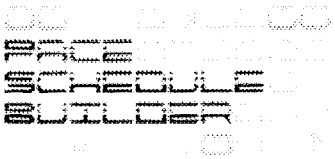


Pace University
New York, New York

CS 389
May 5, 2004

Viktor Geller Naseer Haniff Nikita Lukish

Design Document v3.0

Pace Schedule Builder ("PSB")

Table of Contents

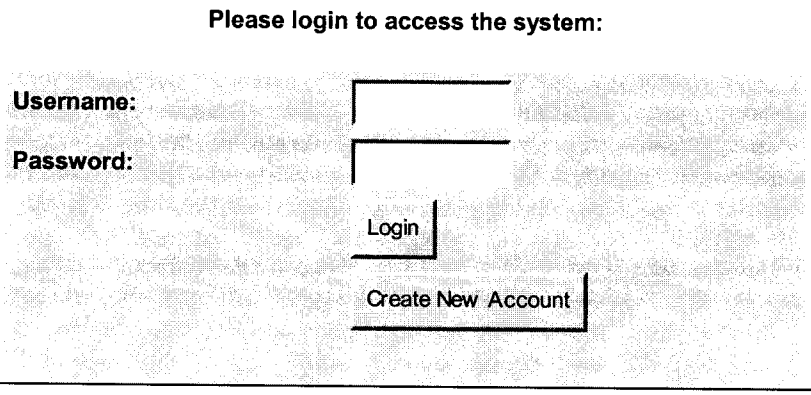
Objectives

The main objective of the system is to provide an automated way to create schedules for students prior to registration. The system will eliminate the routine and tedious work of rooting out class conflicts in the schedule, automatically create personalized schedules based on constraints and preferences specified by the user and automatically generate a visual representation of the schedule(s) which can be sent to a printer or saved for later use.

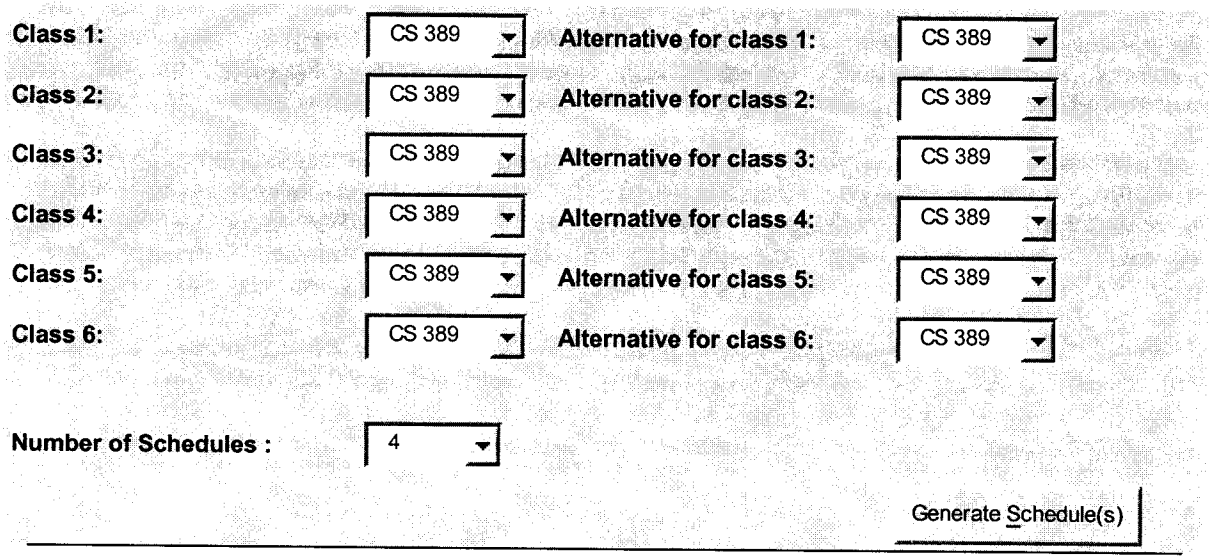
The system will integrate with the Pace ClassSchedule system as a means of getting course data and registration information. It is entirely possible for other universities to write their own versions of this component to work with their databases. A database will hold all login information for the users as well as schedules they designated to save. The database connectivity will be done via JDBC, a standard part of Java.

This project will be built using the Java programming language and will incorporate Sun's Java-based technologies and APIs such as JSP, Servlets and JDBC.

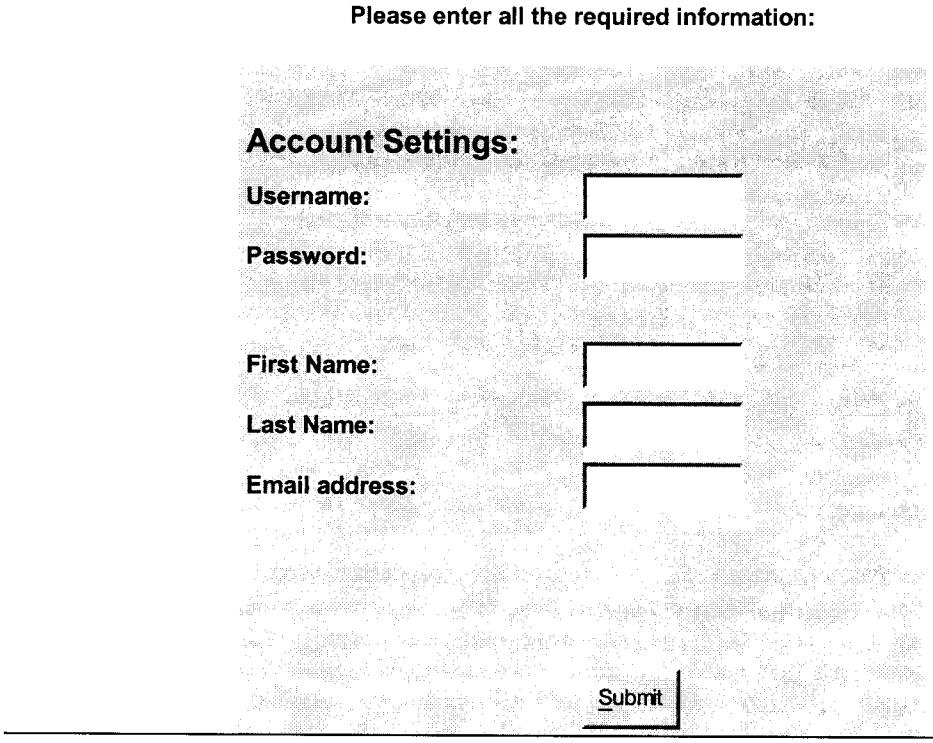
Graphical User Interface



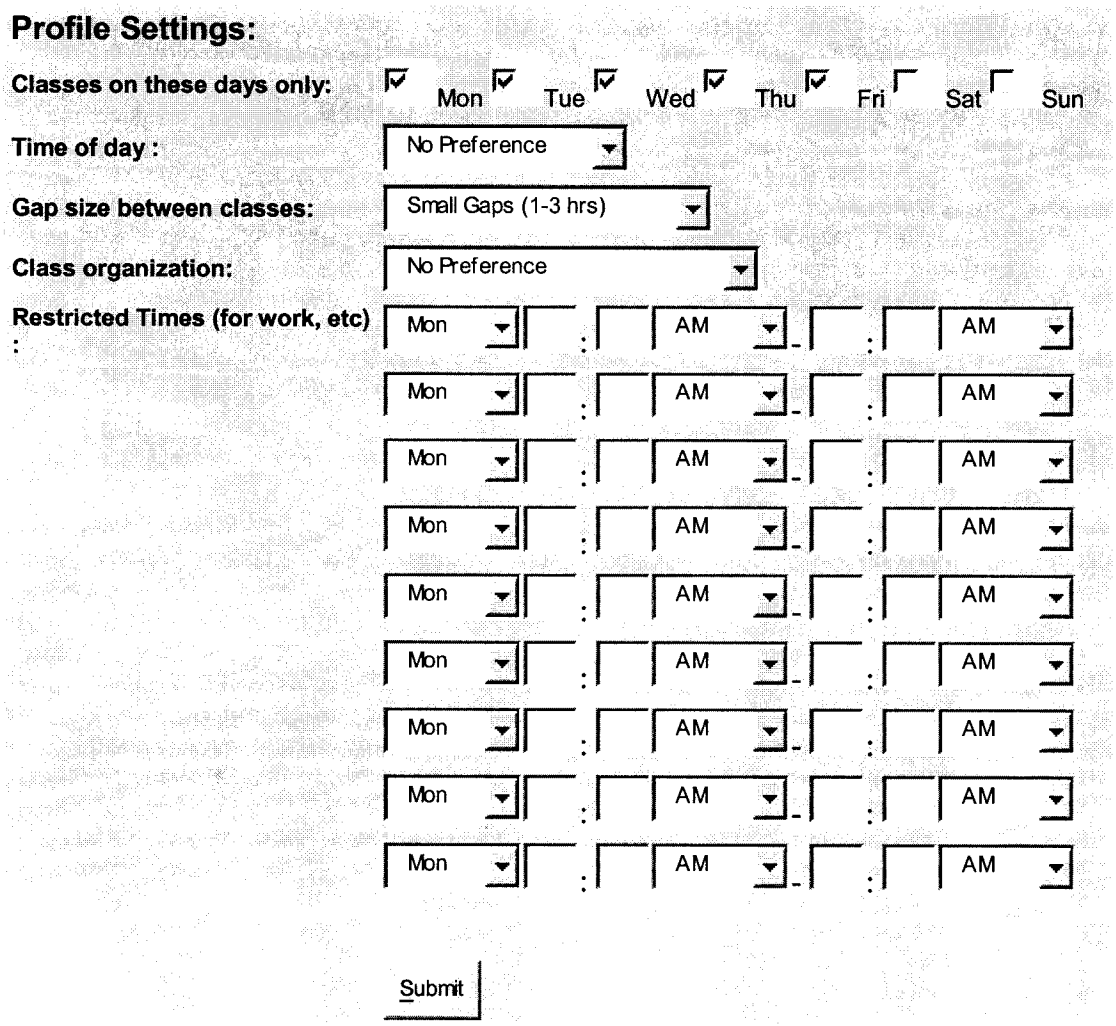
Our Fist Screen or form will be displayed when the user will activate our software, Pace ScheduleBuilder, and this screen will then appear that contains the *User Name* field with its textbox that follows, a *Password* field with its own textbox and also the *Login* and *Create New Account* Buttons. What the user will then proceed to do is simply fill in the Username fields and the Password fields and then proceed to click login. Furthermore, if the user does not have an account already established, then the user will then simply click on the create new account button. Moreover, if the login button is selected, then the class selection page will be displayed. If the new account button is selected then account settings will be displayed.



This screen will be displayed after the login button was activated. What this screen will then display are the possible choices of any six *classes* with their drop down box that follows each class set in their own field and also their choice of an *alternative class* box that also follows in each of their fields. Furthermore, the user will also have another drop down box that has the *number or preferred schedules* to be generated from either 1-6 choices. The last button on the form will be the *Generate Schedule(s)* button which will then produce all possible schedules.

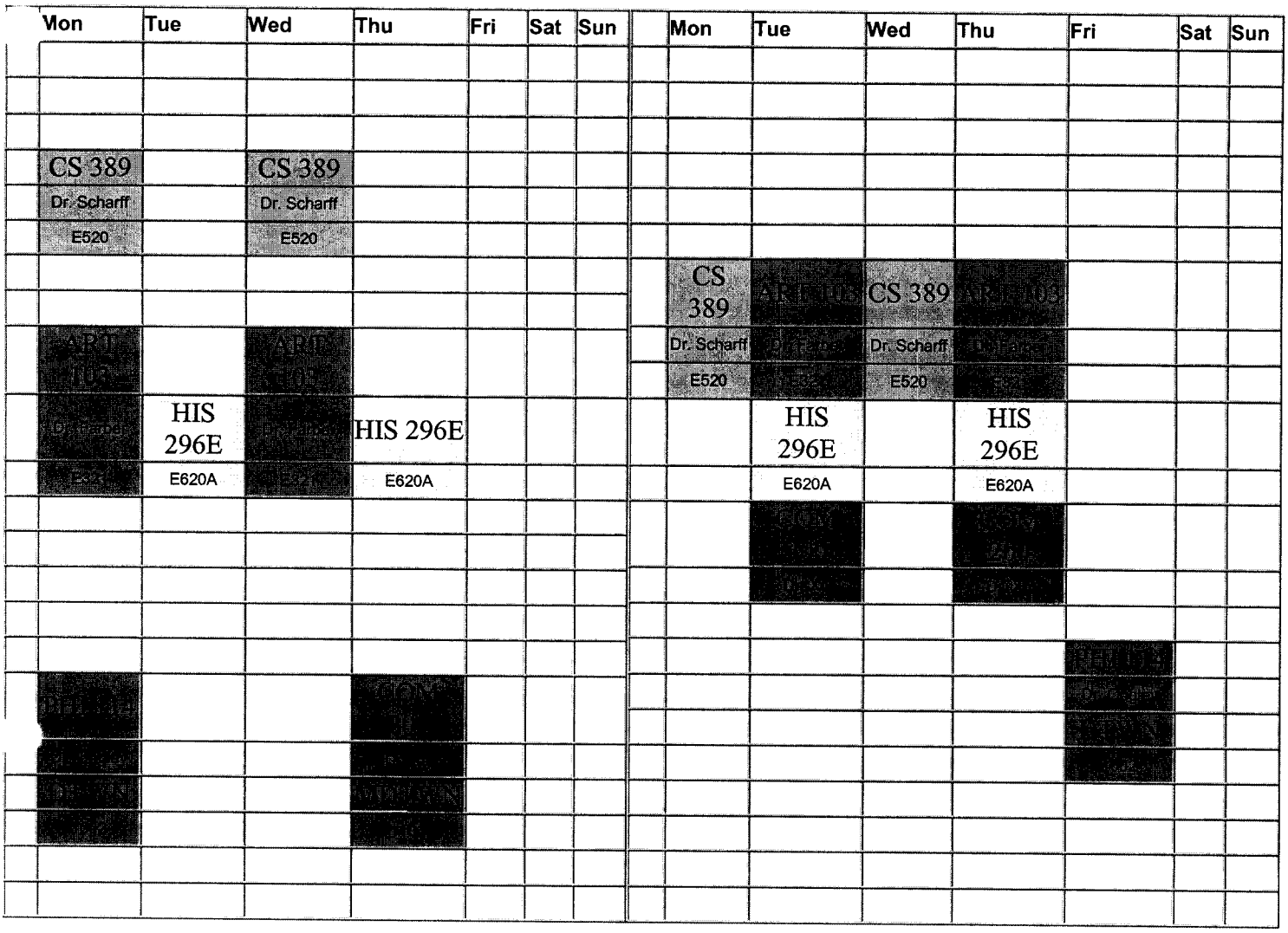


This screen will be displayed if the user does not have an account and has chosen to create a new account by activating that button. What appears on this form are all of the required data from the user such as *Username* and its text field, *Password* and its text field, *First name*, *Last Name* and the *Email Address* with each of their own text field. After the user has entered all required data, the last button, the *"submit"* button, will be to submit their date into the database.



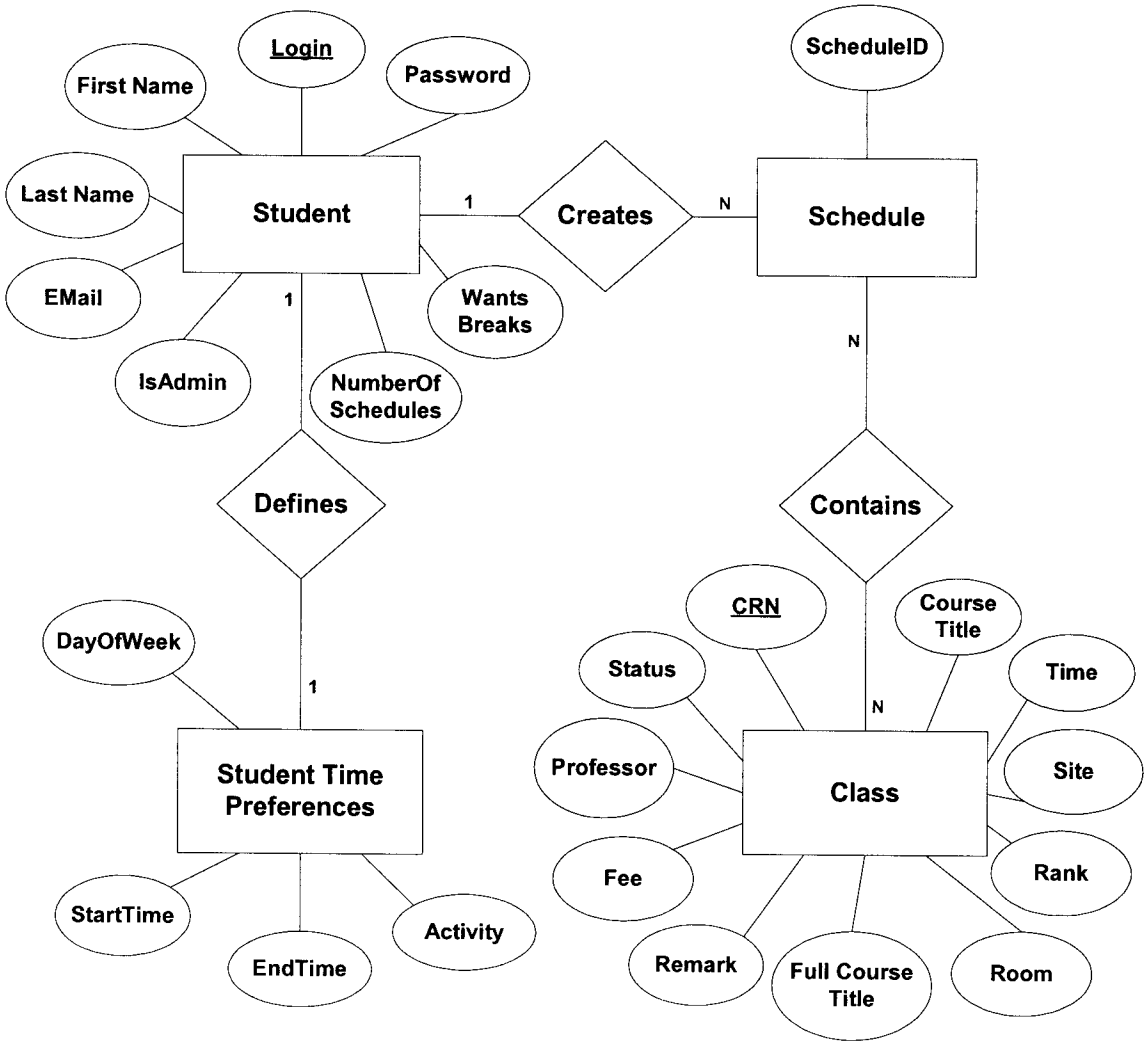
This screen will be activated after the user has set up their account, or will be displayed after their account has been processed and submitted into the database. What this screen contains are many data fields and also a new feature, which are checkboxes that exemplifies our emphasis for user convenience. What this page will display is as follows: a selection of *Days* to be selected on which days they will prefer to attend class with each day and their checkbox to follow including Saturday and Sunday, and *Time Of Day* to have class with a drop down function, a G*ap Size Between Class*, meaning how much time between each class, a *Class Organization* drop down box meaning which order they would like to have their classes and also most importantly, their *Work Hours* with each day having their own drop down box with a text field for each time and another drop down box that follow each time slot to assign whether its *am* or *pm*. The last function again on this form, which is on most of the forms, is the final button which is the *submit* button which will be submitted into the database and kept for consideration when generating the possible choices of the schedules.

Schedule 1 **Schedule 2**



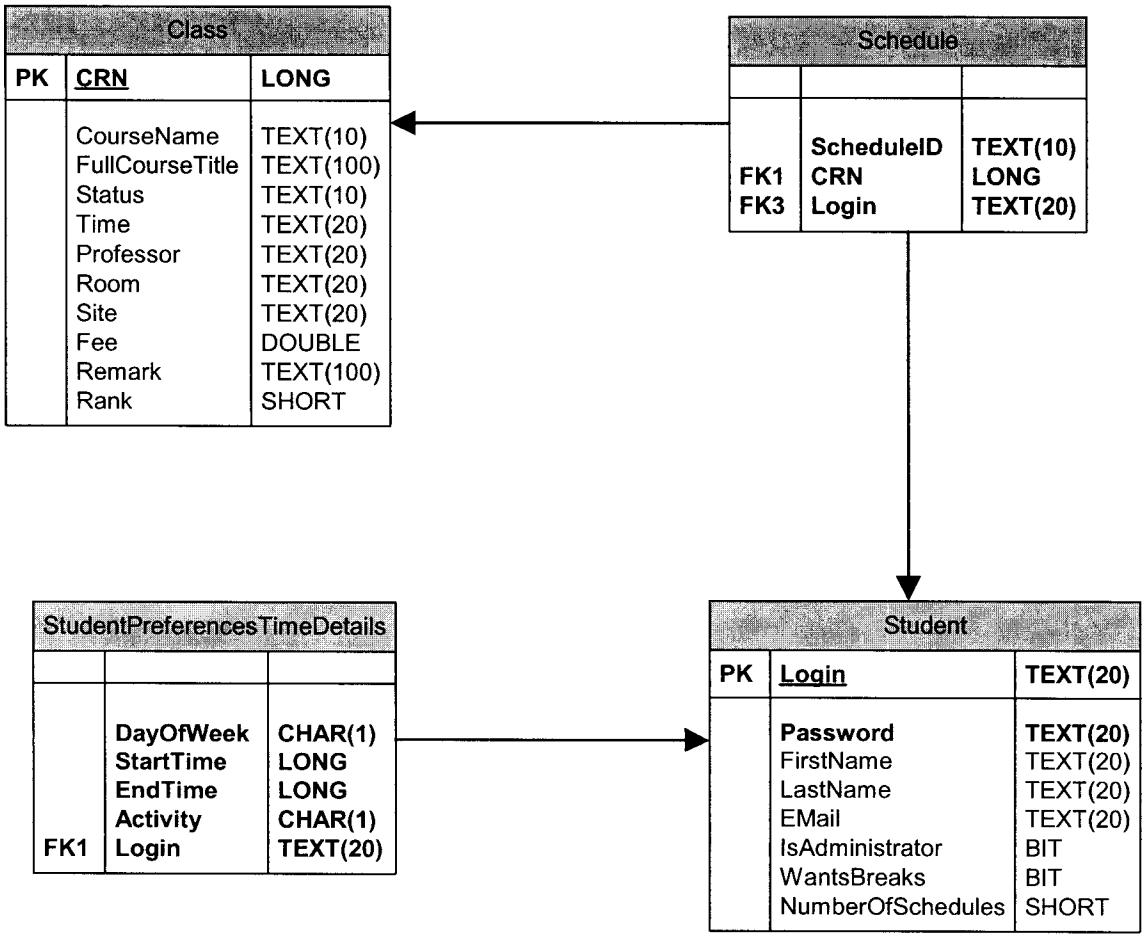
One of our last forms will be the actually production of the schedule or the actual possibilities of what his or her schedule may look like. What this form will display is basically their schedule already produced on a weekly basis chart and will have three functions, which two are of checkboxes, and one is a button. The function that are checkboxes are basically the *Save Schedule* option which will allow the user to save that desired schedule if he or she likes it and the last function will be the Continue button which will proceed to the next step.

ERD Diagram



Created with Microsoft Visio.

Database model diagram



PK stands for primary key, FK stands for foreign key. Required fields are in bold.

Created with Microsoft Visio.

Database Tables

The following is a preliminary description of database tables, along with their corresponding CREATE statements.

Student Table

The Student table contains all information pertaining to the student, who has successfully completed registration process. The Student table also contains student's preferences, such as whether he wants his/her schedule to include breaks or the number of schedules he wants the system to generate.

```
CREATE TABLE Student (
Login  VARCHAR(20) NOT NULL,
Password  VARCHAR(20) NOT NULL,
FirstName VARCHAR(20),
LastName VARCHAR(20),
Email  VARCHAR(20),
IsAdministrator NUMBER(1),
WantsBreaks NUMBER(1),
NumberOfSchedules NUMBER(2),
PRIMARY KEY (Login));
```

Schedule Table

The Schedule table contains all schedules stored by all registered users of the system.

```
CREATE TABLE Schedule(
ScheduleID  VARCHAR(10) NOT NULL,
CRN  LONG NOT NULL,
Login  VARCHAR(20) NOT NULL);
```

Class Table

The Class table contains all information for a particular class.

```
CREATE TABLE Class (
CRN VARCHAR(10),
```

```
CourseName VARCHAR(10),
FullCourseName VARCHAR(100),
Status VARCHAR(10),
Time VARCHAR(20),
Professor  VARCHAR(20),
Room VARCHAR (5),
Site VARCHAR(100),
Fee NUMBER(6,2),
Remark VARCHAR(100),
Rank NUMBER(2),
PRIMARY KEY (CRN));
```

StudentPreferencesTimeDetails Table

The StudentPreferencesTimeDetails table contains all time constraints a student chooses to enter. This allows for a convenient and easy way to store time of day or days of the week that are either desirable or undesirable (such as work hours) for a student. Activity field is used to differentiate between whether time stored is desirable or not.

```
CREATE TABLE StudentPreferencesTimeDetails (
Login VARCHAR(20) NOT NULL,
DaysOfWeek CHAR(1) NOT NULL,
StartTime INT NOT NULL,
EndTime INT NOT NULL,
Activity CHAR(1) NOT NULL);
```

Algorithms

Schedule generating module algorithms:

The overall schedule-generating procedure is expected to function in the following manner:

1. Obtain a list of courses with course information for each class
2. Purge the list of classes deemed invalid due to profile constraints
3. Assign a numeric rank to each class
4. Sort the list based on the class rank
5. Loop through and generate schedules

1. Obtain a list of classes with class information for each course

This procedure loops through the list of specified classes and retrieves the required information for each course:

```
For each class in classList {
      Get HTML from Pace.edu classschedule with specified class
            parameter
      Find starting pattern for "<table>" tag
      Find ending pattern for "<table>" tag
      Remove everything before starting pattern and after ending patter
      Split HTML string into tokens along "<td>" tags
      For each token {
            If token number > 11 {
                  Set token values to fields
                  Add course to courseList
            }
      }
}
```

2. Purge the list of classes deemed invalid due to profile constraints

This procedure loops through courseList obtained from the previous step and removed courses deemed invalid due to profile preference restrictions:

```
For each course in courseList {
      For each restrictedTime in restrictedTimeList {
            If course.time overlaps restrictedTime  {
```

```
                    Remove course from courseList
                }
        }
}
```

3. Assign a numeric rank to each class

A numeric rank for each course is assigned based on the following formula:

```
C_3 * 20 + C_1 - C2 * 5 - abs(course.time * - P_1)/30 -
(length(course.meetingTimes) - P_2) * 10
```

Profile constraints:

P_1 = time of day (Early bird = 420, Morning = 540, Afternoon = 720, Evening = 1020)
P_2 = class organization (Long classes = 3, Short classes = 1)

Class constraints:

C_1 = importance (Required = 200, Optional = 100)
C_2 = availability (# of times course offered)
C_3 = preference (position in the list)

* All times are specified in minutes since 12:00 AM

4. Sort the list based on the class rank

This will be performed using a standard "mergesort" available through the "sort" method in the Java class Collections. This algorithm offers guaranteed n log(n) performance.

Class Collections API:
http://java.sun.com/j2se/1.4.2/docs/api/java/util/Collections.html#sort(java.util.List)

Mergesort algorithm:
http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/merge/mergen.htm

5. Loop through and generate schedules

This is the process which will generate a set of schedules based on the sorted course list:

```
For n = 0 to # of schedules requested
      // start from the first conflict found in last schedule
      For each course >= conflictPosition in courseList {
```

```
            Flag = false
            For each course sc on schedule {
                  // check if already added or is a conflict
                  If (sc conflicts with course) {
                        If (conflictPosition  = 0) conflictPosition =
sc.index
                        flag = true
                  }
                  If (sc.class = course.class) flag = true

            }
            if (flag = false) Add current course to schedule
            }
}
```

Password encryption algorithm:

For password encryption the standard implementation of the SHA one-way digest algorithm will be used. It is available in java.security.MessageDigest.

Class MessageDigest API:
http://java.sun.com/j2se/1.4.2/docs/api/java/security/MessageDigest.html

Class Definitions

```
public class Course {
       private String crn,course,title,status,room,site,fee,remark,instructor;
       private MeetingTimes meetingTimes;

       public Course(String days, String mT)
              public Course(String crn, String course, String title, String status,
       String days, String meetingTimes, String room, String site, String fee, String
       remark, String instructor)
       public String getCrn()
       public String getCourse()
       public String getStatus()
       public String getRoom()
       public String getSite()
       public String getFee()
       public String getRemark()
       public String getInstructor()
       public MeetingTimes getMeetingTimes()
       public boolean overlaps(Course c)
       public MeetingTimes getOverlapPeriods(Course c)
       public String HTMLLinkToPaceCourseInfo()
       public String getJavaScriptLink()
       public String toString()
}

public class Courses extends Vector {
       public String toString()
}

public class DataMiner {
```

```
        public Courses getDataForCourse(String course, String semester)
}

public class MeetingTime {
        private char day;
        private Time startTime;
        private Time endTime;
        public MeetingTime(char day, Time startTime, Time endTime)
        public char getDay()
        public Time getStartTime()
        public Time getEndTime()
        public MeetingTime(char day, String val)
        public String toString()
        public boolean overlaps(MeetingTime m)
        public MeetingTime getOverlapPeriod(MeetingTime m)
}

class MeetingTimes extends Vector {
        public String toString()
}

public class Time implements Comparable {

        private int seconds;

        public int compareTo(Object o)
        public Time(int seconds)
        public Time(int hours, int minutes, String a)
        public Time(String val, String a)
        public String toString()
        public int getTime()
}

public class Schedule {

        private Courses courseList;

        public void addCourse(Course c)
        public void removeCourse(String crn)
        public void removeCourse(Course c)
        public int getTotalGapSize()
}

public class Schedules extends Vector {

}

public class ScheduleGenerator {
        public Schedules generateSchedules(ClassList c)
}

public class PSBCentralServlet extends HttpServlet {

        public void doGet(HttpServletRequest req, HttpServletResponse resp)
        public void doPost(HttpServletRequest req, HttpServletResponse resp)

        public boolean authenticateUser(String uname, String pass)
        public void createNewAccount(String uname, String pass, String fname, String
lname, String email)
        public void modifyAccount(String uname, String pass, String fname, String lname,
String email)
        public void deleteAccount(String uname)
        public void modifyProfile(HttpServletRequest req)
        public void createProfile(HttpServletRequest req)


}
```