

UNIVERSITY OF CINCINNATI

Date: _____

I, _____,
hereby submit this work as part of the requirements for the degree of:

in:

It is entitled:

This work and its defense approved by:

Chair: _____

AN XML-BASED COURSE REGISTRATION SYSTEM

A thesis submitted to the

Division of Research and Advanced Studies
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Department of Electrical and Computer Engineering and
Computer Science
of the College of Engineering

2004

by

Juan Li

B.E., Huazhong University of Science & Technology, 1995

Committee Chair: Dr. Chia-Yung Han

Abstract

Online course registration system is provided by almost all the colleges, universities and training centers. The users require a convenient and efficient system, students want the system to offer accurate information about course offerings, when come to the registration process, they value complete and comprehensive decision-making information. School Administration wants to have control on the curriculum of the degree programs and the course design. In this thesis, an XML-based course registration system has been designed and implemented. It Uses XML to represent the description of a program, this description can be used by the system to specify and check the selection and the completion of the course requirements. It uses XML to describe student record and course information, also it uses XML-based set of tools to manipulate system information to enable the selection of courses checked by the program rules and other criteria. The system has the features of: allowing a degree program to define its curriculum; allowing administration to define the course offering and allowing the students in the program to plan and register for the necessary courses. Apache Tomcat 4.0 is used as the backend web server, Java and Java Servlet is used as the main programming language.

Keywords: Course Registration, Program Rules, Java Servlet, XML and XSL.

Acknowledgement

I would like to express my sincerest gratitude to my advisor Dr. Chia-Yung Han for his guidance, encouragement, and his patience and kindness. His insightful comments and inspiration have been invaluable throughout the preparation of this thesis.

I would also like to thank Professor Dharma Agrawal and Professor Ken Berman for their constructive suggestions and comments on the thesis as well as serving on the committee.

I would like to thank my friends Qi Zhang, Huazhou Liu and Huiqin Yan for their kindly help and encouragement during my stay in University of Cincinnati.

Especially I wish to thank my husband, Zhiyong Xu and my parents, their love and support make my life meaningful and delightful.

Contents

LIST OF FIGURES.....	III
1 INTRODUCTION	1
1.1 MULTIPLE FACETS OF REGISTRATION	2
1.2 TECHNOLOGIES TO IMPROVE SYSTEM PERFORMANCE.....	4
2 BACKGROUND REVIEW	6
2.1 APACHE TOMCAT WEB SERVER AND JAVA SERVLET	6
2.1.1 <i>The Servlet Run-time Environment</i>	7
2.1.2 <i>Servlet Interface and Life Cycle</i>	8
2.1.3 <i>Request and Response Objects</i>	9
2.1.4 <i>Persistent and Shared Data</i>	10
2.2 XML AND EXTENSIBLE STYLE SHEET LANGUAGE (XSL).....	14
2.2.1 <i>DOM & SAX</i>	17
2.2.2 <i>XPath, XSLT and XSL</i>	20
2.2.3 <i>JAXP and Xalan-Java</i>	25
3 PROBLEM STATEMENT	26
4 SYSTEM ARCHITECTURE AND FUNCTIONALITY DESIGN	28
4.1 SYSTEM ARCHITECTURE.....	28
4.1.1 <i>The web server</i>	29
4.1.2 <i>The servlets</i>	29
4.1.3 <i>The JAXP and Xalan-Java package</i>	29
4.1.4 <i>The XML and XSL Files</i>	30
4.2 SYSTEM FUNCTIONALITY AND PROCESSING WORKFLOW.....	31
4.2.1 <i>Modify program rules</i>	32
4.2.2 <i>Modify a course</i>	33
4.2.3 <i>Student Login</i>	34
4.2.4 <i>Search courses</i>	35

4.2.5	<i>Add/Drop class</i>	36
5	SYSTEM IMPLEMENTATION AND RESULTS	39
5.1	REPRESENTING INFORMATION IN XML.....	39
5.2	MODIFY PROGRAM RULES & COURSE INFORMATION.....	43
5.3	STUDENT LOGIN	47
5.4	SEARCHING COURSE ACCORDING TO DIFFERENT CRITERIA.....	49
5.5	STATISTICS FUNCTION ON STUDENT’S RECORD AGAINST PROGRAM RULES	53
5.6	ADD/DROP CLASSES.....	54
5.7	SESSION TRACKING AMONG DIFFERENT SERVLETS AND WEB PAGES.....	58
6	CONCLUSION AND FUTURE WORK	61
	BIBLIOGRAPHY	63

List of Figures

Figure 1 The Curriculum Requirements for Computer Science Major of One University	1
Figure 2 Multiple Faucets of Course Registration System.....	3
Figure 3 DOM Parser Build an Explicit Object Tree	19
Figure 4 SAX Parser Builds an Implicit Object Tree	19
Figure 5 System Architecture and main functionality	28
Figure 6 System Major Functionality and Workflow	31
Figure 7 Modify Major Rules Function	33
Figure 8 Modify Course Function.....	34
Figure 9 Login and Show Student Information	35
Figure 10 Search course according to the search criteria	36
Figure 11 Add/Drop Class Function	37
Figure 12 A student information presented in XML file.....	40
Figure 13 A Course information presented in XML	41
Figure 14 A major rules presented in XML.....	43
Figure 15 Administration Submit the Rules of Major XX to the System.....	44
Figure 16 The Rules of Major XX	45
Figure 17 Administration Submit the Information of Course X1 into the System.....	46
Figure 18 The Detailed information of Course X1.....	47
Figure 19 System Showing the information of Student A after Login	48
Figure 20 The XSL to extract course information from XML file according to requirements and show in HTML page.	51
Figure 21 The course search results if search criteria is major="CS" and core_course="yes".....	52
Figure 22 Rules of major XX and the statistic results of a student A's course record	54
Figure 23 The XSL file taking in the Add/Drop course list and modifying student record in the XML file	56
Figure 24 The error message of Add/Drop course failure because of not fulfilling prerequisite.....	57
Figure 25 The result of successfully adding course 012 and 014, the statistic results is updated accordingly.....	58

Figure 26 Java codes of setting session attributes	59
Figure 27 Java codes of retrieving session attributes	60

1 Introduction

Course registration is a common step in any educational/degree program. It is a part of a planning process, which normally requires basic functionalities for decision support and decision-making. Registering for courses is seemingly a simple process. Registering for well-structured degree programs with clearly defined courses is simple. What is not simple is when the program provides many selections and choices. The selection of a set of courses would have to satisfy the basic requirements established by the program. These requirements are often embedded in course sequences and pre-requisites. For instance, Figure 1 shows the requirements of Computer Science major of one university:

Program Requirements

The Computer Science curriculum provides training in practical programming skills, modern approaches to systems construction, underlying abstractions and mathematical theory. The first year curriculum includes: Calculus, Physics, Orientation to Computer Science, introductory programming courses and Professional Development. All students must also take three quarters of Freshmen English and Breadth of Knowledge classes. In the fifth year, students work under the guidance of a faculty member to develop a design project that helps to develop their individual experimental and design abilities in a specific area of computer science.

First Year

Orientation to Computer Science	3
Computer Science 1 & Lab	4
Computer Science II	3
General Physics I,II & Labs	10
Breadth of Knowledge	6
Professional Development I	1
TOTAL	51

Second Year

Data Structures	3
Matrix Methods	3
General Physics III & Lab	5
TOTAL	35

Third Year

Programming Languages & Methodologies	3
Pre-Junior Seminar for CS	2
Discrete Math	3
Computer Organization	3
Design & Analysis of Algorithms I	3

Figure 1 The Curriculum Requirements for Computer Science Major of One University

As a planning process, one has to be able to access all the available resources, i.e., the list of course offerings and the time schedule as well as the description of the courses. This

information normally comes in the form of course catalog. Visualizing the availability of the courses is important.

The basic requirements for the program should be explicitly presented. These will form the basis for making selection of the courses to take[1]. In addition to the program requirement, selection is also hinged on both the past record and the current status of the student. In structured situations, required courses are selected based on the time schedule. For elective courses, students can make selection based on interest, schedule, and requirements. Many factors have to be considered carefully.

Nowadays, everything is done on-line. There is a need to have a system that makes this process simple and efficient. A registration process online would consist of the following major steps. First, the student has to log on to the system with his personal identification information. Different sets of information should be available for browsing and making selections. Changes should be easily made or undone. Alternatives can be planned and time schedules as well as time conflicts should be made clear in supporting the course selection.

1.1 Multiple Facets of Registration

Course Registration is an activity that actually involves different stakeholders, students, faculty, and administration. Each of these constituents has a different demand. For instance, students need to be able to make accurate selections that would fulfill their

program study plan. Faculty and administration need to have control on the course design and offering. Any degree program should fulfill a well-defined set of rules with respect to both the amount and the content of the subject matters[2].

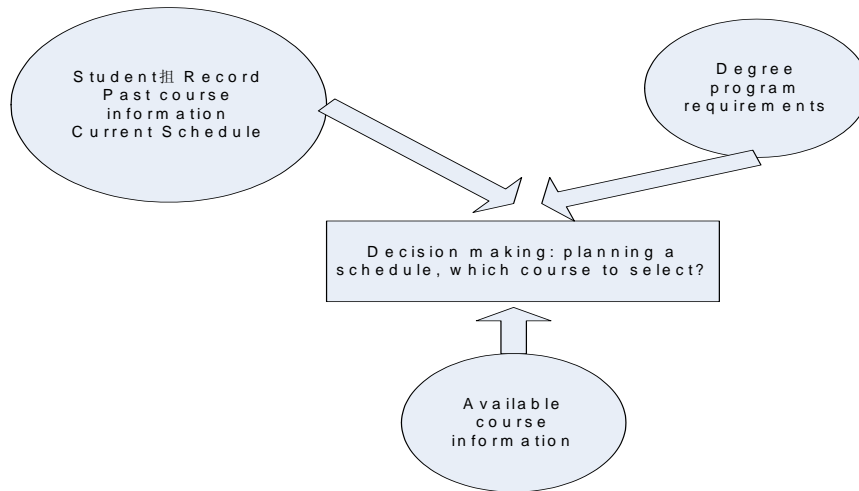


Figure 2 Multiple Faucets of Course Registration System

Student expectations for the course registration system mainly include the following[3]:

- Accurate information about course offerings

Students request more complete information about the professor, the time, the level, the course requirements, and syllabus, less drilling down for course details, and links to catalog course descriptions.

- Improved registration processes.

When it comes to the registration process, students value convenience, responsiveness, and justice. Students value complete and comprehensive decision-making information when planning a schedule. They expect our systems to do their duty and to be smarter than they appear to be. For example, students expect the scheduling of their courses so that irresolvable schedule conflicts don't exist with required courses, a good registration system knows a student's current schedule and won't allow them to add a class which has

time conflict with his schedule. It knows what major a student is pursuing, and the requirements of the major, it can provide related assistance with course selection.

- To retrieve selected data or features in a portal based on function, role, or personal preference

University administration officers and faculties expectations for the course registration system mainly include the following issues:

- Integration of systems in use throughout the enterprise
- Enrollment demand projection system for courses and majors
- Relational student databases needed for robust on-line programs such as prerequisite check.

In order to design a convenient and efficient online course registration system, all the above issues should be taken into consideration.

1.2 Technologies to Improve System Performance

XML stands for the eXtensible Markup Language. It is designed to meet the challenges of large-scale electronic publishing. It is a technology for supporting richly structured documents over the World Wide Web. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations. The reason we choose XML language here is its ability to support for a variety of user-defined data types. With the support for data types:

- It is easier to describe permissible document content;
- It is easier to validate the correctness of data;

- It is easier to define data facets (restrictions on data);
- It is easier to define data patterns (data formats);
- It is easier to convert data between different data types.

With XML, we can easily define complex data like student information, course information and major rules. For any attribute of the data that is not easily defined as a conventional data type like integer, array, string, etc, we can simply use our own tag to define this attribute. We will use a sample to show this ability in Chapter 2. XML language comes with XSL, which stands for eXtensible Style Sheet Language. XSL supports XPath and other data transforming ability which makes the processing and transforming of data in XML format very flexible and convenient. We will give detailed introduction in the next chapter.

The rest of this thesis is organized as follows: Chapter 2 gives a technique review including Java Servlet, Apache Web Server and XML. Chapter 3 states out the problem our system is going to solve and the proposed solution. Chapter 4 describes the architecture and functionality of this course registration system. Chapter 5 describes the system implementation and demonstration; and finally, Chapter 6 gives the conclusion and future work.

2 Background Review

In this chapter, we will review the system implementation environment and tools to define the data structure: Apache Tomcat web server, Java Servlet and XML standards suite.

2.1 Apache Tomcat web server and Java Servlet

Apache Tomcat is the most popular platform for deploying Java-based Web applications. Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and Java Server Pages technologies. The version that we use as the server for our web based course registration system is Tomcat 4.0. Tomcat 4.0 implements a new servlet container (called Catalina) that is based on completely new architecture. The 4.0 releases implement the Servlet 2.3 and JSP 1.2 specifications. Tomcat 4.0 contains significant features, including[4]:

- JMX based administration features
- JSP and Struts based administration web application
- Performance and memory efficiency improvements
- Enhanced manager application support for integration with development tools
- Custom Ant tasks to interact with the manager application directly from build.xml scripts

The 4.0 servlet container (Catalina) has been developed from the ground up for flexibility and performance. Version 4.0 implements the final released versions of the Servlet 2.3 and JSP 1.2 specifications[5]. Servlets are the Java platform technology of choice for

extending and enhancing Web servers. Servlets provide a component-based, platform-independent method for building Web-based applications, without the performance limitations of CGI programs. And unlike proprietary server extension mechanisms (such as the Netscape Server API or Apache modules), servlets are server and platform-independent. This leaves you free to select a "best of breed" strategy for your servers, platforms, and tools.

Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. Servlets can also access a library of HTTP-specific calls and receive all the benefits of the mature Java language, including portability, performance, reusability and crash protection. Today servlets are a popular choice for building interactive Web applications. Third-party servlet containers are available for Apache Web Server, Microsoft IIS, and others.

Servlet containers are usually a component of Web and application servers, such as BEA WebLogic Application Server, IBM WebSphere, Sun Java System Web Server, Sun Java System Application Server, etc.

2.1.1 The Servlet Run-time Environment

A servlet is a Java class and therefore needs to be executed in a Java VM by a service we call a servlet engine. The servlet engine loads the servlet class the first time the servlet is requested, or optionally already when the servlet engine is started. The servlet then stays loaded to handle multiple requests until it is explicitly unloaded or the servlet engine is

shut down. Some Web servers are implemented in Java and have a built-in servlet engine. Other Web servers, such as the Apache Group's Apache, require a servlet engine add-on module. The add-on intercepts all requests for servlets, executes them and returns the response through the Web server to the client. All Servlet API classes and a simple servlet-enabled Web server are combined into the Java Servlet Development Kit (JSDK).

2.1.2 Servlet Interface and Life Cycle

A servlet is a Java class that implements the Servlet interface. This interface has three methods that define the servlet's life cycle:

- Public void `init(ServletConfig config)`. This method is called once when the servlet is loaded into the servlet engine, before the servlet is asked to process its first request.
- Public void `service(ServletRequest request, ServletResponse response)`. This method is called to process a request. It can be called zero, one or many times until the servlet is unloaded. Multiple threads (one per request) can execute this method in parallel so it must be thread safe.
- Public void `destroy()` This method is called once just before the servlet is unloaded and taken out of service.

The Servlet API is structured to make servlets that use a different protocol than HTTP possible. The `javax.servlet` package contains interfaces and classes intended to be protocol independent and the `javax.servlet.http` package contains HTTP specific interfaces and classes. We give out an example, named `ReqInfoServlet` here, it extends a class named `HttpServlet`. `HttpServlet` is part of the JSDK and implements the Servlet interface plus a number of convenience methods. We define our class like this:

```
import javax.servlet.*;
import javax.servlet.http.*;

public class ReqInfoServlet extends HttpServlet {
    ...
}
```

An important set of methods in `HttpServlet` is the ones that specialize the service method in the `Servlet` interface. The implementation of service in `HttpServlet` looks at the type of request it's asked to handle (GET, POST, HEAD, etc.) and calls a specific method for each type. This way the servlet developer is relieved from handling the details about obscure requests like HEAD, TRACE and OPTIONS and can focus on taking care of the more common request types, i.e. GET and POST.

2.1.3 Request and Response Objects

The `doGet` method has two interesting parameters: `HttpServletRequest` and `HttpServletResponse`. These two objects give us full access to all information about the request and let us control the output sent to the client as the response to the request. With CGI we can read environment variables and `stdin` to get information about the request, but the names of the environment variables may vary between implementations and some are not provided by all Web servers. The `HttpServletRequest` object provides the same information as the CGI environment variables, plus more, in a standardized way. It also provides methods for extracting HTTP parameters from the query string or the request body depending on the type of request (GET or POST). We can access parameters the same way for both types of requests. Other methods give us access to all request headers

and help us parse date and cookie headers. We get an `OutputStream` or a `PrintWriter` from the `HttpServletResponse`. The `OutputStream` is intended for binary data, such as a GIF or JPEG image, and the `PrintWriter` for text output. We can also set all response headers and the status code, without having to rely on special Web server CGI configurations such as Non Parsed Headers (NPH).

2.1.4 Persistent and Shared Data

One of the more interesting features of the Servlet API is the support for persistent data. Since a servlet stays loaded between requests, and all servlets are loaded in the same process, it's easy to remember information from one request to another and to let different servlets share data. The Servlet API contains a number of mechanisms to support this directly. We'll look at some of them in detail below. Another powerful mechanism is to use a singleton object to handle shared resources.

2.1.4.1 Session Tracking

An `HttpSession` class was introduced in the 2.0 version of the Servlet API. Instances of this class can hold information for one user session between requests. We can start a new session by requesting an `HttpSession` object from the `HttpServletRequest` in `doGet` or `doPost` method:

```
HttpSession session = request.getSession(true);
```

This method takes a boolean argument. True means a new session shall be started if none exist, while false only returns an existing session. The `HttpSession` object is unique for one user session. The Servlet API supports two ways to associate multiple requests with a session: cookies and URL rewriting.

If cookies are used a cookie with a unique session ID is sent to the client when the session is established. The client then includes the cookie in all subsequent requests so the servlet engine can figure out which session the request is associated with. URL rewriting is intended for clients that don't support cookies or when the user has disabled cookies. With URL rewriting the session ID is encoded in the URLs servlet sends to the client. When the user clicks on an encoded URL, the session ID is sent to the server where it can be extracted and the request associated with the correct session as above. To use URL rewriting we must make sure all URLs that sent to the client are encoded with the `encodeURL` or `encodeRedirectURL` methods in `HttpServletResponse`.

An `HttpSession` can store any type of object. A typical example is a database connection allowing multiple requests to be part of the same database transaction, or information about purchased products in a shopping cart application so the user can add items to the cart while browsing through the site. To save an object in an `HttpSession` we use the `putValue` method:

```
...  
Connection con = driver.getConnection(databaseURL, user, password);  
session.putValue("myappl.connection", con);  
...
```

In another servlet, or the same servlet processing another request, we can get the object with the `getValue` method:

```
...  
HttpSession session = request.getSession(true);  
Connection con = (Connection) session.getValue("myappl.connection");
```

```
if (con != null) {  
    // Continue the database transaction  
    ...
```

We can explicitly terminate (invalidate) a session with the invalidate method or let it be timed-out by the servlet engine. The session times out if no request associated with the session is received within a specified interval. Most servlet engines allow user to specify the length of the interval through a configuration option.

2.1.4.2 Request Attributes and Resources

The servlet API has two more mechanisms for sharing data between servlets: request attributes and resources. The `getAttribute`, `getAttributeNames` and `setAttribute` methods are added to the `HttpServletRequest` class (or to be picky, to the `ServletRequest` superclass). They are primarily intended to be used in concert with the `RequestDispatcher`, an object that can be used to forward a request from one servlet to another and to include the output from one servlet in the output from the main servlet. The `getResource` and `getResourceAsStream` in the `ServletContext` class gives access to external resources, such as an application configuration file. The `ServletContext` methods, however, can provide access to resources that are not necessarily files. A resource can be stored in a database, available through an LDAP server, anything the servlet engine vendor decides to support. The servlet engine provides a context configuration option where we specify the root for the resource base, be it a directory path, an HTTP URL, a JDBC URL, etc.

2.1.4.3 Multithreading

Concurrent requests for a servlet are handled by separate threads executing the corresponding request processing method (e.g. `doGet` or `doPost`). It's therefore important

that these methods are thread safe. The easiest way to guarantee that the code is thread safe is to avoid instance variables altogether and instead pass all information needed by a method as arguments. For instance:

```
private String someParam;
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {
```

```
    someParam = request.getParameter("someParam");  
    processParam();  
}
```

```
private void processParam() {  
    // Do something with someParam  
}
```

is not safe. If the `doGet` method is executed by two threads it's likely that the value of the `someParam` instance variable is replaced by the second thread while the first thread is still using it.

A thread safe alternative is:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {
```

```
    someParam = request.getParameter("someParam");  
    processParam(someParam);  
}
```

```
private void processParam(String someParam) {  
    // Do something with someParam
```

}

Here the processParam gets all data it needs as arguments instead of relying on instance variables.

2.2 XML and Extensible Style Sheet Language (XSL)

Extensive Markup Language (XML) is a markup language for structured documentation. Markup language is a mechanism to identify structures in a document. Structured documents are documents that contain both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption, etc.). Almost all documents have some structure.

XML is a text-based markup language that is fast becoming the standard for data interchange on the Web. It is a cross-platform, software and hardware independent tool for transmitting information[6]. XML is a markup language much like HTML. As with HTML, data are identified by tags (identifiers enclosed in angle brackets, like this: <...>). Collectively, the tags are known as "markup". But unlike HTML, XML tags says what the data means, rather than how to display it. Where an HTML tag says something like "display this data in bold font" (...), an XML tag acts like a field name. It puts a label on a piece of data that identifies it (for example: <message>...</message>). Tags can also contain attributes -- additional information included as part of the tag itself, within the tag's angle brackets. The following example is a university's course information, stored as XML:

Example 1. An XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.2 (http://www.xmlspy.com) by Juan Li (University of Cincinnati) -->
<courseinfo xmlns="http://collegestudents.com/namespace"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://collegestudents.com/namespace
C:\Li\Thesis\courseinfo.xsd">
  <courses>
    <name>Datastructure</name>
    <college>Engineering</college>
    <major>CS</major>
    <call_number>002</call_number>
    <level>200</level>
    <credits>3</credits>
    <area>DataStructure</area>
    <required_level>required</required_level>
    <instructor>Berman</instructor>
    <description>A course to learn build data model to solve
problems</description>
    <prerequisite_course>
      <course_number>003</course_number>
      <course_number>007</course_number>
    </prerequisite_course>
    <quarter>Spring</quarter>
    <time>
      <schedule>
        <Day>Mon</Day>
        <btime>9:30</btime>
        <etime>10:30</etime>
      </schedule>
      <schedule>
```



```
        <Day>Wed</Day>
        <btime>9:30</btime>
        <etime>10:45</etime>
    </schedule>
</time>
<place>swift608</place>
<core_course>y</core_course>
</courses>
</courseinfo>
```

The 1st line in the document - the XML declaration - defines the XML version and the character encoding used in the document. In this case the document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set. The 2nd line is a comment line, which is similar to the comment line in HTML. The 3rd line is a processing instruction line, which says the schema used to display the XML file in web browser. The 4th line describes the root element of the document. The lines between tag <courses> and </courses> describe 16 child elements of the root (name, call number, etc.). And the tag </courses> defines the end of the root element.

In HTML, the tags used to mark up documents are predefined. The author of HTML documents can only use tags that are defined in the HTML standard (like <p>, <h1>, etc.). XML allows the author to define his own tags and his own document structure[7]. The tags in the example above (like <name> and <time>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

2.2.1 DOM & SAX

An application involving with XML will have two parts: the parser deals with the XML file and the application consumes the content of the file through the parser. A parser is a software component that sits between the application and the XML files. Its goal is to shield developer from the intricacies of the XML syntax. The parser and the application must share a common model for XML data. In practice, the common model is always some variation on a tree in memory that matches the tree in the XML document. The parser reads the XML document and populates the tree in memory. This tree built by the parser is an exact match of the tree in the XML document. The application manipulates it as if it were the XML document. In fact, for the application, it is the XML document.

There are two basic ways to interface a parser with an application: using object-based interfaces and using event-based interfaces[8]. The standard for object-based interface is DOM, Document Object Model, published by W3C. The standard for event-based interface is SAX, Simple API for XML, developed collaboratively by the members of the XML-DEV mailing list and edited by David Megginson.

The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. DOM has defined classes of objects to represent every element in an XML file. There're objects for elements, attributes, entities, text, and so on. Using DOM, the parser explicitly builds a tree of objects that contains all

the elements in the XML document. This is probably the most natural interface for the application because it is handed a tree in memory that exactly matches the file on disk.

The SAX is the event-driven, serial-access mechanism that does element-by-element processing. The parser does not explicitly build a tree of objects using SAX. Instead, it reads the file and generates events as it finds elements, attributes, or text in the file. With a SAX parser, events are not related to user actions, but to elements in the XML document being read. There are events for elements starts, element ends, attributes, text contents, entities and so on. A major disadvantage of this approach is that SAX does not support random-access manipulation of the document – people see the tokens once, in document order, and that's it. If anything in the middle needs to be referenced, that information needs to be stored by some code for later retrieval. But SAX uses less memory and is more efficient.

Example 2 is a list of students, with their names and IDs, presented in an XML document. The DOM parser reads this document and gradually builds a tree of objects that matches the document. Figure 2 illustrates how the tree is built by DOM parser. Figure 3 illustrates how the SAX parser generates events as it progresses along the documents and builds the tree implicitly.

Example 2. An XML File Fragment

```
<?xml version="1.0"?>  
<studentsinfo>  
<student>
```

```

<name>Alica</name>
<ID>001</ID>
</student>
<student>
<name>Jason</name>
<ID>002</ID>
</student>
</studentsinfo>

```

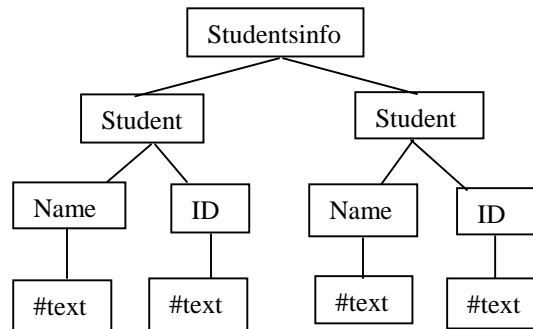


Figure 3 DOM Parser Build an Explicit Object Tree

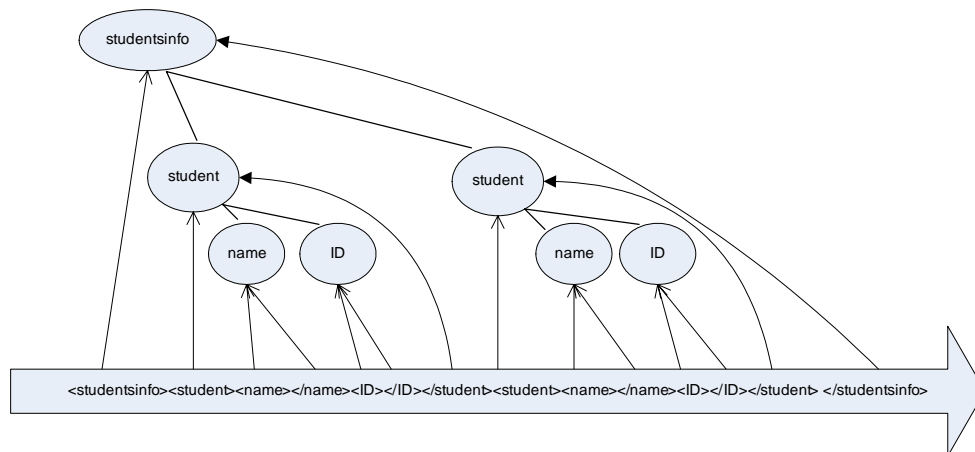


Figure 4 SAX Parser Builds an Implicit Object Tree

As Figure 4 illustrates, taken together, the events describe the document tree to the application. An opening tag event means “going one level down in the tree”, whereas a

closing tag means “going one level up in the tree.” SAX is the most natural interface for a parser. Indeed, the parser simply has to report what it sees. In practice, both forms of interfaces are helpful but they serve different goals. DOM may be used in case other code or applications need to explore and possibly alter the document's contents. When the task processes the document on a straight-line flow-through basis, for example, if a XML document needs to be parsed directly into a database for storage -- SAX may provide a more direct interface to the parser.

2.2.2 XPath, XSLT and XSL

XSL stands for Extensible Style Sheet Language. It is an important standard that achieves several goals.

- Specify an addressing mechanism, namely the path of a hierarchical structure (XPath), so you can identify the parts of an XML file that a transformation applies to.
- Specify tag conversions or transformations (XSLT), so you convert XML data into a different format.
- Specify display characteristics, such as page sizes, margins, and font heights and widths, as well as the flow objects on each page (XML-FO). Information fills in one area of a page and then automatically flows to the next object when that area fills up. That allows you to wrap text around pictures, for example, or to continue a newsletter article on a different page.

2.2.2.1 XPath

In general, an XPath expression specifies a pattern that selects a set of XML nodes[9].

The XPath specification is the foundation for a variety of specifications, including XSLT

and linking/addressing specifications like XPointer. So an understanding of XPath is fundamental to a lot of advanced XML usage. XSLT templates use the patterns specified by XPath when applying transformations. XPointer, on the other hand, adds mechanisms for defining a point or a range, so that XPath expressions can be used for addressing.

The nodes in an XPath expression refer to more than just elements. They also refer to text and attributes, among other things. In fact, the XPath specification defines an abstract document model that defines seven different kinds of nodes, root, element, attribute, namespace, text, comment, and processing instruction.

2.2.2.2 Basic XPath Addressing

An XML document is a tree-structured (hierarchical) collection of nodes. Like a hierarchical directory structure, it is useful to specify a path that points to a particular node in the hierarchy. (Hence the name of the specification: XPath). In fact, much of the notation of directory paths is carried over intact:

- The forward slash (/) is used as a path separator.
- An absolute path from the root of the document starts with a /.
- A relative path from a given location starts with anything else.
- A double period (..) indicates the parent of the current node.
- A single period (.) indicates the current node.

The full range of XPath expressions take advantage of the wildcards, operators, and functions that XPath defines. By definition, an unqualified XPath expression selects a set of XML nodes that matches that specified pattern. For example, /HEAD matches all top-

level HEAD entries, while `/HEAD[1]` matches only the first. But XPath expressions can also contain one of several wildcards to broaden the scope of the pattern matching:

So far, all of the patterns we've seen have specified an exact number of levels in the hierarchy. For example, `/HEAD` specifies any HEAD element at the first level in the hierarchy, while `/*/*` specifies any element at the second level in the hierarchy. To specify an indeterminate level in the hierarchy, use a double forward slash (`//`). For example, the XPath expression `//PARA` selects all PARA elements in a document, wherever they may be found. The `//` pattern can also be used within a path. So the expression `/HEAD/LIST//PARA` indicates all PARA elements in a sub tree that begins from `/HEAD/LIST`. XPath expressions yield either a set of nodes: a string, a boolean (true/false value), or a number. Expressions can also be created using one of several operations on these values.

A node has a string-value, which is a sequence of unicode characters. For a text node, this is the text as it appears in the source XML document, except that the XML parser will have replaced end-of-line sequence by a single new line (`#xA`) character. For a comment, it is the text of the comment, minus the delimiters. For a processing instruction, it is the data part of the source processing instruction, not including the white space that separates it from the PI Target. For an attribute, it is the attribute value. For a root or element node, it is defined as the concatenation of the string value of the element and text children of this node. Or to look at it another way: the concatenation of all the PCDATA contained in the element (or for the root node, the document) after stripping out all

markup. The definition of node string value in XPath is different from the DOM, where the node Value property in these cases is null.

2.2.2.3 XSLT

In an XSL transformation, an XSLT processor reads both an XML document and an XSLT style sheet. Based on the instructions the processor finds in the XSLT style sheet, it outputs a new XML document or fragment thereof[10]. There's also special support for outputting HTML. With some effort most XSLT processors can also be made to output essentially arbitrary text, though XSLT is designed primarily for XML-to-XML and XML-to-HTML transformations. XSLT's ability to move data from one XML representation to another makes it an important component of XML-based electronic commerce, electronic data interchange, metadata exchange, and any application that needs to convert between different XML representations of the same data.

2.2.2.4 XSLT Style Sheet Documents

A XSLT document contains template rules. A template rule has a pattern specifying the nodes it matches and a template to be instantiated and output when the pattern is matched. When a XSLT processor transforms an XML document using an XSL style sheet, it walks the XML document tree, looking at each node in turn. As each node in the XML document is read, the processor compares it with the pattern of each template rule in the style sheet. When the processor finds a node that matches a template rule's pattern, it outputs the rule's template. This template generally includes some markup, some new data, and some data copied out of the source XML document. XSLT uses XML to describe these rules, templates, and patterns. The root element of the XSLT document is

either a style sheet or a transform element in the <http://www.w3.org/1999/XSL/Transform> namespace. By convention this namespace is mapped to the `xsl` prefix. Each template rule is an `xsl:template` element. The pattern of the rule is placed in the `match` attribute of the `xsl:template` element. The output template is the content of the `xsl:template` element. All instructions in the template for doing things such as selecting parts of the input tree to include in the output tree are performed by one or another XSLT elements. These are identified by the `xsl:` prefix on the element names. Elements that do not have an `xsl:` prefix are part of the result tree.

The following example shows a very simple XSLT style sheet with one template rule. It sets the output format as text. The template rule matches the root element and selects the value of attribute *level_requirements* of each element `<major>`. The output is the results of applying the template in this XSL document to the contents of the *majorreqs.xml*.

Example 3: A simple XSLT style sheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:my="http://collegestudents.com/namespace"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">

  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:for-each select="my:major">
      <xsl:value-of select="my:level_requirements"/>, <xsl:text></xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

2.2.3 JAXP and Xalan-Java

The Java API for XML Processing (JAXP) enables applications to parse and transform XML documents using an API that is independent of a particular XML processor implementation[11].

Xalan-Java (named after a rare musical instrument) fully implements the Extensible Stylesheet Language (XSL) and the XML Path Language (XPath)[12]. Xalan-Java performs the transformations specified in the XSL stylesheet and packages a sequence of SAX events that may be serialized to an output stream or writer, used to build a DOM tree, or forwarded as input to another transformation. Xalan-Java includes the following features[13]:

- Implements the relevant W3C specifications: XSL and XPath
- Can process Stream, SAX or DOM input, and output to a Stream, SAX or DOM.
- Transformations may be chained (the output of one transformation may be the input for another).
- May be run from the command line for convenient file-to-file transformations.

The XML and XSL introduced above will be used in our system to implement data models.

3 Problem Statement

Any course registration system would serve three kinds of people: students, faculty and school administration. It processes three main types of information: students' information, course details and major rules. Students' information should include the student's ID, PIN, name, finished course information and current schedule etc. Course information should include course name, call number, level, prerequisite, contents, schedule, etc. Major requirements should include detailed program rules which students need to satisfy to get the degree. The registration system would help students to make decision during the registration process, as student would need to know about his finished course information, his major requirements and detailed course information to make decision. The actual listing of the courses would need to be updated frequently. The system should allow faculty to update courses from time to time and school administrative personnel to change program rules accordingly. Our problem is to **design an XML-based course registration system which:**

- allows a degree program to define its curriculum;
- allows administration to define the course offering; and
- allows the students in the program to plan and register for the necessary courses.

We can see the information contained in the system could be very complex. The system needs to handle all these information flexibly with a set of functionalities that include not only the data storage, retrieving, modification and transforming, but also frequent data exchanging among different data set. Thus, a key issue in our solution is how to organize

the data. As discussed in Chapter 2, XML and its related processing tools can provide the data structure and means to implement and process the information flexibly and effectively. Our proposed solution uses XML as the major data structure to define the student, course and program information in our system. More specifically,

- Use XML to represent the description of a program. This description can be used by the system to specify and check the selection and the completion of the course requirements.
- Use XML to describe student record and course information.
- Use XML-based set of tools to manipulate system information to enable the selection of courses checked by the program rules and other criteria.

4 System Architecture and Functionality Design

This chapter presents the system architecture and functionality of our XML-based course registration system. Section 4.1 briefly describes the system requirements, Section 4.2 describes system major functionalities and workflow.

4.1 System Architecture

The course registration system has a client /server architecture, as shown in Figure 5.

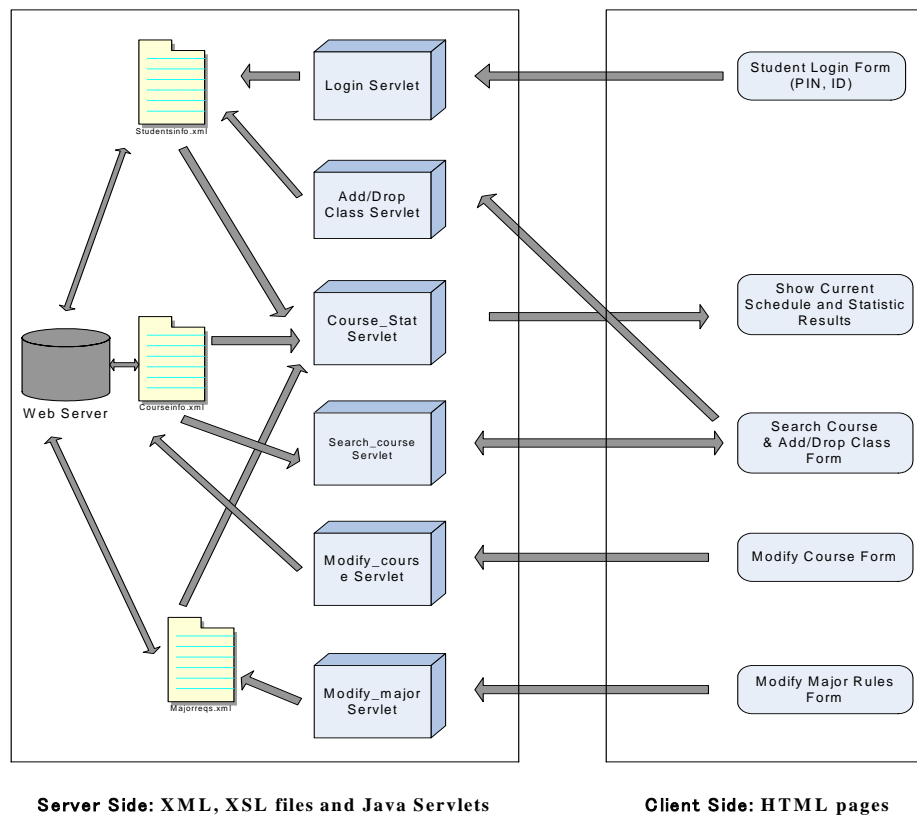


Figure 5 System Architecture and main functionality

We use Apache Tomcat 4.0 as the backend web server, Java as the programming language, and Java Servlet as the main technology to implement our web applications.

Also we add in some special Java package, JAXP and Xalan-Java, to handle the special data file, XML and XSL. XML DOM is used as the data model to store data like course information, student information and major rules, XSL files is used to transform the data and exchange data between different data set. The following briefly describes the major function of each system component:

4.1.1 The web server

- Hosts all the system information that can be accessed in the course registration process, like student information, course information and major rules.
- Hosts all the XML files and XSL files which contain all the data definition and data transform and exchange information according to different data requirements.
- Hosts the servlet class files which are running in the backend server to create dynamic web pages according the students and administrative requirements.

4.1.2 The servlets

The servlets class files are running in the backend server, handling the requests from the front-end web users, processing the XML and XSL files on the web server and generating web pages showing the information requested by the user.

4.1.3 The JAXP and Xalan-Java package

- Xalan-Java performs the transformation specified in the XSL files and packages a sequence of events that may be serialized to an output stream or writer, used to build a DOM tree, or forwarded as input to another transformation.

- The JAXP package supports the XML 1.0 recommendation and contains all parser functionality used to transform the XML data.

4.1.4 The XML and XSL Files

- *studentsinfo.xml* contains the detailed information of each student, such as ID, PIN, name, major, college, accomplished course information, current schedule, etc.
- *courseinfo.xml* contains detailed information of each course, such as course name, call_number, level, credits, prerequisite course, instructor, contents, etc.
- *majorreqs.xml* contains the rules of each major, student needs to fulfill this rules in order to get the degree.
- The XSL files contain all the data transform and exchange information.

4.2 System Functionality and Processing Workflow

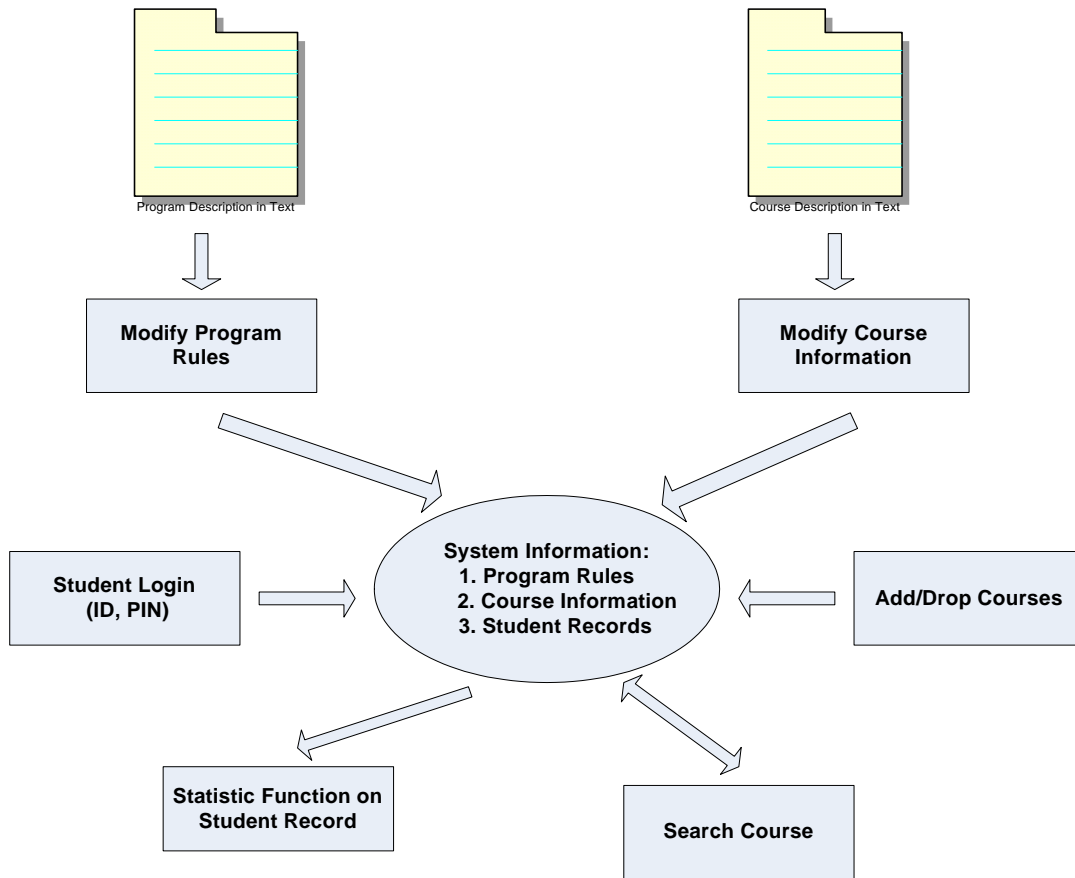


Figure 6 System Major Functionality and Workflow

The major functionalities include modify program rules, modify course information, student login, search courses, add/drop courses, and show student record. The whole system can be divided in two parts: the faculty/administrative part and the student part.

University Administration part includes the following functionalities:

- **Check/Modify Program Rules** the university administration officer can check and modify the criteria and course requirements of a specific major, thus to control the program rules effectively.

- **Check/Modify Course Information** faculty can check and modify the course information, such as the course level, content, whether it should be a required course, etc.

Student part includes the following functionalities:

- **Login** students can login into the system after they entered the valid ID and PIN, system will show detailed information about this student, such as his name, college, department, major, the courses he finished, his current course schedule, etc.
- **Statistic function** according to the students' department, major, the system can show out the criteria and requirements of his major, his finished course, his current schedule, thus to give them some view of what kind of courses they should choose to meet the degree requirements.
- **Search course function** can search for courses according to the students preference, such as major, topic and level preferences, etc, system can show detailed course information once the student know the call number of a course.
- **Add/Drop class** students can add or drop courses, the system can do some check such as to avoid time conflict in schedule, check prerequisite of the courses thus to make sure the students are eligible to register the courses.

4.2.1 Modify program rules

The modify program function, its workflow is shown in Figure 7, allows school administrative to modify the major rules. A user enters the name of the major he wants to modify and password. System verifies his password and accesses the *majorreqs.xml*, if there exists a major with the same name, this is to modify the rules of an existing major, otherwise, it is to add the rules of a new major into the system. The modify interface

gives out a HTML form allowing the user to modify major requirements, such as the total credit hours, the credits hours for each specific level, topic, core course requirements, etc.

After submitting, system will access the *majorreqs.xml* again to update the information.

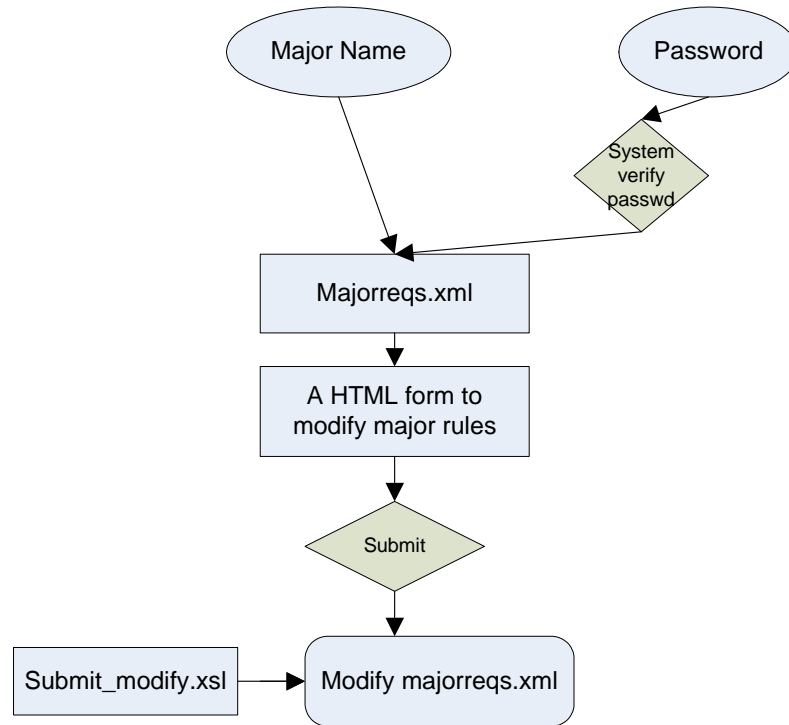


Figure 7 Modify Major Rules Function

4.2.2 Modify a course

This function allows faculty users to modify the course information. As shown in Figure 8, user enters the call number of the course he wants to modify and password. System verifies his password and accesses the *courseinfo.xml*. If there exists a course with the same call number, this is to modify an existing course, otherwise, it is to add a new course into the system. The modify interface gives out a HTML form allowing the user to modify course information, such as course name, level, credit hours, etc. After inputting

all the information and submit, system will access the *courseinfo.xml* again to update the course information.

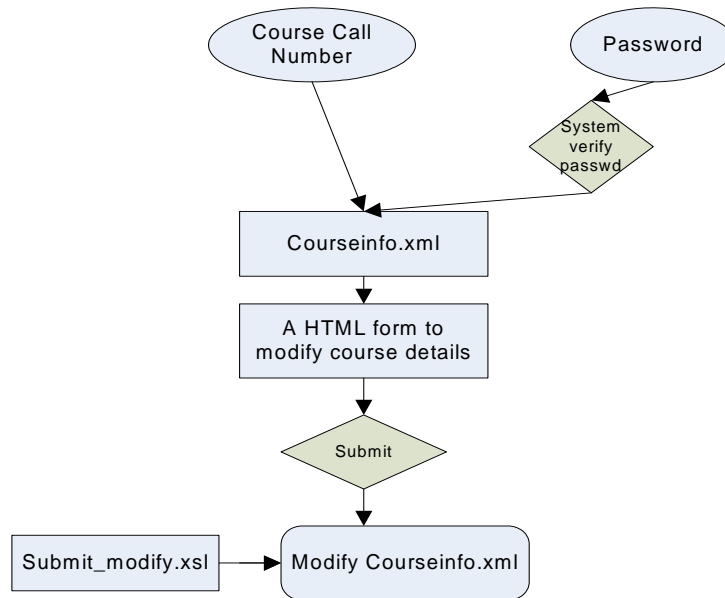


Figure 8 Modify Course Function

4.2.3 Student Login

A student tries to login into the system by entering his ID/PIN information. System will first access the *studentsinfo.xml* file, if there's a record has the same ID and PIN as entered, the login is valid and system gets the according information of this student from *studentsinfo.xml* file, this includes name, major, college, past course information and current course information. Then system will use the student's major information to access the *majorreqs.xml* file, get out the major requirements, also the system will do a statistic on the student's past course information and current course information. Finally, system will output all these information in a HTML page. Figure 9 shows the workflow of this process.

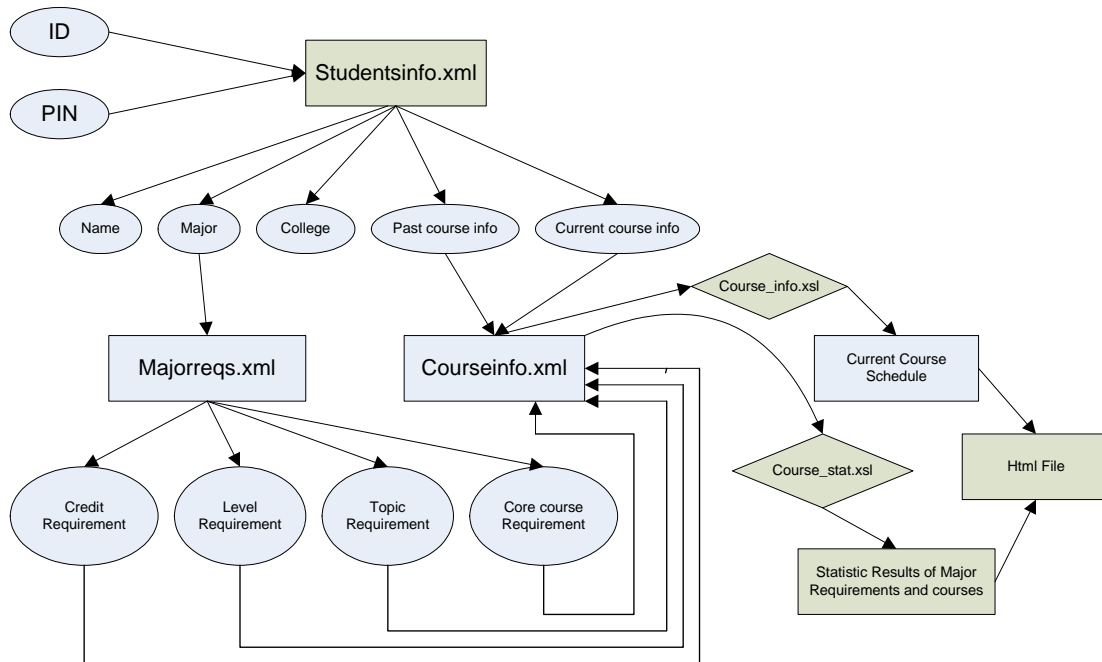


Figure 9 Login and Show Student Information

4.2.4 Search courses

A user searches the course information using this function. This interface (see Figure 10) allows the user to set 6 criteria for the course searching: Major, Call number, Level, Topic, Required level, whether it is a core course. The user can use either of these criteria or use several of them at the same time as a combined search, the system will access the *courseinfo.xml* to find out all the courses that fulfill the requirements, then it will present the information of these courses in a HTML page. Students can use this function to search courses according to their preference and choose a course according to the searching results.

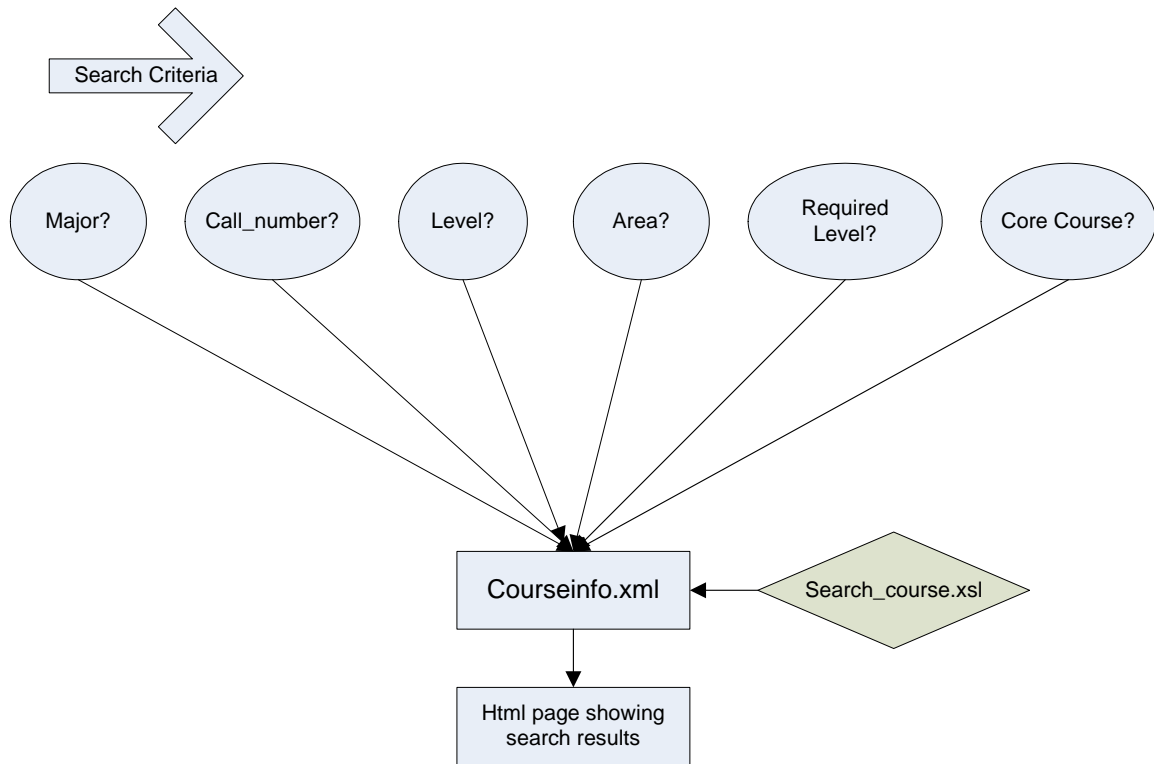


Figure 10 Search course according to the search criteria

4.2.5 Add/Drop class

The interface of this function is integrated with the *Search_course* function thus to allow users conveniently check course information and add/drop classes. Figure 11 shows its flow. This interface gives two lists, in one list students enter the call number of the courses they want to register, in the other list students enter the call number of the courses they want to drop from their current courses. The servlet will take this information in and do some checking.

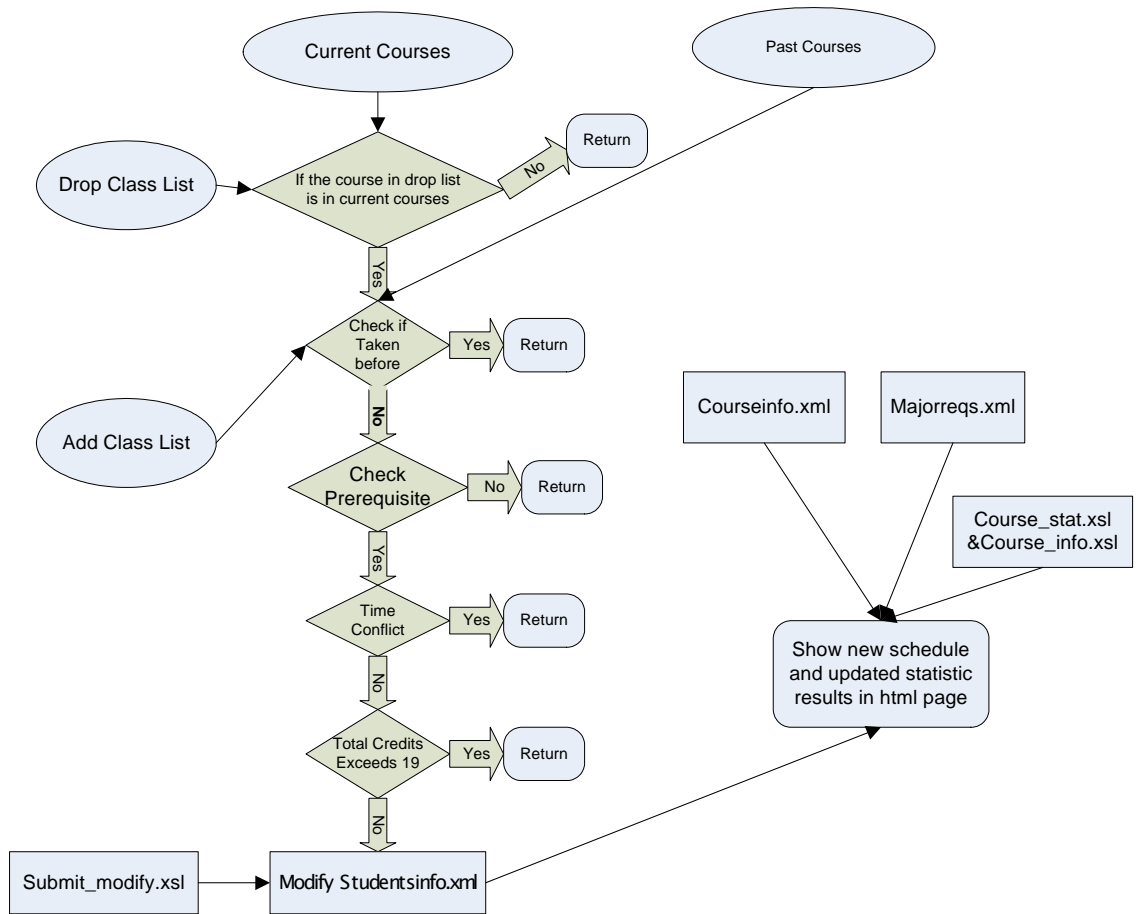


Figure 11 Add/Drop Class Function

First, it will check the drop list, if any of them is not in the student’s current courses, the system throws out an error message and fails the process; the system then checks the add list and the student past course information, if the student has taken any of the courses before, it throws out an error message and fails the process; system checks every course in the add list against the *courseinfo.xml* to find out prerequisite course for each course, if any of the prerequisite courses has not yet been accomplished by the student, it throws out an error message and fails the process; system checks all the courses in the add list and in his current courses, if there’s a time conflict, it throws out an error message and

fails the process; system adds up all credit hours of the courses in the add list and his current courses, minuses credit hours of all the courses in the drop list, if the total credits exceeds 19, it throws out an error message and fails the process. If all these check passes, the system will access the *studentsinfo.xml*, add all the courses in the add list to the current courses and take off all the courses in the drop list from the current courses, also the system will access the *majorreqs.xml* and *courseinfo.xml* files again to get an up-to-date course statistic results and present the up-to-date schedule and statistic results in the HTML page.

5 System Implementation and Results

This chapter explains system implementation in detail. It also demonstrates the results and screen shots showing the major functionality. Section 5.1 describes how to present student, course information and major rules in XML. Major functionality such as transforming students information into HTML file, searching course according to different criteria, doing statistic function by accessing and exchanging among different data set, getting add/drop class list from HTML form and updating the information in XML file, modifying course information and major rules are described in sections 5.2 - 5.6. Section 5.7 discusses how to track the consistent information among different web pages by Session tracking.

5.1 Representing information in XML

We will define several key elements to represent the information that is needed for the course registration system: student element, course element, major element, etc.

In every XML file below, we just show the structure of one record, for instance, record of one student, one course, rules for one major, the other information has the same structure but different value, we simply omit them here for briefness.

A student element includes the following information:

Name, college, major, stud-ID, password, takencourses, and currentcourse. Takencourses indicates a list of courses the student has taken and currentcourses indicates a list of courses the student is currently taking.


```

<students>
  <name>Juan Li</name>
  <college>Engineerig</college>
  <major>CS</major>
  <stud-ID>111</stud-ID>
  <password>111</password>
  <takencourses>001</takencourses>
  <takencourses>002</takencourses>
  <takencourses>003</takencourses>
  <takencourses>004</takencourses>
  <takencourses>005</takencourses>
  <takencourses>006</takencourses>
  <takencourses>007</takencourses>
  <takencourses>008</takencourses>
  <currentcourses>016</currentcourses>
</students>

```

Figure 12 A student information presented in XML file

A course element includes the following information:

Name, college (which college offers this course), major (which major offers this course), call number (normally consisting a code that includes college_code-dept_code-course_code), level, credits (credit hour of this course), area (which area this course belongs to), required_Level (whether this course is required for this major), instructor, contents (course description), prerequired_course (which courses should be finished before taking this course), quarter (which quarter this course is offered), time (schedule of this course), place, core_course (whether this course is a core course of this major or not).

```

<courses>
  <name>Datastructure</name>
  <college>Engineering</college>
  <major>CS</major>
  <call_number>002</call_number>
  <level>200</level>
  <credits>3</credits>
  <area>DataStructure</area>
  <required_level>required</required_level>
  <instructor>Berman</instructor>
  <description>A course to learn build data model to solve
problems</description>
  <prerequisite_course/>
  <quarter>Spring</quarter>
  <time>
    <schedule>
      <Day>Mon</Day>
      <btime>9:30</btime>
      <etime>10:30</etime>
    </schedule>
    <schedule>
      <Day>Wed</Day>
      <btime>9:30</btime>
      <etime>10:45</etime>
    </schedule>
  </time>
  <place>swift608</place>
  <core_course>y</core_course>
</courses>

```

Figure 13 A Course information presented in XML

A major element is used to describe the rules of a program, it includes the following information:

name, college (to which college this major belonging to), topic_reqs (topic requirements), level_reqs (level requirements), corecourse_reqs (core course requirements), credits_reqs (credit hour requirements).

```
<majors>
  <name>CE</name>
  <college>Engineering</college>
  <topic_reqs>
    <areas>
      <area_name>ComputerArchitecture</area_name>
      <levels>
        <level>400</level>
        <credits>9</credits>
      </levels>
    </areas>
    <areas>
      <area_name>ProgrammingLanguage</area_name>
      <levels>
        <level>200</level>
        <credits>3</credits>
      </levels>
      <levels>
        <level>400</level>
        <credits>3</credits>
      </levels>
    </areas>
  </topic_reqs>
  <corecourse_reqs>
    <credits>12</credits>
  </corecourse_reqs>
```

```

<level_reqs>
  <levels>
    <level>100</level>
    <credits>15</credits>
  </levels>
  ...

  <levels>
    <level>400</level>
    <credits>24</credits>
  </levels>
</level_reqs>
<credits_reqs>160</credits_reqs>
</majors>

```

Figure 14 A major rules presented in XML

5.2 Modify Program Rules & Course Information

The *Modify_major* servlet implements the function of modifying the rules of a program.

The user inputs the major name and password, any the system will pull out the HTML form letting the user modify the corresponding rules of a major, if the major name the user inputted is not existing, this will be considered as adding rules of a new major into the system. For example, if there exists a new major, called XX, with the following rules:

Rule-1: Program name is XX

Rule-2: Student's main college affiliation: YY

Rule-3: Total credit hours for the program: 180

Rule-4: Core course credit hours for the program: 24

...

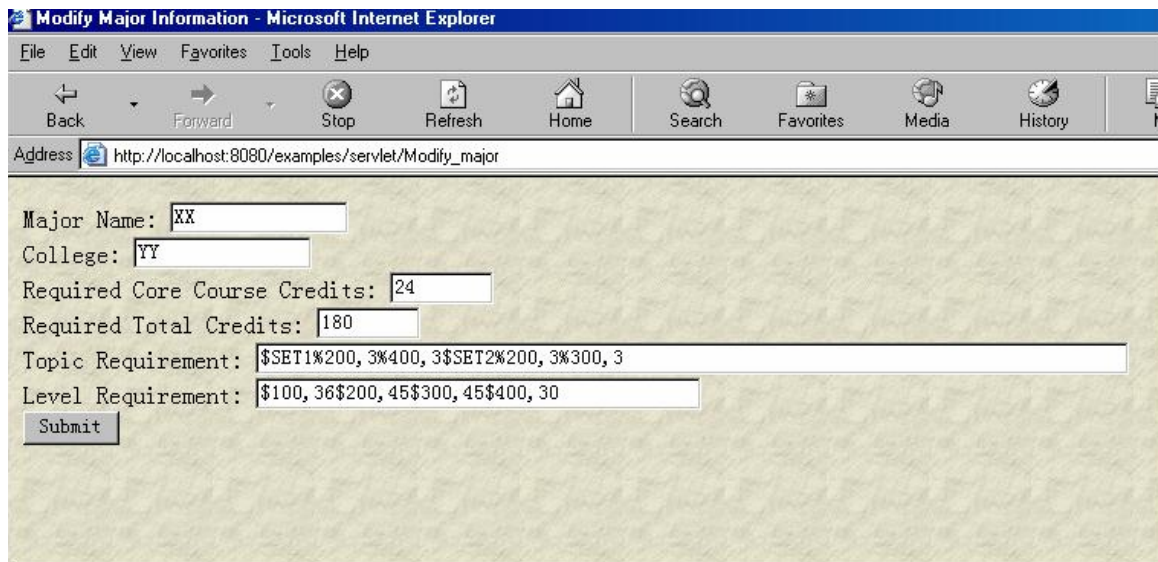
Rule-5: Course level requirement: Level 100: 36 (credit hours) Level 200: 45
Level 300: 45 Level 400: 30

Rule-6: Curriculum requirement: two sets of requirements: SET1 and SET2.

For SET1: students need to finish at least 3 credit hours for the level 200 course and 3 credit hours for the level 400 course.

For SET2; students need to finish at least 3 credit hours for the level 200 course and 3 credit hours for the level 300 course, etc.

All these information of the major XX are on the paper, system gives out an interface of a HTML form, the school administration just needs to input the above information and submit:



The screenshot shows a Microsoft Internet Explorer browser window titled "Modify Major Information - Microsoft Internet Explorer". The address bar displays "http://localhost:8080/examples/servlet/Modify_major". The form contains the following fields and values:

- Major Name: XX
- College: YY
- Required Core Course Credits: 24
- Required Total Credits: 180
- Topic Requirement: \$SET1%200, 3%400, 3\$SET2%200, 3%300, 3
- Level Requirement: \$100, 36\$200, 45\$300, 45\$400, 30

A "Submit" button is located at the bottom left of the form.

Figure 15 Administration Submit the Rules of Major XX to the System

After submit, the system will record the rules of this new major in an XML file, next, when a user queries the rules about this major, system will extract the information from the XML file and show the results:

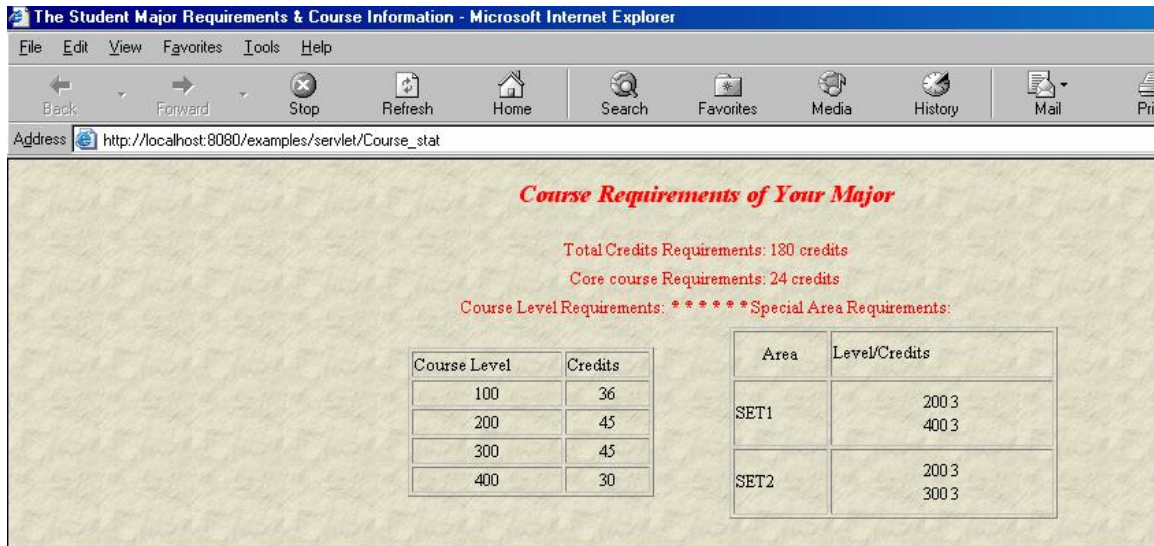


Figure 16 The Rules of Major XX

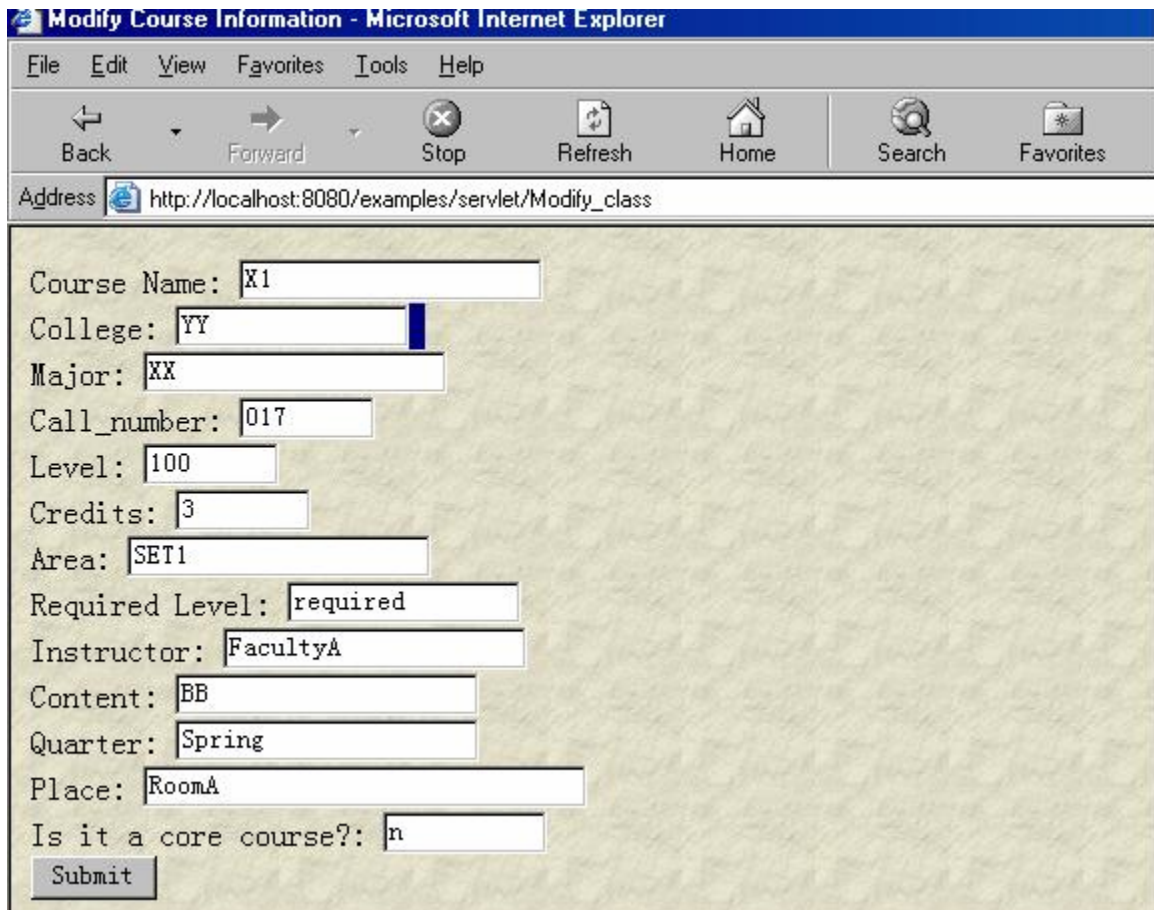
The rules of major XX will be applied to the student who is in this major when system does statistic function on this student's record.

The *Modify_course* servlet implements the function of modifying the course information. User needs to input the password and call number of the course he wants to modify into the system. System will pull out an HTML form letting the user input the information of a course. If the call number does not exist in the system, this will be considered as adding a new course to the system. For example, if there is a course with the following information:

Course name: X1	College: YY	Major: XX
Call number: 017	Level: 100	Credit hour: 3
Course Area: SET1	Required level: required	Instructor: Faculty A
Content: BB	Quarter: Spring	Place: Room A

And it is a core course of this major, etc.

The information of this course X1 are all on paper in hard copy, the administration needs to input these information in the following form and submit:



The screenshot shows a Microsoft Internet Explorer browser window titled "Modify Course Information - Microsoft Internet Explorer". The address bar displays "http://localhost:8080/examples/servlet/Modify_class". The main content area contains a form with the following fields and values:

Course Name:	X1
College:	YY
Major:	XX
Call_number:	017
Level:	100
Credits:	3
Area:	SET1
Required Level:	required
Instructor:	FacultyA
Content:	BB
Quarter:	Spring
Place:	RoomA
Is it a core course?:	n

At the bottom of the form is a "Submit" button.

Figure 17 Administration Submit the Information of Course X1 into the System

System will store this course information in an XML file. When a user queries the detailed information about this course next time, system will extract this information from the XML file and show the results:

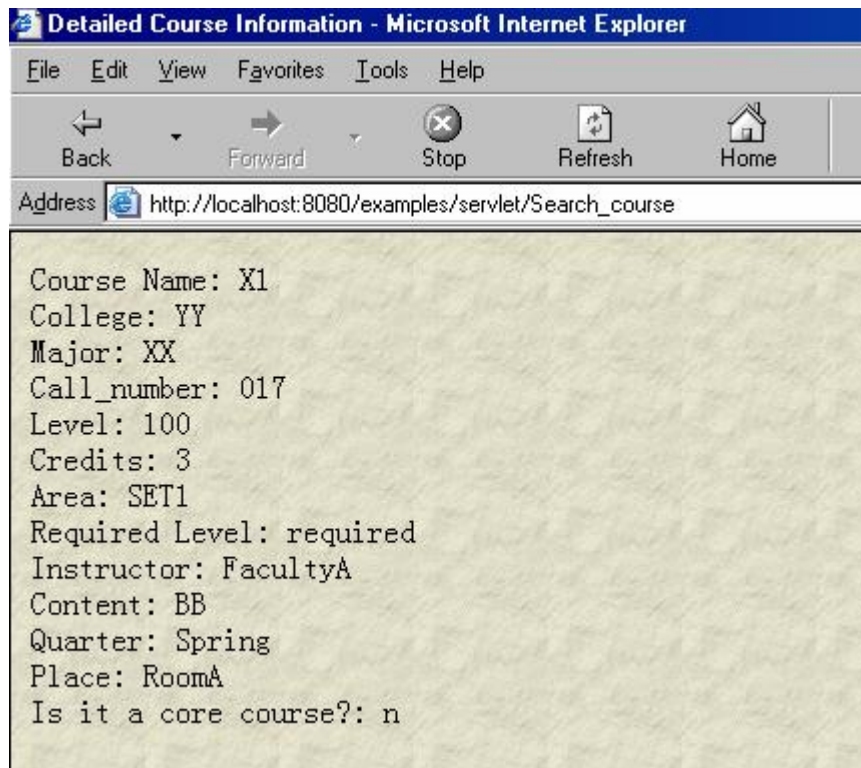


Figure 18 The Detailed information of Course X1

The course X1 is added into the system and available for students to register.

5.3 Student Login

The *Student_login* servlet implements the function of showing students information after the student enters ID and PIN information to login. Let's look at an example of the student A login into the system.

Student Name: *StudentA* College: *YY*

Your Current Course Schedule

Call_ID	Title	Place	Time
019	X3	RoomC	Tus9:30-10:30 Thu9:30-10:45
020	X4	RoomD	Tus8:30-9:30 Thu8:30-9:30

Details of your major requirement and your taken courses.

Your Course Statistics and Criteria

Total credits of your taken course: 6
 Credits of your core course: 3

To complete: 174
 To Complete: 21

To complete after this quarter: 168
 To complete after this quarter: 18

Area	To Complete	Count-In This Quarter
SET1	200, 3 400, 3	200, 3 400, 3
SET2	200, 0 300, NaN	200, 0 300, NaN

Level	Credits	To Complete	Count-In This Quarter
100	3	33	33
200	3	42	42
300	0	45	42

Learning Opportunities!

Choose a scope for course information:

Major: Area: Level: Requirement:

Call_ID: Core_Course:

Come Here: Add or Drop Classes!

Add List	Drop List
1 <input type="text"/>	1 <input type="text"/>
2 <input type="text"/>	2 <input type="text"/>
3 <input type="text"/>	3 <input type="text"/>
4 <input type="text"/>	4 <input type="text"/>
5 <input type="text"/>	5 <input type="text"/>
6 <input type="text"/>	6 <input type="text"/>

Figure 19 System Showing the information of Student A after Login

We can see from the above figure, Student A's record is showed after he login into the system: the student name, college, his current course information (current schedule), the statistic results of his past record against the rules of his major. Since this student's major is XX, we can see the rules of major XX apply in the statistic results here: Student A has finished 6 credit hours courses, since the total credit hour requirements for major XX is 180, so he still needs to finish 174 credit hours; since he's taking 6 credit hour courses in the current quarter, so he has to finish another 168 credit hours after this quarter. He has finished 3 credit hours on core course, since this requirement for major XX is 24, so he has to finish 21 credit hours for core course, since he is taking 1 core course in the current quarter, so after this quarter, he still needs to finish another 18 credit hours on core course. Also the system gives out statistic results against the category requirements and course level requirements of major XX.

5.4 Searching Course According to Different Criteria

Search_class servlet implements the function of searching courses according to the search criteria. Figure 20 shows the XSL file used to access the *courseinfo.xml* file and get out course information fulfilling search requirements. We omit the definition of variable *s2 ~ s6* for briefness because they are similar to definition of variable *s1*.

...

1. `<xsl:key name="major" match="my:courses" use="my:major"/>`
2. `<xsl:param name="Major"/>`
3. `<xsl:key name="area" match="my:courses" use="my:area"/>`
4. `<xsl:param name="Area"/>`
5. `<xsl:key name="level" match="my:courses" use="my:level"/>`
6. `<xsl:param name="Level"/>`
7. `<xsl:key name="req_lev" match="my:courses" use="my:required_level"/>`
8. `<xsl:param name="Req_lev"/>`
9. `<xsl:key name="call_id" match="my:courses" use="my:call_number"/>`
10. `<xsl:param name="Call_id"/>`
11. `<xsl:key name="core_course" match="my:courses" use="my:core_course"/>`
12. `<xsl:param name="Core_course"/>`

13. `<xsl:template match="/">`
14. `<html>`
15. `<body background="/examples/images/parchment.gif"/>`
16. `<head>`
17. `<title>Course Information</title>`
18. `</head>`
19. `<body>`

20. `<xsl:variable name="s1">`
21. `<xsl:choose>`
22. `<xsl:when test="$Major">`

23. `<xsl:for-each select="key('major', $Major)">`
 24. `<xsl:value-of select="my:call_number"/>`
 25. `</xsl:for-each>`
 26. `</xsl:when>`
 27. `<xsl:otherwise>`
 28. `<xsl:for-each select="//my:courses">`
 29. `<xsl:value-of select="my:call_number"/>`
 30. `</xsl:for-each>`
 31. `</xsl:otherwise>`
 32. `</xsl:choose>`
 33. `</xsl:variable>`

34. `<xsl:variable name="s2"/>`
 35. `<xsl:variable name="s3"/>`
 36. `<xsl:variable name="s4"/>`
 37. `<xsl:variable name="s5"/>`
 38. `<xsl:variable name="s6"/>`
 39. `<h6 align="left">The courses matching your`
 40. `requirements</h6>`
 41. `<table width="680">`
 42. `<tr>`
 43. `<td>Call_ID</td>`
 44. `<td>Name</td>`
 45. `<td>Major</td>`
 46. `<td>Level</td>`
 47. `<td>Req_lev</td>`
 48. `<td>Credits</td>`
 49. `<td>Core_course</td>`
 50. `<td>Pre_requisite</td>`
 51. `</tr>`
 52. `
</br>`
 53. `<xsl:for-each select="//my:courses">`

```

54. <xsl:if test="contains($s1, my:call_number) and contains($s2,my:call_number)
    and
55. contains($s3,my:call_number) and contains($s4,my:call_number) and
    contains($s5,my:call_number) and
56. contains($s6,my:call_number)">
57. <tr>
58. <td><xsl:value-of select="my:call_number"/></td>
59. <td><xsl:value-of select="my:name"/></td>
60. <td><xsl:value-of select="my:major"/></td>
61. <td><xsl:value-of select="my:level"/></td>
62. <td><xsl:value-of select="my:required_level"/></td>
63. <td align="center"><xsl:value-of select="my:credits"/></td>
64. <td align="center"><xsl:value-of select="my:core_course"/></td>
65. <td align="center"><xsl:value-of select="my:prerequisite_course"/></td>
66. </tr>
67. </xsl:if>
68. </xsl:for-each>
69. </table>
70. <br><br></br></br>
71. <form method="POST">
72. <a href="http://localhost:8080/examples/servlet/FirstServlet"
    target="_blank"><u>Please enter the call
73. number if you want to check detailed course information: </u></a>
74. <INPUT type="text" maxLength="8" size="8" name="call_id"/>
75. </form>
76. </body>
77. </html>
78. </xsl:template>
79. </xsl:stylesheet>

```

Figure 20 The XSL to extract course information from XML file according to requirements and show in HTML page.

Lines 20 ~ 33 define variable S1, that is, the set of courses whose major is satisfying the search criteria: major. Lines 34 ~ 38 define variables s2 ~ s6, that is the set of courses satisfying the search criteria, Level, Credits, Call number, Required level and core course, respectively. We just omit the detailed definition here for briefness. Lines 53 ~ 56 selects the set of courses which are the intersection of s1 ~ s6, that is, the courses satisfying all the six search criteria.

Figure 21 shows the search results if the search criteria are CS major course and core course.

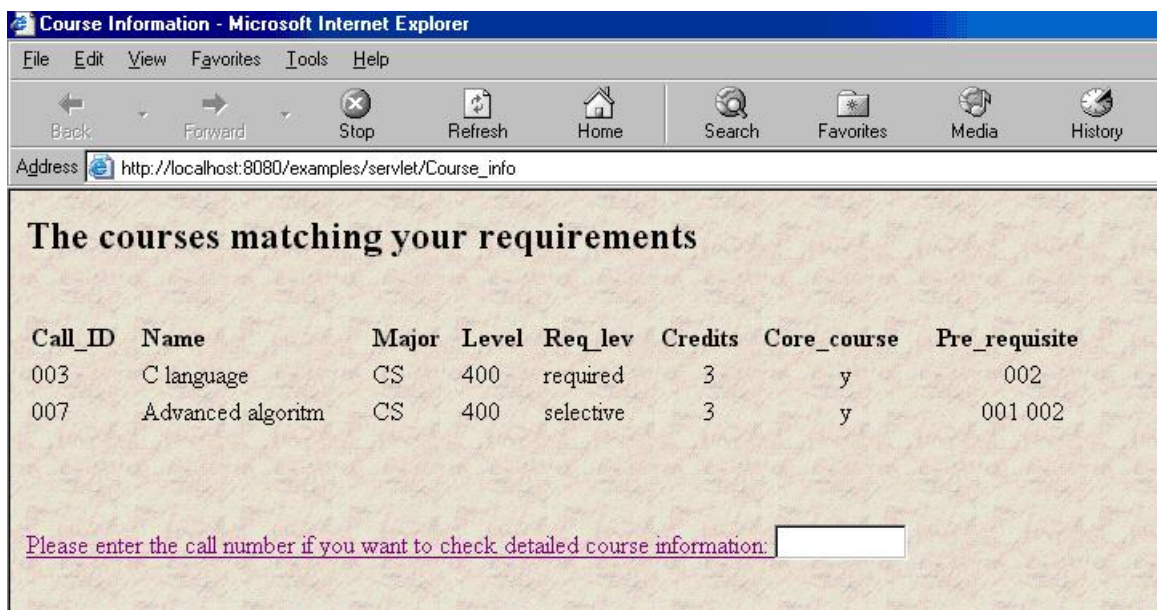


Figure 21 The course search results if search criteria is major="CS" and core_course="yes"

5.5 Statistics Function on Student's Record against Program

Rules

Course_stat servlet implements the function of doing statistic on student's record against his program rules. System will first access the *Majorreqs.xml* file to extract the rules of a specific major and save them in a text file. The *Course_stat Servlet* will process this information and set them as parameters when accessing the *studentsinfo.xml* file and comparing the student's past and current course information against these requirements. System shows the statistic results in a HTML page.

The Figure 22 shows the statistic results of Student A's record against the rules of his major XX. The upper frame of the page is showing the requirements of major XX, the lower frame of the page shows student A's course record against these rules.

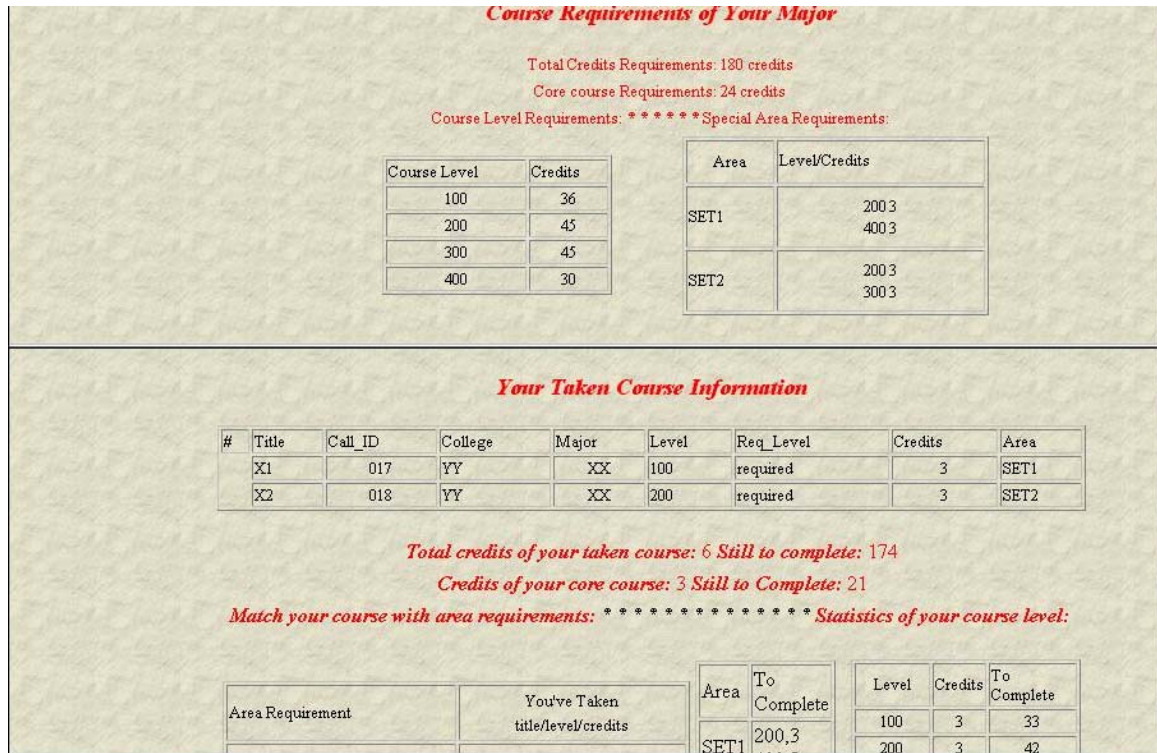


Figure 22 Rules of major XX and the statistic results of a student A's course record

5.6 Add/Drop Classes

Course_info servlet implements the function of add/drop courses. When in the Add/Drop process, the system not only needs to read in the add course list and drop course list from the HTML form and modify the student record, it also needs to commit a series of check to see if the user can validly add or drop these courses. In order to do this, the system needs first to get the student's course information including taken course and current course information, also the system needs to access the *courseinfo.xml* and get some information about the add-in courses, like prerequisite, schedule, etc. After all this validation, the system can successfully add/drop courses according to the lists and write back to the student record. The results of modifying student record (Add/Drop courses) can be immediately taken by the *course_stat* servlet and thus gives out the updated course

statistic results. We will attach the source Java code of Add/Drop course validation at the end of the thesis, the following XSL file shows how to take in the add/drop list and modify student record in the *studentsinfo.xml*.

1. ...
2. `<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>`
3. `<xsl:key name="mod" match="my:students" use="my:stud-ID"/>`
4. `<xsl:param name="modid"/>`
5. `<xsl:param name="c_course0"/>`
6. `<xsl:param name="c_course1"/>`
7. `<xsl:param name="c_course2"/>`
8. `<xsl:param name="c_course3"/>`
9. `<xsl:param name="c_course4"/>`
10. `<xsl:param name="c_course5"/>`
11. `<xsl:param name="c_course6"/>`
12. `<xsl:template match="/">`
13. `<xsl:variable name="tomod" select="key('mod', $modid)"/>`
14. `<studentsinformation>`

15. `<xsl:for-each select="//my:students">`
16. `<xsl:if test="my:stud-ID != $modid">`
17. `<xsl:copy-of select="."/>`
18. `</xsl:if>`
19. `</xsl:for-each>`

20. `<students xmlns="http://collegestudents.com/namespace"`
21. `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">`
22. `<name><xsl:value-of select="$tomod/my:name"/></name>`
23. `<college><xsl:value-of select="$tomod/my:college"/></college>`
24. `<major><xsl:value-of select="$tomod/my:major"/></major>`
25. `<stud-ID><xsl:value-of select="$tomod/my:stud-ID"/></stud-ID>`
26. `<password><xsl:value-of select="$tomod/my:password"/></password>`


```

27. <xsl:for-each select="$tomod/my:takencourses">
28. <takencourses><xsl:value-of select="."/></takencourses>
29. </xsl:for-each>
30. <xsl:if test="$c_course0">
31. <currentcourses><xsl:value-of select="$c_course0"/></currentcourses>
32. </xsl:if>
33. <xsl:if test="$c_course1">
34. <currentcourses><xsl:value-of select="$c_course1"/></currentcourses>
35. </xsl:if>
36. <xsl:if test="$c_course2">
37. <currentcourses><xsl:value-of select="$c_course2"/></currentcourses>
38. </xsl:if>
39. <xsl:if test="$c_course3">
40. <currentcourses><xsl:value-of select="$c_course3"/></currentcourses>
41. </xsl:if>
42. <xsl:if test="$c_course4">
43. <currentcourses><xsl:value-of select="$c_course4"/></currentcourses>
44. </xsl:if>
45. <xsl:if test="$c_course5">
46. <currentcourses><xsl:value-of select="$c_course5"/></currentcourses>
47. </xsl:if>
48. <xsl:if test="$c_course6">
49. <currentcourses><xsl:value-of select="$c_course6"/></currentcourses>
50. </xsl:if>
51. </students>
52. </studentsinformation>
53. </xsl:template>
54. </xsl:stylesheet>

```

Figure 23 The XSL file taking in the Add/Drop course list and modifying student record in the XML file

In Figure 23, Lines 15 ~ 19 copy every students record except the one who just modified his registration courses (the student who just submit the registration form) to the target XML file, Lines 22 ~ 50 write the current record of that student (the student who just submit the registration form and has updated current course list) to the target XML file.

The Figure 24 and Figure 25 show the results of Add/Drop classes. When we tried to add in Course 016, the system gives out an error message saying this course not able to be added because it has a prerequisite 014 that the student has not completed yet. Figure 25 shows the results of successfully adding courses 012 and 014, the statistical results has also been updated accordingly.



Figure 24 The error message of Add/Drop course failure because of not fulfilling prerequisite

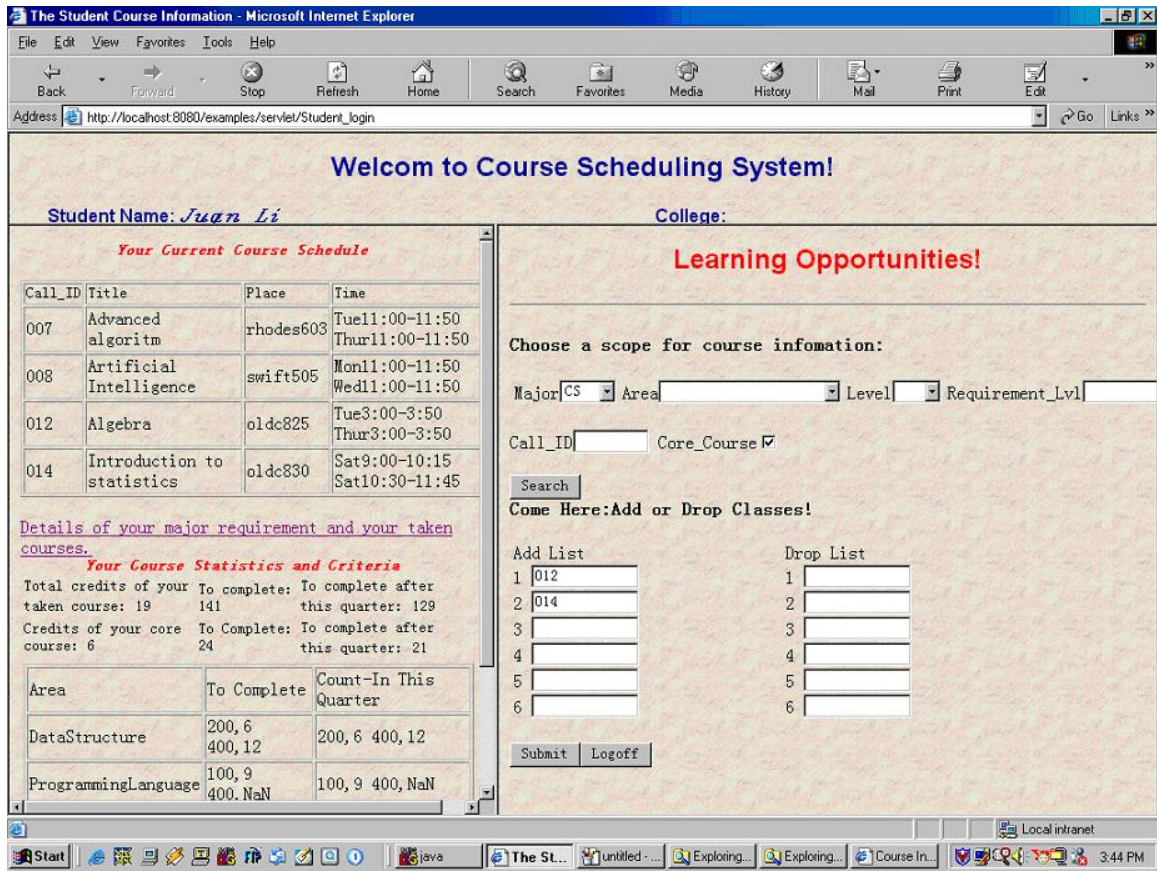


Figure 25 The result of successfully adding course 012 and 014, the statistic results is updated accordingly

5.7 Session Tracking among Different Servlets and Web Pages

During course registration, the user may go through several web pages to finish the process, a student may check his major requirements, his detailed course statistic data to make a decision. Even the same web page needs several servlets work together to get the result. The system should be able to keep persistent data. That is, to remember information from one request to another and to let different servlets share data. We use session tracking to do this.

Once the student login into the system, his personal information like name, major, past course information and current course information should be saved by the system as session attribute, then outside of this login servlet, these attributes can also be accessed. The following Java codes show the process of setting and retrieving session attributes among different servlets.

```
...
String Name=is.readLine().trim();
String College=is.readLine().trim();
String Major=is.readLine().trim();
Vector t_courses = new Vector(); //taken courses
Vector c_courses = new Vector(); //current courses
StringTokenizer tt = new StringTokenizer(is.readLine().trim(), ",");
    while (tt.hasMoreTokens())
{
    t_courses.add(tt.nextToken());
}
StringTokenizer tc = new StringTokenizer(is.readLine().trim(), ",");
    while (tc.hasMoreTokens())
{
    c_courses.add(tc.nextToken());
}

session.setAttribute("studname", Name);
session.setAttribute("college", College);
session.setAttribute("major", Major);
session.setAttribute("takencourses",t_courses);
session.setAttribute("currentcourses",c_courses);
...

```

Figure 26 Java codes of setting session attributes

...

```
else if(req.getParameter("action").equals("Submit"))
{
    HttpSession session= req.getSession(true);
    Vector C_course=(Vector)session.getAttribute("currentcourses");
    Vector P_course=(Vector)session.getAttribute("takencourses");
    ...
}
...
```

Figure 27 Java codes of retrieving session attributes

If in any of the servlets, the value of session attribute is changed, we need to update the session attribute by resetting the same attribute.

6 Conclusion and Future Work

We have presented an XML-based solution to the course registration problem. The course registration system assembles the course information, student information and program rules together in the registration process. The system has the following features:

- It has statistic function which can do comprehensive statistics on the students past and current course information against their major rules, thus to give the student some idea what kind of courses they need to fulfill their major requirements.
- It has search function which helps the user to find course information according to their preference.
- In the registration process, it has the check function to avoid invalid adding or dropping courses.
- It has modify_course function which allows administration to modify the cause offering or add new course into the system.
- It has modify_major function which allows the administration to modify the requirements of a program or add rules of a new program into the system.

Compared with the other course registration system, this system gives users accurate information about course offerings, improved registration process not only able to validate student's course registration but also gives them information to help in decision making. The system also offers administration the interface to modify course and program rules. The XML language used to implement the student and course information,

the rules of a major, and the XSL language used to transform and exchange data give system the ability of flexible data modeling and processing.

During the progress of this project, several interesting issues come up that need further investigation. The most important and challenging one is to make the system more intelligent. Currently, the system can only give out statistical results to help decision making, if we can build an agent able to use this result and take in all the rules of a major to give out advice in registration, the system will be more powerful and convenient.

Another problem is the storage of the data in XML format. Traditional relational database system is not suitable for XML data. Some middleware have been developed to transfer data between XML documents and relational databases[14]. Like XML-DBMS, it maps the XML document to the database according to an object-relational mapping in which element types are generally viewed as classes and attributes[15], user needs to use XML-based mapping language to specify customize this mapping. That means XML data needs to be transformed before they are written to the database and data extracted from the database needs to be transformed back to XML, obviously it adds complexity to data manipulation, which is the most advantageous attribute of XML data model. So we expect a matured object oriented database system convenient to store XML data.

Bibliography

- [1] Summary Results of Registration Process Review, University of South California, Nov. 2003. http://registrar.sc.edu/html/news/reg_proc_rec.pdf
- [2] Online Enrollment and Administration System. K. Kit, C. Mo, L. Yeung & L. Hong, Hongkong University of Science and Technology. <http://www.cs.ust.hk/faculty/fred/FYP/DB-development/Guidelines/sample-reports/db-proposal.pdf>.
- [3] Administrative and Course Management System Vendors Take Up the Challenge. <http://www.syllabus.com/article.asp?id=6389>
- [4] Tomcat User's Guide. <http://jakarta.apache.org/tomcat>
- [5] Developing Applications with Tomcat. <http://jakarta.apache.org/tomcat>
- [6] XML Tutorail. <http://www.w3schools.com/xml/default.asp>
- [7] XML by Example. Benoit Marchal QUE Publishing House, December, 1999
- [8] Document Object Model (DOM) & Simple API for XML (SAX). <http://www.perfectxml.com/domsax.asp>
- [9] XPath Tutorial. <http://www.zvon.org/xxl/XPathTutorial/General/examples.html>
- [10] XSL Tutorial. <http://www.vbxml.com/xsl/tutorials/intro/default.asp>
- [11] Java API for XML Parsing 1.1.1. <http://java.sun.com/xml/jaxp/dist/1.1/docs/api/index.html>
- [12] Xalan Java Version 2.6.0. <http://xml.apache.org/xalan/index.html>

[13] Xalan-Java overview. <http://xml.apache.org/xalan-j/overview.html>

[14] XML Database Products: Middleware.

<http://www.rpbouret.com/xml/ProdsMiddleware.htm>

[15] XML-DBMS Middleware for Transferring Data between XML Documents and

Relational Databases. <http://www.rpbouret.com/xmldbms/>