Table of Contents:

~~ Part One ~~

I. Abstract

In this report, I will be presenting a detailed description of the Universal Multimedia Access Interface (UMAI). The description includes the tools that were used to make this interface possible as well as explanations and snapshots of the interface. The complete list of files used in this project is presented at the end of the report, in Appendix B.

Moreover, the appendices include other work that was completed during this training period. Appendix A presents a review on various devices and their parameters. Appendix C includes weekly progress reports on what work I was engaged in at the time. Finally, Appendix D includes a research report I have completed on Ambient Intelligence.

II. Introduction

This report begins by introducing the environment of the interface, i.e. the tools that were used to make the interface dynamic. Section III briefly describes the Apache Tomcat Server used, the JavaServer Pages (JSP) which incorporate HTML syntax and java code, as well as the MySQL database for storing all the data.
Section IV gives an overview of what the project is and its purpose, explaining the interface in detail by providing snapshots of all aspects of the interface. Throughout the description, I have included parts of code to explain how certain methods are accomplished (i.e. saving data from web page to web page). This should facilitate understanding how it is done.
Finally, the complete list of files used is included at the end of the report in Appendix B.

III. Setting the Environment

A. Apache Tomcat

The Tomcat server is an open-source Java based web application container that was created to run Servlets and JavaServer Pages (JSP) in web applications to create dynamic, interactive web sites. In the case of this project, Tomcat 4.1.30 was used [1, 3].

1. Running the Tomcat 4.0 Servlet/JSP Container

In order to install and run Tomcat 4.0, the following was done:

- Downloading and Installing a Java Development Kit (JDK) release from: http://java.sun.com/j2se/
- Setting an environment variable JAVA_HOME to the pathname of the directory into which the JDK release was installed [3, 4]

2. Starting Up Tomcat 4.0 via an Environment Variable

- Setting an environment variable CATALINA_HOME to the pathname of the directory in which Tomcat 4.0 was installed
- Executing the shell command:

```
$CATALINA_HOME/bin/startup.sh          (Unix)
```

- After startup, the default web applications included with Tomcat 4.0 were available at: http://wwwtest.dsp.toronto.edu:8080/

3. Shutting Down Tomcat 4.0 via an Environment Variable

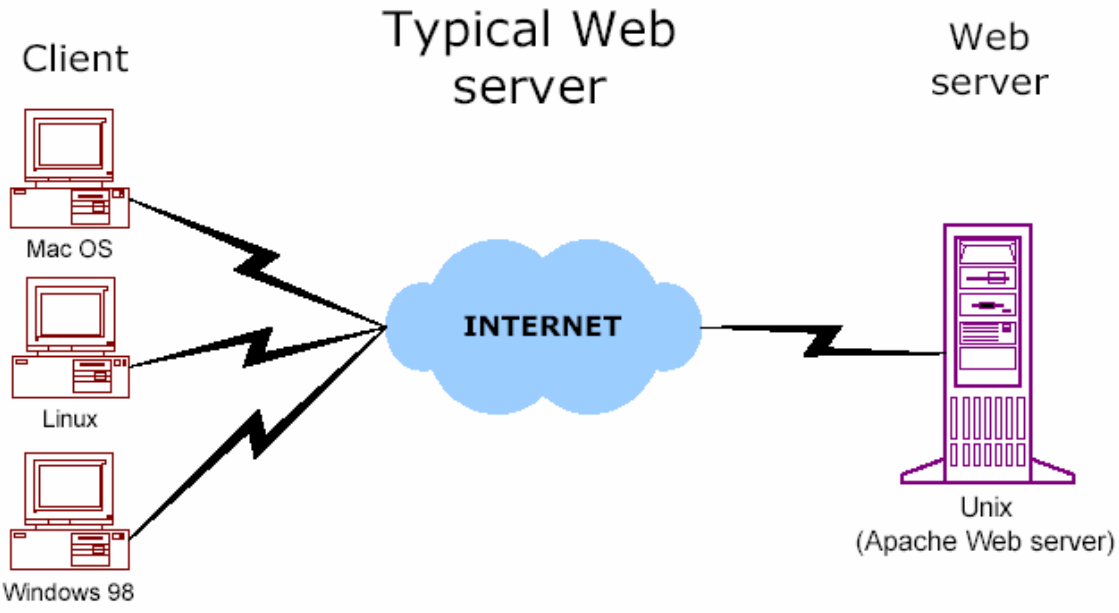
- Setting an environment variable CATALINA_HOME to the pathname of the directory into which Tomcat 4.0 was installed [1]
- Executing the shell command:

```
$CATALINA_HOME/bin/shutdown.sh          (Unix)
```

B. JavaServer Pages (JSP)

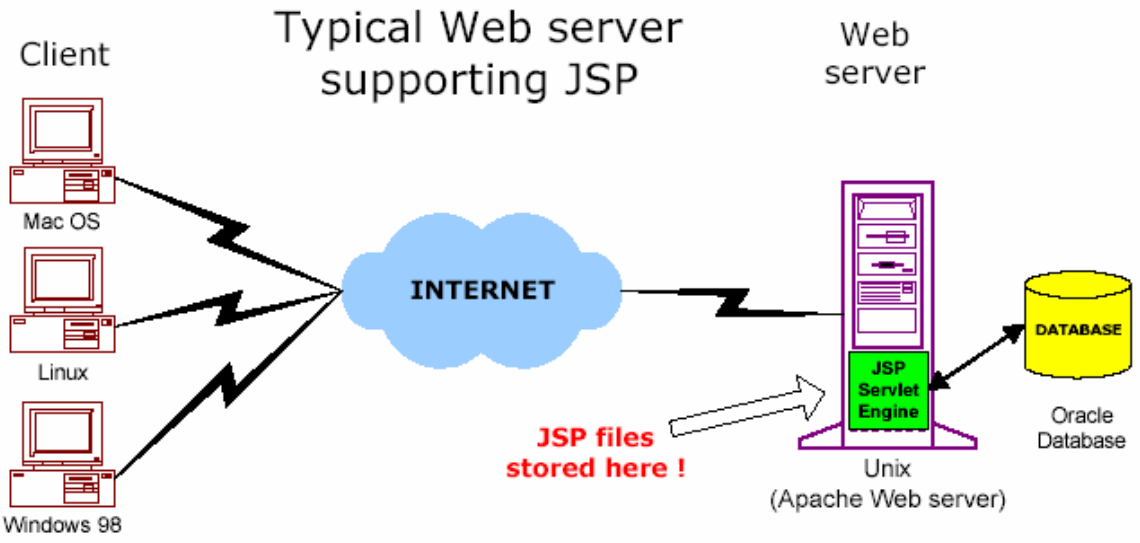
1. Overview

JavaServer Pages (JSP) is a scripting language based on Java for developing dynamic Web pages and allowing server side development. JSP files are HTML files with special tags containing Java source code that provides the dynamic content [1, 6].



The following shows the Apache web server running on Unix, with different clients connecting to the server via the Internet

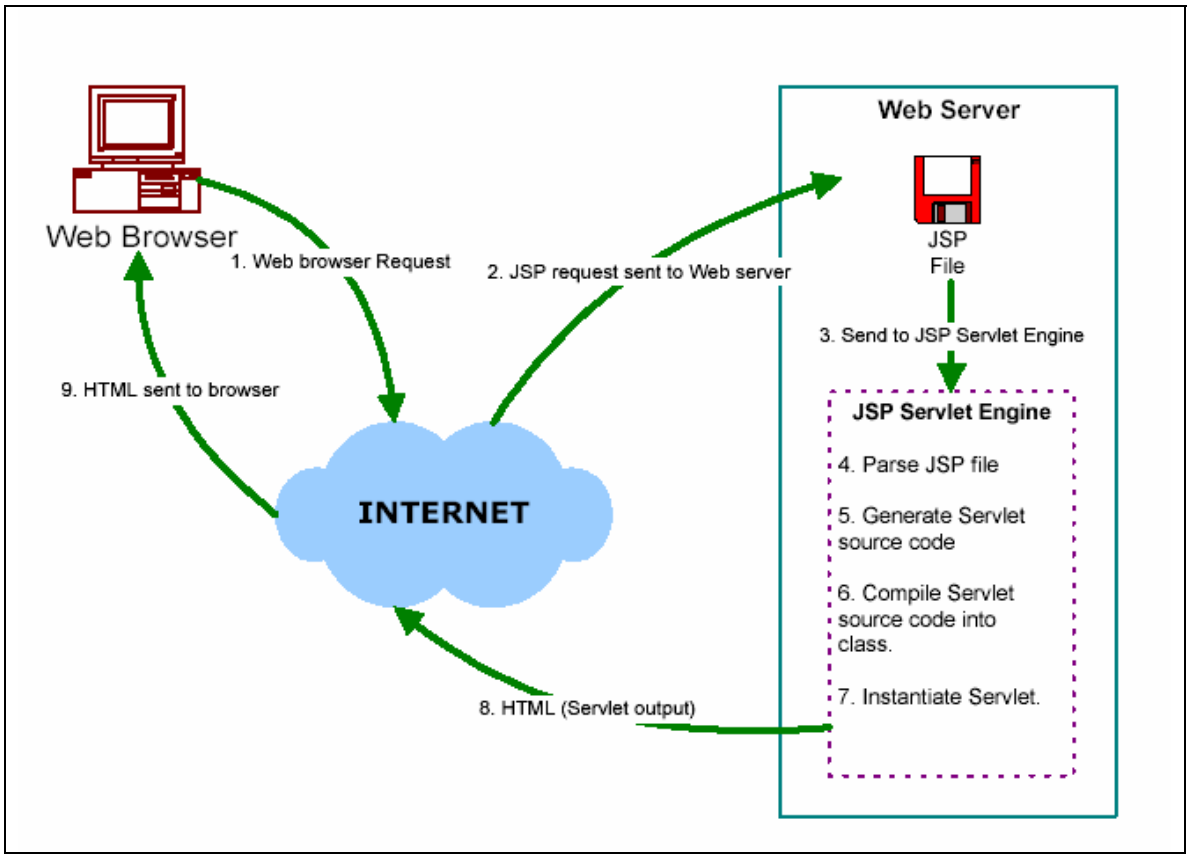
Java can be used to write applications that provide dynamic content using data from database requests, which is what is needed for this particular project [2, 5].



The JSP Servlet Engine is connected to a database and supports JSP files. JSP source code runs on the web server in the JSP Servlet Engine which dynamically generates the HTML and sends the HTML output to the client's web browser

2. JSP Architecture

As it was mentioned above, JSPs are HTML pages with special JSP tags embedded within them that contain Java code. The file extension is .jsp and the JSP engine parses the .jsp file and creates a Java servlet source file. It then compiles the source file into a class file, as shown in the diagram below.



3. Setting Up the JSP Environment

- Downloading the JSP Environment from: http://java.sun.com/products/jsp/download.html
- Placing the project directory in the "webapps/ROOT" directory under the installed Tomcat directory. To view the executed file, the browser address becomes: http://wwwtest.dsp.toronto.edu:8080/JSPFile.jsp

4. JSP Tags Used

- A declaration tag allows the declaration of variables and methods. It is presented as follows: **<%! …; %>**
- An expression tag allows the embedded Java expression to be displayed. It is presented as follows: **<%= … %>**
- A JSP directive tag allows for files and libraries to be included for processing by the JSP Servlet Engine. It is presented as follows: **<%@ directive … %>**. One of the page directive attributes used in the project is:

Language	Which Language the File Uses	<%@ page language = "java" %>
Import	Import all classes in a java package into the JSP page.	<%@ page import ="java.util.*" %>

- A scriptlet tag encloses any valid Java code and is called a scriptlet. It is presented as follows: **<% ... %>** [2]

5. Implicit Objects

- A request object is normally used in looking up parameter values and storing them from web page to web page. It is presented as follows [2]: **<% String Str = request.getParameter("Name"); %>**. This code snippet stores the parameter *Name* in the string *Str*.

C. MySQL Database

After mentioning some information about the Tomcat server and the JavaServer Pages (JSP) used in this project, a little time will be devoted to the MySQL database used. The database essentially contains all the data needed in the project and this information is saved in tables. To access the information, a connection has to be made to the database. A connection session includes the SQL statements that are executed and the results that are returned over that connection [1].
To establish a connection with a database using Java code in the JSP file, a call has to be made to the method DriverManager.getConnection. This method takes a string containing a URL. The DriverManager class, referred to as the Java Database Connectivity (JDBC) management layer, tries to locate a driver that can connect to the database represented by that URL. The Driver method connect uses this URL to actually establish the connection. In this specific project, I was making a connection with the "grp2" database where all the data has been saved. The following Java code establishes the connection [1]:

```
<%
Connection conn = null;
try {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        //Below is the URL used for connection
        conn=DriverManager.getConnection("jdbc:mysql://localhost.localdomain
        /grp2?user=grp2&password=grp2");
      }
//In case of exceptions or errors, a message will be displayed
catch (SQLException ex) {
            out.print("SQLException: " + ex.getMessage());
            out.print("SQLState: " + ex.getSQLState());
}
%>
```

IV. The Project

A. Overview

The tools mentioned above, namely the Tomcat Server, JavaServer Pages, and MySQL were used in the implementation of the "Universal Multimedia Access Interface" (UMAI) project. The purpose of this project is to allow users, mainly students, to search for courses and view lecture slides and other media types like video. This provides the students with the ability to catch up on missed classes, for example, and view lecture videos online without missing out on in-class explanations.
For the purpose of this project, the courses are saved in a table in the MySQL database. The table includes information about the course, such as course title, professor, department, university, course number, and the media type(s) associated with it (slides, video, etc…).
The powerful tool of this project is that the student can log in and look for courses in various universities and different departments. This means that this interface is not only limited to one university offerings but is stretched out over a wide set of available universities. Moreover, the student can look up courses offered by specific teachers or simply by course name/number. This search will yield a table with the possible matches. From here, the student is free to select the course that he/she is looking for or just browse for what is available.
Basically, what the "Universal Multimedia Access Interface" project provides is a means for each student to access the media that is provided by courses and be able to download it and see it.

B. Detailed Description

As mentioned earlier, the working directory of this project is saved in the "webapps/ROOT" folder of the installed Tomcat server. In the "ROOT" folder, I created my working directory and named it "interface". Within this folder are all the files and pictures that are associated with the interface.
To begin, open a web page and type the following URL to access the interface:

http://wwwtest.dsp.toronto.edu:8080/interface/login.jsp

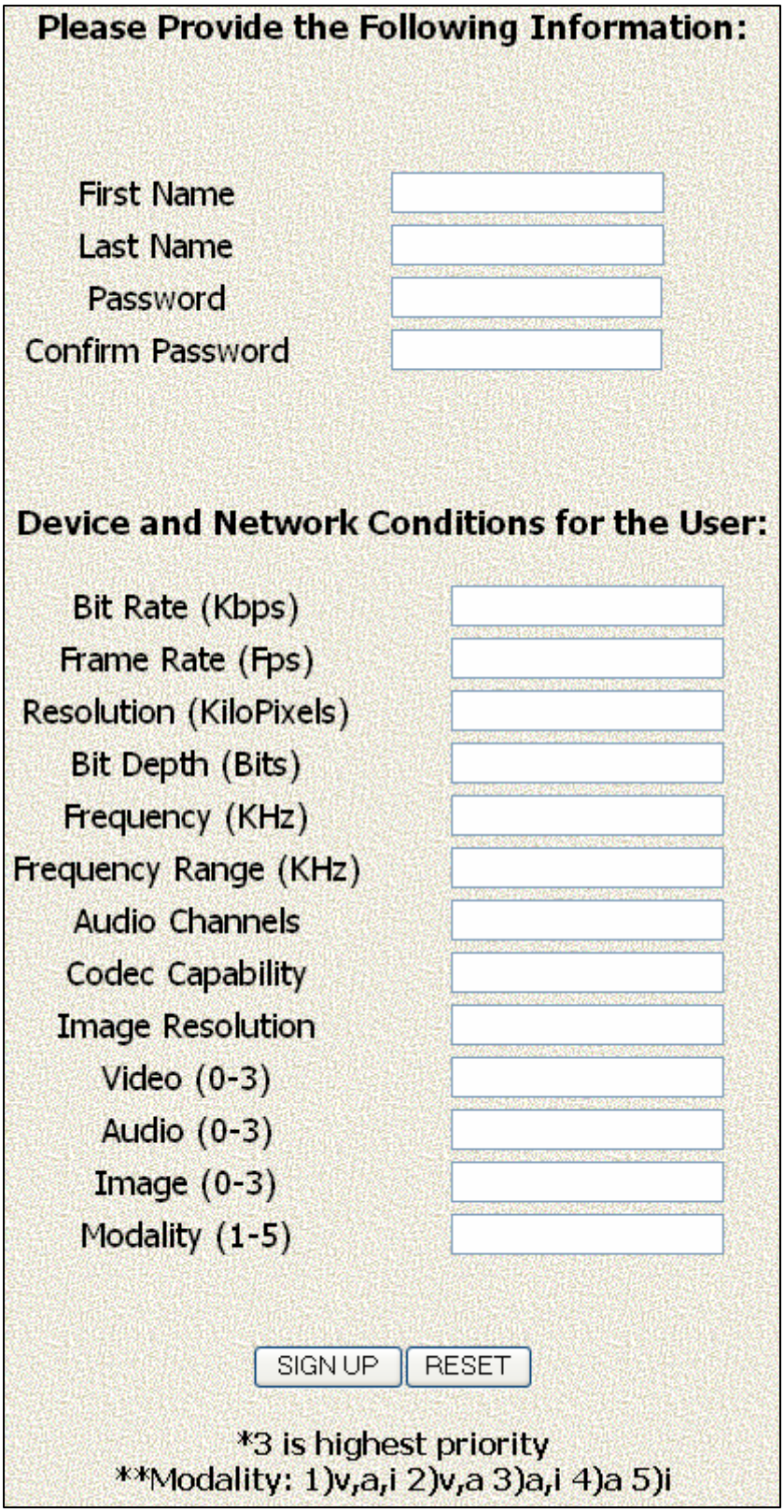
As shown above, I specify the port number "8080" and my folder "interface" where I saved my information. To access the login page, the above URL is required.

- The first name, last name, and password are required fields that must be filled in to enter the site. If the student has already registered, then he/she just has to fill in those 3 fields and click "LOGIN" to enter
- A new student user should click "NEW USER" to register
- "RESET" simply erases what has been typed in the 3 fields

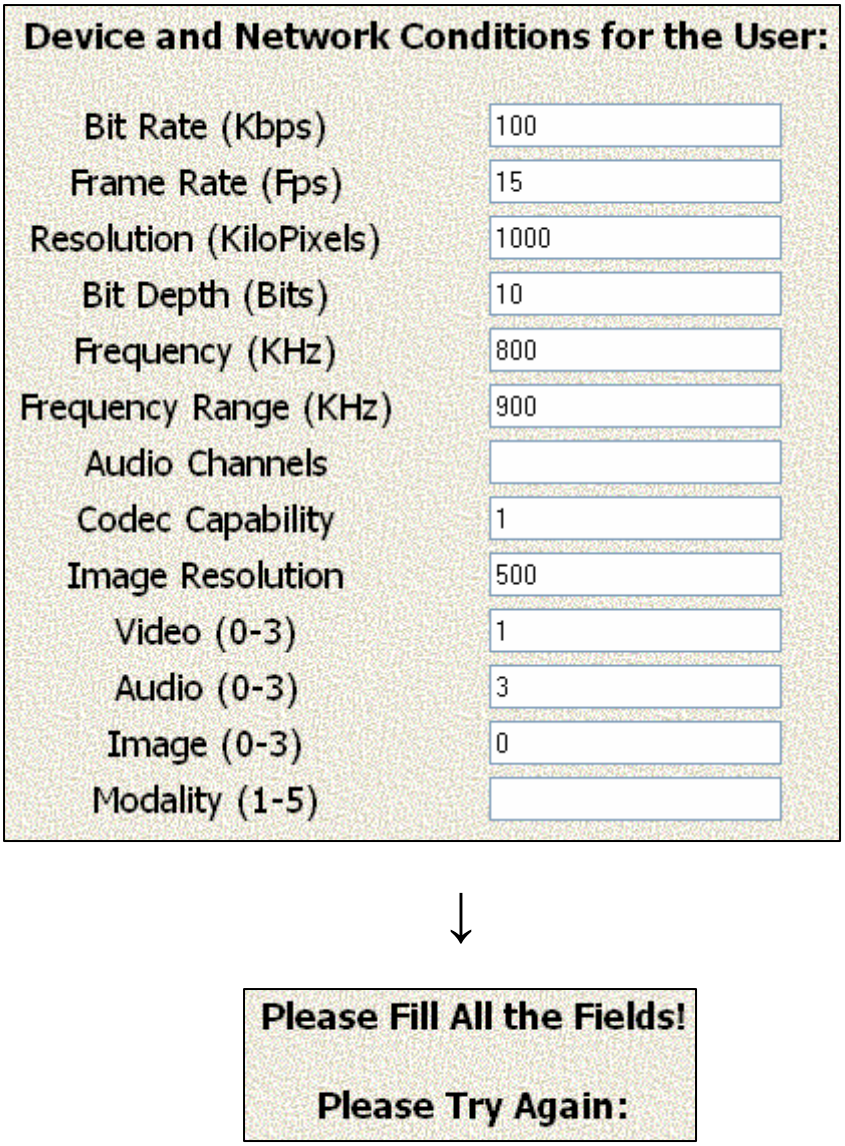
Clicking on either "NEW USER" or "LOGIN" loads the "main.jsp" page. The following code scriptlet allows this to happen:

```
<form action="main.jsp" method=post>
<input type="submit" name="submit" value="LOGIN">
<input type="submit" name="submit" value="NEW USER">
<input type ="reset" value ="RESET">
```

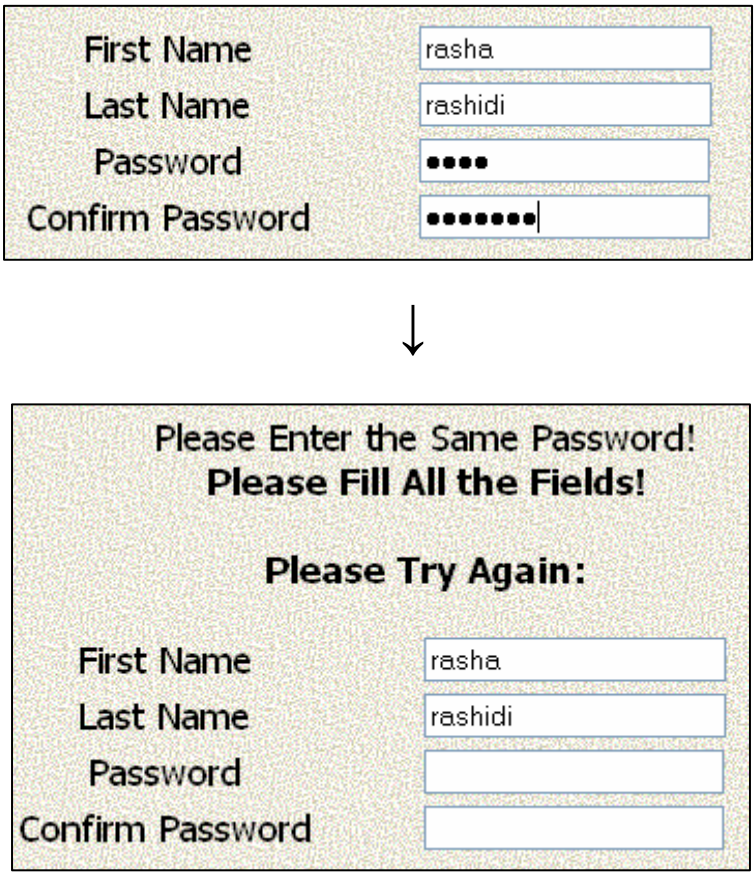
By clicking on "NEW USER" the URL now becomes:
http://wwwtest.dsp.toronto.edu:8080/interface/main.jsp



In this page, the user is requested to enter his/her first name, last name, and a unique password. Following, the user must specify values for his/her device and network conditions. Those values are important when the user later downloads or views media online. Once the values are entered, they will be stored. Therefore, once he/she wants to download some media, the server will check the user's set of values and compare them against a fixed set of "sheets" with standard values for the same parameters. Hence, the closest sheet to the user's sheet will be selected and the user will be provided with that set of conditions for downloading media or viewing it online.
Once the set of parameters are provided, the user can click on "SIGN UP" to register or click on "RESET" to fill out this sheet again.



As shown above, if some parameters were left empty, the user will be requested to refill all the fields. On the other hand, even if all the parameters were filled, if the "password" and "confirm password" fields were not matching, the user will be requested to fill the fields again.



Once all the fields are filled, and considering that the "password" and "confirm password" fields match, the user will be registered in the database.
The code that was utilized to add the new user to the database tables is show below. As it can be seen, the first step is to make sure that the user does not already exist and this is why a MySQL statement is first executed to check that the user is not listed. If no matches of this person were found, the next step would be to insert the new user, along with the parameter values he/she provided earlier, into two tables – *user* and *profile*. I will talk about the profile table shortly.

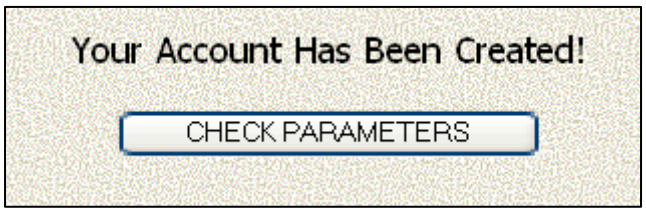
```
Statement st = conn1.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);

ResultSet rs = st.executeQuery("SELECT first_name, last_name, password FROM user where
first_name= '"+fname+"' AND last_name= '"+lname+"' AND password= '"+pass1+"' ");
int match=0;
while (rs.next())
        match=1;
if (match==0){

int update = st.executeUpdate("INSERT INTO user SET first_name= '"+fname+"', last_name=
'"+lname+"', password= '"+pass1+"', bitrate= '"+bitrate+"', framerate= '"+framerate+"', resolution=
'"+resolution+"', depth= '"+depth+"', freq= '"+freq+"', freq_range= '"+freq_range+"', channels=
```

```
'"+channels+"', codec_capability= '"+codec_capability+"', image_resolution= '"+image_res+"',
video= '"+video+"', audio= '"+audio+"', image= '"+image+"', modality1= '"+modality1+"' ");

int update2 = st.executeUpdate("INSERT INTO profile SET id=1, fname= '"+fname+"', lname=
'"+lname+"', lastfile1=0, lastfile2=0, lastfile3=0 ");

}
```

Below is a look into the MySQL table *user*, which displays correctly the added user: "rasha rashidi".

```
mysql> select * from user;
+---------+------------+-----------+----------+-----------+------------+---
------+
| bitrate | first_name | last_name | password | framerate | resolution | de
lityl |
+---------+------------+-----------+----------+-----------+------------+---
------+
|       7 | azadeh     | kushki    | maz      |         7 |          7 |
    7 |
|     100 | rasha      | rashidi   | rasha    |        15 |       1000 |
    2 |
+---------+------------+-----------+----------+-----------+------------+---
------+
2 rows in set (0.00 sec)
```

Once the user is saved successfully in the database, the following will be displayed on the web page:



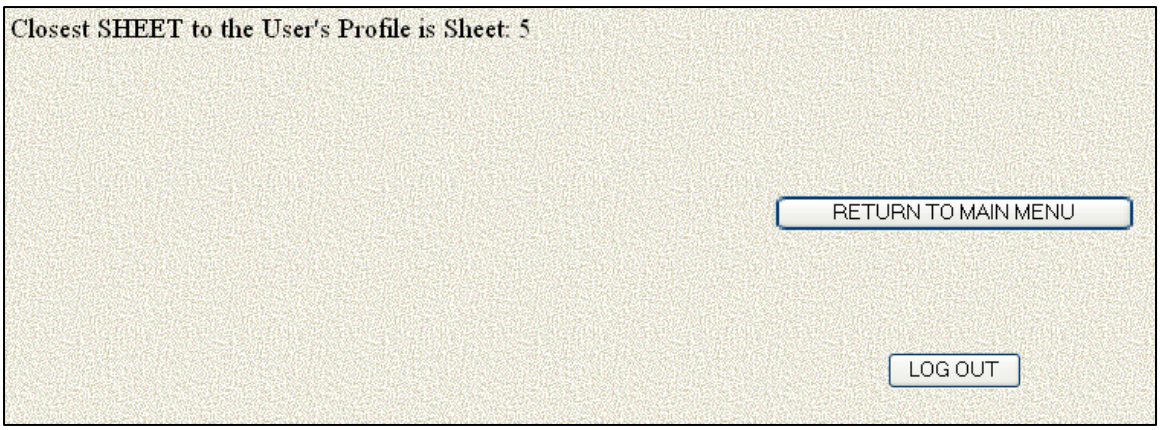
Earlier, I mentioned that the user is requested to provide values for his/her device and network conditions for the purpose of downloading media and watching it. The purpose of the user supplying this data is so that he/she will be provided with the best available download conditions. What this means is that once the user submits the values as it was done above, the server checks the user's submitted values against a set of standard "sheets" that are saved in the database. The sheets contain the same information (i.e. bit rate, frame rate, resolution…) but the values vary from sheet to sheet so as to cover as many types of devices as possible (the image resolution for a PDA may differ from that for a notebook, etc…)[1]. When the user clicks on "CHECK PARAMETERS", the server

[1] A Device Parameters Review can be found in Appendix A

performs a "filtering" algorithm[2] whereby the sheets are retrieved from the database and checked against the user's values. From here, one sheet is picked which corresponds most closely to the user's values and these values will be used when the user downloads media. I saved 18 sheets in the database, each with unique values that cover a wide range of devices. Below is a snapshot of the sheets saved in the *sheet* table.

```
mysql> select * from sheet;
+-----------+----------+---------+-----------+------------+----
| sheet_num | modality | bitrate | framerate | resolution | dep
+-----------+----------+---------+-----------+------------+----
|         1 |        1 |      50 |        15 |         12 |
|         2 |        3 |      50 |         0 |          0 |
|         3 |        4 |      50 |         0 |          0 |
|         4 |        5 |      50 |         0 |          0 |
|         5 |        2 |      50 |        15 |         12 |
|         6 |        1 |    1000 |        30 |         25 |
|         7 |        3 |    1000 |         0 |          0 |
|         8 |        4 |     128 |         0 |          0 |
|         9 |        5 |     128 |         0 |          0 |
|        10 |        1 |    1000 |        15 |         25 |
|        11 |        1 |    2000 |        30 |        101 |
|        12 |        1 |     500 |        15 |         12 |
|        13 |        1 |     256 |        15 |         12 |
|        14 |        1 |     128 |        15 |         12 |
|        15 |        2 |      50 |        15 |         12 |
|        16 |        3 |      50 |         0 |          0 |
|        17 |        2 |     512 |        15 |         12 |
|        18 |        3 |     250 |         0 |          0 |
+-----------+----------+---------+-----------+------------+----
18 rows in set (0.03 sec)
```

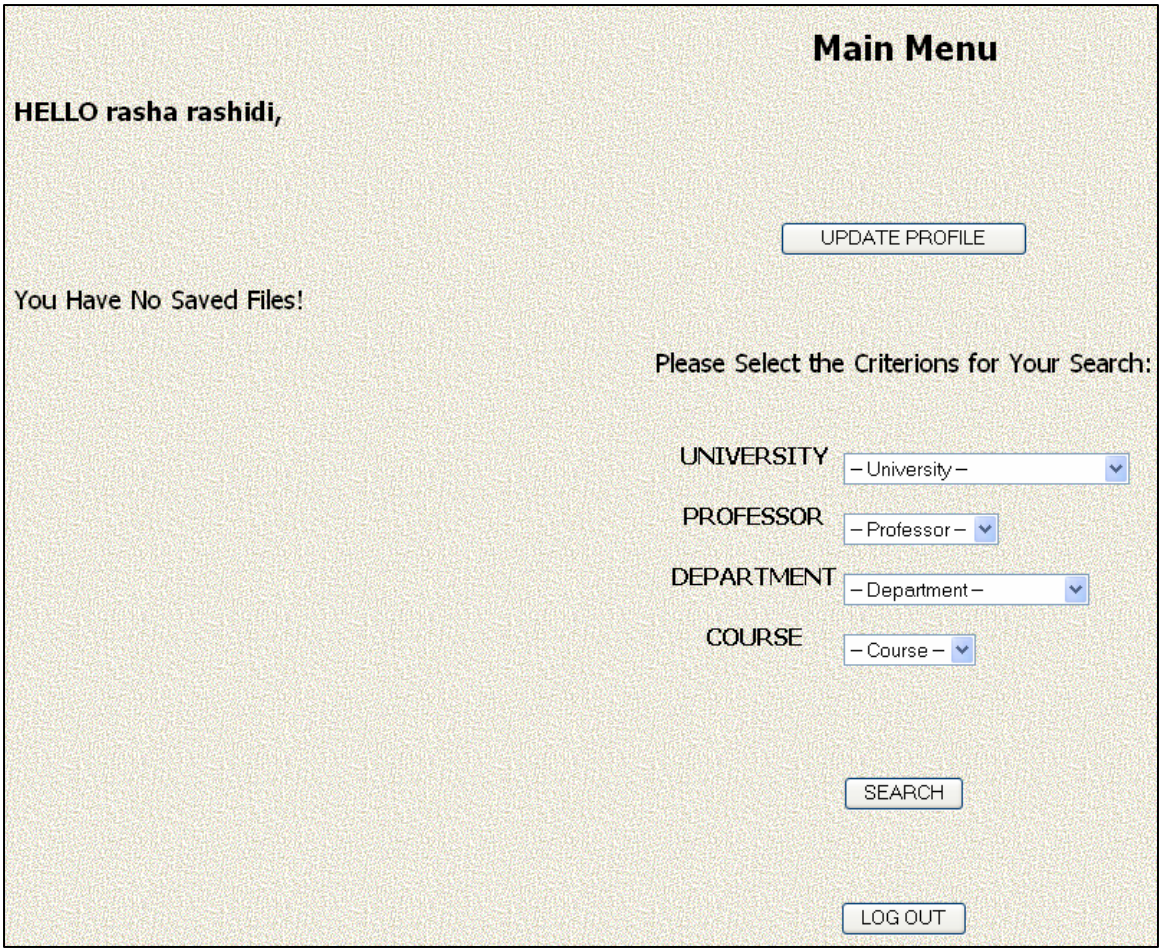
This is the result of the "filtering" algorithm:



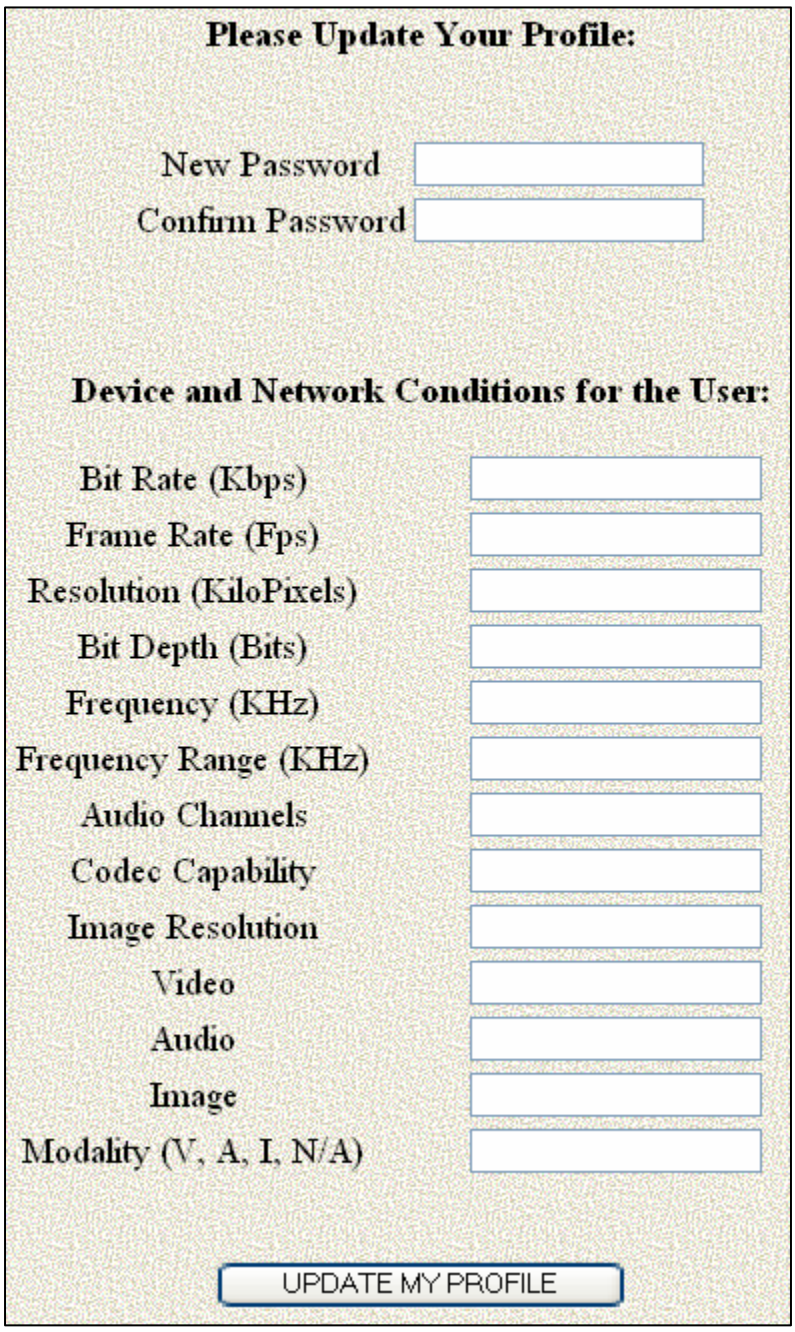
[2] The "filtering" algorithm can be found in the *params.jsp* file in Appendix B

The above shows that the parameter values of sheet 5 in the database were chosen as the download parameters when the user wants to download or view media.

By clicking on “RETURN TO MAIN MENU”, the user can view the main page, where search options and profile updates are possible (i.e. the user’s parameters).

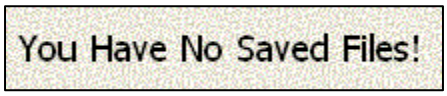


The user “rasha rashidi” can update her profile by selecting “UPDATE PROFILE”.



Here, the user may choose to change the password or even the device and network conditions.
After completing this sheet, the user clicks on "UPDATE MY PROFILE", whereby the new values replace the older ones and the user returns to the main menu.
Also note that none of the fields here should be empty and the "password" and "confirm password" fields must match.

Returning to the main menu, a note on the left hand side states that there were no saved files:



Being a new user, the student has not yet accessed or saved any course files yet. So, this message is valid as long as the user does not add course files. However, once the student starts adding files, he/she will be allowed to view the three latest saved files. Earlier in this report, I mentioned that when the new user is registered, his/her data is saved in two tables in the database – the *user* table as well as the *profile* table. This is where the *profile* table becomes useful. Once the user is registered, the data is also passed to the *profile* table. This table saves the last three added files by the user. Therefore, when the user logs on, the server searches for this user in the *profile* table and returns the last three files found. Since no more than three files are allowed to be saved, once the user saves three files, the next file to be saved will replace the oldest file in the table in a first-in-first-out (FIFO) manner.
The following code shows how this is done:

```
<%   …

else if (acc1!=0 && acc2!=0 && acc3!=0)                    //There are 3 files saved already
{
        file1=result.getInt(4);
        file2=result.getInt(5);

        //"load" here corresponds to the file that was selected by the user after the search was made
        result3 = add.executeUpdate("UPDATE profile set lastfile1='"+load+"'    where fname='"+first+"' AND lname='"+last+"' ");
         result4 = add.executeUpdate("UPDATE profile set lastfile2='"+file1+"'    where fname='"+first+"' AND lname='"+last+"' ");
         result5 = add.executeUpdate("UPDATE profile set lastfile3='"+file2+"' where fname='"+first+"' AND lname='"+last+"' ");
}
… %>
```

A snapshot of the *profile* table is shown below:

```
mysql> select * from profile;
+------+--------+---------+-----------+-----------+-----------+
| id   | fname  | lname   | lastfile1 | lastfile2 | lastfile3 |
+------+--------+---------+-----------+-----------+-----------+
|    1 | azadeh | kushki  |         0 |         0 |        26 |
|    1 | rasha  | rashidi |         0 |         0 |         0 |
+------+--------+---------+-----------+-----------+-----------+
2 rows in set (0.00 sec)
```

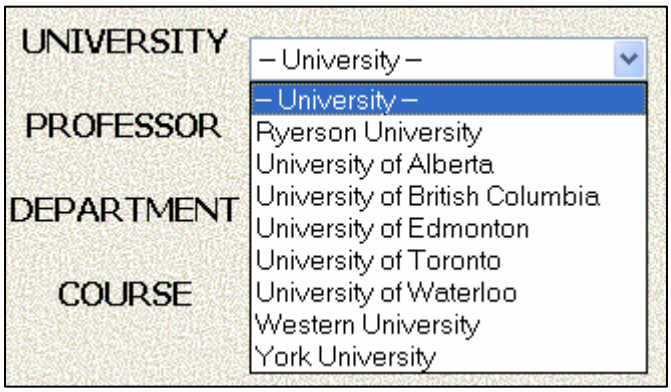
As it can be seen, the values under lastfile1, lastfile2, and lastfile3 are all integers. What those integers correspond to are the course file numbers. For example, the user "azadeh kushki" has accessed and saved only one file – file 26. This means that in the courses table (the table is called *metadata*), the 26th course listed has been saved for this user. Each course in the *metadata* table has an id number next to it that specifies the number of that course in the table. As more courses are added to the table, the id numbers will be the key to differentiate them.
To understand this better, below is a snapshot of part of the *metadata* table. The "id" field is used as the key to store files in the *profile* table:

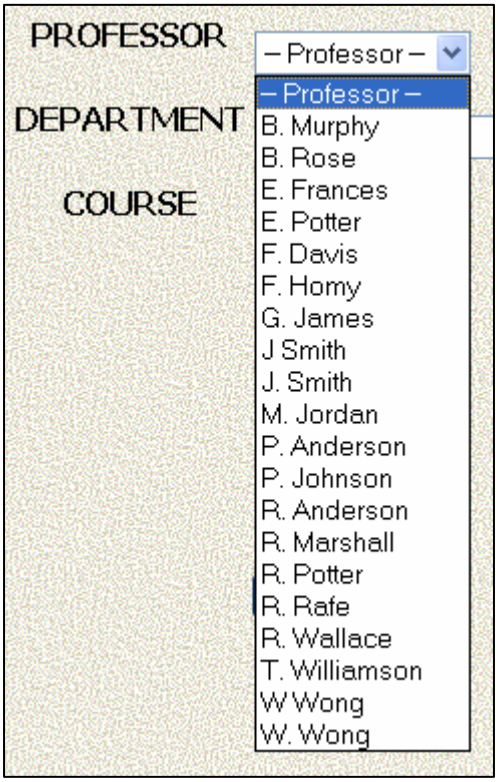
```
mysql> select * from metadata;
+----+---------------------------------------+---------------------------------------+
----------+------+------------+----------+-----------------------+
| id | title                                 | description                           |
quality   | size | format     | language | department            |
+----+---------------------------------------+---------------------------------------+
----------+------+------------+----------+-----------------------+
|  1 | ECE461 Internetworking Lecture 1      | Week one lectures updated Jan 19, 2004 |
bad       | NULL | powerpoint | English  | Computer Engineering  |
|  2 | CSC343 Introduction to SQL            | Week one lectures updated Jan 5, 2004  |
```

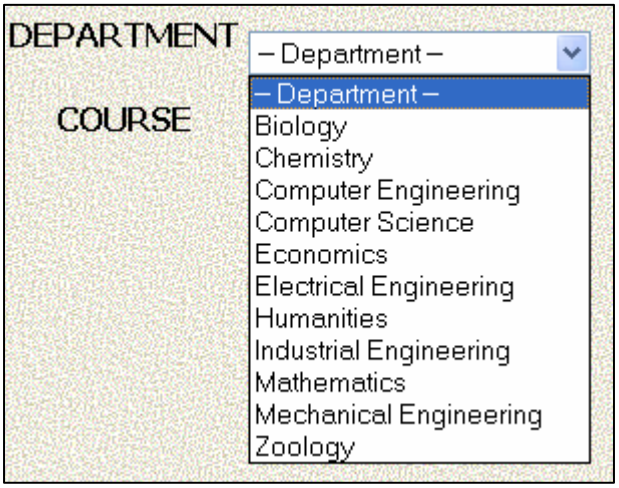
To add files to the user profile, a search has to be made and a course has to be selected. To perform the search, the following can be done is available:

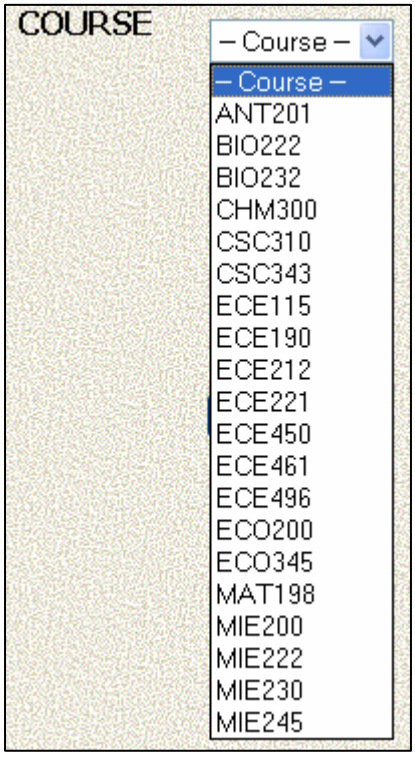
The search can be done by selecting one of the four fields, or any combination of all four and clicking "SEARCH".

The list of possible universities, departments, professors, and courses is shown below:









The data within each box is retrieved from the database of course by accessing the *metadata* table which includes information about all of the courses available and the data pertaining to the courses. For example, the university that the course is given in, the professor teaching the course, the title of the course, etc...

Therefore, the way that the above was able to be done was simply by calling MySQL statements that sort the table according to the field required – for example, university. Hence, the table would become sorted according to university and I could then output the distinct university name in a drop down box as shown here.

The code used for retrieving the university name and displaying it in a drop down menu is provided below. Similar code is used for the rest of the searches:

```
<% …

Statement sort1 = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);

ResultSet res1    = sort1.executeQuery("SELECT distinct university FROM metadata order by
university");
%>

<form method=post action="search.jsp">
<br>
<table align="center">
<tr><td><center><h4>UNIVERSITY</h4></center></td>
<form><td>
<select name="uni">
<option value= "UNI">-- University --
<%
            String university="";
            while (res1.next()) {
                        university=res1.getString(1);
%>
                         <option value="<% out.print(university); %>" > <%out.print(university);
<%
             }
%>
```

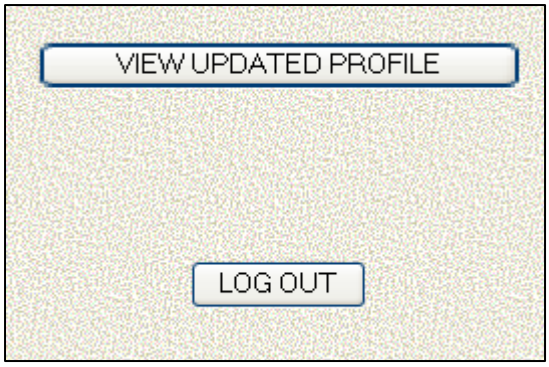
For demonstration purposes, I selected all the courses that are enlisted in the University of Toronto and clicked “SEARCH”:

	Course Code	Title	University	Professor	Department	Media Type
○	ECE461	ECE461 Internetworking Lecture 1	University of Toronto	J. Smith	Computer Engineering	Slides
○	CSC343	CSC343 Introduction to SQL	University of Toronto	R. Potter	Computer Science	Slides
○	MIE200	Dynamics - week 1	University of Toronto	R. Wallace	Mechanical Engineering	Audio
○	MIE200	Dynamics - week 2	University of Toronto	R. Wallace	Mechanical Engineering	Audio
○	MIE222	Mechanics of Solids 1-Lec 1-5	University of Toronto	W. Wong	Industrial Engineering	Slides
○	MIE222	Mechanics of Solids 1-Lec 6-10	University of Toronto	W. Wong	Industrial Engineering	Slides
○	MIE230	Engineering Analysis - Lec 1	University of Toronto	G. James	Industrial Engineering	Online Lecture
○	MIE230	Engineering Analysis - Lec 2	University of Toronto	G. James	Industrial Engineering	Online Lecture
○	MIE245	Differential Equations	University of Toronto	F. Davis	Mechanical Engineering	Slides
○	MIE245	Differential Equations	University of Toronto	F. Davis	Mechanical Engineering	Online Lecture
○	ECE496	Design Project - Introduction	University of Toronto	P. Anderson	Computer Engineering	Online Lecture
○	ECE496	Design Project - Writing Seminar	University of Toronto	P. Anderson	Computer Engineering	Online Lecture
○	CSC310	Information Theory Assignment 1	University of Toronto	R. Marshall	Computer Science	slides
○	ANT201	Evulotion of Man	University of Toronto	J Smith	Humanities	Slides
○	ECE450	Software Engineering	University of Toronto	F. Homy	Computer Engineering	Slides
○	CSC343	Database Review	University of Toronto	R. Rafe	Computer Science	slides

ADD FILE

As it can be seen, the search has listed the courses that can found in the University of Toronto. By selecting any one of those files and clicking “ADD FILE”, the course file number will be saved under the “lastfile3” field in table *profile*.

I have selected Software Engineering and the following is displayed:



Now, by clicking on “VIEW UPDATED PROFILE”, I will be able to see my main page with the selected file added to my profile:

Last Accessed File(s):

	COURSE CODE	TITLE	UNIVERSITY	DEPARTMENT	PROFESSOR	MEDIA TYPE
○	ECE450	Software Engineering	University of Toronto	Computer Engineering	F. Homy	Slides

DOWNLOAD

In the MySQL database, table *profile* table has been updated for the user “rasha rashidi”. As shown below, the id of the “Software Engineering” course has been saved in the “lastfile3” field so as to display the course file in the main page when “rasha rashidi” logs on.

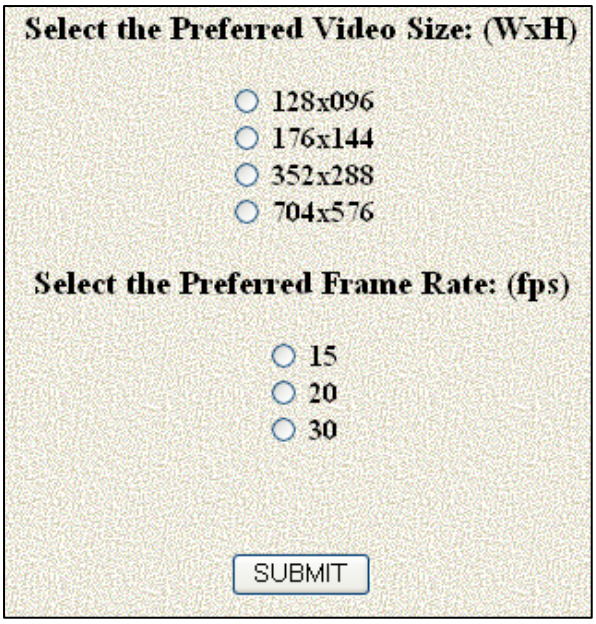
```
mysql> select * from profile;
+------+--------+---------+-----------+-----------+-----------+
| id   | fname  | lname   | lastfile1 | lastfile2 | lastfile3 |
+------+--------+---------+-----------+-----------+-----------+
|    1 | azadeh | kushki  |         0 |         0 |        26 |
|    1 | rasha  | rashidi |         0 |         0 |        29 |
+------+--------+---------+-----------+-----------+-----------+
2 rows in set (0.00 sec)
```

I will add more files for display:

	COURSE CODE	TITLE	UNIVERSITY	DEPARTMENT	PROFESSOR	MEDIA TYPE
○	MIE230	Engineering Analysis - Lec 1	University of Toronto	Industrial Engineering	G. James	Online Lecture
○	BIO232	Zoology - Introduction	University of British Columbia	Zoology	W Wong	Online Lecture
○	ECE450	Software Engineering	University of Toronto	Computer Engineering	F. Homy	Slides

DOWNLOAD

As you can see, I have added another two files the same way I added the first one. One of the fields above is called “MEDIA TYPE”. This means that the corresponding course is available for downloading and viewing in the media type specified. The course “Software Engineering” provides slides for download, whereas the other two courses provide the actual lecture online for viewing making the type a video.
From here, what can be done is to select one of the three files and click “DOWNLOAD” to be able to view the selected media. For this project, I have made a simple demo that plays a video. The video is not relevant to the course. However, it is just a demonstration of how the interface will be working in later stages to support the different types of media for each course. Upon clicking “DOWNLOAD”, the following is displayed:

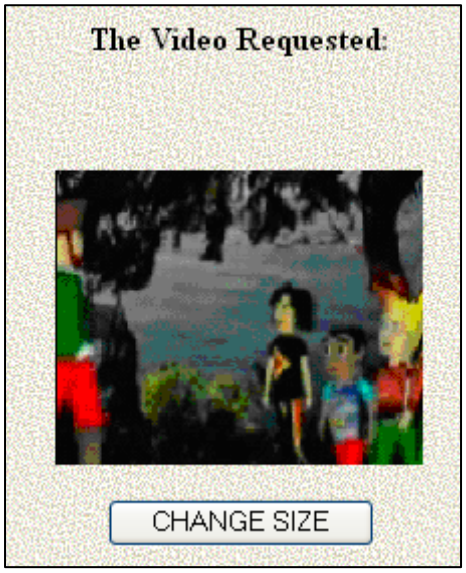


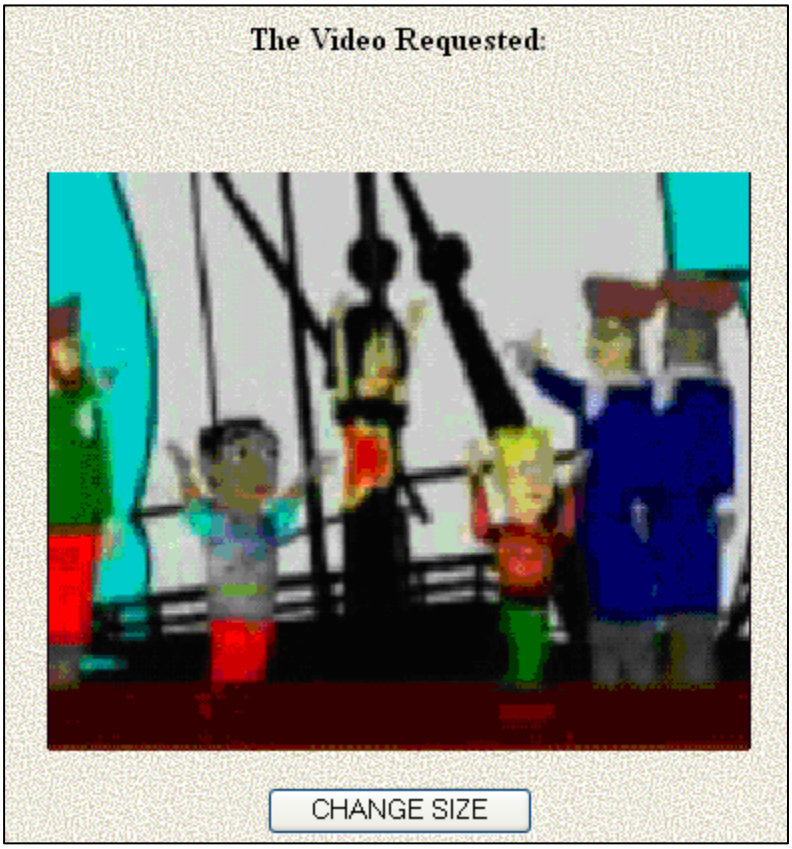
The video I have chosen to display is located in a directory different from my working directory. To view the video, I need to encode it first and parse it using the "ffmpeg" encoder. By selecting the preferred size and frame rate, those values will be embedded in the encoding command that will encode the video and play it afterwards. The code for the encoding is:

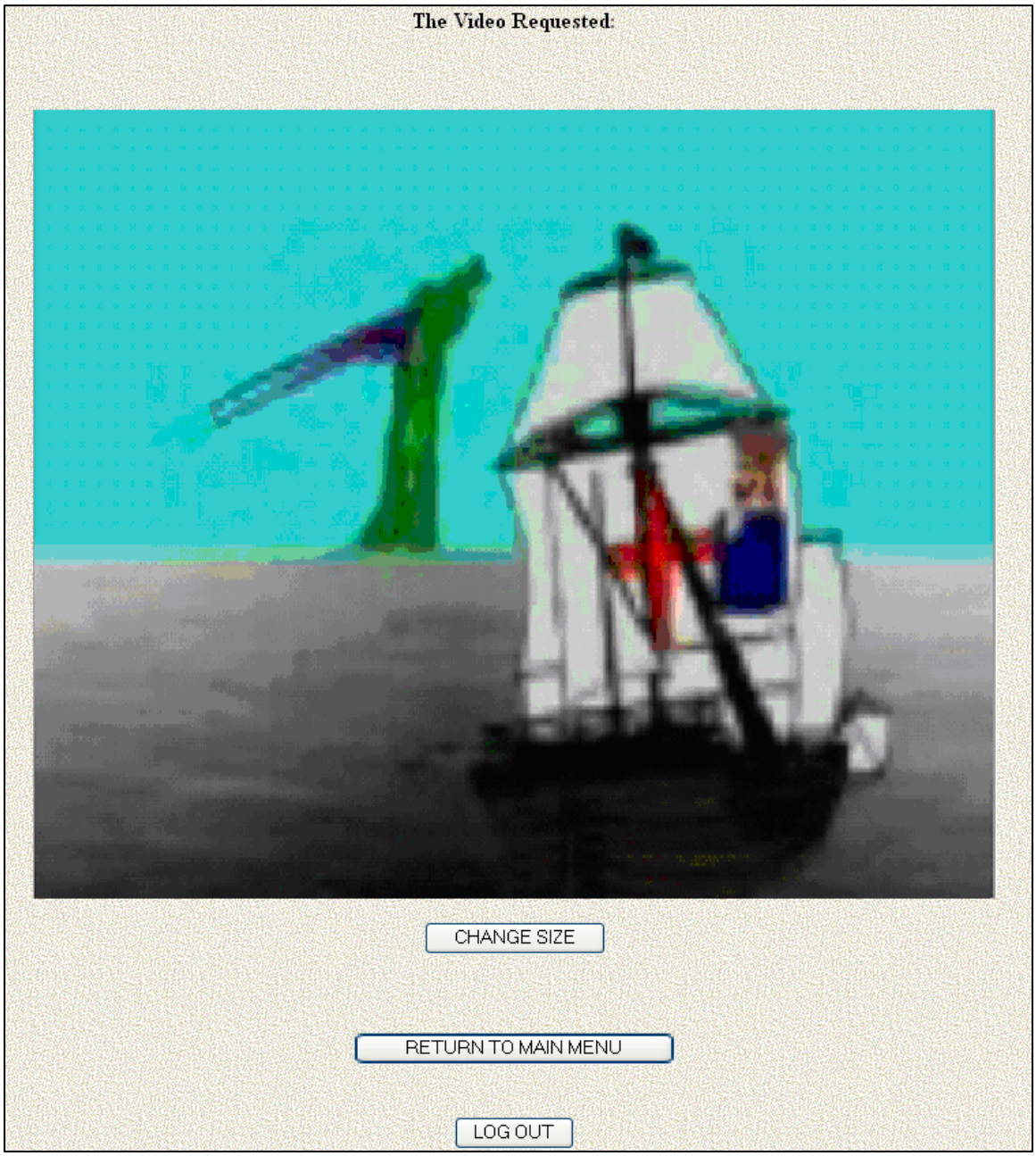
```
//Retrieving the size value the user selected
String size          = request.getParameter("size");
//Retrieving the frame rate value the user selected
String frame     = request.getParameter("frame");

String f                 = request.getParameter("f");
String l                 = request.getParameter("l");
String p                = request.getParameter("p");

boolean display = true;
//The encoding command
String[] cmd={"ffmpeg", "-i",
"/home/grp2/mazenDoNotTouch/mpeg4ip-1.0/ffmpeg-0.4.9-pre1/pirates.mpg", "-s", "sqcif",
"-r", "15", "/home/grp2/tomcat/webapps/ROOT/interface/videos/rasha2.mpg"};

cmd[4] = size;
cmd[6] = frame;
int ch;
String s,s1;
display = false;
Process p2 = Runtime.getRuntime().exec(cmd);
try{
        p2.waitFor();
}
catch (Exception e) {
            out.print("ERROR!");
}

BufferedReader stdInput = new BufferedReader(new InputStreamReader(p2.getInputStream()));
BufferedReader stdError = new BufferedReader(new InputStreamReader(p2.getErrorStream()));

try{
%>
        <center><b>Encoding the Video...</b></center>
<%
}

catch(IOException t) {
        t.printStackTrace();
}
%>              //Displaying the video on the web page
                  <center><img dynsrc="../interface/videos/rasha2.mpg"
                  width=<% out.print(w); %> height=<% out.print(h); %>
                  start="fileopen" loop="infinite"></center>
```

The above code encodes the video I have selected and outputs it to my directory to a folder I called “video”. The following is shown after I have selected the video size to be 128x96 pixels and the frame size to be15f/s:

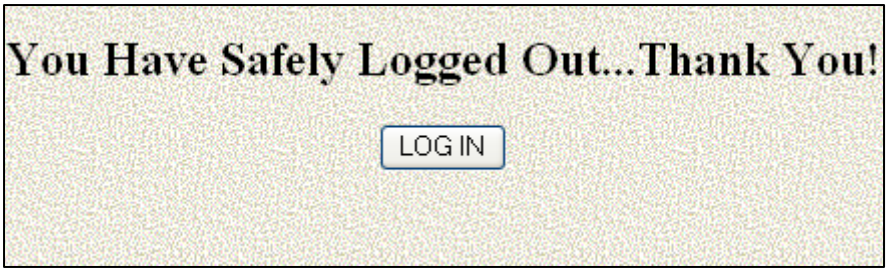
The video will play in the web page as shown below. If the user wishes to change the size of the video display, clicking on “CHANGE SIZE” will make this possible. There are four size options available and choosing each one in turn will yield the outputs shown here:







The user can either log out from this page or return to the main menu and perform more searches or updates to the user profile. Upon clicking “LOG OUT”, the following is displayed, whereby clicking on “LOGIN” again will load the first page – “login.jsp”:



C. Future Work

As it can be seen from the detailed description of this project, users are able to log into the site or register as new users. They can search for courses online in different universities and add up to three course files to their profile. There is also the ability to download the media type associated with the course. This is where more work can be implemented. As I mentioned before, the video display was simply a demonstration of how a video file can appear on a web page after it has been encoded. Moreover, as it was shown in some figures above, in the course files, there is a field called media type which specifies if the file contains slides, a video, etc…
This is where some work can be implemented in the future. Instead of simply writing out "slides" or "online lecture", a real file can be stored in the media type field in the database table. This file can be a PowerPoint lecture or an mpeg video file. Whatever the file is, the student can select the specific course file and click "DOWNLOAD" and what will be visible is this file.
Moreover, there could be an administrator user who has control over this website. This means that he/she has the authority to upload new course files into the database, delete some old ones, look up registered users and have the permissions to access their data and alter/delete/modify it. Basically, the administrator will make sure that data is kept up to date and have control on who registers in to the site.

V. References

[1] W3 Schools, [Online]. Available: http://www.w3schools.com/

[2] Community for Multi Skilled Developers, [Online]. Available: http://www.visualbuilder.com

[3] The Apache Jakarta Project, [Online]. Available: http://jakarta.apache.org/tomcat/

[4] Jeff Hanson, Server Side Coding, [Online]. Available: http://www.sitepoint.com/subcat/java

[5] Sun Microsystems, Java Technology, [Online]. Available: http://java.sun.com/

[6] Dave Kristula, HTML: An Interactive Tutorial for Beginners, [Online]. Available: http://www.davesite.com/webstation/html/

~~ Part Two ~~

I. Abstract

Part Two of this report deals with the JSP implementation of the user interface for Epresence on PCs, SmartPhones and PDAs. Different specifications for each device are discussed and the underlying algorithm is explained. In addition, illustrations include diagrams and snapshots that will aid in the understanding of the material. A complete list of codes used in this segment of the project is presented at the end of the report, in Appendix A for Part Two. Appendix B contains weekly reports on the work's progress.

II. Introduction

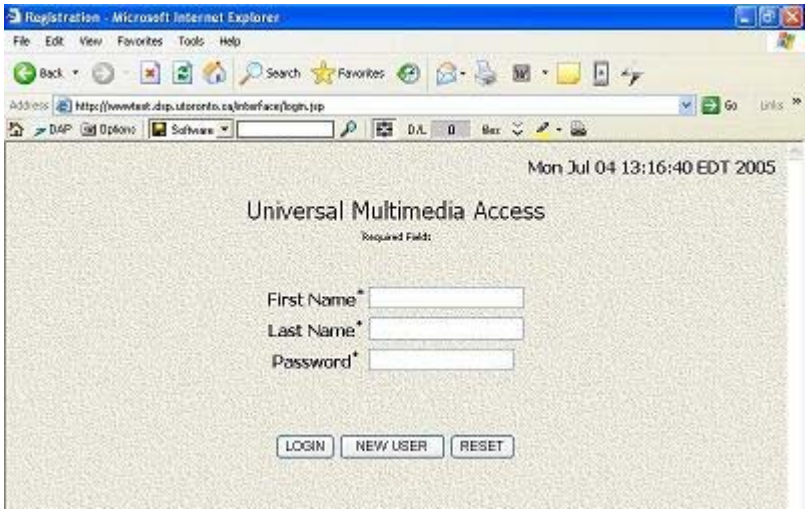
The project description first familiarizes readers with the login as well as the presentation layout of Epresence. The interactivity of the webpages is briefly mentioned and the JavaScript algorithm that is the backbone of the JSP codes is disected. Section B then moves on to the implementations written for Windows Mobile 2003 SmartPhones. An overview of the restrictions of SmartPhones is given and a means of virtually testing the JSP pages is illustrated. Screen shots and demonstrations from the simulations are included. Section C changes gear to a Windows Mobile 2003 platform on PDAs and finally, possible future improvements are discussed in Section D.

III.The Project

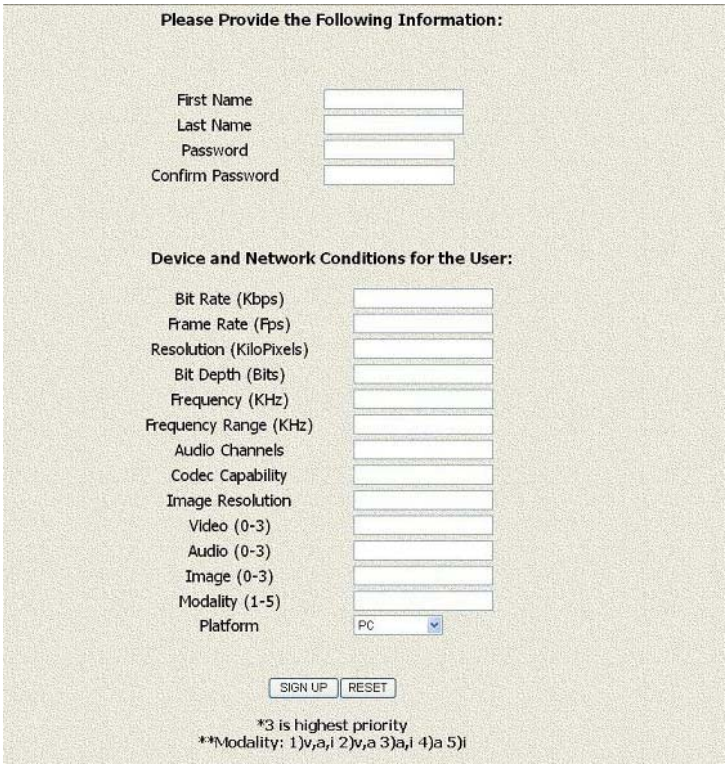
A. Viewing Archived Lectures on Epresence

1. Accessing a User Account

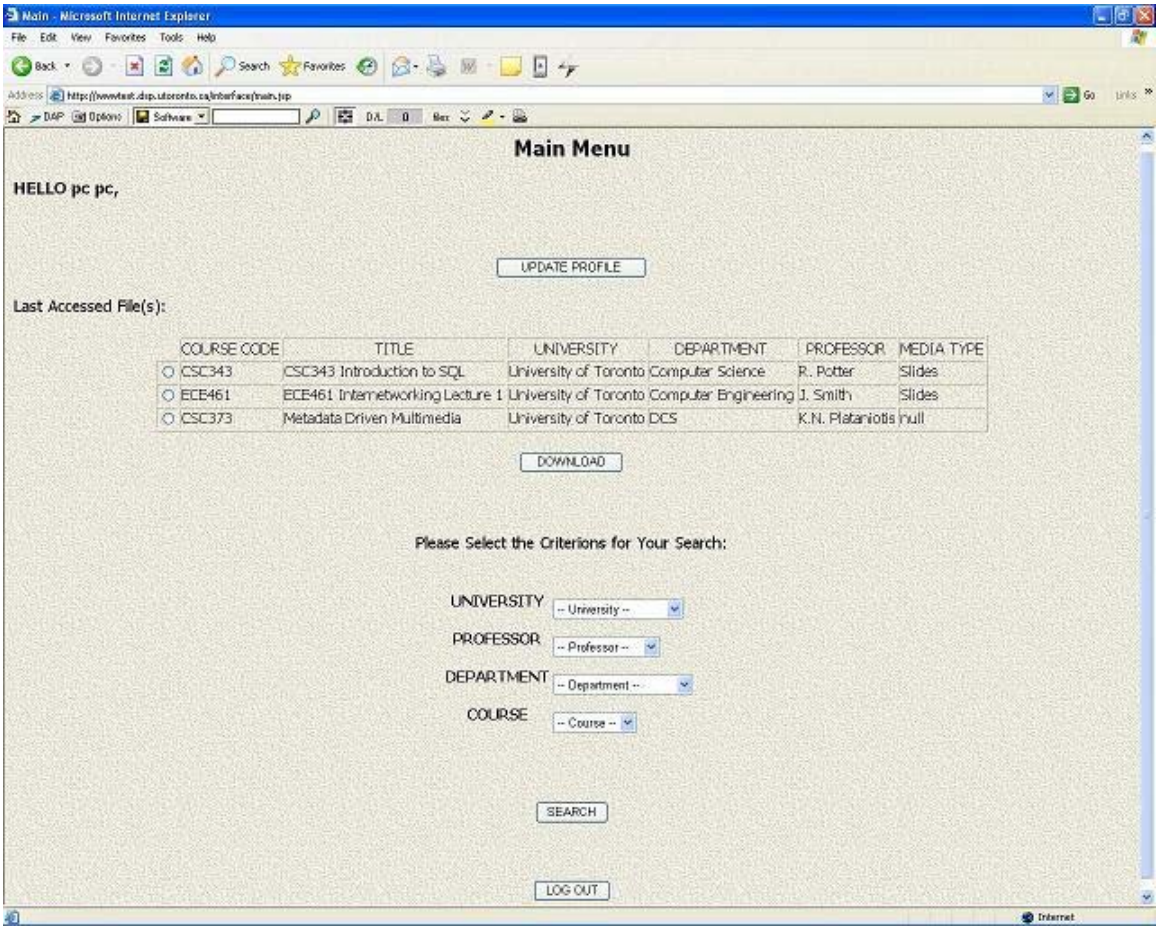
This is the login interface, with all three fields set as required fields to be filled in. New users can click on the "New User" button to create an account.



A new user is asked to fill out the following form in order to create an account.

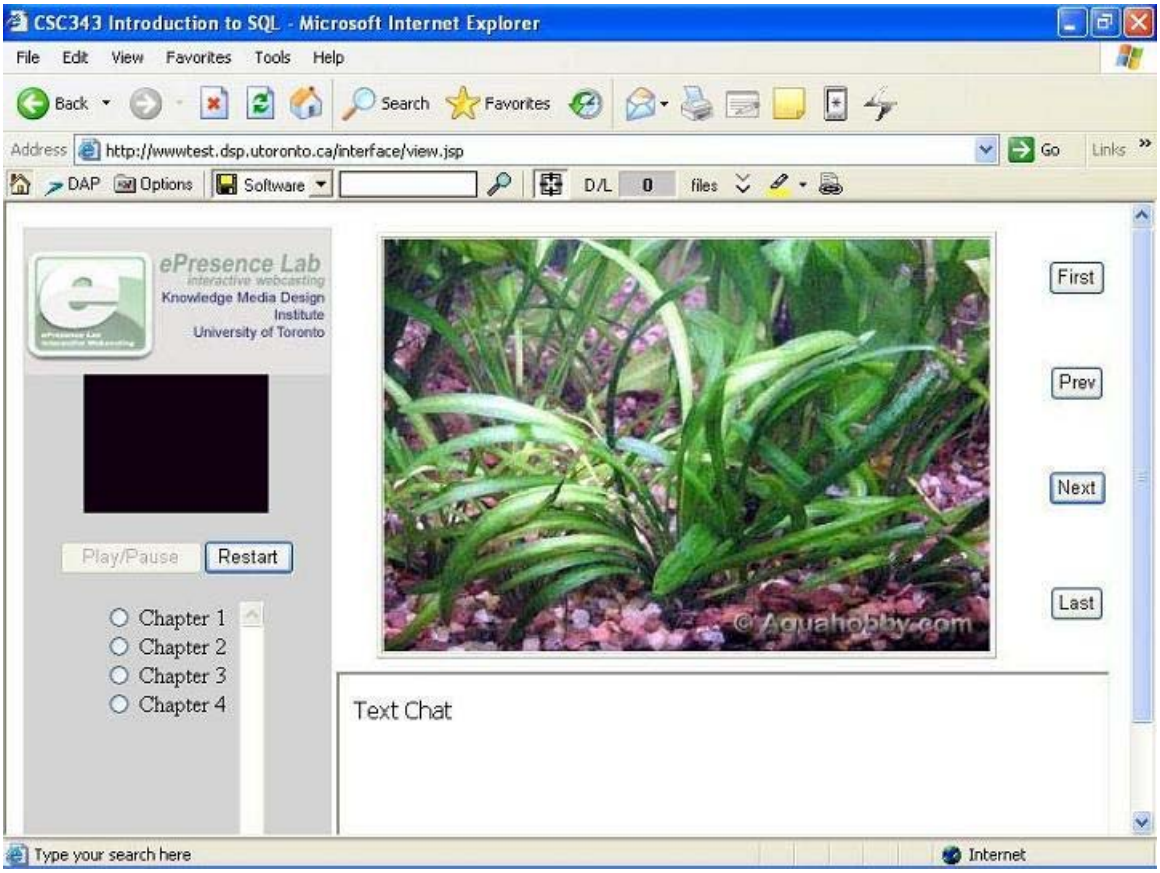


After logging in or creating a new account, the user is presented with his or her most recently accessed archives, the choice of updating his or her profile of device and network conditions, and the option of searching for a specific lecture yet to be viewed.



Once a lecture is chosen with either of the two methods mentioned above, the user is asked to select the preferable resolution. The options are shown below.

Finally, the archived lecture is displayed according to the resolutions chosen by the user. (The example of a user viewing “small” resolution with a frame rate of “10” on a PC platformis demonstrated in this snap shot. The lecture is CSC343 Introduction to SQL.)



2. User Interaction with Epresence Presentations

An Epresence presentation can be a combination of one or all of video, sound, slides and text-chat, depending on user preference and client device limitations (ie. processing rate, resolution, network speed, etc.). The interface shown above in step 5 is typical of the PC platform with all the media types present. A Personal Digital Assistant (PDA), for example, or a SmartPhone will have a different interface based on its screen estate and pocket Internet Explorer capabilities (to be discussed later).

The basic features of the media are similar across all three platforms. The video and sound files are streamed by the Darwin Streaming Server over an RTSP protocol, which will provide the user with almost instant viewing of the lecture rather than a slow website that needs to download the media files in their entirety before playing back. On the PC interface, the movie is embedded into the web page, saving the user the hassle of entering the RTSP URL him- or herself. Currently, PDAs and SmartPhones that run on Windows Mobile 2003 SE, which is what this project focuses on, do not support the <embed> tag required for RTSP links. Therefore, for now these devices will have to rely on playback

capabilities of Windows Media Player 9. Possible solutions to this scenario will be discussed in the Future Works section.

The user is given control over the media elements. The video can be played, paused and restarted along with sound and slideshow. When both movie and slides are displayed, the two are synchronized: the slide will change as the video plays out. The time of the slide change will have been recorded as time stamps from the live lecture. The chapter headings underneath the video allow the user to jump to any section of the slideshow, but as soon as the user chooses a chapter, the video and the slideshow would become out of synch. The same can be said about the 'Next,' 'Prev,' 'First,' and 'Last' buttons to the right of the slideshow (see screen shot from step 5 of the above section); they allow the user to browse through the slides freely, but will bring the presentation out of synch. The difference with these buttons, however, is that the video and slideshow will go back to synchronized display as soon as the video catches up to the slides.

3. Synchronization Algorithm

The synchronization of the video, sound and slideshow is accomplished through JavaScript. The backbone of the algorithm is the **setTimeout ("function_to_be_called", interval x)** function, which allows a function by the name in quotation marks to be called after a period of x seconds. To create a slideshow, **setTimeout** is placed inside **"function_to_be_called"** so that it will call itself again and again, but every time the interval will be reset according to the time stamps. In addition, a variable is kept to record the numerical ID of the current slide in the sequence. Since the video and slideshow are synchronized, a difficulty arises if the user chooses to bring them out of synch. If the user decides to browse through the slide, then the timer to call the slideshow function is simply cancelled until the video catches up. But when the video is paused, the timer has to be cancelled and the amount of time paused has to be recorded in order to calculate the appropriate slide to show when un-paused as well as the duration of that slide. Please refer to the appendix for complete algorithm and comments.

B. Migrating to SmartPhones

1. What are the features of a SmartPhone?

~The Software~
The newest version of the Microsoft software that runs SmartPhones is Windows Mobile 2003. This software includes the following features.

- Pocket Outlook
- Unified Inbox
- Contacts
- Calendar
- Pocket Internet Explorer
- MSN Messenger
- Windows Media Player
- Smartphone Games
- Security
- Predictive Text Input
- Smartdial
- ActiveSync

Fig. 1 Motorola MPx200

~The Screen Estate~
A Smartphone screen size is 176 x 220 pixels (see Fig. 2 for the Home and Start screen). When viewing a web page in pocket Internet Explorer, the usable content area is reduced to 171 x 175 pixels due to the scroll bars, the title bar and the menu bar (see Fig. 2 again). To ensure that the dreaded horizontal scroll bar does not occur, any fixed-width element should be set to 160 pixels or less.

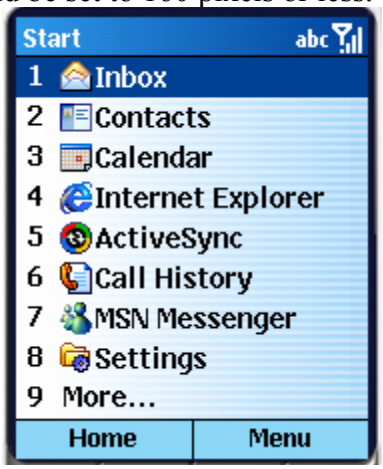


Fig. 2 Home Screen, Start Screen

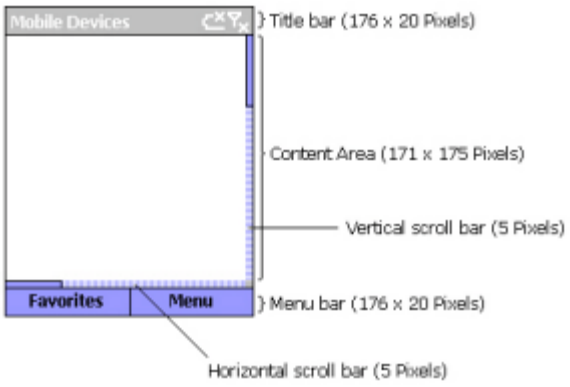


Fig. 3 Screen Estate in pocket Internet Explorer

~User Inputs~

The buttons and their functionalities on a SmartPhone are summarized in Fig. 4.

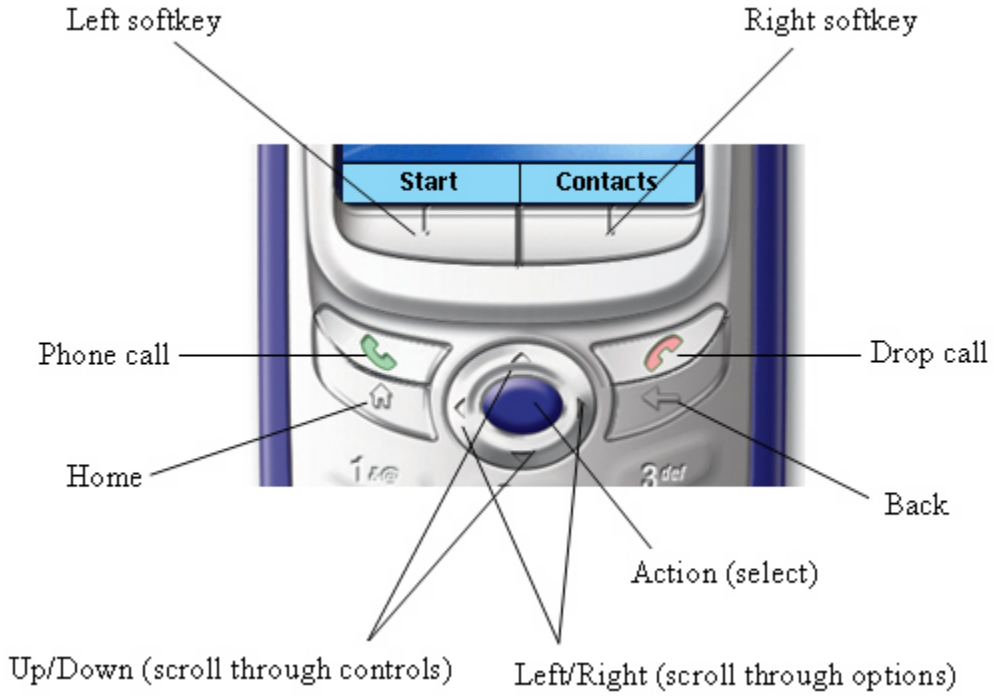


Fig. 4 Buttons and controls on a SmartPhone

For text input, users must rely on the numeric keypad to enter both numbers and alphabets. The input mode is indicated on the title bar as Fig. 5 shows.

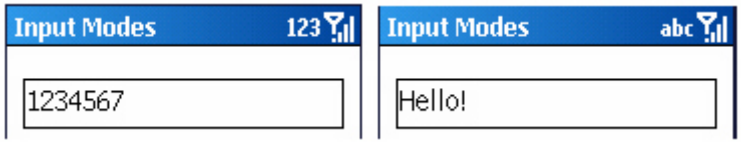


Fig. 5 Providing visual cues to indicate the input mode

2. How is pocket Internet Explorer different from Internet Explorer?

The Microsoft® Pocket Internet Explorer is a full-featured Internet browser, optimized for mobile devices with small, vertically oriented displays. HTML functionality is equivalent to that of Microsoft Internet Explorer version 4.01. A fit-to-screen option dynamically resizes Web pages to minimize horizontal scrolling. Microsoft® Pocket Internet Explorer has been updated to support the following:

Feature	Description
Fixed width layout	Dynamically fits text to device screen, eliminating need for user to scroll horizontally to read text.
Cookies	50 elements, 4 KB maximum each.
Supported protocols	IPv4, IPv6 HTTP1.1 (http, https), FTP, FTPIR, NNTP, file://.
XML	Uses the Windows CE MSXML parser.
Hypertext Markup Language (HTML)	Equivalent to version 3.2, with support for tables and forms. Cascading Style Sheets No Virtual Reality Modeling Language (VRML).
Scripting	Windows CE JScript 5.5 (ECMA-262 compliant).
Audio formats	WAV.
File formats	GIF, JPEG, BMP, XBM, PNG, HTML, TXT. No support for animated GIF (GIF89a).
Offline browsing	Page caching, subscriptions.
Encryption	64-bit and 128-bit SSL.
Security	SSL2, SSL3, TLS 1.0, SGC.
User authentication	Basic, NTLM.

Pocket Internet Explorer does not support the following:

- Printing
- Java Virtual Machine
- Downloading of ActiveX® controls
- Context menu (right-click menu) support
- Intelligent forms for automatic completion of user input and saved passwords
- Font downloading
- Data binding
- Multiple windows

Complete HTML reference for Windows Mobil-Based SmartPhones can be found in the SmartPhone Developer's Guide at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/DevGuideSP/html/Dev_Guide_for_Windows_Mobile-Based_Smartphones.asp

3. Which SmartPhones run on Windows Mobile 2003?

The Microsoft Corp. website is currently advertising the following phones:

- Audiovox SMT 5600 – Windows Mobile 2003 SE
- mate SP3 – Windows Mobile 2003 SE
- mate SP3i –Windows Mobile 2003 SE
- Motorola MPx220 – Service Provider: Cingular / TIM (Brazil), Windows Mobile 2003 SE
- Samsun SCH-i600 – Service Provider: Verizon Wireless, Windows Mobile 2002
- SP-i600 SmartPhone – Service Provider: Sprint, Windows Mobile 2003
- Voq Professional Phone –Windows Mobile 2003

Further information on these products and downloads for SmartPhone accessories can be found at http://www.microsoft.com/windowsmobile/devices/smartphone/americas.mspx

4. What is the SmartPhone 2003 SDK?

SDK for Windows Mobile 2003-based Smartphones is created by Microsoft Corp. for the development of applications for Windows Mobile 2003-based Smartphones, leveraging the Microsoft® .NET Compact Framework, Windows CE. NET 4.2, an improved emulator and new ATL support.

The Emulator is the main tool needed for this project. It mimics a real SmartPhone in every way, except it appears on the desktop. All features of a SmartPhone are included, such as Calendars, Games, Windows Media Player 9 Series, Contacts and pocket Internet Explorer.

The soft keys, numeric keypad and the Left/Right/Up/Down/Selection buttons can be controlled with the mouse. For text input, both the keypad on the emulator and the keyboard can be used.

The web pages created for Epresence will be tested on this Emulator to verify that they will function properly on a real SmartPhone; however, since it is emulation, Internet loading speed on the Emulator is much slower than on the actual phone.

System Requirements for the Installation of the SDK are:

- Windows 2000 Service Pack 3 or Windows XP

- Microsoft ActiveSync 3.7.1
- eMbedded Visual C++ 4.0
- eMbedded Visual C++ 4.0 Service Pack 3

The downloadable files and additional instructions on the installation can be found at http://www.microsoft.com/downloads/details.aspx?FamilyId=A6C4F799-EC5C-427C-807C-4C0F96765A81&displaylang=en

5. How to start the SmartPhone Emulator and connect to the Internet?

The SmartPhone Emulator can be launched with the following methods (see Fig. 6 for what the Emulator looks like on the desktop).

- eMbedded Visual C++: compile a simple "Hello world" program and choose the device as the Emulator
- Command Prompt: execute Emul.exe under the Emulation folder of SmartPhone 2003 in your installation location
- Shortcut from the Start Menu: the emulator should be under Microsoft SmartPhone 2003 SDK -> SmartPhone 2003 Emulator

Fig. 6 SmartPhone 2003 Emulator

Once the Emulator is up, go through these steps to connect it to the Internet.

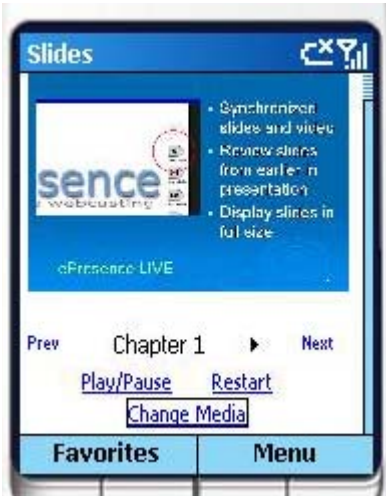
1. Click on "Start"
2. Go to "8. Settings" and choose "9. More…"
3. Choose "2. Data Connections"
4. Click on "Menu" and choose "1. Edit Connections"
5. Choose "3. Proxy Connections"
6. From "Menu," choose "1. Add"

7. Give the connection a description
8. Choose Connects from “My Corporate Network” (it may be called something else, such as “Work”, depending on the computer’s configuration)
9. Choose Connects to “The Internet”
10. Leave the other options
11. Click Done until your back at the Start screen
12. Browse the internet

6. How to view JSP pages created for Epresence?

To view JSP pages, simply start the Tomcat server and enter the URL into the SmartPhone Emulator as one would on a regular browser. Note, however, that if “localhost” is typed in, the Emulator will not recognize it; instead, type in the computer’s name. The next series of images demonstrates what a user would see for a makeshift archived lecture.

1. The user, after finishing viewing the lecture slides, chooses to see the slide for Chapter 1.



2. He or she then scrolls down using the arrows to select the option of switching to a different medium.

3. The other option is to go to “Menu” and choose “Address Bar.”

Continued Next Page…

4. Assume that the main page has the above URL. Note: "pcthomaswu" has to be used on the Emulator instead of "localhost."

5. On the main page, the user chooses to see the lecture video. Note that pocket Internet Explorer (pIE) gives link focus.

6. The user selects "View Movie."

7. pIE asks for confirmation to download the video.

8. pIE launches Windows Media Player 9 Series (WMP9) to play the video.

7. The control consul of WMP9 is displayed at the end of the video and users can use the back button to return to the main page.

7. What is the optimal text size to use for PowerPoint slides?

Slides containing different sizes of Arial and Times New Roman text were created in Microsoft PowerPoint and these slides were saved directly in JPEG format (under Save As… in PowerPoint). This is probably the easiest method to retrieve lecture slides as

JPEG images. Then, these images were coded into web pages with their widths set at 160 pixels (the maximum width before the horizontal scroll bar appears) and they are viewed with the SmartPhone 2003 Emulator as in Fig. 7 and 8.

Fig. 7 Various font sizes (regular and bolded) for Arial

Fig. 8 Various font sizes (regular and bolded) for Times New Roman

It can be concluded that in general a non-serif font face, such as Arial, looks better than a serif font like Times New Roman, which tend to become segmented when shrunk. Also by using the smallest words appearing on the SmartPhone's Home screen (see Fig. 2) as a standard of measurement, the Arial font would require size 66, preferably bolded when written in Microsoft PowerPoint, and if Times New Roman must be used, it should be at least size 72 and bolded.

C. Migrating to Pocket PCs / PDAs

1. What are the features of a PDA?

~The Software~
PDAs or Pocket PCs that have been recently available on the market use the same software as the new SmartPhones: Windows Mobile 2003, with support for context menus, drop down lists and a few other controls. For a comprehensive list of Windows Mobile 2003-based Pocket PCs, go to
http://www.microsoft.com/windowsmobile/devices/pocketpc/ppc/americas.mspx

~The Screen Estate~
A Pocket PC screen size is typically 240 by 320 pixels (see Fig. 9 and 10). The usable content area in pIE, however, is only 229 by 255 pixels. With these dimensions, the

optimal size for displaying content is a maximum width of 220 pixels, which eliminates the appearance of the horizontal scrollbar.

Fig. 9 Home Screen

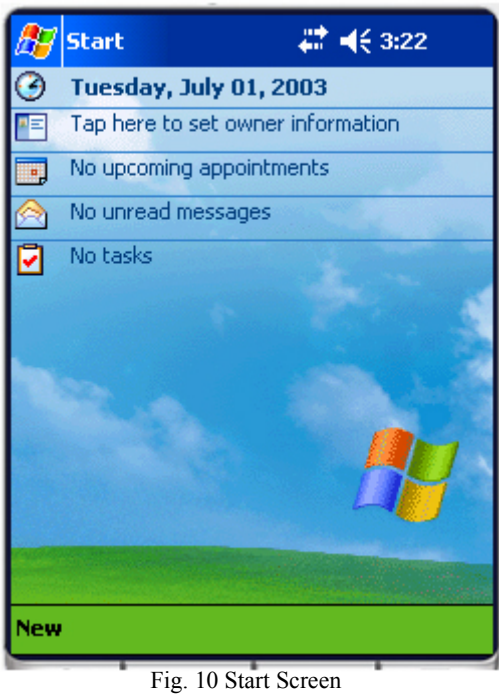


Fig. 10 Start Screen

~User Inputs~

All Windows Mobile 2003-based PDAs come equipped with a touch sensitive screen and a pointer. Sometimes they have physical key pads like Symbol Technologies' MC50, but most of the time, Pocket PC have a built-in Software-based Input Panel (SIP). When activated, it consumes 240 by 81 pixels of screen space (see Fig. 11).

Fig. 11 PDA with SIP activated

2. Viewing Epresence Presentations on PDAs

An actual Pocket PC was purchased in order to test the web pages, so there was no need for any development in a Pocket PC SDK. The following (Fig. 12 – 14) are screen shots of what the PDA interface looks like.

Fig. 12 Media preference page for an archived lecture on a PDA

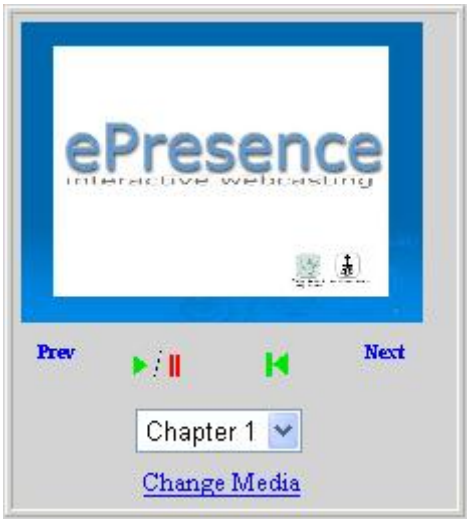


Fig. 13 Viewing the slides on a PDA

Fig. 14 Text Chat (to be implemented later) with the PDA's SIP activated

D. Future Work

1. Embedding Sound and Video into Web Pages for Mobile Devices

Embedding sound and video into the web pages would allow users to bypass the page on which he or she has to click "View Video" in order to open Windows Media Player, and the pages showing the lecture slides would be able to have the lecturer's voice as accompaniment. Unfortunately, embedding objects requires ActiveX controls for the object, in this case, Windows Media Player 9. In Windows Mobile 2003, such control does not exist and Microsoft has decided that users would not be able to download any ActiveX element. The good news is Windows Media Player 10 Mobile will have the necessary ActiveX control for embedding the Windows Media Player object. Microsoft has already announced that Windows Media Player 10 Mobile will be available on many new SmartPhones over the next few months.

The following have Windows Media Player 10 Mobile installed:

- SmartPhone: Audiovox SMT-5600 from Cingular
- Pocket PC: Dell Axim X50

The following Pocket PCs are upgradeable to Windows Media Player 10 Mobile:

- HP iPAQ rx3100
- HP iPAQ rx3400
- HP iPAQ rx3700
- Dell Axim X30

2. Video and Sound Streaming for PDAs and SmartPhones

Simply embedding the media is not, in fact, the optimal solution to the user-friendliness issue on PDAs and SmartPhones. Streaming an embedded medium would be. The current set up of the Darwin Streaming Server can only stream to a QuickTime Player, not a Windows Media Player (WMP) because although WMP is RTSP compliant, it only supports the streaming of .asf files, but Darwin only streams .mov files. Why not ask the user to download QuickTime? One, there is no Windows Mobile version of QuickTime and two, this project's aim is to minimize inconvenience to the client.

A possible solution comes with Microsoft Media Server (MMS), which is a compatible server with Windows Media Player. Unfortunately, MMS is not an open-source product of Microsoft Corp., and the use of MMS related servers from the Internet, especially those without credentials, should be scrutinized.

IV. Reference

Microsoft Corporation. (n.d.). *Windows Mobile Software*. Retrieved June 30, 2005, from http://www.microsoft.com/windowsmobile/about/tours.mspx

Microsoft Corporation. (n.d.). *Windows Media Player 10 Mobile*. Retrieved June 30, 2005, from http://www.microsoft.com/windows/windowsmedia/player/windowsmobile/default.aspx

Microsoft Corporation. (n.d.). *Windows Mobile-Based SmartPhones Available in the Americas*. Retrieved June 30, 2005, from http://www.microsoft.com/windowsmobile/devices/smartphone/americas.mspx

Microsoft Corporation. (n.d.). *What's New [in Windows Mobile 2003 SE]*. Retrieved June 30, 2005, from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/devguidesp/html/sp_conwinmobile_programming.asp

Microsoft Corporation. (n.d.). *Download Details: SDK for Windows Mobile 2003-Based SmartPhones*. Retrieved June 30, 2005, from http://www.microsoft.com/downloads/details.aspx?FamilyId=A6C4F799-EC5C-427C-807C-4C0F96765A81&displaylang=en

Sophonix. (2002, November 8). *Development and Content for Desktop and Mobile*. Retrieved June 30, 2005, from http://www.sophonix.com/articles/spcontent.asp

Wei-Meng Lee. (2004, January 7). *Design Considerations for Microsoft SmartPhones*. Retrieved June 30, 2005, from http://www.oreillynet.com/pub/a/wireless/2004/01/07/smartphone.html

~~ Part Three ~~

I. Abstract

This report provides a detailed description of the implementation of the Universal Multimedia Access Interface (UMAI). The description includes various technologies used to implement the algorithms and overall ideas, accompanied by explanations, examples and screenshots. Finally in the appendices there are all source files.

II. Introduction

The work done is based and is extending Rasha Rashidi's work, so the assumption was made that the reader is familiar with it. This report start by introducing the slight modifications added to Rasha's work. Those were introduced to accompany the future extension (i.e. extending the MySQL database, modifying some of the JSP code).

The report then goes to discuss creating the Digital Item (DI) and the Digital Item Adaptations (DIA), which are both XML meta-data.

It continues by discussing the various XSLT style sheets and finally putting all the pieces together. How JSP was used to apply the transformations to the XML files and generating both different User Interfaces and command line arguments for the transcoder.

III. The Project

A. Overview

The project is an extension to Rasha Rashidi's work. There were a few modifications added and a lot of separate additions. After a user logs in and selects a lecture they want to view, the movie is no longer hard-coded. There is a database that has a collection of all the available lectures (may they be pre-recorded or live streaming), along with all the relevant information, e.g. University, professor, etc. The server takes all necessary information and creates the DI that will be used to generate the interface, and the DIA, used to generate the command line arguments.
There is also a database storing all the XSLT style sheets with their relevant parameters, e.g. resolution, modality, etc. After the DI and the DIA are generated two sheets, that match the user preferences the closest, are picked. The first one is applied to the DI to produce the web interface and the second one is applied to the DIA to produce the command line arguments to the transcoder.
After that the appropriate interface is displayed and the transcoded media is shown accommodating the user preference and abilities as close as possible, providing high QoS.

B. Initial Modifications

The modifications done to Rasha's work were introduced to mainly accommodate the future development of the project and to optimize certain steps since everything should be running in real time.

One of the changes that need explanation was converting the algorithm that selects the closest sheet. While the actual algorithm remains the same, its implementation was moved from JSP/MySQL queries to a single MySQL query. That was done because JSP runs on the java virtual machine which tends to be slower than MySQL which is optimized for data retrieval. The downside of this is that the query is huge, not user friendly and hard to modify over and over again, but this is justifiable by the increase of speed.

Another significant extension is changing the flow when the user clicks download. The last hard-coded webpage is remove and after the user has made his/hers choice on resolution or frames per second, he/she is taken to the dynamic JSP-based webpage (view.jsp) that generates the interface and also does the transcoding on the fly and then delivers the adapted multi-media content to the user. See 3.6 for more details.

C. Digital Item

The Digital Item is a piece of XML meta-data that describes everything about a lecture (i.e. speaker, university, number of chapters, slides, etc). There is not a strict outline of the DI. It varies greatly depending on the content. Here follows a detailed description of the DI used with explanation for the purpose of each tag.

The root element of the document is <DIDL>. Right after it is a <DECLARATIONS> element. Within it goes any Mpeg-7 data that might be needed to describe the video, e.g. motion descriptors. After that is the <CONTAINER> tag which encapsulates all the metadata for this specific lecture.

The first couple of tags in the <CONTAINER> are <DESCRIPTOR>. Those provide general information for the lecture, e.g. university, professor name, course name, etc.

Two of the descriptors are different, as they have a reference in them. The first one is with id "LOGO", and stores the logo of the lecture. The second one is with id "PLAYLIST" and stores the location of the play list of videos that will be passed to WMP. After that come a series of <ITEM> tags, each one corresponding to a different chapter. The ID property of an ITEM specifies its order in the presentation.

Each item has the information about the video, audio and slides in that chapter. First the chapter title is described by a descriptor. The video meta-data is stored in a <COMPONENT> with an id that is "VIDEO". Tat component stores information about the description of the video, its format and length as well as the locations of a thumbnail and the video itself.

Audio information is stored in the same way, only that the <COMPONENT> storing it has an id “AUDIO”.

Similarly, the slides are described by a <COMPONENT> with an id “SLIDES”. Within it every slide is represented by an <ANCHOR> tag. Its property, FRAGMENT, holds the information of the duration of the slide. Within the anchor there is a descriptor that specifies the overall position in the sequence of slides, and is followed by the location of the slide.
An example DI can be found in Appendix A.

D. Digital Item Adaptation

The DIA is similar to the DI in the sense that it is also descriptive XML meta-data. Its purpose is completely different though. It is used to store the user preferences and the capabilities of the devices they are using. That information is later used in the transcoding to provide the best available multimedia quality to the user. As with the DI, the DIA vary greatly depending on their use, so here follows a description of the used tags and their functions. For example of how the tags were used refer to DIA.xml in Appendix A.

<UsageEnvironment> - Describes various dimensions of the usage environment, including user characteristics, terminal capabilities, etc.
<UserCharacteristics> - Describes the User characteristics in terms of general User information, content preference, presentation preferences, accessibility characteristics, etc.
<UserInfo> - Describes a general characteristics of a User such as name and contact information.
<UserCharacteristics xsi:type=”PresentationPreferencesType”> - Tool that describes the audio-visual presentation preferences of a User
<ModalityConversion> - Describes the modality conversion preferences of a User.
<Conversion> - Tool for describing the choices of a modality conversion
<From> - Describes the original modality
<To> - Describes the destination modality
order - Describes the order of the conversion corresponding to the original modality and destination modality above. Conversion of high order is carried out before conversion of low order. The smaller the value is, the higher the order is, except that when value is 0, the corresponding conversion is forbidden.
weight – describes the weight of the conversion corresponding to the original modality and destination modality above. It is used to scale the highly-subjective conversion boundaries between different modalities. The weight has the non-negative real type. Its default value is 1.0.

<PresentationPriority> - Describes the presentation priority of resources according to the preference of a User.

<GeneralResourcePriorities> - Describes the presentation priorities for resources in a general way. In this case the User has some general knowledge of the content source, e.g. original modalities or genres.

<ModalityPriorities> - Describes the presentation priorities for all resources of certain modalities.

<Modality> - Describes a certain modality.

<TerminalCapabilities> - Describes the capabilities of the terminal in terms of decoding and encoding capabilities, input-output capabilities, and device properties.

<TerminalCapabilities xsi:type="CodecCapabilitiesType"> - Tool for describing the encoding and decoding capabilities of the terminal.

<Decoding> - Describes the decoding capability of the terminal.

<TerminalCapabilities xsi:type="InputOutputCapabilitiesType"> - Tool for describing the input-output capabilities of the terminal.

<Display> - Tool that describes the display capabilities of the terminal.

bitsPerPixel – Describes the bits-per-pixel, i.e., the color depth, of the display.

refreshRate – Describes the refresh rate of a display in Hz.

<Resolution> - Describes the horizontal and vertical resolution of the display in pixels.

horizontal – Describes the horizontal resolution.

vertical – Describes the vertical resolution.

<AudioOut> - Tool that describes the audio output capabilities of the terminal.

samplingFrequency – Describes the sampling frequency in units of Hz.

bitsPerSample – Describes the number of bits per sample the output device supports

lowFrequency – Describes the lower value of the frequency range in units of Hz.

highFrequency – Describes the upper value of the frequency range in units of Hz.

numChanels – Describes the number of output channels the speakers support.

<UsageEnvironment xsi:type="NetworkCharacteristicsType"> - Tool for describing the network characteristics in terms of capabilities and conditions.

<NetworkCharacteristics> - Describes the static capabilities and time-varying conditions of a network.

<NetworkCharacteristics xsi:type="NetworkCapabilityType"> - Tool for describing the static capabilities of a network.

maxCapacity – Describes the maximum bandwidth capacity of a network in bits/sec.

minGuaranteed – Describes the minimum guaranteed bandwidth of a network in bits/sec.

<NetworkCharacteristics xsi:type="NetworkConditionType"> - Tool for describing time-varying conditions of a network.

<AvailableBandwidth> - Describes the available bandwidth of a network.
minimum – Describes the minimum available bandwidth in bits per second of the network during the window of time expressed by interval
minimum – Describes the minimum available bandwidth in bits per second of the network during the window of time expressed by interval
average – Describes the average available bandwidth in bits per second of the network during the window of time expressed by interval.
interval – Describes the window size in milliseconds that corresponds to the calculation of AvailableBandwith attributes: minimum, maximum and average.

E. XSL Transformations

Meta-data (DI and DIA) by itself is just storage. To utilize that store we will use XSLT and XPath. XSLT is a language for transforming XML documents into XHTML documents or to other XML documents and XPath is a language for navigating in XML documents used by XSLT. In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document. The fact that XSLT uses pattern matching techniques makes it faster to transform data than JSP that is why it is our preference to use XSLT as much as possible.

There are two types of XSLT sheets written for this project. The first is the one applied to the DI to generate the XHTML interface, whereas the second type is applied to the DIA to generate the command line options to the transcoder. We will look in more detail at both types.

1. Interface

The sheets used to create the UI normally have a general pattern, with the only difference being the XHTML look of the webpage. Here we will show some of the more common routines used.

The XSLT sheets that show the interface is essentially an XHTML document with some of its parameters (media locations, media durations, etc.) generated by XSL transformations. These transformations are applied to the Digital Item since all necessary information is stored there.

There are several sheets for each interface, but these differ only cosmetically. The only differences are the size of the video, slides, etc. and the actual XSLT content stays the same for each interface.

Sheets for different platforms differ again only in the XHTML content and the presentation method (since for example PDAs don't have WMP 10 ActiveX controls), but again the routines to extract the information from the DI are the same.

The following code extracts the durations of the slides, converting them to milliseconds. Extracting the actual slide locations is done with a similar routine.

```
                    <xsl:for-each select="//didl:ANCHOR">
                     <xsl:sort
select="didl:DESCRIPTOR[@ID='SLIDE_NUMBER']/didl:STATEMENT"></xsl:sort>
                        <xsl:choose>
                           <xsl:when test="position()=last()">
                              <xsl:value-of select="@FRAGMENT * 1000"></xsl:value-of>
                           </xsl:when>
                           <xsl:otherwise>
                              <xsl:value-of select="@FRAGMENT * 1000"></xsl:value-of>,
                           </xsl:otherwise>
                        </xsl:choose>
                    </xsl:for-each>
```

The next piece of code creates an ActiveX control to play the video of the first chapter, getting the location from the DI using XSLT.

```
                    <object classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95" height="96"
id="movie" width="128">
                        <param name="FileName">
                           <xsl:attribute name="value">
                               <xsl:value-of
select="//didl:ITEM[@ID='1']/didl:COMPONENT[@ID='VIDEO']/didl:RESOURCE/@REF"></xsl:value-o
                           </xsl:attribute>
                        </param>
                        <param name="ShowControls" value="0"></param>
                    </object>
```

The following generates a radio button for every chapter, so the user can skip to any chapter he/she wants.

```
                        <xsl:for-each select="//didl:ANCHOR">
                           <input name="slide_radio" onFocus="showSlide(1)" type="radio"
value="1">
```

```
                    <xsl:attribute name="onFocus">
                        showSlide(<xsl:value-of
select="didl:DESCRIPTOR[@ID='SLIDE_NUMBER']/didl:STATEMENT"></xsl:value-of>)
                    </xsl:attribute>
                        Chapter <xsl:value-of
select="didl:DESCRIPTOR[@ID='SLIDE_NUMBER']/didl:STATEMENT"></xsl:value-of>
                </input><br></br>
            </xsl:for-each>
```

The full list of XSLT sheets can be found in Appendix A.

2. Transcoding

There is just one transcoding style sheet and it is very straight forward. It produces the command line arguments to ffmpeg, without appending the "ffmpeg" and the input file in the beginning.

The style sheet that performs this is called command.xsl and can be found in Appendix A. It creates a XML document with the root level being <conversion>. It has at most 2 children tags (<command>). Each one is the command line option to ffmpeg . The first one is the video options and the second one – audio. Depending on the chosen modality and presentation preference some of these may not appear (e.g. the user wants only audio but not video). Also this XML layout allows for the two commands to be combined into one should the input file contain both audio and video, or they can be used separately when the input are two files – one for video and one for audio.

F. Combining everything using JSP

Putting everything together was naturally done with Java, since JSP is serving the website. Most of that work is done in the view.jsp file, which can be found in Appendix A.

When we arrive at view.jsp we already have enough details that allow identifying a unique user and a unique lecture. Some of the data (like framerate and size of video) is set in the beginning because of the fact that it might not come from the database but from the user preference from the previous page.
Using this data the program first finds the closest suitable sheet. The algorithm used was not changed but only the implementation. As a result there is one query that finds the closest list.

```
SELECT id, location, param_sum
FROM    (SELECT s.id, s.location,
                    (ABS(s.modality-ui.modality)/ui.modality+ABS(s.bitrate-ui.bitrate)/ui.bitrate+

ABS(s.framerate-ui.framerate)/ui.framerate+ABS(s.resolution-ui.resolution)/ui.resolution+
                     ABS(s.depth-ui.depth)/ui.depth+ABS(s.freq-ui.freq)/ui.freq+
```

```
                    ABS(s.channels-ui.channels)/ui.channels+
                    ABS(s.image_resolution-ui.image_resolution)/ui.image_resolution+

ABS(s.video-ui.video)/ui.video+ABS(s.audio-ui.audio)/ui.audio+ABS(s.image-ui.image)/ui.image
                )AS param_sum
        FROM user ui,(SELECT s.id,s.modality,s.bitrate,s.framerate,s.resolution,s.depth,s.freq,s.channels,
                            s.image_resolution,s.video,s.audio,s.image,s.location
                        FROM sheet s, user ui
                        WHERE ui.modality >= s.modality AND ui.bitrate >= s.bitrate AND
                            ui.framerate >= s.framerate AND ui.resolution >= s.resolution AND
                            ui.depth>= s.depth AND ui.freq>=s.freq AND ui.channels>=s.channels
AND
                            ui.image_resolution>=s.image_resolution AND ui.video>=s.video AND
                            ui.audio>=s.audio AND ui.image>=s.image AND ui.id=userID AND
vwidth=width AND
                            vheight=height AND s.stream=stream
                    )s
        WHERE ui.id=userID          )result_table
WHERE param_sum=(SELECT min(param_sum)
                FROM    (SELECT s.id, s.location,(ABS(s.modality-ui.modality)/ui.modality+
                            ABS(s.bitrate-ui.bitrate)/ui.bitrate+
                            ABS(s.framerate-ui.framerate)/ui.framerate+
                            ABS(s.resolution-ui.resolution)/ui.resolution+
                            ABS(s.depth-ui.depth)/ui.depth+ABS(s.freq-ui.freq)/ui.freq+
                            ABS(s.channels-ui.channels)/ui.channels+
                            ABS(s.image_resolution-ui.image_resolution)/ui.image_resolution+
                            ABS(s.video-ui.video)/ui.video+ABS(s.audio-ui.audio)/ui.audio+
                            ABS(s.image-ui.image)/ui.image) AS param_sum
                        FROM user ui,(SELECT
s.id,s.modality,s.bitrate,s.framerate,s.resolution,s.depth,

s.freq,s.channels,s.image_resolution,s.video,s.audio,s.image,    9
                                                    s.location
                                                FROM sheet s, user ui
                                                WHERE ui.modality >= s.modality AND ui.bitrate >=
s.bitrate AND
                                                    ui.framerate >= s.framerate AND ui.resolution >=
s.resolution AND
                                                    ui.depth>= s.depth AND ui.freq>=s.freq AND
                                                    ui.channels>=s.channels AND
                                                    ui.image_resolution>=s.image_resolution AND
                                                    ui.video>=s.video AND ui.audio>=s.audio AND
ui.image>=s.image
                                                    AND ui.id=userID AND vheight=height AND
vwidth=width AND
                                                    s.stream=stream
                                                )s
                        WHERE ui.id="+userID+"      )result_table
                )
```

The name of the result is stored in a string variable called *sheet*.
The transformation that generates the command line options is only one, so it is known.

Using the same data, about the user and the lecture, the program creates the DI and DIA as strings, inserting the necessary information where it is needed.

Now that it has everything necessary (the DI, DIA and the transformation to be applied to them) the program proceeds to apply the transformations.

The string of the DIA and DI is transformed into a StringReader and later into a StreamSource, the sheet is loaded as a StreamSource object. After which the program receives a new TransformFactory and generates a new Transformer. With it, the program goes to apply the transformation.

```
Reader xslReader = new StringReader(didl);

File transform = new File("/home/grp2/jakarta-tomcat-

5.5.9/webapps/ROOT/interface/"+sheet);
StreamSource    sTransform = new StreamSource(transform);
TransformerFactory    tFactory = TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer(sTransform);
StreamSource source = new StreamSource(xslReader);
StringWriter strWriter = new StringWriter();
StreamResult result= new StreamResult(strWriter);

transformer.transform(source, result);
String strOutput = strWriter.getBuffer().toString();
```

After this the result of the transformation is stored in strOutput as a string.
This is done first for the command line options to ffmpeg. Then the command line is executed to produce new files which are then included in the DI instead of the original ones.

After that the transformation that generates the interface is applied to the Digital Item to produce the webpage. The resulting string is printed using out.print();

This implementation is straight forward, but requires one special note due to the fact that the PDA and SmartPhone interfaces consist of several pages.

G. Future Work

Several ideas could not be implemented due to hardware and software limitations.
One of them is embedding the video and sound in a webpage on a PDA or a SmartPhone. This was due to the fact that currently there does not exist a multi-media control for the pocket internet explorer. However this should change soon when Microsoft Windows Mobile 5.0 comes out. The new operating system for Pocket PCs, which includes WMP

10 and an ActiveX control for it, is already out but the device vendors haven't shipped Pocket PCs or allowed upgrades yet. This is expected to happen in fall 2005, so streaming video and audio to a Pocket PC would not be a problem then. Visually, instead of the current interface for video

We would have the more user-friendly version:

IV. References

[1] W3 Schools, [Online]. Available: http://www.w3schools.com/

[2] The Apache Jakarta Project, [Online]. Available: http://jakarta.apache.org/tomcat/

[3] Sun Microsystems, Java Technology, [Online]. Available: http://java.sun.com/

[4] Working with XML – Tutorial, [Online]. Available: http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/TOC.html

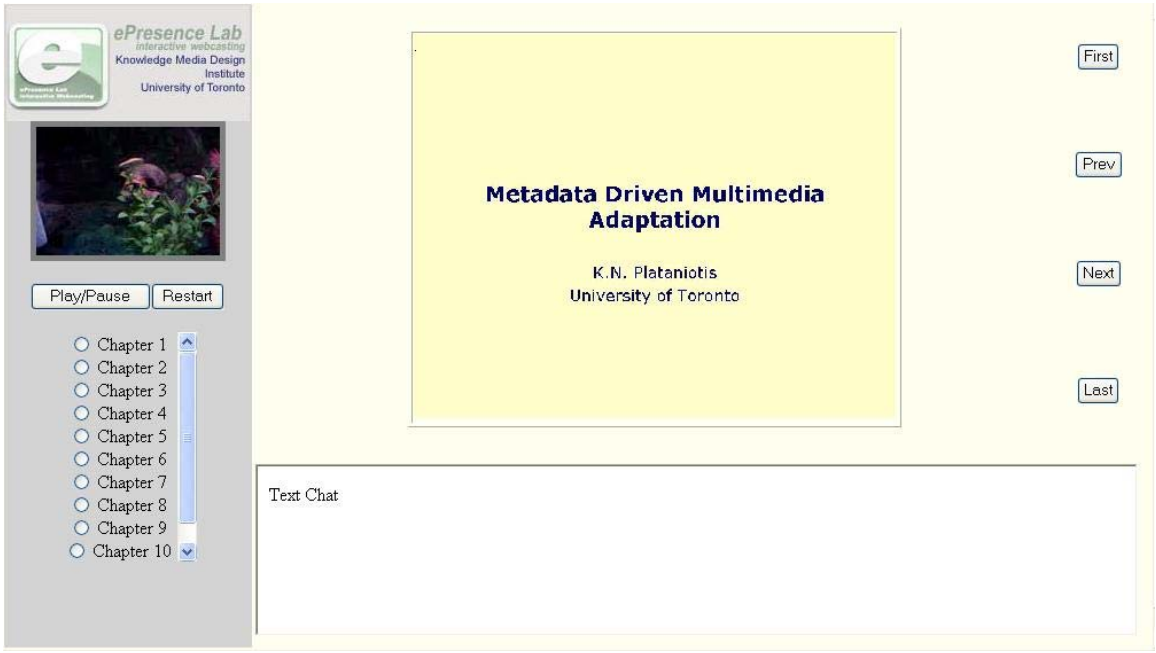
~~ Appendices ~~

I. Appendices for Part Three

A. Appendix B

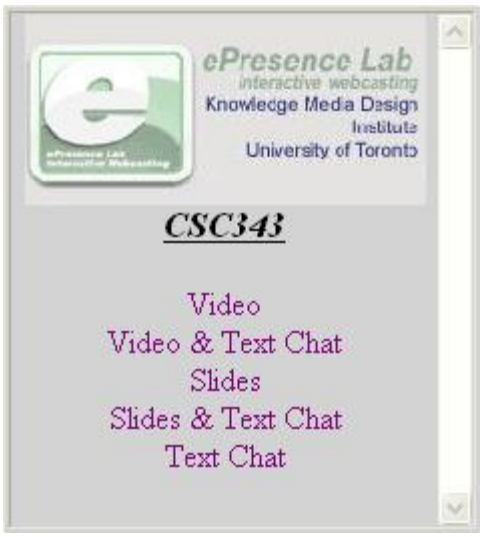
This appendix contains several screenshots of the different interfaces.

1. PC Interface

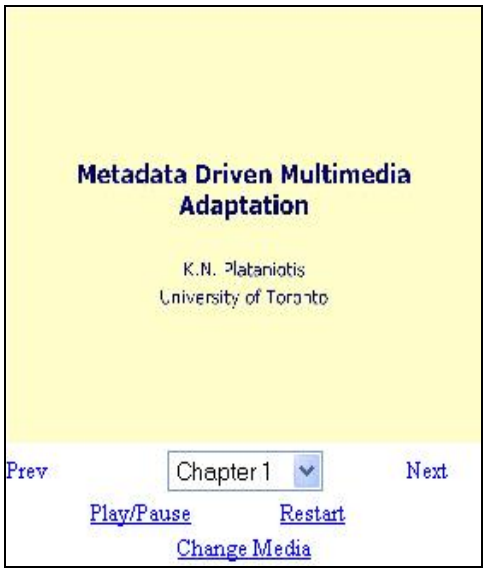


2. PDA Interface

Main page



Slideshow



Current video interface

Future video interface

3. SmartPhone Interface