MyXML: An XML based template engine for the generation of flexible web content

Engin Kirda and Clemens Kerer
Distributed Systems Group
Technical University of Vienna
Argentinierstrasse 8/184-1
A-1040 Wien, Austria
{E.Kirda,C.Kerer}@infosys.tuwien.ac.at
http://www.infosys.tuwien.ac.at/

Abstract: MyXML is an XML/XSL based template engine that facilitates the rapid development of flexible, database-backed web sites. A strict separation between content, layout and business logic of a web site is enforced. As a consequence, changes in the layout no longer affect the content or the logic of a web site and vice verse. On the basis of our web service engineering experiences with the Vienna International Festival Web Service we designed and implemented MyXML to provide a language independent abstraction of frequently needed web functionality. Also the creation of dynamic web content based on CGI and database access as found in many of today's web sites is supported. MyXML can be used with arbitrary programming languages. Additionally, MyXML can be used to directly generate XML content for further processing from relational databases.

Introduction

World Wide Web services and hypermedia applications are information systems that are based on a highly dynamic and flexible medium. They often have to provide interactive and up-to-date content. Managing and maintaining a web based service can be a difficult task once the size of the presented information exceeds a certain limit. Flexibility in terms of layout and functionality is a basic requirement for web applications. Applying changes to both static and dynamic components of a web service while keeping a common look (corporate design) and functionality can be a tedious and difficult task.

The last couple of years have shown the emergence of new technologies and tools for the creation of static and dynamic web content. A large variety of WWW services, applications and tools have been flooding the Internet for the last few years. The rapid development of flexible and scalable web sites is becoming a key success factor in the online commerce business.

Our group has been managing, hosting and building the web presence of the Vienna International Festival (VIF) since 1995 (Schranz 1997, Schranz 1997a, Schranz 1998). The site of this international festival consists of many dynamically updated information pages, bilingual content, some multimedia data and several interactive services. One requirement for such a service is the high level of necessary flexibility both in respect to layout and functionality changes. This requirement comes from the fact that the VIF look and feel has to change *every year*. We found that this task cannot be satisfactorily achieved by the deployment of popular web engineering tools. In the past, we designed and developed tools (HTML++ (Barta 1995), JESSICA (Barta et al. 1998)) to provide web engineering and flexibility support for a complete web service. One drawback of our tools, like many other web engineering tools, has been the unsatisfactory support for a complete layout, content and logic separation.

The wide acceptance of the XML (W3C 2000) standard by developers and the large growth in its use recently has brought with it proposals and tools to enable a strict layout, content and business logic separation in web applications (Mazzocchi 2000, Mazzocchi 2000a, Mazzocchi 2000b). Nevertheless, most of the current popular technologies for building web applications like PHP, ASP, JSP, JavaScript, Perl, Python and Java servlets still do not have support for this separation. The layout, content and the logic are mixed together in one or more files which complicates the task of applying changes. If you have to do layout modifications, for example, you have to analyze the source code and to try to locate the encoded layout information. Then, you have to modify your layout and this often forces you to alter your functionality. To cut down the development time when migrating the VIF web site to a completely different look and feel and to be able to adapt rapidly to changing layout requirements, we developed an

XML/XSL-based template engine. The MyXML template engine attempts to solve this flexibility problem by using a combination of XML, XSL (W3C 2000a) and the MyXML language for developing web services. The system architecture is programming language and database independent. We have implemented a prototype that has support for Java servlets and that has an interface to the MySQL freeware DBMS (tcx 2000). The latest version of MyXML is downloadable at http://www.infosys.tuwien.ac.at/myxml/. Our system aims at automating processes that are frequently needed in building static as well as dynamic web sites.

The MyXML Template Engine

MyXML uses a layered approach to achieve a separation between the static and dynamic information offered in a web site and the layout it is presented in.

The content is represented in special MyXML documents. These documents are well-formed XML documents that contain the structured content. MyXML documents can also be based on a document type definition (DTD) (W3C 2000b) that defines the content's overall structure. The content itself can be enriched with special MyXML elements which represent template functions like variables, loops, and database queries and enable the user to add dynamic content to a web site. These elements are defined in the MyXML namespace and are described in the next section. All the necessary layout information is added to the content defined in MyXML files as separate XSL documents. XSL transformations are much more powerful than cascading style sheets (CSS) (W3C 2000c) and provide a higher level of layout flexibility. Context information can be used in the layout definition rules and it enables the processing of elements only if they appear in a predefined context (e.g., if they have a certain parent element, if they have an attribute with a given value etc.). XSL style sheets can also be used to add static content, like headers and footers, to the documents. Arbitrary XSL style sheets can be used by MyXML as long as they import the MyXML style sheet. This style sheet defines the processing of the elements of the MyXML namespace.

The MyXML engine is responsible for the dynamic generation of content as well as the automatic creation of *hooks* for the business logic. The engine distinguishes between static and dynamic MyXML documents. A document is considered as being static if all tags from the MyXML namespace can be resolved at compile time. A dynamic document, on the other hand, contains tags which cannot be resolved at compile time (e.g., user defined variables or CGI parameters which have to be provided at runtime). If a tag cannot be resolved during the compilation, the MyXML engine generates source code encapsulating the layout information and the functionality of the dynamic MyXML document. We generate Java classes from dynamic MyXML documents in our reference implementation, but any other language can be supported as well.

The MyXML process starts with a MyXML document as input. Having checked the well-formedness of this document, a pre-processing style sheet is applied to add layout information and/or static content. In the next step, the generated intermediary file is parsed by the MyXML engine and the tags from the MyXML namespace (see Section 'The MyXML Namespace') are resolved. Another XSL transformation can then be applied to the output of the MyXML engine to generate the final (usually HTML) document. This post-processing style sheet can only be applied to static MyXML documents as the MyXML engine generates Java source code for dynamic ones.

We demonstrate the use of the MyXML system with the example of a search form that lets you search for musicals in the Vienna International Festival web site. All musicals containing a certain keyword are retrieved from the database and are displayed in a dynamic web page in a given layout. The keyword is transmitted as a CGI parameter and is processed by the servlet which is generated by the MyXML template engine. (Fig. 1) shows the MyXML code that implements this functionality.

```
<?xml version="1.0"?>
<!DOCTYPE VIF>
<VIF xmlns:myxml=".../ns/myxml">
  <query>
    <myxml:sql>
      <myxml:dbcommand>SELECT * FROM VIF_EVENTS WHERE title LIKE
        <myxml:cgi>musical_title</myxml:cgi>
      </myxml:dbcommand>
      <db_titel><myxml:dbitem>title</myxml:dbitem></db_titel>
    </myxml:sql>
  </query>
</VIF>
```

Figure 1: Example MyXML file to search in a database

Note that there is a strict separation of data and layout as only the content and its structure are defined in the MyXML document. This example demonstrates the use of CGI parameters and the handling of SQL queries. The CGI parameter is used to construct the query string and after the query is executed, the title field is extracted from the result set. In the next step, an XSL style sheet is used for adding a simple layout to the data. The search result is displayed in a table. (Fig. 2) shows the XSL style sheet used for formatting the output.

```
<?xml version="1.0"?>
<xsl:style sheet version="1.0"
  xmlns:xsl=".../Transform"
  xmlns:myxml=".../ns/myxml">
  <xsl:import href="myxml.xsl"/>
  <xsl:output method="html" indent="yes"/>

  <xsl:template match="query">
  <html><h2>The result of your search is:</h2>
    <table><xsl:apply-templates/></table>
  </html>
  </xsl:template>

  <xsl:template match="db_titel">
    <tr><td><xsl:apply-templates/></td></tr>
  </xsl:template>
</xsl:style sheet>
```

Figure 2: XSL style sheet for formatting the output

The style sheet generates HTML output and adds a heading to the document. For every record in the query's result set, a new row is added to the table. In real-world style sheets, more complex rules would be applied to create a more sophisticated layout and element matching. The MyXML engine parses the resulting file in the last step, classifies the document as being dynamic since the CGI parameter cannot be resolved at compile time and, thus, generates Java source code from it. The code generated for the presented example is shown in (Fig. 3).

```
public class VIF {
  protected HttpServletRequest request = null;
  protected ResultSet SQL0 = null;

  public VIF(HttpServletRequest request) {
    this.request = request; }

  protected String getCGIParameter(String paramName) {
    return request.getParameter(paramName); }

  protected ResultSet processSQLStatement(String select, String user,
      String pwd, String url, String driver) {
      // do sql query using JDBC here! }

  public void printHTML(PrintWriter pw) {
    pw.println("<html> <h2> The result of your search is: </h2> <table>");
    printHTMLSQL0(pw); pw.println("  </table> </html>");
  }

  public void printHTMLSQL0(PrintWriter pw) {
    try { SQL0 = processSQLStatement("SELECT title, isbn_nr FROM
            VIF_EVENTS WHERE title LIKE "+getCGIParameter("musical_title")+
            ";", "user", "pwd", "connect", "dbdriver");
      while (SQL0.next()) {
        pw.println("  <tr> <td>"+SQL0.getString("title")+"</td> </tr>");
      }
    } catch (SQLException se) {// ignore exception and continue operation }
  }
}
```

Figure 3: Generated Java Source Code

All that the servlet responsible for the business logic of the website has to do now, is to create a new instance of this class and to call its *printHTML()* method. The advantage of this solution is that a modification of the layout, the data structure or the business logic does not force a modification of the other components. Once the layout has been changed, the VIF Java class only has to be regenerated again using the MyXML engine. We presented a simple example using several elements defined in the MyXML namespace. The next section describes the complete set of elements in the MyXML namespace and discusses their usage.

The MyXML Namespace

The MyXML namespace defines all elements which provide template functionality for MyXML documents. A complete discussion of each element is beyond the scope of this paper. We give a general overview of the elements in the MyXML namespace. A detailed description of the MyXML namespace and example code can be found in (Kerer & Kirda 2000).

The **<myxml:single>** element describes a single variable which can be used arbitrary times in a MyXML document. The value of a *single* element is determined at runtime and the same value is used whenever the element appears. A possible use of the *single* element would be to print a customized welcome text depending on who (e.g., members vs. guests) is currently logged in.

The **<myxml:cgi>** element supports direct access to CGI parameters within a MyXML document. The name of the *cgi* element has to correspond to the name of the CGI parameter it refers to (e.g., the name of the input field in an HTML form). In the above example, we used the *cgi* element to define a database query based on the user input.

The **<myxml:loop>** and **<myxml:multiple>** elements provide iteration and array variable functionality. For all values provided as input for the *multiple* element, the part of the document enclosed in the loop element is processed. The representation of a user's shopping cart is a perfect example for the use of the *loop* and *multiple* elements. Loops can be cascaded. Thus, loops within other loops can be used, for example, to print a table containing all the books in a bookstore along with a list of authors for each book. *single* and *cgi* elements can be used withing the *loop* element for defining functionality and templates.

The **<myxml:attribute>** element is used for dynamically defining an attribute of the parent element which is not in the MyXML namespace. For instance, the *href* atribute of an HTML *<IMG>* tag could be dynamically defined using an element of this type.

The **<myxml:sql>** element represents a database query. It is very similar to the *loop* element. The document fragment enclosed by the *sql* element is processed for every record in the query's result set. Access to database fields is provided by the **<myxml:dbitem>** element. The query to be executed is defined by the **<myxml:dbcommand>** element and can contain other MyXML elements from the MyXML namespace like variables or CGI parameters. Additional elements include the **<myxml:dbconnect>** and **<myxml:dbdriver>** elements which can be used for defining database specific information such as the login and driver information for an ODBC DBMS. The *sql* element can be used for generating XML from a relational DBMS.

Related Work

We have identified several similar approaches and concepts. Template based web engineering tools are not new and there are several popular tools in the market. Most of these technologies are not XML aware but HTML-oriented. Their architectures and generation capabilities have been designed for HTML and do not take advantage of XML.

ColdFusion (Allair 2000) and Webmacro (WebMacro 2000), for example, provide custom scripting languages to embed dynamically generated content into web pages and allow the user to define page templates. The templates, however, are based on HTML and suffer from the flexibility problems we have mentioned. There is no logic and content separation.

HTML++ (Barta 1995) and Jessica (Schranz 1998, Schranz 1998a) are tools which use an object-oriented approach for web site development. They define *classes* and *objects* which provide the content for the template. Source code generation in Perl is provided for CGI scripts but the layout and content information is coded together in an HTML-like language.

The Cocoon project (Mazzocchi 2000) tackles the content and layout separation problem in web sites by using a servlet engine for the real-time application of XSL style sheets to XML files. The Cocoon project proposes two technologies for providing flexible and layout independent dynamic content in web pages; XSP (eXtensible Server Pages) and DCP (Dynamic Content Processor). XSP is completely based on XML/XSL technology and uses XSL tag libraries and associated code generation style sheets (logic sheets) to generate compileable source code. We, in comparison, provide hooks for the business logic so the dynamic content generation does not have be defined in a logic sheet but can be developed independently of the XML/XSL technology. DCP uses a simpler approach than XSP but is an interpreted language and thus, has a performance drawback. DCP is intended to support only dynamic content whereas our system supports the generation of both dynamic and static documents. While DCP is intended to be an easy to use dynamic content generator, XSP has a better performance than DCP since no interpretation is required and compiled pages are cached by the system.

MyXML in comparison to XSP and DCP aims not only at separating logic from content, but also at supporting web sites backed by DBMSs and automating the implementation of frequently needed functionality. MyXML is intended to be simple enough to use and learn but to be powerful enough to provide support for CGI and SQL.

Summary and Future Work

The MyXML system provides a flexible and easily maintainable framework for web site development based on the XML and XSL technologies. The integration of these technologies enables a strict separation between content, layout and business logic. The MyXML namespace provides template elements which support the dynamic generation of content at runtime. CGI parameters and database access support is provided by the MyXML engine. No source code is included in MyXML documents as opposed to many popular web tools. The MyXML engine processes MyXML documents and can either generate a static result document or a piece of source code in an arbitrary programming language.

A common layout can easily be applied to both static and dynamic web pages. We currently use Java in our reference implementation as the output language. The integration of other languages such as Perl or PHP are possible.

Although many web template based tools exist, to our knowledge, only XSP/DCP currently support this clear separation. Layout and content are often tightly coupled together in most popular web tools. Our contribution is a tool that facilitates web site development and maintenance. It provides support for the rapid development of flexible and layout independent web sites by clearly separating the content from the logic.

One of our next goals is to support another programming language besides Java. We are currently evaluating the MyXML template engine in two web engineering projects involving large sets of static and dynamic information. We plan to integrate XML schemas (W3C 2000d) instead of (or in addition to) DTDs in the future. Besides support for XSL transformations, we are also planning to integrate XSL formatting objects into the MyXML template engine.

References

Schranz, M. (1997). *Vienna International Festival 1996 – managing culture on the WWW*, Museums and the Web:An International Conference, Los Angeles, California.

Schranz, M. (1997a). *Management process of WWW services: an experience report*. 9th International Conference on Software Engineering and Knowledge Engineering, Madrid, Spain, 16-23.

Schranz, M. (1998). *Engineering Flexible World Wide Web Services*. Symposium on Applied Computing, Atlanta, Georgia, 80.

Barta, R. (1995). *What the heck is HTML++: Salvation for the Souls of Webmasters*. Technical report. Technical University of Vienna.

Barta, R., & Schranz, M. (1998). *JESSICA – An Object-Oriented Hypermedia Publishing Processor*. Computer Networks and ISDN Systems, 30(1-7):281.

W3C (2000). *Extensible Markup Language (XML) 1.0 Recommendation*. http://www.w3c.org/TR/1998/REC-xml-19980210. TEchnical report. The World Wide Web Consortium.

Mazzocchi, S. (2000). The Cocoon Project Home Page, http://xml.apache.org/cocoon/.

Mazzocchi, S. (2000a). eXtensible Server Pages (XSP) Processor, http://xml.apache.org/cocoon/xsp.html.

Mazzocchi, S. (2000b). Dynamic Content Processor (DCP), http://xml.apache.org/cocoon/dcp.html.

W3C (2000a). eXtensible Stylesheet Language 1.0 Recommendation, http://www.w3.org/TR/xsl/, Technical report, The World Wide Web Consortium.

tcx (2000), The MySQL DBMS, http://www.mysql.com, t.c.x. datakonsult AB.

W3C (2000b). XML Specification DTD, http://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm, Technical report, The World Wide Web Consortium.

W3C (2000c). Cascading Style Sheets, http://www.w3.org/Style/CSS/, Technical report, The World Wide Web Consortium.

Kerer, C., & Kirda, E. (2000). MyXML Home Page, http://www.infosys.tuwien.ac.at/myxml/.

Allair (2000). Coldfusion Home Page, http://www.coldfusion.com.

Webmacro (2000). Webmacro Home Page, http://www.webmacro.org.

Schranz, M. (1998a). *World Wide Web Service Engineering - Object Oriented Hypermedia Publishing*, PhD thesis, Technical University of Vienna.

W3C (2000d). *XML Schema Part 1 and II working groups*, http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2/, Technical report, The World Wide Web Consortium.