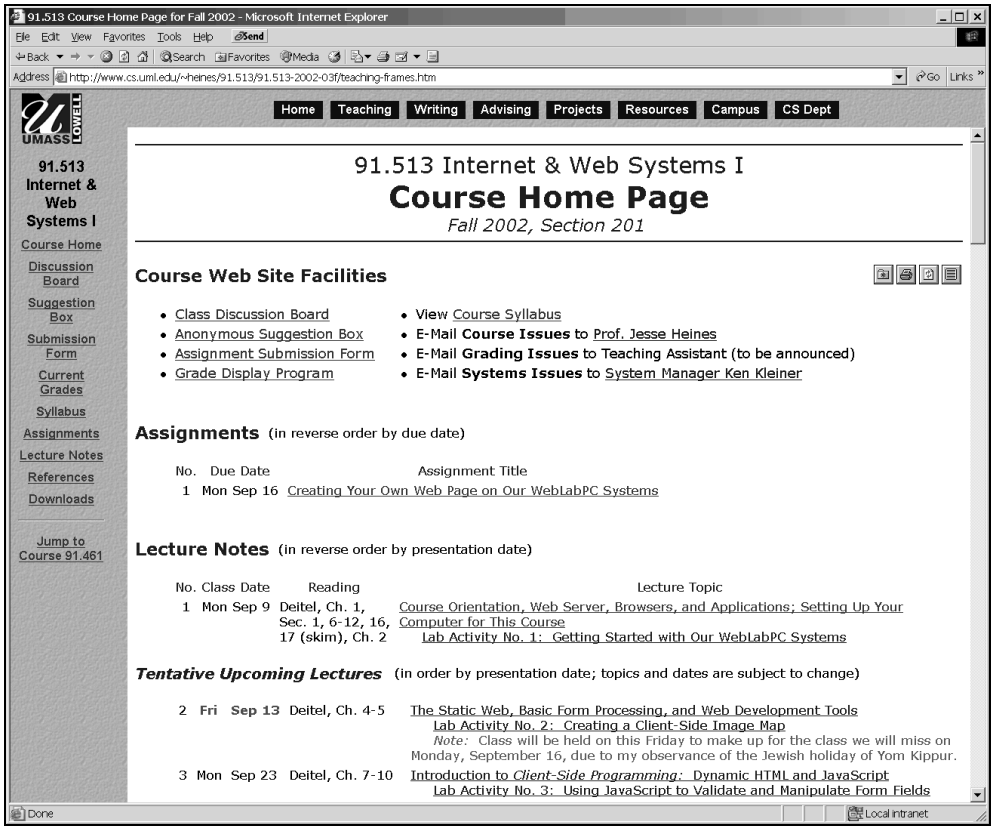

Figure 3. A typical course home page.

Listing 5: XML data used to generate the listing of lecture notes in Figure 3.

```
 1  <?xml version="1.0" ?>
 2  <!--
 3    513-lecs.xml, Jesse M. Heines, UMass Lowell Computer Science, heines@cs.uml.edu
 4  -->
 5  <classes>
 6    <class number="1">
 7      <date>2002-09-09</date>
 8      <reading><![CDATA[Deitel, Ch. 1,<br />Sec. 1, 6-12, 16,<br />17 (skim), Ch. 2]]></reading>
 9      <lecture>
10        Course Orientation, Web Servers, Browsers, and Applications; Setting Up Your
11        Computer for This Course
12      </lecture>
13      <activity>Getting Started with Our WebLabPC Systems</activity>
14    </class>
15    <class number="2" status="upcoming" suppresslinks="false">
16      <date format="bold" color="red">2002-09-13</date>
17      <note>
18        Class will be held on this Friday to make up for the class we will miss on
19        Monday, September 16, due to my observance of the Jewish holiday of Yom Kippur.
20      </note>
21      <reading>Deitel, Ch. 4-5</reading>
22      <lecture>
23        <![CDATA[The Static Web, Basic Form Processing, and Web Development Tools]]>
24      </lecture>
25      <activity>Creating a Client-Side Image Map</activity>
26    </class>
      ... code for the other classes is analogous to what appears above ...
99  </classes>
```

You can also extract data from the XML file to add to the lecture notes shown in Figure 1. For example, you might want to indicate the topics covered in the lecture and the list of related readings, information which currently only exists in the index on my course Web site. Pulling this data from the XML file would once again guarantee that the two places in which this information appears always agree with each other, eliminating inconsistencies and "out-of-synch" problems. As you might suspect, you can also use the data in this file to address the page navigation issues discussed above.

Thus, the encoding data in XML files and using that to drive the Web site rather than straight HTML provides a wealth of functionality and levels of consistency and integrity much more difficult to achieve in non-data-driven course Web sites. In all fairness, however, I must recognize that some will object to the code-centric view one must adopt to work with XML and will see that as a step backward from today's sophisticated WYSIWYG HTML editors. I concede that this is a legitimate concern today, but some XML editors with well-engineered graphical user interfaces to exist today, and we at UMass Lowell are exploring techniques that will allow the required XML data to be specified via Web forms.

6 Student-Centered Course Web Sites

When course Web sites are data-driven and their formatting code is independent of their content, one can begin to explore alternative ways to generate their content, knowing that the display format will remain consistent regardless of who serves as the content author. For example, one might envision a model of course Web site development and maintenance that puts students in the center of the process as *course scribes*. In addition to providing a way to get timely information on-line without burdening professors with yet another responsibility, this approach might get students more actively involved with the subject matter and its treatment in the course. Such involvement would undoubtedly enrich the individual scribes' learning as well as that of their fellow students following the classic adage: "If you really want to learn something, teach [or explain] it to someone else."

We also expect that when students rather than professors put material on-line, there will be more discussion among peers about the accuracy of the Web site content. It would be ideal if this activity fostered an environment in which students learn from and critique each other's work in the same way that journalism students do when putting together a campus newspaper or literature students do when editing submissions to a literary magazine. Students readily accept that they and their peers make mistakes, while they implicitly – although, of course, erroneously – assume that whatever their professors post is correct. Such discussions are one of the fundamental tenets of cooperative learning, which has been shown to produce significant positive effects on student achievement (Slavin, 1996).

7 References

Apache Software Foundation (2002). Apache XML Project. xml.apache.org

Gehringer, E. F. (2002). To see or not to see: access restriction on course Web sites. *Proceedings of the 2002 American Society for Engineering Education Annual Conference and Exposition*, Montreal, Quebec, Session 1520, June 16-19, 2002.

Grankovska, S. & Heines, J. M. (2003). Course Web sites: state-of-the-art. *U.S.D.L.A. Journal* 16(12). teaching.cs.uml.edu/~heines/techrpts/papers/GrankovskaHeines_USDLAJournal_online.pdf

Heines, J. M. (2000). Evaluating the effect of a course Web site on student performance. *Journal of Computing in Higher Education* (Fall 2000) 12(1):57-83. www.cs.uml.edu/~heines/academic/papers/2000jche

Heines, J. M. (2003). XSL. In H. Bidgoli (Ed.), *The Internet Encyclopedia*. To be published by John Wiley & Sons.

Jordan, D. K. (1997). *Evaluation of a Web site as a teaching tool in MMW-1 (track C)*. Unpublished manuscript. University of California, San Diego.

Martin, Didier, and twelve other authors (2000). *Professional XML*. Birmingham, UK: Wrox Press, Ltd.

Meggison, David (2002). SAX. www.saxproject.org

Slavin, R. E. (1996). Research for the future: research on cooperative learning and achievement: what we know, what we need to know. *Contemporary Educational Psychology* 21(1):43-69. www.successforall.net/resource/research/cooplearn.htm

World Wide Web Consortium (2002a). XML Software. www.w3.org/XML/#software

World Wide Web Consortium (2002b). What is XSL? www.w3.org/Style/XSL/WhatIsXSL.html