Tradeoffs and Guidelines for Selecting Technologies to Generate Web Content from Relational Data Stores

Frank Sigvald Haug, Graduate Student
Graduate Programs in Software
University of St. Thomas
Fshaug@stthomas.edu

Saeed K. Rahimi, Ph.D.
Graduate Programs in Software
University of St. Thomas
Skrahimi@stthomas.edu

Abstract

In order to evaluate approaches for offering relational database classes on the web, we recognized the need for extracting web page content from relational database management systems. "Content" in this context refers to dynamic XML data that is reformatted into HTML pages.

This paper describes the issues, advantages, and limitations of two approaches for extracting web content dynamically. It is based on the findings of a research project that focused on implementing a small, representative set of transformations from a common relational database.

The study focused on deployment of information (not creation or modification). The first approach relied on the vendor-supplied support for SQL/XML in Microsoft SQL Server 2000. The second used a custom servlet written in Java. Emphasis was placed on identifying guidelines and tradeoffs between vendor-supplied, minimal scripting solutions and custom developed programmatic solutions.

The Research Project Description

The Web introduced the end-user to a whole new world of possibilities with respect to how information is accessed and stored. Increases in communication bandwidth, and computer processing power, memory and storage have made remote access to information practical and reliable. Acceptance of standards such as HTML [1] encouraged the separation of information formatting and content. Other standards, such as XML [2], have further formalized this separation, by allowing authors to separate semantic information from data and also provide a uniform mechanism for validating both semantic and syntactic content.

While these standards are useful for documents created by hand, they are much more powerful when used to create content dynamically from a live system. Here, additional standards such as XSL [3], XPath [4], and XDR [5] provide a framework for transforming and selecting content based on semantic information.

There is a large selection of facilities implementing these standards to choose from when designing a dynamic content web system. These facilities make it possible, and practical to dynamically generate different up-to-date views of information interactively or by batch processes at predefined intervals after the content has changed. Choosing the right alternatives from the available facilities requires careful analysis and planning.

Limited Project Scope

The project focused on dynamic retrieval and encoding of information. This avoided standard database issues such as concurrency, locking, etc. and allowed the project to focus on the issues related to the specific retrieval and encoding implementations rather than the issues involved in traditional database applications.

The project focused on two approaches: a vendor-supplied approach using Microsoft's MS SQL XML facility in SQL Server 2000, and a custom-Servlet approach written in Java using the current JDK (1.3), JDBC (2.0), JSDK (1.1) and deployed using the Apache Tomcat servlet container from the Jakarta project.

Common Components (Used by Both Approaches)

While the project evaluated two different approaches, the facilities implementing standard and platform services were shared between the two approaches as much as possible. In particular, the same operating system, web server, and relational database services were used.

Both approaches ran on the same machines at the same time, using the same tables, database, and Microsoft SQL Server 2000 RDBMS. Both used Microsoft's Internet

Information Server (IIS) version 4.0, running on Windows NT Server 4.0 Service Pack 6a for the web and platform services, and both used Microsoft's Internet Explorer 5.5 as the client. The latest version of Microsoft's XML/XSL parsing service was used for the client (however, due to vendor requirements, an older version of this was also used in parallel with the latest version on the web server for the vendor supplied approach).

Common Functionality

Because the MS SQL XML facility is a commercial quality application, it supports more features and facilities than could be implemented by the second approach within the limits of the research project. However, a core set of functionality was selected to be implemented to enable comparisons and to act as a baseline for any future facility implementations or feature enhancements.

Both facilities respond to requests for content that is dynamically generated from database queries. These queries can optionally include parameters that are specified per execution. The queries and parameters can be specified in files stored on the server (template and properties files) or passed as part of the http URL request. (For example selecting values based on an HTML form input field, such as customer id, PIN, etc.). Both approaches have essentially the same limits with respect to query/result size and complexity.

Encoding results for SQL queries can be done in many ways, and the MS SQL XML facility has several different options available. These options are actually implemented in the SQL Server database, and therefore are also available for the custom servlet approach. This would not be useful for comparison purposes however, since the MS SQL XML support is included as part of SQL Server 2000. In other words, the Java approach was implemented using SQL Server 2000 to simplify comparisons, but in practice was intended to demonstrate an alternative that was not dependent on using MS SQL Server 2000. (Why re-implement a facility if it is already there?)

Results for the custom servlet were encoded by creating one XML element per row, with column names stored as attribute names and column values stored as attribute values within that element (equivalent to one of the supported formats from MS SQL XML).

For example:

```
<row EMP_NAME="Frank" EMP_ID="123" />
<row EMP_NAME="Saeed" EMP_ID="456" />
```

Architectural Overview

Both approaches share a common architectural structure (See Figure 1). The components can be grouped into one of three areas: Client, Server, and Web Site (within Server). The Client Area refers to the components that exist or are delivered to the computer running the client (i.e. the web browser). The Server Area refers to the components that exist on the server (as opposed to the client). Lastly, the Web Site Area refers to the components that exist on the Server side but are part of a normal web site (as opposed to the components that are specific to one approach).

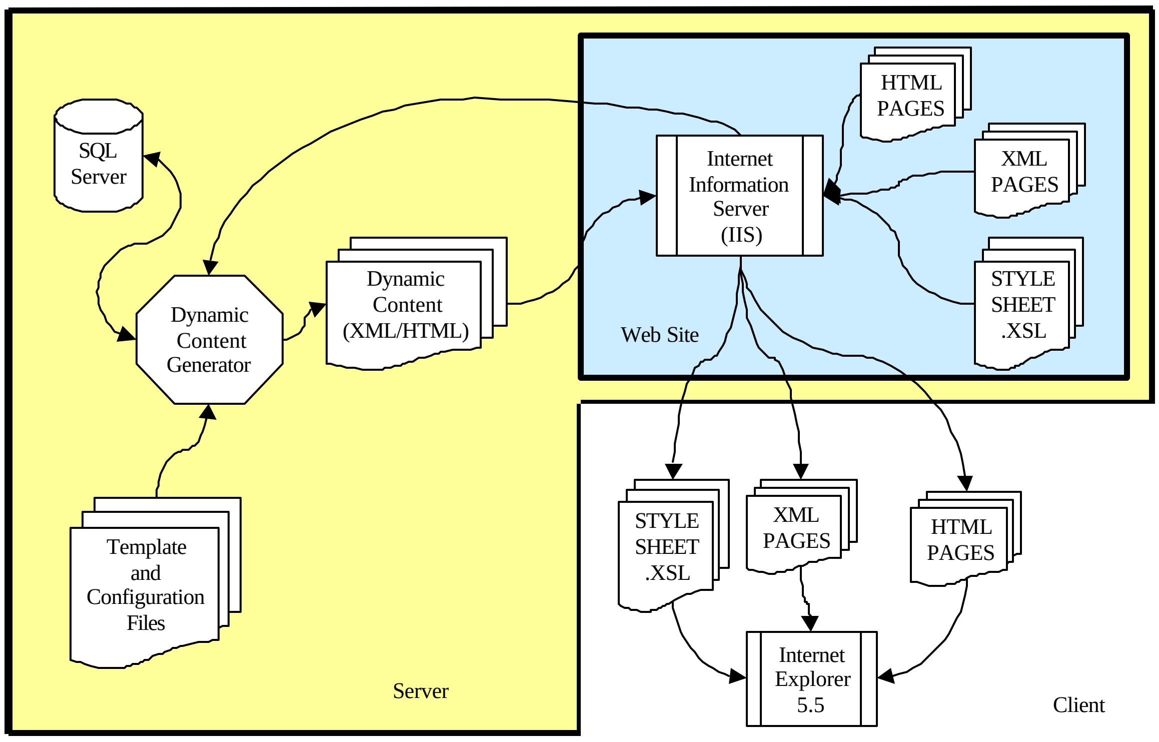

Figure 1 - Common Architecture overview

Two of the areas are essentially unchanged for both approaches, namely the Client Area and Web Site Area (see Figure 1). The Client Area simply contains Internet Explorer, and the content that is ultimately delivered to it. The Web Site Area contains the web server software itself (IIS) and the normal files that make up a web site (static HTML, XML, XSL, CSS, etc.). While the configuration and content of the files in the web site will be different for different approaches, the arrangement and types of components is common to both.

The Server Area includes the database, the implementation of each specific approach, any files necessary to configure or direct the given approach, and the actual content generated by the approach (see Figure 1). Generically speaking, the client requests content from the web site. The web site determines if the request refers to its "normal" files and if so, it delivers the appropriate content. If the request is for dynamic content, the web site sends the appropriate request to the appropriate components in the Server Area. The Dynamic Content Generator in Figure 1 is a generic placeholder for the specific components that are responsible for processing this request for a specific approach. These components use various files and database requests to generate and format the dynamic content, which is then returned to the web site and ultimately to the client.

MS SQL XML Architecture

Figure 2 shows the architecture specific to the vendor-supplied approach. Here the MS SQL XML component (a set of DLLs supplied by Microsoft) takes the place of the Dynamic Content Generator from Figure 1. It is implemented as an ISAPI redirector DLL, which means that it can be plugged directly into IIS and IIS can then forward requests to it and receive responses from it.

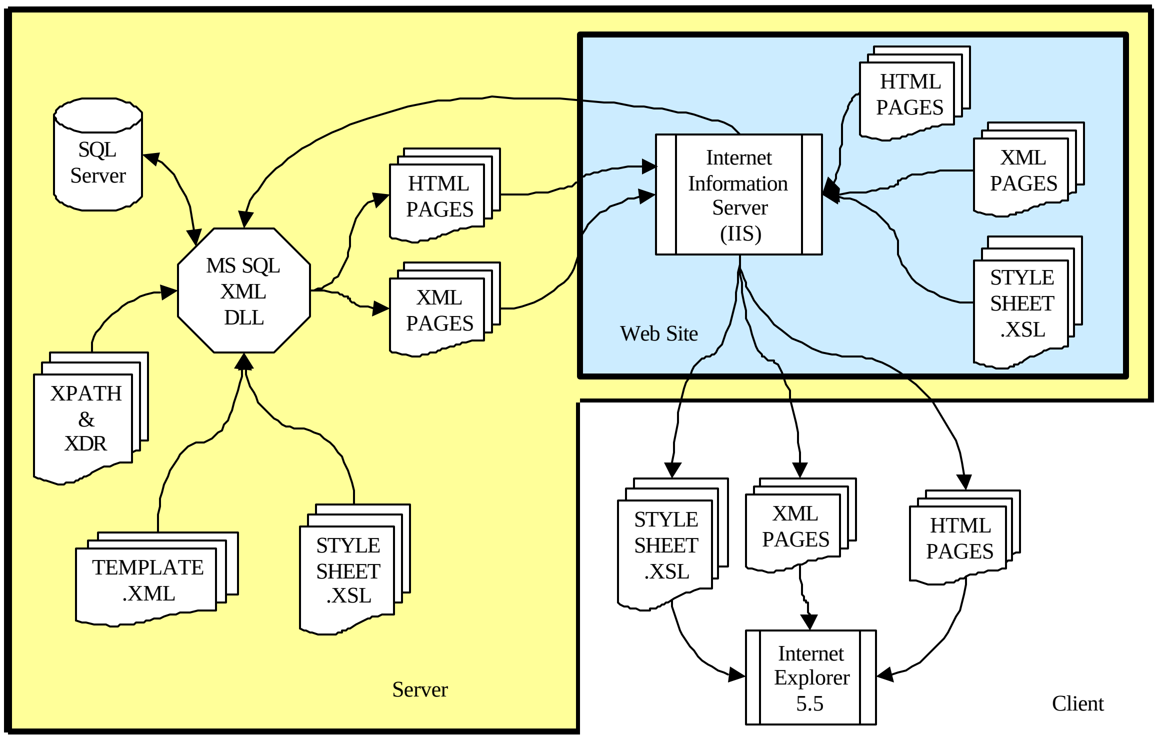

Figure 2 - MS SQL XML Architecture overview [5]

The MS SQL XML facility supports different request types and techniques. These requests can use XPath and XDR information stored in files as well as template queries stored in template files to direct its database requests. The content returned from the database can be translated using XSL files to convert the raw XML content returned from the database into content (usually HTML or XML), which is then returned to the web site. The web site can then return this dynamic content directly, or include additional content from its "normal" files. For example, dynamic content can be embedded into static pages using server side includes, frames or data islands, or it can be bundled with static content such as CSS, XSL, image files, etc.

Custom Servlet Architecture

Figure 3 shows the architecture specific to the custom servlet approach. Here, the combination of an ISAPI redirector DLL, Tomcat server, and JSQLXML3 servlet take the place of the Dynamic Content Generator from Figure 1.

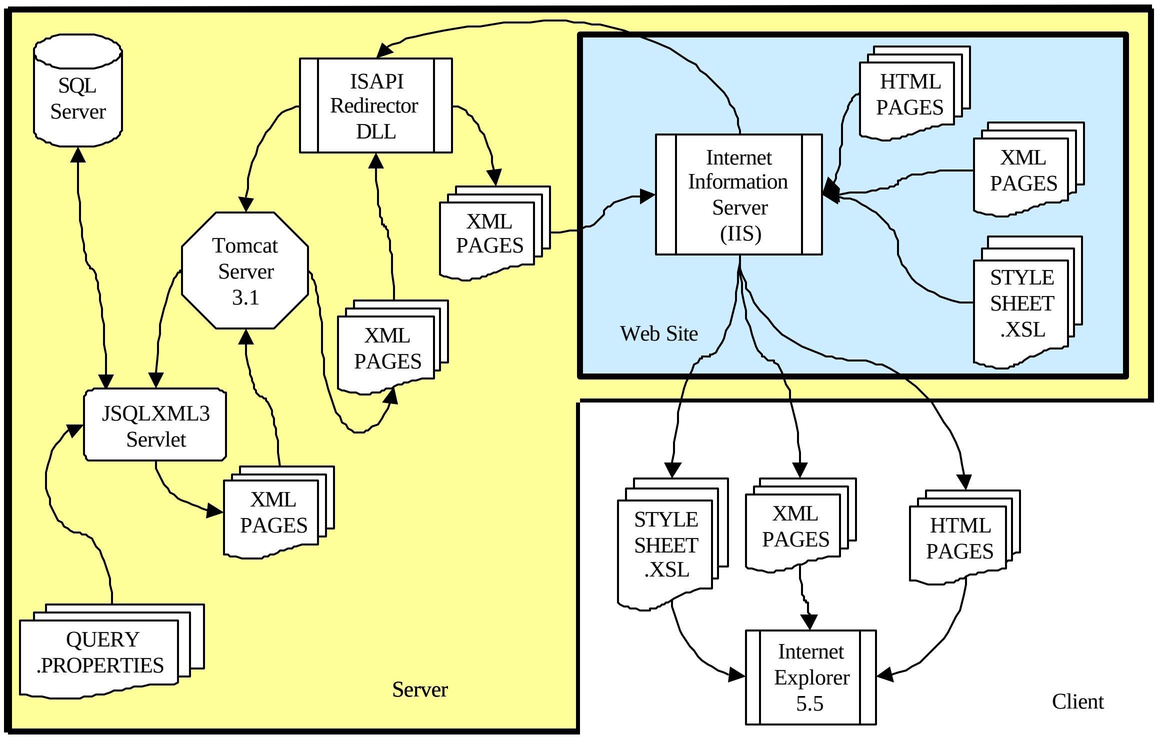

Figure 3 - Custom Servlet Architecture overview

The Tomcat Server is a servlet container that also supports Java Server Pages (JSP). It is essentially a scaled-down web server capable of hosting Java servlets. It consists of many files (not shown here) and requires non-trivial configuration not described here. Unlike the MS SQL XML DLL in Figure 2, Tomcat is not implemented as an ISAPI redirector DLL. However, it has a separate ISAPI DLL that allows the Tomcat Server to be plugged in similar to the first approach.

The JSQLXML3 Servlet is the custom software written for the project. It mimics a subset of the functionality supplied by the MS SQL XML facility. It uses Java "Properties" files to define queries and JDBC to connect to the database. The results are returned to the Tomcat server XML content, which is then forwarded to the web site to be used the same as the content in MS SQL XML approach.

It is important to notice that, unlike the MS SQL XML approach, there is no server side XSL processing in the custom servlet approach. This is not because it was not possible, but rather because it was determined to be out of scope for the project. There are several XSL/XSLT implementations available to choose from and it would not be difficult to add this functionality in a real deployment.

MS SQL XML Facility Details

The vendor-supplied facility is implemented in two places: the MS SQL XML facility and the SQL Server 2000 database engine.

The MS SQL XML facility is installed as part of the SQL Server 2000 installation and configured separately. Configuration results in creating a virtual directory in the web server. Queries can be specified using any of four different types of access to the virtual directory, URL queries, template queries, XPath queries, and POST queries.

The SQL Server database engine supports new syntax for SQL queries and other extensions see [6] for further details. This new syntax includes a set of clauses that can be appended to a normal SQL select statement to reformat the results into an XML data stream. The syntax that this stream conforms to is dependent upon the particular syntax used (see Table 1).

The general syntax for the clause is:
FOR XML { RAW | AUTO | EXPLICIT } [, XMLDATA] [, ELEMENTS]
[, BINARY base64]}

Table 1 - MS SQL XML Select Syntax Extensions

Clause	Description	Example
RAW	<row> elements where the column names are attributes and the column values are the values for those attributes in each <row> element.	<row dno="100" dname="Accounting"/>
AUTO	<Table-name>elements containing as nested child elements based upon the tables in the from clause and columns in the select clause	<dept dno="100" dname="Accounting"> <emp ssn="10" dno="100" /> </dept>
EXPLICIT	Involves using column aliases and a complex naming convention to direct the structure of the XML document	too complex to show here: uses expressions like "[dept!2!id!id]", where the "!" separates pieces in a traversal of the tree in some specific sense
XMLDATA	This prepends an XDR for the query as part of the results returned	See XDR specification [5]
ELEMENTS	Used to return columns as child elements rather than attributes	<row> <dno>100</dno> <dname>Accounting</dname> </row>
BINARY base64	Because XML is a TEXT encoded format, this option specifies that binary content should be converted to base64 (MIME).	

In addition to the new SQL extensions, the MS SQL XML facility also supports XPath queries (limited to very simple XPath expressions). XPath queries are an alternative to SQL queries that use XDR annotated schemas describing the format of XML data to support search and filter expressions. Using XPath relies on understanding the XDR [5], XPath [4], and XML Schema [7] specifications.

While XPath queries themselves are not difficult to use, they are difficult to describe briefly. Just to provide a flavor of the XPath queries, the SQL query:
SELECT * FROM STUDENT WHERE STU_ID<=200
would be translated to:
STUDENT[(@STU_ID <=200)=true()]

Both SQL and XPath Queries can be specified either in template files or as part of the URL. The queries differ in format, readability, complexity, and to a certain extent power of expression. See Table 2 for examples of URLs requesting URL or template queries, and Figure 4 for an example of a SQL Query in a Template file.

Table 2 - MS SQL XML Query Request Examples

Request Type	Example
URL	Http://Server/VDir?sql=SELECT+*+FROM+dept+FOR+XML+AUTO
Template	Http://Server/Vdir/templates/query1.xml
URL w/ parameter	Http://Server/VDir?sql=SELECT+*+FROM+student+where+stu_id=1001001+FOR+XML+AUTO
Template w/ parameter	Http://Server/Vdir/templates/query1.xml?stuid=1001001

```
<?xml version ='1.0' encoding='UTF-8'?>
<!-- Schedule_For_Student2.xml - a template -->
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql" sql:xsl="/Student2.xsl" >
       <sql:header>
               <sql:param name="studid"></sql:param>
       </sql:header>
       <sql:query >
       select  pers.fname
                      ,pers.lname
                      ,semester.sem_year
                      ,course.crs_name
                      ,section.sec_number
       from    student, pers, semester, course, section,takes
       where   student.stu_id=@studid
       and     student.stu_id=pers.per_id
       and     student.stu_id=takes.stu_id
       and     takes.sec_id=section.sec_id
       and     section.sem_id=semester.sem_id
       and     section.crs_id=course.crs_id
       order by
       pers.fname,pers.lname,semester.sem_year,course.crs_name,section.sec_number
       for xml raw
       </sql:query>
</ROOT>
```

Figure 4 - Sample SQL Template Query (query1.xml)

Benefits

- Support for SQL queries (coded easily by users familiar with SQL)
- Support for simple URL queries (coded easily by users familiar with web programming)
- Support for the template files (provide encapsulation, readability, and potential reuse)
- Support for XDR generation (don't have to create from scratch)
- Support for XDR annotated schemas (provide an abstraction from the database model, and physical details)
- Support for XPath queries (provide an abstraction from the actual database syntax, coded easily by XML programmers not necessarily familiar with SQL)
- Server-Side XSL processing is automatically supported

Disadvantages

- URL syntax is difficult since spaces and other characters must be escaped.
- URL syntax exposes the query to the user
- Template files require escaping of XML special characters in query and parameters. (e.g. &lt;= for <=.
- Template files mean additional files required for the query (in addition to the HTML, ASP. XSL, etc. files).
- Format generated by SQL "FOR XML" clause is simple for simple queries but complex for intermediate queries (table and column aliases can affect things, order of columns can effect things).
- XPath queries are new and strange compared to SQL.
- XDR Schemas can have complex issues when resolving name space (same column different table) and need to be updated when schema changes.
- XPath queries can be to complex for the MS SQL XML to process, even when they look simple and the XDR is properly defined.
- Debugging XML, XSL, XPath, template, HTML, and ASP in combination can be quite difficult. Syntax errors are easy to find, but semantic and logical errors across files are hard for the author to detect.
- Database Connection is per Virtual Directory; no support for changing on a per request basis.
- Incorrect File, Directory, and IP Permissions can cause unusual failures.

Custom Servlet Facility Details

The custom servlet facility also relies on implementation from two places: the services supplied by the Jakarta Tomcat 3.1. Servlet Container, and the web application developed for this project (named JSQLXML3).

Jakarta is the name for the project from Apache that includes servlets and Java Server Pages (JSP), as well as other related products. Tomcat is part of the Jakarta project, namely the environment for running Java Server Pages (JSP) as well as the container for running Servlets in. It can be run as a stand-alone application, in-process server (DLL) or NT service. It can also be embedded in a web server (such as Apache, JWS, or IIS). For this project it was run as an NT Service and then (using an ISAPI Extension DLL to redirect IIS requests to Tomcat) embedded in IIS. Tomcat uses configuration files to control web applications. A web application consists of one or more servlets, HTML files, configuration files, Java archive files (JAR) and Java class files.

The servlet itself is a simple implementation consisting of a single class (less that 500 source lines long) mimicking the FOR RAW behavior in MS SQL XML. The servlet used the JSDK classes: HttpServlet, HttpRequest and HttpResponse for communication and JDK/ JDBC to perform the query. Parameter processing and output was simplistic and relied on simple string processing.

The web application supports SQL queries and parameters specified in either URL, GET, or POST formats. The servlet is initialized when Tomcat first loads it and is then shared for all requestors. Database access was limited to SQL SELECT statements, and therefore no complex synchronization was required.

In order to use this approach, IIS 4.0 must be installed, and Tomcat must be installed on the same machine (or at least the ISAPI redirector does). Tomcat can be configured to run in several different ways, and unfortunately this is not well documented in one single place. The redirector is configured as an ISAPI filter. Tomcat configuration files must be setup to control how URL's are redirected and mapped within Tomcat. Additionally, root directories for Web Applications must be added as Virtual Directories in IIS (or be defined under an existing virtual directory, but this requires more complex configuration). This controls how URL's are redirected from IIS to Tomcat.

Behavior similar to the SQL templates in MS SQL XML by using Java Properties files. Tomcat can be configured to direct all requests under a given path to the same servlet. The servlet can then call standard functions to determine if any additional information is specified after the servlet path and interpret it as it chooses.

For this project, the servlet checked for additional information after the path to the servlet and interpreted this as the name of a Java Properties file. Properties files are similar to windows "INI" files and consist of "name equals value" pairs. This allows the user to store different SQL queries in different properties files and then refer directly to the file containing the desired query in the URL (just like template files).

See Table 3 for examples of URLs requesting URL or properties file queries, and Figure 5 for an example of a SQL Query in a properties file.

Table 3 - JSQLXML3 Query Request Examples

Request Type	Example
URL	Http://Server/JSQLXML3/JSQLXML3?sql=SELECT+*+FROM+dept
Properties	Http://Server/JSQLXML3/JSQLXML3/qry1.properties
URL w/ parameter	Http://Server/JSQLXML3/JSQLXML3?sql=SELECT+*+FROM+student+where+stu_id=1001001
Properties w/ parameter	Http://Server/JSQLXML3/JSQLXML3/qry1.properties?stuid=1001001

```
head   =      <?xml version="1.0"?>\n\
                     <?xml-stylesheet type="text/xsl" href="../ Student2.xsl"?>

sql           =      select  pers.fname \
                     ,pers.lname \
                     ,semester.sem_year \
                     ,course.crs_name \
                     ,section.sec_number \
       from   student, pers, semester, course, section,takes \
       where  student.stu_id=@studid \
       and    student.stu_id=person.per_id \
       and    student.stu_id=takes.stu_id \
       and    takes.sec_id=section.sec_id \
       and    section.sem_id=semester.sem_id \
       and    section.crs_id=course.crs_id \
       order by  \
       pers.fname,pers.lname,semester.sem_year,course.crs_name,section.sec_number

root   =      root
```

Figure 5 - Sample SQL Properties File Query (query1.properties)

Benefits

- Support for single, simple SQL queries was coded easily.
- Support for simple URL queries required no additional programming.
- Although it was not done for this project, adding support for other third-party processors (such as server-side XSL/XSLT) should be trivial.
- More complex formatting of output would be possible, while not trivial.
- Coding support similar to other MS SQL XML formats is possible but not trivial.
- Support for database connections as different user is possible (though not without changing current code).
- Solution portable to other web servers that Tomcat supports (Apache, JWS, or IIS)
- Solution portable to other databases that JDBC supports (ORACLE, etc.)
- Solution portable to other Operating Systems (Unix, Linux, Microsoft Windows 98/95, etc.)
- Solution can operate without web server as stand-alone JSP/Servlet container or embedded in some other environment such as Java2 Enterprise Edition Application Servers (WebLogic, BEA, etc.)

Disadvantages

- URL syntax is difficult since spaces and other characters must be escaped.
- URL syntax exposes the query to the user.
- Development environment is lacking; the Tomcat server needed to be shutdown in order to update servlets (partially addressed in latest release but not completely).
- Coding support for XPath Queries would be non-trivial.
- No direct support for Server-Side XSL.
- Debugging is possible, but difficult.
- Coding support for XDR Schemas would be non-trivial.
- Servlet execution is extremely fragile in Tomcat: unexpected exceptions kill the Java Virtual Machine, which kills Tomcat and all servlets.
- Incorrect File, Directory, and IP Permissions can cause unusual failures.
- Using Java Virtual Machine 1.3 for Tomcat Service requires user to remain logged on in order to use service (there is a free fix for this but not part of Tomcat. package)

Tradeoffs

Usability

While both approaches were usable, the MS SQL XML approach had the advantage of being more complete. Given more time and a serious attempt at creating a production quality system, the Java approach could easily provide enough features to be equally usable.

While the MS SQL XML approach is more powerful, it is also potentially more complex. Using the features such as XSL, XML templates, XDR, and more complicated FOR XML clauses can be very difficult to configure correctly.

The Tomcat approach is equally complex with respect to configuring the servlet to IIS mappings. Perhaps this would be easier if a different web server were used.

Adaptability

The Tomcat approach should work equally well in any RDBMS and also on several operating systems. There is nothing in the approach that requires NT or precludes other OS and RDBMS platforms from participating.

The MS SQL XML approach is out of necessity tied to the Windows NT/Windows 2000, IIS, and MS SQL Server platforms.

Maintainability

While servlets are applications requiring all of the expected maintainability issues, ultimately the maintainability of both approaches would reside in the URL's and equivalent template files.

Tools for verifying and debugging these files would be crucial to maintenance, and sadly are non-existent at this time.

Scalability

The MS SQL XML approach is scalable in that there is no need for additional coding to expand the system within reason. Support is limited to a single database per URL/template, but there is nothing preventing these facilities from being used with other traditional web development approaches such as servlets, ASPs and other web components.

The Tomcat approach is only limited in its current implementation, and could easily be expanded to support multiple databases per request. Also it should be easy to move the servlets to multiple servers.

Performance for both approaches on queries appeared to be bottlenecked by the browser, not the processing of results on the server.

MS SQL XML Strategies

- Use simple SQL queries in template files first. URL queries are not easier than copying/cutting and pasting from a similar template file. Also, URL queries, even when encoded in the href of an HTML file are more difficult to read.
- As a data model stabilizes and as experience grows with XDR, start using generated XDRs to try to perform the same queries using XPath. As experience (and support for more advanced XPath features) grows, decide which format to use based on maintainers, variability of the model and queries, and any performance results you experience. (I have no performance data, but it seems reasonable that some queries might perform better in one format than another based on some kind of empirically determined criteria).
- Use the "XML FOR" clause appropriate to the task at hand
- In general, use FOR RAW when joining tables or fully qualified columns heavily, and FOR AUTO when simpler queries are used.
- Recognize that FOR RAW means XSL based on ATTRIBUTES (,ELEMENT is an AUTO only feature).

Try to focus on one approach consistently to start off with, and add others as needed.

Custom Servlet Strategies

- Use simple SQL queries in template files first. - same as for MS SQL XML.
- Given more time, create an XML Document Type Definition (DTD) for template-like file format and use JAXP to read it.
- Add support for XSLT processing on the server-side.
- Develop on a system other than the production system (since Tomcat must be stopped to allow updated code to take effect).

References

1. "HyperText Markup Language (HTML) 4.01 Specification", November 2000, http://www.w3.org/TR/html401/

2. “Extensible Markup Language (XML) 1.0 Second Edition”, November 2000, http://www.w3.org/TR/REC-xml

3. “Extensible Stylesheet Language (XSL) Version 1.0”, November 2000, http://www.w3.org/TR/xsl/

4. "XML Path Language (XPath) Version 1.0", November 2000, http://www.w3.org/TR/xpath/

5. "XML-Data Reduced Draft Version 0.21", November 2000, http://www.w3.org/TR/1998/NOTE-XML-data-0105/

6. "XML and Internet Support Overview", SQL Server 2000 Online Help, October 2000 RTM

7. " XML Schema Part 2: Datatypes", November 2000, http://www.w3.org/TR/xmlschema-2/