DESIGN AND IMPLEMENTATION OF REGISTRATION SYSTEM
FOR THE ENGINEERING EXTENSION SERVICE
AT AUBURN UNIVERSITY

Technical Report 96-03

Feng Li

Directed By Dr. W. H. Carlisle

Department of Computer Science and Engineering
Auburn University

Auburn, Alabama

May 23, 1996
PROJECT ABSTRACT

DESIGN AND IMPLEMENTATION OF REGISTRATION SYSTEM
FOR THE ENGINEERING EXTENSION SERVICE
AT AUBURN UNIVERSITY

Feng Li
MCSE, May 23, 1996

Directed by Dr. W. II. Carlisle

In this project software is developed to convert a registration database from an old registration system to a new registration system. This software makes it feasible for the Engineering Extension Service (EESV) to upgrade their registration system. EESV Internet registration is also developed in this project. It makes it possible for clients to check new courses and register for them over the Internet. The Internet software takes advantage of Internet technology to greatly improve work efficiency and cut costs in the EESV. It is implemented using the Internet language called Java. New features and properties of the Java language and environment are explored in this study.
ACKNOWLEDGEMENTS

I would like to extend my sincere appreciation to Dr. W. H. Carlisle for serving as a major adviser during my graduate studies and for providing guidance and support for the research described in this project. I would also like to thank Dr. K. H. Chang and Dr. J. H. Cross for their guidance and direction. I would like to thank my wife Ying Xiao, my daughter Li, Fanfei and my family in China for their constant support and for keeping my goals focussed during the course of my graduate studies.

Especially, I would like to thank Mr. T. Pilsch (Director of Engineering Extension Service at Auburn University) for providing me an opportunity to work in this research area.
TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES vii

1 INTRODUCTION 1

2 REGISTRATION SYSTEM DATABASE CONVERSION 3

2.1 Background and Motivations 3

2.2 Introduction of Database Conversion Project 7

2.3 Choosing a Software Tool For the Database Conversion 10

2.4 Database Structure and Database Conversion Program Analysis 12

2.4.1 Analysis of Aceware Registration System Database 13

2.4.2 Analysis of MPFW Registration Database 18

2.5 Database Conversion Program and Result Verification 29

2.5.1 Program Functions and Interfaces 29

2.5.2 Result Verification 32

3 AN INTERNET REGISTRATION SYSTEM 37

3.1 Background and Motivations 37

3.2 Brief Introduction to Java 40
3.2.1 Features of Java Language
3.2.2 Java and Internet
3.3 Implementation of EESV Internet Registration system
   3.3.1 Client / Server Data Transition Model
   3.3.2 Database Connections
   3.3.3 Java Classes and Program Interfaces
3.4 Direction of Using EESV Internet Registration System
4 CONCLUSIONS AND FUTURE WORK
   4.1 Conclusions
   4.2 Future Work
BIBLIOGRAPHY
APPENDIX
LIST OF FIGURE

1  MeetingPro For Windows Registration System  4
2  Data Flow Chart for the Database Converting Program  9
3  Data Controls in Database Converting Program  11
4  MPFW Database Structure and Data Flow Chart  19
5  The Interface of EESV Database Converting Program  30
6  Personal Information in Aceware  33
7  Registration Information in Aceware  34
8  Personal Information in MPFW After Data Converting  35
9  Registration Information in MPFW After Data Converting  36
10 Diagram of Information Exchange Between User and WWW Server  45
11 Structure of Event Link List  50
12 Author’s Homepage  52
13 New Course Window  53
14 Window For Information About this Application  54
15 Window For Information About How to Use This Application  55
16 Window For Course Registration  56
LIST OF TABLE

1  VB Data Controls In Database Converting Program  12
2  Database Table Names.dbf Structure  14
3  Database Table Confmas.dbf Structure  15
4  Database Table Course.dbf Structure  17
5  Database Table Event.dbf Structure  21
6  Database Table Subevent.dbf Structure  22
7  Database Table Fees.dbf Structure  22
8  Database Table Person.dbf Structure  24
9  Database Table Company.dbf Structure  25
10 Database Table Registr.dbf (Registration) Structure  26
11 Database Table ARCash.dbf (Payment) Structure  27
12 Database Table ARMast.dbf (Invoice master) Structure  27
13 Database Table ARTran.dbf (Invoice items) Structure  28
14 Database Table Creditht.dbf (Credits history) Structure  28
15 Java Class Event_datasource.java Interface  47
16 Java Classes in EESV Internet Program  51
CHAPTER 1
INTRODUCTION

The motivation behind the project is that the Engineering Extension Service at Auburn University (EESV) wants to upgrade an old registration system and develop an Internet registration application mechanism. In the past, the EESV used a DOS version for a registration system which ran on a local area network within EESV offices. This local area network uses the PC-NFS [1] LAN software, and it is connected to the Engineering College Network at Auburn University. The previous registration system, called "Aceware" [7], was developed by Aceware Company in 1990. It is a DOS application. One of problems with the system is that it does not have a user-friendly interface. However, the most serious problem is that defects in this software system make the registration system incorrectly perform some functions, for example, group registration and the accounting system. In addition, as EESV has developed rapidly since 1995, the old registration system could not meet newer requirements. After doing research about registration system software market, EESV has decided to implement a new MS-Windows based registration system.

The Internet has grown at a remarkable pace. The recent growth rate of computers connected to the Internet can be characterized as nothing short of explosive. Currently in the United States alone well over 9 million computers are part of the global Internet.
The storehouse of information deposited on the Internet is vast. Accessing information resources and communication throughout the Internet has become practical, is easier, faster and cheaper than phone calls, mailing letters or sending documents by fax. The motivation for the Internet registration project comes with the rapid development of Internet applications. EESV wants to use this new technology to extend their services. In addition to upgrading the database system, another aim of this project is to develop a registration system and put it on the Internet. Clients can check courses that are offered along with detailed information about these courses by using the Internet (WWW). This extends information more widely than before, and the cost is much lower than sending course information in the mail or using another media. If clients would like to register for courses, what they need to do is open their Netscape compatible WWW Browser, find our Internet address and look at the EESV homepage. There they can look at the courses the EESV currently offers. If they decide to take certain courses, the only thing they need to do is to click on the "registration course" menu. A registration form will appear. The registration information will be transmitted to the EESV registrar in an electronic mail format after clients confirm the registration form contents.
CHAPTER 2
REGISTRATION SYSTEM DATABASE CONVERSION

2.1 Background and Motivations

In order to solve the old system problems and meet new requirements, EESV has bought new registration system from the software company called "Peopleware Company". The new registration system is called "MeetingPro For Windows" [6] (MPFW for short) which is written by FoxPro 2.6. It is a Windows based registration system running under Window 95 and Window 3.1 on a local area network. As shown in Figure 1, it has very user friendly interface. MPFW make it easy to manage all types of meetings, conferences, trade shows, seminars, classes and training courses. A variety of flexibility and controls have been provided to insure that every meeting is managed perfectly, all registrants handled with efficiency, and all revenue properly accounted.
Figure 1. MeetingPro For Windows Registration System
The first challenge is how to transfer the previous registration data from the old registration system (Aceware) to the new registration system (MPFW) without losing registration information. There are two choices:

(1) We can keep the two registration systems running for a period of time, and input the same data into these two systems. After one or two years, we can smoothly retire the old system. After we did some investigations as to how other companies who bought MPFW software tried to solve this problem, we found that it is actually an approach some companies are using to upgrade their registration systems.

(2) We can develop a program to convert the registration data from the old registration system to the new system. That means we can set a date for switching our registration system without losing any previous registration information. This can save a lot of resources and time compared to the first approach.

Obviously, the second choice sounds much better than the first one. Why then did some companies not do it? That answer is that there are four major difficulties in translation:

(1) the database format,

(2) a database scheme,

(3) complexity of the registration system,

(4) the expense.

First, let us examine the database format problem. The old DOS version registration system database format is dBASE IV. The MPFW registration system database format is FoxPro 2.5. They are two different database formats developed by two
different software companies. The dBASE database is designed by Borland software company and FoxPro is designed by Microsoft corporation. In order to get information from the dBASE database and re-distribute the information into a FoxPro database, we need a software environment which works with both the dBASE and FoxPro formats. After doing some research, the author determined that the Microsoft Visual Basic environment is one of the ideal candidates. We will give a discussion of how Visual Basic works with the two database formats in a later section.

The second challenge is the database scheme. The old Aceware registration system database is not a third normal form database. It has three large database tables to keep track of the registration information which we will discuss later. The new MPFW registration database is a relational third normal form database. It keeps thirty seven relatively small database tables. There is no easy one-to-one relationship between these two system databases. In order to convert the registration data correctly, we have to understand how the two systems work with their databases.

The MPFW is a very complicated registration system. It has thirty seven FoxPro 2.5 database tables. Each database table has an average of thirty fields. It not only keeps track of course information, person and company information, registration information and facility information, but also has an accounting system to keep track of financial information and generate financial reports. The MPFW user manual does not talk about how the system works. In fact, in order to convert the registration data from the old system to the new system, we had to examine each database table very carefully, and figure out how MPFW works. We also had the same difficulty in understanding the
Aceware system.

The last challenge we had is that this project is very expensive. Any mistakes will cost a lot of money and time once the system is put in use. After the data is converted from the old registration system to the new registration system, we do not have the chance to go back and re-convert.

Converting data from the old registration system (Aceware) to the new registration system (MPFW) has been done successfully with four months of data verification and testing.

2.2 Introduction of Database Conversion Project

As we discussed in the previous section, the Engineering Extension Service (EESV) at Auburn University purchased a new registration system to update their old registration system. The new system is called "MeetingPro", which is written by a software company called "Peopleware company". This registration system is a window based application which runs in PC Windows operating systems such as Windows 3.1, Windows 95 and Windows NT. It is a multiple users network application that manages all types of meetings such as conferences, trade shows, seminars, classes and training courses. It is a very complicated and flexible registration software with thirty seven database files. In order to convert correctly the data which is already in the old registration database to MPFW's new registration database, we have to understand how the old and new registration systems work. We need to understand not only the registration information, but also how the two registration systems interact with their databases.
The Visual Basic program was developed to convert data from the old registration system to the new one. The program is called "EESV Database Converting Program". As shown in Figure 2, it gets registration data from the three database tables of the old registration system (Aceware), and then distributes this registration data into the new registration system (MPFW). Since the new system database is a relational database and the old registration database is not relational, during the data conversion, the primary keys for the new registration database are created and added to the database tables.
Figure 2. Data Flow Chart for the VB Database Conversion Program
2.3 Choosing A Software Tool For the Database Conversion

What kind software do we need to implement our database conversion program? There are several choices. One of them is Microsoft Access which is very good at building relational database applications. There are no difficulties in using Microsoft Access to access the data stored in the dBASE format. But it has hard a time when it tries to insert data into a FoxPro database. Microsoft FoxPro is another possible candidate. Because MPFW is developed with the FoxPro software tool, it should be considered for inserting data into the MPFW new registration system database. However, it has problems when it tries to manipulate the dBASE format database of the old registration system (Aceware). The Microsoft Visual Basic 4 (VB4 for short) environment is the best software tool to meet the requirements. Microsoft VB4 has WYSIWYG (What you see is what you get) interface building functions, so it is very easy and fast to build an application. The most important fact is that VB4 has a very important custom control, called data control, for the database conversion task.

Microsoft VB4 has data control which can be hooked to a database with a variety of database formats at design-time or run-time. It is like a bi-direction "data pipe". While one end of the "data pipe" is connected to an external database table, a VB program can send and get data from the other end. The database format can be FoxPro, Paradox dBASE, Btrieve or Access. As show in Figure 2, in the database converting VB program, thirteen data controls are created, and each of them is hooked to one database table dynamically. Table 1 shows the relationship of each connection.
Figure 3. Data Controls in VB Database Conversion Program
<table>
<thead>
<tr>
<th>Custom Control</th>
<th>Registration System</th>
<th>Database Format</th>
<th>Database Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data1</td>
<td>Aceware (old)</td>
<td>dBASE IV</td>
<td>names.dbf</td>
</tr>
<tr>
<td>Data2</td>
<td>Aceware (old)</td>
<td>dBASE IV</td>
<td>course.dbf</td>
</tr>
<tr>
<td>Data3</td>
<td>Aceware (old)</td>
<td>dBASE IV</td>
<td>confmas.dbf</td>
</tr>
<tr>
<td>Data4</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>person.dbf</td>
</tr>
<tr>
<td>Data5</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>company.dbf</td>
</tr>
<tr>
<td>Data6</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>event.dbf</td>
</tr>
<tr>
<td>Data7</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>subevent.dbf</td>
</tr>
<tr>
<td>Data8</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>registration.dbf</td>
</tr>
<tr>
<td>Data9</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>fees.dbf</td>
</tr>
<tr>
<td>Data10</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>armast.dbf</td>
</tr>
<tr>
<td>Data11</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>artrtran.dbf</td>
</tr>
<tr>
<td>Data12</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>arcash.dbf</td>
</tr>
<tr>
<td>Data13</td>
<td>MPFW (new)</td>
<td>FoxPro 2.5</td>
<td>credithl.dbf</td>
</tr>
</tbody>
</table>

Table 1. VB Data Controls In Database Converting Program

2.4 Database Structure and Database Conversion Program Analysis

The Aceware and MPFW registration systems have database tables to keep track of registration data. The old registration system (Aceware) has three database tables, called names.dbf, course.dbf and confmas.dbf. As we discussed before, they are not third normal form relational database tables. Each of the three tables stores different registration information including client’s personal data, course information and transaction data. The new registration system (MPFW) has thirty seven database tables. They are third normal form relational database tables. Each of those database tables has
one or two primary keys, which links those tables together. In the database converting program, we focus only on the most important ten database tables. The reason of this simplification is based on the following assumptions:

1. Some information which MPFW needs can not be found in Aceware.
2. These ten database tables information are enough for MPFW to work as well as Aceware does.
3. Some advanced functions in MPFW can be implemented after the database conversion.

In order to convert correctly the registration data from Aceware to MPFW, we must examine carefully and understand completely the database tables's relation and structure in both systems. The VB Database Converting Program uses thirteen data controls to connect these database tables. It retrieves the data from the three Aceware database tables by data controls 1 to 3, organizes them, and distributes these data into the ten MPFW database tables by the other ten data controls 4 to 13 shown in Figure 3. The following two sections give a more detailed description and analysis of the database tables's structure and the data transformation from Aceware to MPFW.

2.4.1 Analysis of Aceware Registration System Database

The old registration system (Aceware) database is much simpler compared to the MPFW database. It has three database tables, which are names.dbf, course.dbf and confmas.dbf.
Names.dbf Database Table

The names.dbf database table is used in Aceware to keep track of a person’s information including the person’s name, social security number, address, company information and so on. The table has forty-two fields as shown in Table 2. The most important twenty fields of the names.dbf table are discussed.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Salut</td>
<td>Character</td>
<td>Salutation</td>
</tr>
<tr>
<td>2 Name1</td>
<td>Character</td>
<td>First name</td>
</tr>
<tr>
<td>3 Name2</td>
<td>Character</td>
<td>Middle name</td>
</tr>
<tr>
<td>4 Name3</td>
<td>Character</td>
<td>Last name</td>
</tr>
<tr>
<td>5 Suffix</td>
<td>Character</td>
<td>Suffix</td>
</tr>
<tr>
<td>6 Badg_name</td>
<td>Character</td>
<td>Badge name to appear on name tag (nickname)</td>
</tr>
<tr>
<td>7 Ss_no</td>
<td>Character</td>
<td>Social Security Number as primary key</td>
</tr>
<tr>
<td>8 Occupa</td>
<td>Character</td>
<td>Job title</td>
</tr>
<tr>
<td>9 Organiza</td>
<td>Character</td>
<td>Company the person belongs to</td>
</tr>
<tr>
<td>10 Address</td>
<td>Character</td>
<td>First line of address</td>
</tr>
<tr>
<td>11 Address2</td>
<td>Character</td>
<td>Second line of address</td>
</tr>
<tr>
<td>12 City</td>
<td>Character</td>
<td>City name</td>
</tr>
<tr>
<td>13 State</td>
<td>Character</td>
<td>State name</td>
</tr>
<tr>
<td>14 Zip</td>
<td>Character</td>
<td>Zip code</td>
</tr>
<tr>
<td>15 Country</td>
<td>Character</td>
<td>Country name</td>
</tr>
<tr>
<td>16 Dayphone</td>
<td>Character</td>
<td>Daytime phone number</td>
</tr>
<tr>
<td>17 Homephone</td>
<td>Character</td>
<td>Home phone number</td>
</tr>
<tr>
<td>18 Faxphone</td>
<td>Character</td>
<td>Fax number</td>
</tr>
<tr>
<td>19 Sex</td>
<td>Character</td>
<td>Gender</td>
</tr>
<tr>
<td>20 Homeaddr</td>
<td>Logical</td>
<td>To determine the above address is home address</td>
</tr>
</tbody>
</table>

Table 2. Database Table Names.dbf Structure
The primary key of the database table names.dbf is the person's social security number. It is the only foreign key which is used to link to other two database tables in Aceware. Table 2 is very straightforward. If a certain field is not appropriate for a person, it contains null. The "homeaddr" field contains one byte of logical information which is used to determine whether the address is the company's or home address information. Data control, data1, is hooked to names.dbf dynamically when the program is being executed. It retrieves the personal information from names.dbf table.

**Confmas.dbf Database Table**

The confmas.dbf database table stores the course information. It has eighty one fields to keep track of course information, such as course identification number, name, beginning date, ending date, location, account associated with this course, course fee and so on. The most important twenty fields are shown in Table 3.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Active</td>
<td>To test if course is active or inactive</td>
</tr>
<tr>
<td>2</td>
<td>Confno</td>
<td>Course ID, Unique ID</td>
</tr>
<tr>
<td>3</td>
<td>Confname</td>
<td>Course name or conference name</td>
</tr>
<tr>
<td>4</td>
<td>Begdate</td>
<td>Course beginning date</td>
</tr>
<tr>
<td>5</td>
<td>Enddate</td>
<td>Course ending date</td>
</tr>
<tr>
<td>6</td>
<td>Conftime</td>
<td>Time of day</td>
</tr>
<tr>
<td>7</td>
<td>Location</td>
<td>Course location information</td>
</tr>
<tr>
<td>8</td>
<td>Misdcode</td>
<td>Account number associated with course</td>
</tr>
<tr>
<td>9</td>
<td>Minsize</td>
<td>Minimum course size to make a &quot;Go&quot;</td>
</tr>
<tr>
<td>10</td>
<td>Maxsize</td>
<td>Maximum registrations</td>
</tr>
<tr>
<td></td>
<td>Column Name</td>
<td>Data Type</td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>11</td>
<td>RegFee</td>
<td>Numeric</td>
</tr>
<tr>
<td>12</td>
<td>Fee2</td>
<td>Numeric</td>
</tr>
<tr>
<td>13</td>
<td>Coord</td>
<td>Character</td>
</tr>
<tr>
<td>14</td>
<td>Ceus</td>
<td>Numeric</td>
</tr>
<tr>
<td>15</td>
<td>Hours</td>
<td>Numeric</td>
</tr>
<tr>
<td>16</td>
<td>Tch_name</td>
<td>Character</td>
</tr>
<tr>
<td>17</td>
<td>Tch_phone</td>
<td>Character</td>
</tr>
<tr>
<td>18</td>
<td>Tch_ssn</td>
<td>Character</td>
</tr>
<tr>
<td>19</td>
<td>Enroll</td>
<td>Numeric</td>
</tr>
<tr>
<td>20</td>
<td>Cancel</td>
<td>Logical</td>
</tr>
</tbody>
</table>

Table 3. Database Table Confmas.dbf Structure

The primary key of the confmas.dbf database table is the course identification number. Each course has a unique identification number. It is used to link another two database tables to keep track of registration information. There is the field "Active" which controls the registration status. For example, if a course is set to inactive, it can be viewed, but any registration action is not allowed to alter the information. In the VB database conversion program, the data control, Data3, is created and dynamically connected to the confmas.dbf database table to retrieve relative information. Once it finishes its task, it is automatically disconnected from the database table.

**Course.dbf Database Table**

The last database table in Aceware we discuss is the course.dbf database table which actually stores transaction information. It holds information about who registers
what course, how much registration he or she needs pay, how many CEU(Continuing Education Unit) credits he or she has owned and so on. It also keeps track of very important financial information about payments, payment adjustment, receipts, refunds and so on. The course.dbf database table has a total of fifty five fields. In Table 4, the most significant twenty fields are illustrated.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Date</td>
<td>Creation date for this transaction (record)</td>
</tr>
<tr>
<td>2</td>
<td>Character</td>
<td>Person last name</td>
</tr>
<tr>
<td>3</td>
<td>Character</td>
<td>Person first name</td>
</tr>
<tr>
<td>4</td>
<td>Character</td>
<td>Person social security number</td>
</tr>
<tr>
<td>5</td>
<td>Date</td>
<td>Last modified date</td>
</tr>
<tr>
<td>6</td>
<td>Character</td>
<td>Course or conference id number</td>
</tr>
<tr>
<td>7</td>
<td>Numeric</td>
<td>Registration fee associated with a course</td>
</tr>
<tr>
<td>8</td>
<td>Character</td>
<td>Payment adjustment description</td>
</tr>
<tr>
<td>9</td>
<td>Numeric</td>
<td>Payment adjustment amount</td>
</tr>
<tr>
<td>10</td>
<td>Numeric</td>
<td>Total first payment</td>
</tr>
<tr>
<td>11</td>
<td>Character</td>
<td>First payment method</td>
</tr>
<tr>
<td>12</td>
<td>Character</td>
<td>First payment receipt number</td>
</tr>
<tr>
<td>13</td>
<td>Character</td>
<td>Payer name of first payment</td>
</tr>
<tr>
<td>14</td>
<td>Numeric</td>
<td>Total second payment</td>
</tr>
<tr>
<td>15</td>
<td>Character</td>
<td>Second payment method</td>
</tr>
<tr>
<td>16</td>
<td>Character</td>
<td>Second payment receipt number</td>
</tr>
<tr>
<td>17</td>
<td>Character</td>
<td>Payer name of second payment</td>
</tr>
<tr>
<td>18</td>
<td>Numeric</td>
<td>Number of CEU credits awarded for this course</td>
</tr>
<tr>
<td>19</td>
<td>Numeric</td>
<td>Number of contact hours awarded</td>
</tr>
<tr>
<td>20</td>
<td>Logical</td>
<td>To test if this registration is cancelled or not</td>
</tr>
</tbody>
</table>

Table 4. Database Table Course.dbf Structure
The fields "Ss_no" and "Confno" are the foreign keys in the course.dbf table to the names.dbf and confmas.dbf tables. There is no unique primary key in the course.dbf table. In order to improve access efficiency, there are fields to hold person and course information. If more detailed information about a course or person, the course number or person social security number is used to search the confmas.dbf or names.dbf tables. Every record in the course.dbf table is a record of each registration and payment. The records will be modified or overwritten each time the registrar modifies that registration. The field "Entrdate" stores the registration date, at the time the registration fee is added or other information is adjusted. The "Date" field is used to keep track of the date of latest modification.

In the VB database conversion program, the data control, Data3, is created and connected to the course.dbf database table dynamically as shown in Figure 3. It retrieves the registration and payment information from the course.dbf database table. Once it finishes its task, it is automatically disconnected from the database table.

2.4.2 Analysis of MPFW Registration System Database

The new registration system MPFW database with its thirty seven database tables is much more complicated compared to the Aceware database. These tables are all third normal form relational database tables. Each table has one or two unique foreign keys which links it to other tables. Figure 4 shows the database table relations and data flow within MPFW. As in the previous section, the most significant database tables are discussed. In the VB database converting program, the ten data controls are created and
connected to these database tables. We will analyze them and explain how these tables are filled with data information obtained from the Aceware tables.

Figure 4. MPFW Database Structure and Data Flow Chart
Event, Subevent and Fees Database Tables

The MPFW registration system uses the generic terms event and subevent to define and organize the various types of courses and conferences. The event ID identifies a group of related subevents. Each specific course or meeting with a unique set of registrants is a subevent within an event group. The event.dbf table in the MPFW database stores information about event information including event ID, event name and so on. The subevent.dbf table stores a specific subevents's information. It includes the event ID, the subevent ID, the subevent name and so on. The fees.dbf table holds the registration fee type and fee amount associated with each subevent. One subevent might have more than one type of registration fee associated with a different category of registrants. The most important six fields in the event.dbf table and nine fields in the subevent.dbf table are shown in Table 5 and Table 6. Table 7 shows the most significant five fields in the fees.dbf database table.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Eventid</td>
<td>Character</td>
<td>Event ID, Primary key</td>
</tr>
<tr>
<td>2 Name</td>
<td>Character</td>
<td>Event name</td>
</tr>
<tr>
<td>3 Begin_date</td>
<td>Date</td>
<td>Event beginning date</td>
</tr>
<tr>
<td>4 End_date</td>
<td>Date</td>
<td>Event ending date</td>
</tr>
<tr>
<td>5 Descrip</td>
<td>Memo</td>
<td>Description of event</td>
</tr>
<tr>
<td>6 Status</td>
<td>Numeric</td>
<td>Status (1=Open, 2=Booked, 3=Canceled, 4=Closed)</td>
</tr>
</tbody>
</table>

Table 5. Database Table Event.dbf Structure

21
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventid</td>
<td>Character</td>
<td>Event ID, Primary key</td>
</tr>
<tr>
<td>Subeventid</td>
<td>Character</td>
<td>Subevent ID, Primary key</td>
</tr>
<tr>
<td>Name</td>
<td>Character</td>
<td>Subevent Name</td>
</tr>
<tr>
<td>Descrip</td>
<td>Memo</td>
<td>Description of subevent</td>
</tr>
<tr>
<td>Begin_date</td>
<td>Date</td>
<td>Subevent beginning date</td>
</tr>
<tr>
<td>End_date</td>
<td>Date</td>
<td>Subevent ending date</td>
</tr>
<tr>
<td>Reg_min</td>
<td>Numeric</td>
<td>Minimum number of registrations needed</td>
</tr>
<tr>
<td>Reg_max</td>
<td>Numeric</td>
<td>Maximum number of registrations allowed</td>
</tr>
<tr>
<td>Status</td>
<td>Numeric</td>
<td>Status(1=Open, 2=Booked, 3=Canceled, 4=Closed)</td>
</tr>
</tbody>
</table>

Table 6. Database Table Subevent.dbf Structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventid</td>
<td>Character</td>
<td>Event ID</td>
</tr>
<tr>
<td>Subeventid</td>
<td>Character</td>
<td>Subevent ID</td>
</tr>
<tr>
<td>Priority</td>
<td>Numeric</td>
<td>Order of the fees in the list (more one type of fee)</td>
</tr>
<tr>
<td>Fee</td>
<td>Numeric</td>
<td>Amount of the fee</td>
</tr>
<tr>
<td>Name</td>
<td>Character</td>
<td>Name of the fee</td>
</tr>
</tbody>
</table>

Table 7. Database Table Fees.dbf Structure

The primary key in the event.dbf table is eventid. The primary key in the subevent.dbf table is both eventid and subeventid. The primary key in the fees.dbf table is eventid, subeventid and priority. The information for the event.dbf, subevent.dbf and fees.dbf tables in MPFW can be obtained from the confmas.dbf table in Aceware. Since Aceware registration system does not use event and subevent notions to organize courses
and conferences (it organizes a course as a single "event"), the problem can be solved during the database conversion by treating an event as a subevent. In other words, after database conversion, each event has only one subevent associated with it, and the event and subevent information are identical.

In the VB database conversion program, data controls, Data6, Data7 and Data9, are created and connected to the event.dbf, the subevent.dbf and the fees.dbf tables. The course information which is retrieved by data control Data3 is placed in the event.dbf table and the subevent.dbf table. The course number in the course.dbf table becomes the eventid in the event.dbf table, as well as the eventid and subeventid in the subevent.dbf and fees.dbf tables. The course name goes to the Name field in the event.dbf and subevent.dbf tables. The registration fee information in the course.dbf is transferred into the fees.dbf table. If more than one type of registration fee is defined in the course.dbf table, a new record space is allocated in the fees.dbf table, and it is filled by the new type of fee.

In general, the course information in the Aceware registration system is retrieved by data control (Data3). The course information is organized and distributed by the VB database conversion program into the MPFW registration system through three data controls ( Data6, Data7, Data9). Once it finished transferring the last course information, those data controls (like data "pipes") are disconnected from the database tables.
Person and Company Database Tables

The person and company tables in the MPFW database store the information about people and companies. Table 8 shows the eleven major fields of the person.dbf table. The primary key of this table is personid, which in this case is a person’s social security number. Table 9 shows the four major fields of company.dbf table. The primary key of this table is the companyid, which can join these two tables together.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Lastname</td>
<td>Character</td>
<td>Last name</td>
</tr>
<tr>
<td>2 Firstname</td>
<td>Character</td>
<td>First name</td>
</tr>
<tr>
<td>3 Mi</td>
<td>Character</td>
<td>Middle name</td>
</tr>
<tr>
<td>4 Personid</td>
<td>Character</td>
<td>Person ID, Primary key</td>
</tr>
<tr>
<td>5 Role_type</td>
<td>Numeric</td>
<td>Key role 1=Staff, 2=Instructor, 3=Registrant</td>
</tr>
<tr>
<td>6 Title</td>
<td>Character</td>
<td>Title</td>
</tr>
<tr>
<td>7 Badgenme</td>
<td>Character</td>
<td>Badge name</td>
</tr>
<tr>
<td>8 Jobsitle</td>
<td>Character</td>
<td>Job title.</td>
</tr>
<tr>
<td>9 Companyid</td>
<td>Character</td>
<td>Company ID</td>
</tr>
<tr>
<td>10 Addr1</td>
<td>Character</td>
<td>Address line 1</td>
</tr>
<tr>
<td>11 Addr2</td>
<td>Character</td>
<td>Address line 2</td>
</tr>
</tbody>
</table>

Table 8. Database Table Person.dbf Structure
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Companyid</td>
<td>Character</td>
</tr>
<tr>
<td>2</td>
<td>Name</td>
<td>Character</td>
</tr>
<tr>
<td>3</td>
<td>Name_ext</td>
<td>Character</td>
</tr>
<tr>
<td>4</td>
<td>Addr1</td>
<td>Character</td>
</tr>
</tbody>
</table>

Table 9. Database Table Company.dbf Structure

The person and company information in these two database tables can be obtained from the names.dbf table in Aceware by the data control (Data1). As we discuss, for the data conversion in the event and subevent tables, the VB database conversion program creates data controls, Data4 and Data5. They are connected to the person and company tables and transfer the information. The person's security number in the names.dbf table can be used as the person ID in person.dbf table. The only problem is that there is no company ID in the Aceware names.dbf table. The VB program can create a unique ID for it. The VB program also can determine duplicate company information in the names.dbf table and eliminate it.

Registration, Payment, Invoice and Credit History Database Tables

Tables 10 through 14 show the most significant of fields of the registration, payment, invoice master, invoice item and credit history database tables. The registration database table registr.dbf (shown in Table 10) keeps information of each registration. The primary key is the registration number, which is a unique number for each registration transaction. The payment table ARCash.dbf (shown in Table 11) stores each payment or
refund transaction. The registration number in both the registr.dbf and ARCash.dbf tables is used to link to these two database tables together to keep track of registration and payment data. Every time a payment is entered into the system, there is an unique receipt number created and associated with this payment. The receipt number in ARCash.dbf is the primary key for this table.

Generating an invoice is an option in MPFW. If an invoice is generated, the system automatically assigns an invoice number and stores the invoice information into the master and items database tables (shown in Table 12 and Table 13). The invoice number is a primary key to join these two database tables. The master invoice table (shown in Table 12) has data about the person or company to which the invoice is issued. It also has the invoice amount, previous payment, remaining balance and so on. The invoice items table (shown in Table 13) is a database table which stores information about which registration the invoice is issued.

The credit history database table creditth.dbf (shown in Table 14) stores the credits a person has earned. The primary keys are the combination of person, event and subevent ID’s. This database table keeps track of how many credits were earned by a person for a subevent.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eventid</td>
<td>Event ID</td>
</tr>
<tr>
<td>2</td>
<td>Subeventid</td>
<td>Subevent ID</td>
</tr>
<tr>
<td>3</td>
<td>Regid</td>
<td>Registration ID, Company ID or Person ID</td>
</tr>
<tr>
<td>4</td>
<td>Regno</td>
<td>Registration Number, Primary key</td>
</tr>
<tr>
<td></td>
<td>Field Name</td>
<td>Type</td>
</tr>
<tr>
<td>---</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>1</td>
<td>Invno</td>
<td>Character</td>
</tr>
<tr>
<td>2</td>
<td>Custno</td>
<td>Character</td>
</tr>
<tr>
<td>3</td>
<td>Method</td>
<td>Character</td>
</tr>
<tr>
<td>4</td>
<td>Receiptno</td>
<td>Character</td>
</tr>
<tr>
<td>5</td>
<td>Regno</td>
<td>Character</td>
</tr>
</tbody>
</table>

Table 11. Database Table ARCash.dbf (Payment) Structure

<table>
<thead>
<tr>
<th></th>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Invno</td>
<td>Character</td>
<td>Invoice Number</td>
</tr>
<tr>
<td>2</td>
<td>Custno</td>
<td>Character</td>
<td>Customer ID (Person ID or Company ID)</td>
</tr>
<tr>
<td>3</td>
<td>Invamt</td>
<td>Numeric</td>
<td>Invoice Amount</td>
</tr>
<tr>
<td>4</td>
<td>Paidamt</td>
<td>Numeric</td>
<td>Amount paid</td>
</tr>
<tr>
<td>5</td>
<td>Balance</td>
<td>Numeric</td>
<td>Remaining balance</td>
</tr>
</tbody>
</table>

Table 12. Database Table ARMast.dbf (Invoice master) Structure
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invno</td>
<td>Character</td>
<td>Invoice number</td>
</tr>
<tr>
<td>Custno</td>
<td>Character</td>
<td>Customer ID (Person ID or Company ID)</td>
</tr>
<tr>
<td>Eventid</td>
<td>Character</td>
<td>Event ID</td>
</tr>
<tr>
<td>Subeventid</td>
<td>Character</td>
<td>Subevent ID</td>
</tr>
<tr>
<td>Regno</td>
<td>Character</td>
<td>Registration number</td>
</tr>
<tr>
<td>Price</td>
<td>Numeric</td>
<td>Registration fee</td>
</tr>
</tbody>
</table>

Table 13: Database Table ARTran.dbf (Invoice items) Structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personid</td>
<td>Character</td>
<td>Person ID</td>
</tr>
<tr>
<td>Eventid</td>
<td>Character</td>
<td>Event ID</td>
</tr>
<tr>
<td>Subeventid</td>
<td>Character</td>
<td>Subevent ID</td>
</tr>
<tr>
<td>Credits</td>
<td>Numeric</td>
<td>Number of credits earned</td>
</tr>
</tbody>
</table>

Table 14: Database Table Creditht.dbf (Credits history) Structure

The information required by these database tables (Table 10 to Table 14) in MPFW can be transferred from the course.dbf database table in Aceware. The VB database converting program creates five data "pipe" to read from these database tables. The information is retrieved by the data "pipe" (Data2), and organized by the VB program. Then, the organized information is distributed by another five data "pipes" (Data8, Data10 to Data 13) into the registration, payment, invoice master, invoice items and creditht database tables. The unique registration and invoice number are generated by the VB program as primary keys for the registration, invoice master and invoice items.
tables. Other data converting actions of the EESV database conversion program are similar to what we discussed in the previous sections.

2.5 Database Converting Program and Result Verification

2.5.1 Program Functions and Interfaces

The EESV database conversion program is developed with Microsoft Visual 4 Professional Edition. It contains two forms and one code module. Form2, shown in Figure 3 in the previous section, is a Form which users cannot access. It is used to place the data controls. As we discussed, the data controls act as data "pipes", with data 1 through data 3 connected to source database tables in Aceware, data4 through data13 are connected to destination database tables in MPFW. As shown in Figure 5, Form 1 is the interface between users and the VB program. We will discuss the principal components of the interface.
Figure 5. The Interface of EESV Database Conversion Program
(1) **Date Control Option**

There are two pull down lists with captions of "Month" and "Year". The user can select a month and year by using these two lists. The VB program will convert the data in Aceware subsequent to the date into MPFW. Any Aceware information before that date will be ignored during data conversion. This function gives the user a way to filter out old information and improve the efficiency of the new registration system.

(2) **Source and Destination Options**

The source destination driver and directory selection interfaces give the user more flexibility in where the source and destination database could be located. For example, one database can be on your local hard disk and the other can across a network.

(3) **Control Panel**

The control panel contains five command push buttons to control starting and ending of the program. The "Start Program" button is used to initialize the program after date, divers and directories are chosen. The "Terminate Program" button is to terminate the program after the job has been done. The "Try Again" button is to re-initialize the program to re-do data conversion without quitting the VB program. The "Information" button is to provide help information about how to use this program.

(4) **Program Information**

The program information window provides program status information. For
example, it tells the user when it waits for input data or user’s actions, and when its execution is reading or writing a database. If an error occurs, it shows the error information to help the user understand the problem and the programmer to debug the program.

2.5.2 Result Verification

In the Aceware registration system, there are about 10000 person records in the names.dbf table, about 17000 registration and payment transaction records in the course.dbf table and about 400 course and conference records in the confmas.dbf table. It takes about 5 hours for the database conversion program to convert all the data from Aceware to MPFW.

It took about four months to test the correctness of the data conversion program. Most testing work focused on verifying the consistency of the data information in both registration systems. Once an inconsistency was detected, the program needed to be debugged and modified. We continued to do a new test each time the program was modified, repeating the test and debug cycle until no further errors were detected.

Data in the Registration Systems

Figures 6 and 7 show a person’s record and registration record in the Aceware database, and Figures 8 and 9 show same information in the MPFW after data conversion.
DEO: BR  *** EDIT REGISTRANT INFORMATION *** ENTERED 950928 (YYMMDD)

< Print Label? F >

LAST UPDATE: 10/17/95

BADGE NAME

SAL Ms. F Susan M C. L Atherton SS/ID# 000-00-2061

TITLE Visiting Associate Dean

COMPANY/ORG Suffolk University Total Courses Taken= 1

ADDRESS School of Management <Home Address? N>

ADDRESS 2 Beacon Hill 8 Ashburton Place

CITY Boston STATE MA ZIP 02108-2770 DAY ( ) +

NIGHT ( ) -

Optional: Dues Expire // UDPC2 <undefined> FAX ( ) -

Fields: GIST Phone Indiv Memb. F OK to MAIL? Y

COMMENT FLAG SOURCE

Org Code Job Code MAIL CODES 1 2 3 4 5

+----------------------------------------+
| Here's what we have on file for Atherton |
+----------------------------------------+

(E)dit record >(N)o change (add course)

<ESC> - QUIT/RETURN

F10 to view courses taken

Figure 6. Personal Information in the Aceware
Figure 7. Registration Information in the Aceware
Figure 8. Personal Information in MPFW After Data Conversion
Figure 9. Registration Information in MPFW After Data Conversion
Figure 10  Diagram of Information Exchange Between User and WWW Server
CHAPTER 3
AN INTERNET REGISTRATION SYSTEM

3.1 Background and Motivations

During the past year a revolutionary change has been happening within the world's telecommunications networks. It is called the World Wide Web, and its growth has been nothing less than phenomenal. Of course, no revolution comes out of nowhere, and so it is with the World Wide Web. Developments, both technological and social, have paved the way for the emergence of the Web. Computers, once confined only to those who could master the arcane mysteries of programming languages, are now part of the everyday world of business people, artists, and schoolchildren. The Internet, a set of protocols that permitted universities and other large organizations to exchange data, has now become the hottest trend of the nineties. The World Wide Web ties the two together, breaking the physical barriers of Cyberspace to establish the foundation of a global electronic village. The World Wide Web provides a means of accessing the resources of the Internet without requiring the user to know how those resources are transmitted and sorted. The Web hypermedia paradigm expands the potential of the low-cost method of providing information, opinions, and art to a world-wide audience of millions.
The motivation behind an EESV Internet registration system project is the rapid development and increasing use of the Internet WWW for information access. EESV wants to take advantage of this new technology to expand its service. EESV used to send letters or call its potential clients. It was very difficult and expensive to advertise services because it requires a lot of human resources and it is not easy to know exactly who will be interested in the services.

Developing an Internet registration system is one of the best solutions to these problems. The advantages of an Internet registration system can be summarized as follows:

(1) No limitation of pre-assuming categories for our potential clients.

(2) No geographic or time limitations.

(3) Low-Cost

(4) Fast

By putting new courses on the Internet, anyone who is interested in our courses can look at them and find the one he or she needs. Anyone can access a new course database. There is no geographic limitation in the service. People anywhere in the world can query the new course database through the Internet. They can do it any time and from anywhere. An Internet registration system will provide a low-cost way to distribute course information. The Internet registration information is stored in an WWW server at Auburn University Engineering Network. Only the database tables which store the course information are copied to the WWW server, and these represent about 100 k bytes. This costs much less than other information distribution approaches. There is no paper
work required to distribute course information once the Internet registration is implemented. When course information is updated or new courses are added into the registration database at EESV, one need only to copy the updated course database to the WWW server. This can be done automatically. The course information in the WWW server is updated when the course database at EESV is updated, so clients can always access the most recent course information at EESV.

How can one implement an Internet registration system? It is very easy to create HTML files which contain new course descriptions, these HTML files need to be manually modified if new courses are added. This is time consuming work. The content of these HTML files are basically static. In order to solve this problem and create an Internet application with nice interfaces, Java applets are used to build an Internet application which allows a clients to query the new course database through the Internet. This Internet registration application was completed under supervision and direction of Dr. Carlisle in the Department of Computer Science and Engineering at Auburn University. Currently, information about the software can be accessed at the author’s homepage (www.eng.auburn.edu/~lifeng1) for alpha testing. It will be linked to EESV homepage for beta testing. This Internet registration application has very nice GUI interfaces, so it is very easy for our clients to operate it. It does not assume that users have any Internet knowledge.

For security reasons, we do not want users to interact directly with EESV’s registration database tables at the EESV offices. The new course related database tables are copied to the WWW server, and users are only allowed to retrieve information from
these database tables at the WWW server. These database tables can be read, but they cannot be modified. We reserve the detailed discussion about the system implementation and difficulties encountered for later sections.

3.2 Brief Introduction to Java

The Java environment provides the means for distributing dynamic content through applets in hypertext documents, platform independent stand-alone applications, and protocol handlers. With this functionality comes the means to develop the future of the Internet such as intelligent agents, interactive 3D worlds, self-updating software and multimedia utilities. Java provides this functionality through its object-oriented structure, robust environment, and multithreaded capabilities. Because of these features, Java can be employed for creating demanding applications, such as VRML engines and intelligent agents.

3.2.1 Features of Java Language

Java Is Object-Oriented The fundamental unit in object-oriented computing is the object. Object templates in Java are called classes. A Java class is a grouping of code and storage that models in software the behavior of objects. Java code is organized by classes. Each class defines a set of methods that define the behavior of an object and a set of variables that keep the state of that object. Controlling access to variables and methods is accomplished through access modifiers. Java defines four levels of access modifier[3]: public, protected, private, and friendly.
**Java Is Simple** Java is very similar in syntax to C++, but it is much simpler. It was deliberately designed to look and feel like C++, which means C++ programmers will have an easier time converting to the new language. But they will also have to change some old habits. The differences between Java and C++ can be summarized as follows,

- no structure or unions
- no preprocessor
- no pointers
- no multiple inheritance
- no stand-alone functions
- no goto
- no operator overloading
- no automatic coercion

Java has eliminated structures and unions. There is no pre-processor, Java uses interfaces instead of header files. Individual functions have been dropped from Java. Any functions needed must be encapsulated in a class. Java drops multiple inheritance and replaces it with interfaces. Java has also removed the goto, operator overloading, automatic coercion, and pointers.

**Java Is Static Typed** In a Java program, the type of the objects (number, characters, arrays, etc.) must be defined. This helps programmers find potential problems early because type errors can be detected when a program is complied.
Java Is Compiled And Architecture Neutral  When running a Java program, it is first compiled to byte-code. Byte-codes are very similar to machine instructions, so Java program can be very efficient. However, byte-codes are not specific to a particular machine; they can run on any platform that supports Java. It is not necessary to recompile a Java source code to run on a new platform.

Java Is Multi-Threaded  A major element of the Java architecture is its inclusion of multithreading at every level. Multithreading begins at the syntactical level with synchronization modifiers included in the language. At the object level, the class libraries allow the creation of threaded applications by inheriting classes developed for this purpose. Actually, the Java run-time environment uses multithreading in areas such as background garbage collection to speed performance while retaining usability. Programming in a multi-threaded environment is usually difficult because many things can happen at the same time or in an unpredictable order. Java provides easy-to-use features for synchronization that make programming easier.

3.2.2 Java and Internet

The Internet is a gigantic network which connects hundreds of thousands of computers together. It is a heterogenous network, because those computers are of different types and run different operating systems. They communicate with each other using the TCP/IP protocol. The common protocol means that a Unix workstation can
communicate just as easily with a PC running Windows as it can with another Unix workstation. Because there are so many types of computer systems connected to each other, there is a need for a language that is not tied to a particular platform. Java programs are compiled to a byte-codes, which means that they can run on any platforms that support the Java environment. Java program can be downloaded to any computers on the Internet and executed without worrying about the system type.

Java is a good language for Internet programming. Currently, HotJava and Netscape 2.0 World-Wide Web browsers are able to download compiled Java programs as Java applets. The Java applet is part of a Web page. The Java applets are downloaded to local computers and executed automatically when the Web pages are exposed. Executing Java applets is secure because Java programs are compiled into byte-code instructions, which can be verified before executing them.

3.3 Implementation of EESV Internet registration system

The goal of the EESV Internet registration system is to allow users to look up EESV's course information by accessing course related database tables and to register for a course via the Internet. Courses and conferences are organized as event and subevent concepts. Each event can have more than one subevents associated with it. Each subevent can have more than one fee type associated with it. The Internet registration system should allow users to check the subevent information associated with an event and the fee information associated with a subevent. Also, it can allows users to immediately
register for a course if they find a particular course they are interested in.

3.3.1 Client / Server Data Transition Model

As we have discussed, the course related database tables and compiled Java program are stored in the WWW server at Auburn University. The information that is sent back and forth between the user's computer and the WWW server at Auburn University is a client / server system based on the Internet protocol HTTP.
Figure 10  Diagram of Information Exchange Between User and WWW Server
As shown in Figure 10, when a user wants to look at EESV’s course information, a request to load a Java program is sent from user’s end to the WWW server using the HTTP protocol. The compiled Java program is sent to user’s computer by the WWW server at Auburn University. The compiled Java program is attached to Web page as an applet and it is sent to user’s computer by HTTP protocol. HTTP protocol is actually implemented by the TCP/IP protocol suite.

The Java applet must be verified by the client computer to check if the applet intends to do some dangerous actions on the client computer. After the applet passes the verification, it is executed by the client’s computer. When a Java applet needs course information in the database tables at the WWW server end. It creates data "pipes" to connect to the database tables at the WWW server end. After extracting the information by those "pipes" over the Internet, it closes those data "pipes". These data "pipes" are implemented by using the URL class in Java library network package (net.java).

3.3.2 Database Connections

Java Data Source Classes During the implementation of the EESV Internet registration system, one of the difficulties was how to access and query the database in the WWW server. The WWW server runs in the UNIX environment. Our course related database tables are FoxPro which is designed for the PC environment. Currently, we know of no database engine which runs in the UNIX environment and can access a FoxPro database.

This problem is solved by writing our own Java classes to connect to the FoxPro
database tables. For an example, in order to access and query the event.dbf table, the Java class called "event_datasource.java" is created to be hooked to the event.dbf database table. The complete class definition of "event_datasource" is shown in the Appendix. The following table lists some important data members, member functions and brief descriptions in the interface.

<table>
<thead>
<tr>
<th>A. M.</th>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>private String</td>
<td>Private variable to hold event ID</td>
</tr>
<tr>
<td>2</td>
<td>public String</td>
<td>Constructor</td>
</tr>
<tr>
<td>3</td>
<td>public int</td>
<td>To get total number of records in database table</td>
</tr>
<tr>
<td>4</td>
<td>public int</td>
<td>To get current record index</td>
</tr>
<tr>
<td>5</td>
<td>public String</td>
<td>To get event ID of current record</td>
</tr>
<tr>
<td>6</td>
<td>public String</td>
<td>To get event name of current record</td>
</tr>
<tr>
<td>7</td>
<td>public String</td>
<td>To get event beginning date of current record</td>
</tr>
<tr>
<td>8</td>
<td>public String</td>
<td>To get event ending date of current record</td>
</tr>
<tr>
<td>9</td>
<td>public void</td>
<td>To move pointer to first record in database table</td>
</tr>
<tr>
<td>10</td>
<td>public void</td>
<td>To move pointer to previous record if there is</td>
</tr>
<tr>
<td>11</td>
<td>public void</td>
<td>To move pointer to next record if there is</td>
</tr>
<tr>
<td>12</td>
<td>public void</td>
<td>To move pointer to last record if there is</td>
</tr>
</tbody>
</table>

Table 15. Java Class event_datasource.java Interface
When an event_datasource class is downloaded to the client computer and an instance of event_datasource Java class is created, the object of event_datasource class is hooked to the event database table in the WWW server over the Internet by URL data "pipes". The public methods are defined in the class to make the manipulation of records in the event table feasible and easy.

The careful reader may notice that the private "content" string variable in the class holds data information for the whole event database table. Would it be more efficient if the client program just loaded the useful data in the event.dbf table from the Internet instead of the content of whole database table? In general, the answer is YES. But, since in our particular case, the database tables are quite small (total about 100K). It takes a very short time to download all data over the Internet. Additionally, the complete data, transmitted to the client side, is necessary to construct an event dynamic link list (to be discussed in detail later on). The memory space allocated for the instance of the event_datasource class is freed as soon as the event link list is constructed. So, we do not need worry about memory space on the client side.

The problem of information retrieved from database could also be solved by implementing a database query using the client/server model. That means that the client end sends a query request to the server. The server receives the query and searches the database table on the server side. After obtaining the result, the server sends it back to the client side. We would need to write a separate program and put it in the WWW server’s cgi-bin directory to accomplish this task. The advantage of this approach would be to reduce the Internet traffic. The disadvantage would be poor query performance.
compared to the approach we will discuss.

**Event Link List** For each course database table (event, subevent, fees, facility), a corresponding objects of data source class is created on the client side. The information in these database table is transmitted from server side to client side. To improve the query performance and save memory space, an event link list is built. The memory space allocated for storing these database tables is freed after the event link list is built. Figure 11 shows the structure of an event link list. It grows dynamically and takes less space than the tables. Details can be found in the Appendix.
Figure 11. Structure of Event Link List
3.3.4 Java classes and Program Interfaces

Java Classes    Java is an object-oriented programming language. Java code is organized into classes. The following Table 16 shows the major classes in the EESV Internet registration Java program. The only gui.class is Java applet. It is attached to a HTML file and is first downloaded to the client computer. The Java environment is a run-time dynamically linked environment. Other classes are loaded to the client computer when they are needed. For example, the help.class is not loaded to the client computer until the user wants to read help information and chooses the help menu item.

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gui.class</td>
<td>Graphic User Interfaces</td>
</tr>
<tr>
<td>event_datasource.class</td>
<td>Class to access and query event database table</td>
</tr>
<tr>
<td>subevent_datasource.class</td>
<td>Class to access and query subevent database table</td>
</tr>
<tr>
<td>fees_datasource.class</td>
<td>Class to access and query fees database table</td>
</tr>
<tr>
<td>facility_datasource.class</td>
<td>Class to access and query facility database table</td>
</tr>
<tr>
<td>event_list.class</td>
<td>Event link list to contain subevent, fees and facility</td>
</tr>
<tr>
<td>help.class</td>
<td>Class to hold help information</td>
</tr>
</tbody>
</table>

Table 16. Java classes in EESV Internet registration program

Program Interfaces    Currently the Java program is stored at the WWW server of Auburn University. It can be accessed from Feng Li’s homepage and runs under Netscape 2.0 and HotJava browsers. Figures 12 - 16 show Internet interfaces of this Java program. Figure 12 is the author’s homepage, which is the "door" to enter the Java
program. Figure 13 is new course window which shows the new course information from the course database tables. Figure 14 illustrates the window which shows the background information about the application program. The window showing the information about how to use this application is shown in Figure 15. Figure 16 shows the interface for a course registration.

Welcome To Visit Feng Li's Homepage, Thanks!

Welcome To The Engineering Extension Service At Auburn University.

On-Line Engineering Extension Registration System

The following two push button will give you two choice. If you want to connect to the on-line Engineering Extension New Course Database, you can click the button with caption of "Push Me To Connect Database". If you finish viewing the new courses and wish to disconnect the database, you can click the button with caption of "Push Me To Disconnect Database".

If you have any questions or comments, please feel free to contact at Feng Li, tengan@auburn.edu. Thank you!

Figure 12. Author's Homepage
Figure 13. New Course Window
**Event Information**

<table>
<thead>
<tr>
<th>Event ID:</th>
<th>CONFED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Name:</td>
<td>Intermediate Personal Computing</td>
</tr>
<tr>
<td>Start Date:</td>
<td>10/3</td>
</tr>
</tbody>
</table>

**SubEvent Information**

| Sub-Event ID: | M |
| Sub-Event Name: | |

**Address:**

- City: Chicago
- State: IL

<table>
<thead>
<tr>
<th>Fee Type:</th>
<th>Early Bird Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fee:</td>
<td>$448.25</td>
</tr>
</tbody>
</table>

**Additional Information:**

- City Planners Annual Conference
- City Planners Annual Conference

- Orthopedic Synopsis Foundation
- Advanced Personal Computing
- Introduction to Personal Computing
- Management Skills for the New Supervisor
- Nw-88 Legal & Ethical Issues Conference
- Investing in Residential Real Estate
- Investing in Commercial Real Estate
- Investment Planning for Retirement II
- Investment Planning for Retirement I

- Select | Cancel

- Untrusted Java Applet Window

- Untrusted Java Applet Window
This application is written in Java Beta 1.0 as a part of thesis work of Feng Li who is a graduate student of Computer Science and Engineering at Auburn University. The application was developed under the supervision and direction of Dr. Carlisle, who is a professor of the Computer Science and Engineering Department at Auburn University. The application is designed for Engineering Extension Service at Auburn University (EESV) to allow clients to check and register course(s) through the Internet (WWW). The current version is 1.0.

Figure 14. Window For Information About the Application
Figure 15. Window For Information About How to Use This Application
Figure 16. Window For a Course Registration
3.4 Directions of Using This Application

In this section, instructions about how to use this Internet registration system are discussed. Eight topics are covered. Each of them starts with a question, and is followed by explanations.

(1) **How to connect the on-line new course database?**

The new course database of the Engineering Extension Service at Auburn University (EESV) actually is stored in the WWW server at Auburn University ("www.eng.auburn.edu"). In order to connect to the course database to see what courses are offered by EESV, you select the push button with the caption of "Push Me To Connect To Database" on the "Welcome" HTML Page (shown in Figure 12). The new course information window (shown in Figure 13) will appear after you click this button.

(2) **How to disconnect the on-line new course database?**

After you finish searching course(s) and wish to close or disconnect the course database, you select the push button with the caption of "Push Me To Disconnect Database". After you click this button, the new course window will disappear. You can also accomplish this "close" function by choosing the "Close" menu item under "File" Menu at the top of new course window.

(3) **Where is the new course database stored?**

The EESV new course database tables for the Internet registration system are
actually stored on the WWW server at Auburn University ("www.eng.auburn.edu"). They will be updated frequently to make sure clients can obtain the most recent new course information.

(4) How to check the new courses?

Once the new course window is displayed on the screen, it is arranged in three parts. The upper part is for displaying event information (Event ID, Event Name, Event starting and ending dates). Events can be rotated by the buttons "First", "Previous", "Next" and "Last".

Event When the new course window is first displayed, it automatically shows the first new event record.

Button "First" -- Shows first event record in database.

Button "Previous" -- Shows previous event record in database. Ignored if it is the first record.

Button "Next" -- Shows next event record in database. Ignored if it is the last record.

Button "Last" -- Shows the last event record in database.

The same functionality can be achieved by choosing corresponding menu items under the "Event" menu.

-- First Event.

-- Previous Event.
-- Next Event.

-- Last Event.

**Subevent** The middle part of the new course window is designed for displaying the subevent information associated with an above event. It can also be rotated by "First", "Previous", "Next" and "Last". When an event is changed, it automatically displays the first subevent information associated with the above event.

- **Button "First"**  -- Shows the first subevent record in database.
- **Button "Previous"**  -- Shows the previous subevent record in database. Ignored if it is the first record.
- **Button "Next"**  -- Shows the next subevent record in database. Ignored if it is the last record.
- **Button "Last"**  -- Shows the last subevent record in database.

The same functionality can be achieved by choosing corresponding menu items under the "Subevent" menu.

-- First subevent.

-- Previous subevent.

-- Next subevent.

-- Last subevent.

**Fees** The button part of the new course window is designed for displaying course fee information associated with a subevent. It can also be rotated by "Previous", "Next".
When a subevent is changed, the first fee information associated with the subevent is automatically displayed.

Button "Previous"  -- Shows the previous fee record in database, and ignored if it is the first record.

Button "Next"     -- Shows the next fee record in database, and ignored if it is the last record.

This functionality can also be achieved by choosing the corresponding menu items under the "Fee" menu.

-- Previous Fee.

-- Next Fee.

(5)  **How to search a course quickly?**

If you do not have time to look at all new courses from the first record to last record, you can use the "Search New Course" function to find immediately what you need immediately. After clicking the button "Search New Course" or choosing the menu item under the "Search" menu, a new course name list window will appear. Highlight whatever course you want, and click "OK". The new course name list window will disappear. The course information which you highlighted will be displayed in the course information window.

(6)  **How to register for a course?**

If you want to register and get more information about EESV, choose the
"Register" menu item. A new input HTML form will show up for you. You can input your requirements and click "Send Message" to submit your text information. The EESV Internet registration system will send your text input to an EESV registrar as an e-mail message. You will receive a confirmation letter and a course package.

(7) **How to get help information about the system?**

In order to use this system efficiently, check the on-line help information frequently. This system may be modified slightly without notification. The on-line help information will be updated as the system changes.

(8) **How to report system problem?**

If you encounter any system problems or have any suggestions, please send an e-mail message to Feng Li at the e-mail address:

Feng.Li@eng.auburn.edu or call (334) 884-5801
CHAPTER 4
CONCLUSIONS AND FUTURE WORK

4.1 Conclusions

In this project, registration database conversion software was developed. The software is presently in the operation phase. It converts registration data from an Aceware registration database system to a MPFW registration system. This software makes it feasible for EESV to upgrade their registration system. An EESV Internet registration system was also developed in this project. This system makes it possible to check new courses dynamically and register for them over the Internet. This Internet software takes advantage of the state-of-the-art Internet technology to greatly improve work efficiency and cut costs in the EESV. It was implemented by the new Internet programming language called Java. Some new features and properties of Java language and environment were explored in this study. The Internet registration software is still in test phase.

4.2 Future Work
For the Internet registration software, the course related database tables are copied periodically to a WWW server to guarantee EESV's clients can access the newest course data. This software may be further developed to allow users to access directly the registration course related database tables at the EESV office, provided security of this database can be guaranteed. This improvement would allow the Internet registration system to provide users the newest course information at almost the same time a new course is entered into registration database at the EESV office. One way to implement this direct access software would be to use the cgi-bin programming approach. Another would be to have Java program directly addresses a database server.

If the size of a course related database is very large, the client / server query mechanism is necessary to reduce the Internet traffic. The Java program at the client sends query requests to server and then receives the query results from server instead of downloading database tables.
BIBLIOGRAPHY


APPENDIX

EESV ON-LINE REGISTRATION SYSTEM

JAVA SOURCE PROGRAMS

(1) gui.java

import java.awt.*;
import java.applet.*;

public class gui extends java.applet.Applet {

    private Panel panel0;
    private Button b01,b02;

    private Panel panel1;
    private Panel panel101, panel102,panel103,panel104;
    private Panel panel105, panel106,panel107,panel108;
    private Panel panel141, panel142,panel143,panel144;

    private TextField text11,text12,text21,text31,text32,text41;
    private TextField text51,text61,text71,text72;
    private TextField text411,text421,text431,text432;

    private Label lab11,lab12,lab21,lab31,lab32,lab41;
    private Label lab51,lab61,lab71,lab72,lab81;
    private Label lab411,lab421,lab431,lab432;

    private event_list elist;
    private Frame mainf;
    private MenuBar mb;
    private MenuItem mfile,mselect,subselect,feeselect,regist,mhelp;
    private Menu msearch;
    private MenuItem i11,i12,i13,i21,i22,i23,i24,i31,i32;
    private MenuItem i41,i42,i43,i44,i51,i52,i61,i71;
    private Button b1,b2,b3,b4,b5,b6;
    private Button b7,b8,b9,b10,b11;

    //for search option
    private Frame listframe;
    private List list;
    private Button b20,b21;
private Frame aboutframe, howframe;
private TextArea ta1, ta2;
private Button b22, b23;

private event_data, subevent, fee_data
private event_data cu_event;
private subevent_data cu_subevent;
private fee_data cu_fee;

public void init(){
    
    elist = new event_list();
event_datasource ev = new event_datasource();
subevent_datasource se = new subevent_datasource();
fees_datasource fe = new fees_datasource();
facility_datasource fa = new facility_datasource();
elist.fill(ev, se, fe, fa);
cu_event = elist.get_first_event();
cu_subevent = cu_event.get_first_subevent();
cu_fee = cu_subevent.get_first_fee();
}

/main frame
panel0 = new Panel();
mainf = new Frame("EESV Course(s) View Window");
mainf.setLayout(new GridLayout(16, 1));
this.add(panel0);
mainf.addNotify();

//search list frame
listframe = new Frame("EESV Course(s) List");
listframe.setLayout(new BorderLayout(10, 10));
Panel pa = new Panel();
pa.setLayout(new FlowLayout());

//help "about" frame
aboutframe = new Frame("Help - About This Application");
aboutframe.setLayout(new BorderLayout(10, 10));
Panel pan1 = new Panel();
pan1.setLayout(new FlowLayout());
b22 = new Button("OK");
ta1 = new TextArea();
pan1.add(b22);
aboutframe.add("South", pan1);
aboutframe.add("Center", ta1);
aboutframe.addNotify();
this.add(aboutframe);

//help "how" frame
howframe = new Frame("Help - How To Use This Application");
howframe.setLayout(new BorderLayout(10, 10));
Panel pan2 = new Panel();
pan2.setLayout(new FlowLayout());
b23 = new Button("Done");
ta2 = new TextArea();
pan2.add(b23);
howframe.add("South", pan2);
howframe.add("Center", ta2);
howframe.addNotify();

66
this.add(howframe);

list = new List();

{
  event_data e = elist.get_last_event();
  if(e!=null){
    if(e.eventname!=null){
      list.addItem(new String(e.eventname));
    }
    for( int k=0; k<elist.get_number_event()-1;k++){e = elist.get_previous_event();
      if(e.eventname!=null){
        list.addItem(new String(e.eventname));
      }
    }
  }
}

listframe.add("Center", list);
b20 = new Button("Select");
b21 = new Button("Cancel");
pa.add(b20);
pa.add(b21);
listframe.add("South", pa);
this.add(listframe);

//add button to panel0
b01 = new Button(" Push Me To Connect Database ");
b02 = new Button(" Push Me To Disconnect Database ");
b02.disable();
panel0.add(b01);
panel0.add(b02);
this.add(mainf);

// create menu bar
mb = new MenuBar();

//add file menu
mfile = new Menu("File");
mb.add(mfile);

// add menu items to "file menu"
ill = new MenuItem("Clear");
il2 = new MenuItem("Close");
mfile.add(ill);
mfile.add(il2);

//add event menu
mselect = new Menu("Event");
mb.add(mselect);

// add menu items to "Event menu"
i21 = new MenuItem("First Record");
i22 = new MenuItem("Previous Record");
i23 = new MenuItem("Next Record");
i24 = new MenuItem("Last Record");
mselect.add(i21);
mselect.add(i22);
mselect.add(i23);

67
mselect.add(i24);

//add sub event menu
subselect = new Menu("Sub Event");
mb.add(subselect);

// add menu items to "Subevent menu"
i41 = new MenuItem("First Subevent");
i42 = new MenuItem("Previous Subevent");
i43 = new MenuItem("Next Subevent");
i44 = new MenuItem("Last Subevent");
subselect.add(i41);
subselect.add(i42);
subselect.add(i43);
subselect.add(i44);

//add sub event menu
feeselect = new Menu(" Fee ");
mb.add(feselect);

// add menu items to "Subevent menu"
i51 = new MenuItem("Next Fee");
i52 = new MenuItem("Previous Fee");
feeselect.add(i51);
feeselect.add(i52);

//add search menu
msearch = new Menu("Search");
mb.add(msearch);

// add menu items to "Search New Courses"
i71 = new MenuItem("Search New Courses");
msearch.add(i71);

//add regist menu
regist = new Menu("Registration");
mb.add(regist);

// add menu items to "Sub Event menu"
i61 = new MenuItem("Regist Course");
regist.add(i61);

//add help menu
mhelp = new Menu("Help");
mb.add(mhelp);
mb.setHelpMenu(mhelp);

// add menu items to "help menu"
i31 = new MenuItem("About This Application");
i32 = new MenuItem("How To Use This Application");
mhelp.add(i31);
mhelp.add(i32);

//add menu bar to frame
mainf.setMenuBar(mb);

panel1 = new Panel();
panel01 = new Panel();
panel02 = new Panel();
panel03 = new Panel();
panel04 = new Panel();
panel141 = new Panel();
panel42 = new Panel();
panel143 = new Panel();
panel144 = new Panel();
panel105 = new Panel();
panel106 = new Panel();
panel107 = new Panel();
panel108 = new Panel();

panel01.setLayout(new FlowLayout(FlowLayout.LEFT));
panel02.setLayout(new FlowLayout(FlowLayout.LEFT));
panel03.setLayout(new FlowLayout(FlowLayout.LEFT));
panel04.setLayout(new FlowLayout(FlowLayout.LEFT));
panel141.setLayout(new FlowLayout(FlowLayout.LEFT));
panel142.setLayout(new FlowLayout(FlowLayout.LEFT));
panel143.setLayout(new FlowLayout(FlowLayout.LEFT));
panel105.setLayout(new FlowLayout(FlowLayout.LEFT));
panel106.setLayout(new FlowLayout(FlowLayout.LEFT));
panel107.setLayout(new FlowLayout(FlowLayout.LEFT));

mainf.add(new Label("Event Information",Label.LEFT));
mainf.add(panel101);
mainf.add(panel102);
mainf.add(panel103);
mainf.add(panel11);
mainf.add(new Label("SubEvent Information Associated With The Above Event"));
mainf.add(panel105);
mainf.add(panel106);
mainf.add(panel104);
mainf.add(panel141);
mainf.add(panel142);
mainf.add(panel143);
mainf.add(panel144);
mainf.add(new Label("Fee Associated With The Above Subevent"));

//add label and text to panel01
textField1 = new TextField("",15);
textField2 = new TextField("",5);
label1 = new Label("Event ID:" ,Label.LEFT);
label2 = new Label("Number of Subevent:" ,Label.LEFT);
panel01.add(new Label11);
panel01.add(textField1);
panel01.add(textField2);

//add label and text to panel02
textField3 = new TextField("",45);
label12 = new Label("Event Name:" ,Label.LEFT);
panel02.add(label12);
panel02.add(textField3);

//add label and text to panel03
textField4 = new TextField("",15);
label3 = new Label("Start Date:" ,Label.LEFT);
textField5 = new TextField("",15);
lab32 = new Label("End Date:",Label.LEFT);
panel03.add(lab32);
panel03.add(text32);
panel03.add(lab32);
panel03.add(text32);

//add label and text to panel05
text51 = new TextField("",15);
lab51 = new Label("Sub-Event ID:",Label.LEFT);
panel05.add(lab51);
panel05.add(text51);

//add label and text to panel06
text61 = new TextField("",45);
lab61 = new Label("Sub-Event Name:",Label.LEFT);
panel06.add(lab61);
panel06.add(text61);

//add label and text to panel04
text41 = new TextField("",45);
lab41 = new Label("Location:",Label.LEFT);
panel04.add(lab41);
panel04.add(text41);

//add label and text to panel41
text411 = new TextField("",45);
lab411 = new Label("Address1:",Label.LEFT);
panel41.add(lab411);
panel41.add(text411);

//add label and text to panel42
text421 = new TextField("",45);
lab421 = new Label("Address2:",Label.LEFT);
panel42.add(lab421);
panel42.add(text421);

//add label and text to panel43
text431 = new TextField("",20);
lab431 = new Label("City:",Label.LEFT);
text432 = new TextField("",15);
lab432 = new Label("State:",Label.LEFT);
panel43.add(lab431);
panel43.add(text431);
panel43.add(lab432);
panel43.add(text432);

//add buttons to panel44
b7 = new Button(" <<<<First ");
b8 = new Button(" <<Previous ");
b9 = new Button(" Next>> ");
b10 = new Button(" Last>>> ");
panel44.add(b7);
panel44.add(b8);
panel44.add(b9);
panel44.add(b10);

//add label and text to panel07
text71 = new TextField("",30);
lab71 = new Label("Fee Type:",Label.LEFT);
text72 = new TextField("",10);
lab72 = new Label("Fee :",Label.LEFT);
panel07.add(lab71);
panel07.add(text71);
panel07.add(lab72);
panel07.add(text72);

//add label and text to panel08
b5 = new Button(" < Previous Fee ");
b6 = new Button(" Next Fee > ");
panel08.add(b5);
panel08.add(b6);

//add button to panell
b1 = new Button("<< First ");
b2 = new Button("< Previous");
b3 = new Button(" Next > ");
b4 = new Button(" Last >> ");
b11 = new Button("Search New Courses");
panel11.add(b1);
panel11.add(b2);
panel11.add(b3);
panel11.add(b4);
panel11.add(b11);

// Show the first event when the window is shown

cu_event = elist.get first event();
cu_subevent = cu_event.get first_subevent();
cu_fee = cu_subevent.get first_fee();

// first event information

if(cu_event!=null){
text11.setText(cu_event.eventid);
text12.setText((new Integer((cu_event.get_number_subevent()))).toString());
text21.setText(cu_event.eventname);
text31.setText((new mydate(cu_event.begin_date)).toString());
text32.setText((new mydate(cu_event.end_date)).toString());
} else{
text11.setText("");
text12.setText("");
text21.setText("");
text31.setText("");
text32.setText("");
}

// first subevent information

if(cu_subevent!=null){
text51.setText(new String(cu_subevent.subeventid));
text61.setText(cu_subevent.subeventname);
text411.setText(cu_subevent.facility_addr1);
text421.setText(cu_subevent.facility_addr2);
text431.setText(cu_subevent.facility_city);
text432.setText(cu_subevent.facility_state);
} else {
text51.setText(""); 
text61.setText(""); 
text411.setText(""); 
text421.setText(""); 
text431.setText(""); 
text432.setText(""); 
}

// first fee information
if (cu_fee!=null){
  text71.setText(cu_fee.fname);
  text72.setText("$"+cu_fee.fee.trim());
}else{
  text71.setText("");
  text72.setText(""); 
}

public boolean action(Event evt, Object arg){
  if(evt.target instanceof Button){
    String s= new String((String)arg);

    // Connect database button responses
    // user click"First"
    if (s.equals("Push Me To Connect Database ")){
      database_connect_handler();
      return true;
    }

    // Disconnect database button responses
    // user click"First"
    if (s.equals("Push Me To Disconnect Database ")){
      database_disconnect_handler();
      return true;
    }

    // Event Search button responses
    // user click"Search new courses"
    if (s.equals("Search New Courses")){
      listframe.resize(500,400);
      listframe.move(400,20);
      listframe.show();
      return true;
    }

    // List Frame "Select" button responses
    // user click"Select"
    if (s.equals("Select")){
      event_select_handler();
      return true;
    }

    // List Frame "Cancel" button responses
    // user click"Cancel"
    if (s.equals("Cancel")){
      event_cancel_handler();
      return true;
    }
  }
}
// Help "about" frame " OK " button responses
// user click* OK *
if (s.equals(' OK ')){
    aboutframe.dispose();
    return true;
}

// Help "how" frame " Done " button responses
// user click* Done *
if (s.equals(' Done ')){
    howframe.dispose();
    return true;
}

// Event button responses
// user click "First"
if (s.equals("<< First ")){
    event_first_handler();
    return true;
}

// Event button responses
// user click "Previous" choice;
if (s.equals("< Previous")){
    event_previous_handler();
    return true;
}

// Event button responses
// user click "Next" choice;
if (s.equals(" Next > ")){
    event_next_handler();
    return true;
}

// Event button responses
// user click "Last" choice;
if (s.equals(" Last >> ")){
    event_last_handler();
    return true;
}

// Subevent button responses
// user click "<<<First" choice;
if (s.equals(" <<<First ")){
    subevent_first_handler();
    return true;
}

// Subevent button responses
// user click "<<<Previous" choice;
if (s.equals(" <<<Previous ")){
    subevent_previous_handler();
    return true;
}

// Subevent button responses
// user click "Next>>" choice;
if (s.equals(" Next>> ")){
    subevent_next_handler();
}
return true;
}

// Subevent button responses
// user click "Last>>>" choice;
if (s.equals(" Last>>> ")){
    subevent_last_handler();
    return true;
}

// Fee button responses
// user click "< Previous Fee" choice;
if (s.equals(" < Previous Fee ")){
    fee_previous_handler();
    return true;
}

// Fee button responses
// user click "Next Fee >" choice;
if (s.equals(" Next Fee > ")){
    fee_next_handler();
    return true;
}

// menu action
if(evt.target instanceof MenuItem){
    String s= new String((String)arg);

    // menu "Close" action
    if(s.equals("Close")){
        menu_close_handler();
        return true;
    }

    // menu "Clear" action
    if(s.equals("Clear")){
        menu_clear_handler();
        return true;
    }

    // for event
    // menu "First Record" action
    if(s.equals("First Record")){
        menu_event_first_handler();
        return true;
    }

    // for event
    //menu "Previous Record" action
    if(s.equals("Previous Record")){
        menu_event_previous_handler();
        return true;
    }

    // for event
    //menu "Next Record" action
    if(s.equals("Next Record")){
        menu_event_next_handler();
        return true;
    }
}
// for event
// menu "Last Record" action
if(s.equals("Last Record")){
    menu_event_last_handler();
    return true;
}

// for subevent
// menu "First Subevent" action
if(s.equals("First Subevent")){
    subevent_first_handler();
    return true;
}

// for subevent
// menu "Previous Subevent" action
if(s.equals("Previous Subevent")){
    subevent_previous_handler();
    return true;
}

// for subevent
// menu "Next Subevent" action
if(s.equals("Next Subevent")){
    subevent_next_handler();
    return true;
}

// for subevent
// menu "Last Subevent" action
if(s.equals("Last Subevent")){
    subevent_last_handler();
    return true;
}

// for fee
// menu "Next Fee" action
if(s.equals("Next Fee")){
    fee_next_handler();
    return true;
}

// for fee
// menu "Previous Fee" action
if(s.equals("Previous Fee")){
    fee_previous_handler();
    return true;
}

// for help information
// menu "About This Application" action
if(s.equals("About This Application")){
    // dynamically load help class instance
    help he = new help();
    he.set_about_help(tal);
    aboutframe.resize(500,500);
    aboutframe.move(500,0);
    aboutframe.show();
    return true;
}
//for help information
//menu "How To Use This Application" action
if(s.equals("How To Use This Application")){
    //dynamically load help class instance
    help he = new help();
    he.set_how_help(ta2);
    howframe.resize(500,500);
    howframe.move(520,20);
    howframe.show();
    return true;
}

//for search new courses
//menu "Search" action
if(s.equals("Search New Courses")){
    listframe.resize(500,400);
    listframe.move(400,20);
    listframe.show();
    return true;
}
else{
    return true;
}
}
return false;

//actions for connect database
private void database_connect_handler(){
    b01.disable();
    maininf.setMenuBar(mb);
    maininf.resize(600,750);
    maininf.move(350,0);
    maininf.show();
    b02.enable();
}

//actions for disconnect database
private void database_disconnect_handler(){
    b02.disable();
    maininf.dispose();
    b01.enable();
}

//action for menu
//actions for "close" database
private void menu_close_handler(){
    database_disconnect_handler();
}
//action for "Select"
//actions for "Select" event
private void event_select_handler(){
    listframe.dispose();
    String ss = new String(list.getSelectedItem());
    event_data ev = elist.get_last_event();

    if (ss.equals(ev.eventname)){
        cu_event = ev;
        cu_subevent = cu_event.get_first_subevent();
        cu_fee = cu_subevent.get_first_fee();

        // display event information
        if (cu_event != null){
            text11.setText(new String(cu_event.get_eventid()));
            text12.setText(new Integer((cu_event.get_number_subevent()))).toString();
            text21.setText(cu_event.get_eventname());
            text31.setText(new Integer(cu_event.get_begin_date())).toString();
            text32.setText(new Integer(cu_event.get_end_date())).toString();
            text11.setText('"');
            text12.setText('"');
            text21.setText('"');
            text31.setText('"');
            text32.setText('"');
        }

        // show subevent information
        if (cu_subevent != null){
            text51.setText(new String(cu_subevent.get_subeventid()));
            text61.setText(cu_subevent.get_subeventname());
            text411.setText(cu_subevent.get_facility_addr1());
            text421.setText(cu_subevent.get_facility_addr2());
            text431.setText(cu_subevent.get_facility_city());
            text432.setText(cu_subevent.get_facility_state());
        }

        // first fee information
        if (cu_fee != null){
            text71.setText(cu_fee.get_fee());
            text72.setText(new Integer(cu_fee.get_fee()).toString());
        }

    }
else{
    for (int k = 0; k < elist.get_number_event()-1; k++){
        ev = elist.get_previous_event();
        if (ss.equals(ev.eventname)){
            cu_event = ev;
            cu_subevent = cu_event.get_first_subevent();
            cu_fee = cu_subevent.get_first_fee();
        }
    }
}

77
if(cu_event!=null){
    text11.setText(cu_event.eventid);
    text12.setText((new
        Integer((cu_event.get_number_subevent()))).toString());
    text21.setText(cu_event.eventname);
    text31.setText((new
        mydata(cu_event.begin_date)).toString());
    text32.setText((new
        mydata(cu_event.end_date)).toString());
} else {
    text11.setText("*");
    text12.setText("*");
    text21.setText("*");
    text31.setText("*");
    text32.setText("*");
}

// show subevent information
if(cu_subevent!=null){
    text51.setText(new String(cu_subevent.subeventid));
    text61.setText(cu_subevent.subeventname);
    text411.setText(cu_subevent.facility_addr1);
    text421.setText(cu_subevent.facility_addr2);
    text431.setText(cu_subevent.facility_city);
    text432.setText(cu_subevent.facility_state);
} else {
    text51.setText("*");
    text61.setText("*");
    text411.setText("*");
    text421.setText("*");
    text431.setText("*");
    text432.setText("*");
}

// first fee information
if (cu_fee!=null){
    text71.setText(cu_fee.fname);
    text72.setText("$"+cu_fee.fee.trim());
} else {
    text71.setText("*");
    text72.setText("*");
}

//action for "Cancel"
//actions for "Cancel" event
private void event_cancel_handler(){
    listframe.dispose();
}

//action for menu

78
//actions for "clear" textfield  
private void menu_clear_handler(){

    // clear event information 
    text11.setText(""); 
    text12.setText(""); 
    text21.setText(""); 
    text31.setText(""); 
    text32.setText(""); 

    // clear subevent information 
    text51.setText(""); 
    text61.setText(""); 
    text411.setText(""); 
    text421.setText(""); 
    text431.setText(""); 
    text432.setText(""); 

    // clear fee information 
    text71.setText("");  
    text72.setText("");  
}

//action for menu  
//actions for "first" event 
private void menu_event_first_handler(){
    event_first_handler();
}

//action for menu  
//actions for "previous" event 
private void menu_event_previous_handler(){
    event_previous_handler();
}

//action for menu  
//actions for "next" event 
private void menu_event_next_handler(){
    event_next_handler();
}

//action for menu  
//actions for "last" event  
private void menu_event_last_handler(){
    event_last_handler();
}

//actions for event  
//action handler for "First" 
private void event_first_handler(){
    cu_event = eList.get_first_event();
    cu_subevent = cu_event.get_first_subevent();
    cu_fee = cu_subevent.get_first_fee();
    display_event_info();
}
display_subevent_info();
display_fee_info();

//actions for event
//action handler for "Previous"
private void event_previous_handler()
{
    cu_event  = elist.get_previous_event();
cu_subevent = cu_event.get_first_subevent();
cu_fee     = cu_subevent.get_first_fee();

display_event_info();
display_subevent_info();
display_fee_info();

}

//actions for event
//action handler for "Next"
private void event_next_handler()
{
    cu_event  = elist.get_next_event();
cu_subevent = cu_event.get_first_subevent();
cu_fee     = cu_subevent.get_first_fee();

display_event_info();
display_subevent_info();
display_fee_info();

}

//actions for event
//action handler for "Last"
private void event_last_handler()
{
    cu_event  = elist.get_last_event();
cu_subevent = cu_event.get_first_subevent();
cu_fee     = cu_subevent.get_first_fee();

display_event_info();
display_subevent_info();
display_fee_info();

}

//actions for subevent
//action handler for "First" for subevent
private void subevent_first_handler()
{
    cu_subevent = cu_event.get_first_subevent();
cu_fee     = cu_subevent.get_first_fee();

display_subevent_info();
display_fee_info();
//actions for subevent
//action handler for "Previous" for subevent
private void subevent_previous_handler(){
    cu_subevent = cu_event.get_previous_subevent();
    cu_fee = cu_subevent.get_first_fee();
    display_subevent_info();
    display_fee_info();
}

//actions for subevent
//action handler for "Next" for subevent
private void subevent_next_handler(){
    cu_subevent = cu_event.get_next_subevent();
    cu_fee = cu_subevent.get_first_fee();
    display_subevent_info();
    display_fee_info();
}

//actions for subevent
//action handler for "Last" for subevent
private void subevent_last_handler(){
    cu_subevent = cu_event.get_last_subevent();
    cu_fee = cu_subevent.get_first_fee();
    display_subevent_info();
    display_fee_info();
}

//actions for fee
//action handler for "< Previous Fee" for fee
private void fee_previous_handler(){
    cu_fee = cu_subevent.get_previous_fee();
    display_fee_info();
}

//actions for fee
//action handler for "Next Fee >"
private void fee_next_handler(){
    cu_fee = cu_subevent.get_next_fee();
    display_fee_info();
}
private void display_event_info()
{
    if (cu_event != null)
    {
        text11.setText(String.valueOf(cu_event.eventid));
        text12.setText(String.valueOf(cu_event.eventname));
        text21.setText(String.valueOf(cu_event.begin_date));
        text22.setText(String.valueOf(cu_event.end_date));
    }
    else
    {
        text11.setText('');
        text12.setText('');
        text21.setText('');
        text22.setText('');
    }
}

private void display_subevent_info()
{
    if (cu_subevent != null)
    {
        text51.setText(String.valueOf(cu_subevent.subeventid));
        text52.setText(String.valueOf(cu_subevent.subeventname));
        text411.setText(String.valueOf(cu_subevent.facility_addr1));
        text412.setText(String.valueOf(cu_subevent.facility_addr2));
        text431.setText(String.valueOf(cu_subevent.facility_city));
        text432.setText(String.valueOf(cu_subevent.facility_state));
    }
    else
    {
        text51.setText('');
        text52.setText('');
        text411.setText('');
        text412.setText('');
        text431.setText('');
        text432.setText('');
    }
}

private void display_fee_info()
{
    if (cu_fee != null)
    {
        text71.setText(String.valueOf(cu_fee.fee_name));
        text72.setText(String.valueOf(cu_fee.fee.trim()));
    }
    else
    {
        text71.setText('');
        text72.setText('');
    }
}
import java.io.*;

class fee_data{
    // fee information
    public String feename;
    public String fee;
}

class subevent_data{
    // subevent information
    public String eventid;
    public String subeventid;
    public String subeventname;

    // facility information
    public String facilityid;
    public String facility_name;
    public String facility_addr1;
    public String facility_addr2;
    public String facility_city;
    public String facility_state;

    // fees information
    public int number_of_fees;
    public fee_data fees[] = new fee_data[100];
    private int current = 0;

    // get_number_fees() return number of fees
    // associated with this subevent
    public int get_number_fees(){
        return number_of_fees;
    }

    // get_first_fee() return first fee_data object
    public fee_data get_first_fee(){
        if(number_of_fees!=0){
            current=0;
            return fees[current];
        }else{
            return null;
        }
    }
}
//get_next_fee() return next fee_data object

public fee_data get_next_fee()
{
    if(number_of_fees!=0 && (current+1 <= number_of_fees-1))
    {
        current=current+1;
        return fees[current];
    }
    if(number_of_fees!=0 && (current+1 > number_of_fees-1))
        return fees[current];
    else{
        return null;
    }
}

//get_previous_fee() return next fee_data object

public fee_data get_previous_fee()
{
    if(number_of_fees!=0 && (current-1 >= 0)){
        current=current-1;
        return fees[current];
    }
    if(number_of_fees!=0 && (current-1 < 0 ))
        return fees[current];
    else{
        return null;
    }
}

//get_last_fee() return last fee_data object

public fee_data get_last_fee()
{
    if(number_of_fees!=0){
        current=number_of_fees-1;
        return fees[current];
    }
    else{
        return null;
    }
}

}

class event_data{

    //event information
    public String eventid;
    public String eventname;
    public String begin_date;
    public String end_date;
    public String status;
    private int current;

    //subevent information
    private int number_of_subevent;
    public subevent_data subevent[] = new subevent_data[100];
// constructor
event_data(event_datasource ev,
    subevent_datasource se,
    fees_datasource fe,
    facility_datasource fa) {

    // construct event info
    eventid = new String(ev.eventid());
    eventname = new String(ev.eventname());
    begin_date = new String(ev.begin_date());
    end_date = new String(ev.end_date());
    status = new String(ev.status());
    current = 0;

    // construct subevent info
    int k = 0;
    se.first();
    for(int i = 1; i <= se.numberofrecord(); i++) {
        if(eventid.equals(se.eventid())) {
            subevent[k] = new subevent_data();
            subevent[k].eventid = new String(eventid);
            subevent[k].subeventid = new String(se.subeventid());
            subevent[k].subeventname = new String(se.subeventname());
            subevent[k].facilityid = new String(se.facilityid());

            // construct facility info
            fa.first();
            for(int l = 1; l <= fa.numberofrecord(); l++) {
                if(subevent[k].facilityid.equals(fa.facilityid())) {
                    subevent[k].facility_name = new String(fa.name());
                    subevent[k].facility_addr1 = new String(fa.addr1());
                    subevent[k].facility_addr2 = new String(fa.addr2());
                    subevent[k].facility_city = new String(fa.city());
                    subevent[k].facility_state = new String(fa.state());
                }
            }
        }

        // construct fee info
        fe.first();
        int n = 0;
        for(int v = 0; v < fe.numberofrecord(); v++) {
            if(subevent[k].eventid.equals(fe.eventid())) {
                if(subevent[k].subeventid.equals(fe.subeventid())) {
                    subevent[k].fees[n] = new String(fe.fee());
                }
            }
        }
    }
    fe.next();
    subevent[k].number_of_fees = n;
    k++;
}
se.next();
number_of_subevent = k;
}

// get_first_subevent() return first subevent
public subevent_data get_first_subevent(){
    if(number_of_subevent!=0){
        current=0;
        return subevent[current];
    }else{
        return null;
    }
}

// get_next_subevent() return next subevent
public subevent_data get_next_subevent(){
    if(number_of_subevent!=0 && (current+1 <= number_of_subevent-1)){
        current=current+1;
        return subevent[current];
    }
    if(number_of_subevent!=0 && (current+1 > number_of_subevent-1)){
        return subevent[current];
    }else{
        return null;
    }
}

// get_previous_subevent() return next subevent
public subevent_data get_previous_subevent(){
    if(number_of_subevent!=0 && (current-1 >= 0)){
        current=current-1;
        return subevent[current];
    }
    if(number_of_subevent!=0 && (current-1 < 0)){
        return subevent[0];
    }else{
        return null;
    }
}

// get_last_subevent() return last subevent
public subevent_data get_last_subevent(){
    if(number_of_subevent!=0){
        current=number_of_subevent-1;
        return subevent[current];
    }else{
        return null;
    }
}

// get_number_subevent() return number of subevent
// associated with this event
public int get_number_subevent(){
    return number_of_subevent;
}
class event_node {
    event_node next;
    event_node previous;
    event_data data;

    // constructor
    event_node(event_datasource ev,
                subevent_datasource se,
                fees_datasource fe,
                facility_datasource fa) {
        next=null;
        previous=null;
        data = new event_data(ev,se,fe,fa);
    }
}

class event_list {

    private event_node head;
    private event_node last;
    private event_node current;
    private int number_of_event;

    // constructor
    event_list(){
        head=null;
        last=null;
        current=null;
        number_of_event=0;
    }

    // append method
    public void append(event_datasource ev,
                       subevent_datasource se,
                       fees_datasource fe,
                       facility_datasource fa){
        if(head==null){
            head = new event_node(ev,se,fe,fa);
            last = head;
            current = head;
            number_of_event++;
        }

        if(last.next==null){
            last.next = new event_node(ev,se,fe,fa);
            (last.next).previous = last;
            last=last.next;
            number_of_event++;
        }
    }

    // fill list method
    public void fill(event_datasource ev,
                     subevent_datasource se,
                     fees_datasource fe,
                     facility_datasource fa){
        for(int i=0;i<ev.number_of_record();i++){
            this.append(ev,se,fe,fa);
        }
    }
}
// fresh event list

public void fresh(){
    current = head;
}

// get_first_event() return first event_data object

public event_data get_first_event(){
    if(head!=null){
        current = head;
        return head.data;
    }else{
        current = head;
        return null;
    }
}

// get_next_event() return next event_data object

public event_data get_next_event(){
    if(current!=null && current.next!=null){
        current = current.next;
        return current.data;
    }

    if(current!=null && current.next==null){
        return current.data;
    }else{
        return null;
    }
}

// get_previous_event() return previous event_data object

public event_data get_previous_event(){
    if(current!=null && current.previous!=null){
        current=current.previous;
        return current.data;
    }

    if(current!=null && current.previous==null){
        return current.data;
    }else{
        return null;
    }
}

// get_last_event() return last event_data object

public event_data get_last_event(){
    if(last!=null){
        current=last;
        return last.data;
    }else{
        return null;
    }
}
// get_current_event() return current event_data object

public event_data get_current_event(){
    if(current!=null){
        return current.data;
    }else{
        return null;
    }
}

// int get_number_event() return number of event in list

public int get_number_event(){
    return number_of_event;
}

class mydate{
    private String myear=null;
    private String mmonth=null;
    private String mday=null;
    mydate(String s){
        if(s!=null && s.length()>=8){
            myear = new String(s.substring(0,4));
            mmonth = new String(s.substring(4,6));
            mday = new String(s.substring(6,8));
        }
    }

    // method to return String date

    public String toString(){
        String temp = (new String(mmonth)) + " / " + (new String(mday))
                        + " / " + (new String(myear));
        return temp.trim();
    }
}
(3) event_datasource.java

// ==============================================================
// 1. event_datasource class
// The class "event_datasource" allows us to access the "FoxPro 2.5"
// database "Event", which is stored in WWW server
// "http://www.eng.auburn.edu/~lifengl" or somewhere else.
//
// 2. Some public methods for "event_datasource" class
// (1) event_datasource() // constructor
// (2) int numberOfRecord() // return # of records in database
// (3) void first() // get first record
// (4) void last() // get last record
// (5) void next() // get next record
// (6) void previous() // get previous record
// (7) String eventid() // get content of field "EVENTID"
// (8) String eventname() // get content of field "EVENTNAME"
// (9) String begin_date() // get content of filed "BEGIN_DATE"
// (10) String end_date() // get content of filed "END_DATE"
// (11) String descrpt() // get content of field "DESCRIP"
// (12) String status() // get content of field "STATUS"
//
// ==============================================================

import java.net.*;
import java.applet.*;
import java.lang.*;
import java.io.*;

public class event_datasource {

private String eventid; //field # 1 char *10
private String eventname; // # 2 char *60
private String begin_date; // # 3 date *8
private String end_date; // # 4 date *8
private String descrpt; // # 5 mem *10
private String status; // # 6 num *2

private int numberOf_record;
private int head=514;
private int record_length=169+1;
private int current_record;
private int current_pos;
private String content;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//constructor
event_datasource(){

String s;
StringBuffer con=new StringBuffer();

s="http://www.eng.auburn.edu/~lifengl/event.dbf";
URL url=null;
try {
   url= new URL(s);
} catch ( MalformedURLException e){
    System.exit(0);
}

URLConnection connection = null;
try {
    connection = url.openConnection();
} catch (IOException e){
    System.exit(0);
}

DataInputStream in = null;
try {
    in = new DataInputStream(new BufferedInputStream(
        connection.getInputStream()));
} catch (IOException e) {
    System.exit(0);
}

byte ch[] = new byte[1];
try {
    for(;;){
        ch[0] = in.readByte();
        String c = new String(ch,8);
        con.append(c);
    }
} catch ( IOException e){
    }

content = con.toString();
this.first();

//===-------------------------------------------------------------------------------------
// numberOfrecord();

public int numberOfrecord(){
    number_of_record = (int)((content.length()-514)/record_length);
    return number_of_record;
}

//===-------------------------------------------------------------------------------------
// currentRecord();

public int currentRecord(){
    return current_record;
}

//===-------------------------------------------------------------------------------------
// get field contents

public String eventid(){
    if( eventid.equals(null)){
        return (new String("null"));
    }else{
        return eventid.trim();
    }
}

public String eventname(){
    if( eventname.equals(null)){

    return (new String("null"));
} else{
    return eventname.trim();
}
}

public String begin_date()
    if( begin_date.equals(null)) {
    return new String("null");
} else{
    return begin_date.trim();
}
}

public String end_date()
    if( end_date.equals(null)) {
    return new String("null");
} else{
    return end_date.trim();
}
}

public String descr()
    if( descr.equals(null)) {
    return new String("null");
} else{
    return descr.trim();
}
}

public String status()
    if( status.equals(null)) {
    return new String("null");
} else{
    return status.trim();
}
}

// function to set records;
private void set_record(int num){
    current_record=num;
    current_pos=head+record_length*(num-1);
    if(num>=1 && num<=this.numberofrecord()){
        1. eventid
            eventid=content.substring(current_pos,current_pos+10); 
            current_pos=current_pos+10;
        // 2. eventname
            eventname=content.substring(current_pos,current_pos+60); 
            current_pos=current_pos+60;
        // 3. begin_date
            begin_date=content.substring(current_pos,current_pos+8); 
            current_pos=current_pos+8;
        // 4. end_date
            end_date=content.substring(current_pos,current_pos+8); 
            current_pos=current_pos+8;
        // 5. descr
            descr=content.substring(current_pos,current_pos+10);
} else{
    return null;
}
}
current_pos = current_pos + 10;

// 6. status
status = content.substring(current_pos, current_pos + 2);
current_pos = current_pos + 2;
}

.IsTrue
//get first record
public void first()
if (this.numberOfrecord() >= 1){
    set_record(1);
}

// get next record
public void next(){
    if (current_record < this.numberOfrecord()){
        current_record = current_record + 1;
        set_record(current_record);
    }
}

// get previous record
public void previous(){
    if (current_record > 1){
        current_record = current_record - 1;
        set_record(current_record);
    }
}

// get previous record
public void last(){
    if (this.numberOfrecord() >= 1){
        current_record = this.numberOfrecord();
        set_record(current_record);
    }
}

//End of event_datasource class
//==================================================================

93
(4) subevent_datasource.java

// ================================================================
// 1. subevent_datasource class
// The class "event_datasource" allows us to access the "FoxPro 2.5"
// database "Event", which is stored in WWW server
// "http://www.eng.auburn.edu/~lifengl" or somewhere else.
// 2. Some public methods for "subevent_datasource" class
// (1) subevent_datasource()  // constructor
// (2) int numberOfrecord()  //return # of records in database
// (3) void first()           // get first record
// (4) void last()           // get last record
// (5) void next()           // get next record
// (6) void previous()       // get previous record
// (7) String eventid()      // get content of field "EVENTID"
// (8) String subeventid()   // get content of field "SUBEVENTID"
// (9) String subeventname() // get content of field "EVENTNAME"
// (10) String facilityid()  // get content of field "FACILITYID"
// (11) String begin_date()  // get content of field "BEGIN_DATE"
// (12) String end_date()    // get content of field "END_DATE"
// ================================================================

import java.net.*;
import java.applet.*;
import java.lang.*;
import java.io.*;

public class subevent_datasource {

private String eventid;     //field # 1      char *10
private String subeventid;  //field # 2      char *10
private String subeventname; // # 3        char *60
private String facilityid;  // # 7        char *10
private String begin_date;  // # 8       date *8
private String end_date;    // # 9        date *8

private int number_of_record;
private int head=1506;
private int record_length=340+1;
private int current_record;
private int current_pos;
private String content;

//constructor
subevent_datasource(){
    String s;
    StringBuffer con=new StringBuffer();
    s="http://www.eng.auburn.edu/~lifengl/subevnt.dbf";
    URL url=null;
}
try {
    url = new URL(s);
    } catch ( MalformedURLException e) {
        System.exit(0);
    }

URLConnection connection=null;
try {
    connection=url.openConnection();
    } catch (IOException e){
        System.exit(0);
    }

DataInputStream in = null;
try {
    in = new DataInputStream(new BufferedInputStream(
            connection.getInputStream()));
    } catch (IOException e) {
        System.exit(0);
    }

byte ch[]=new byte[1];
try {
    for (;;) {
        ch[0]=in.readByte();
        String c = new String(ch,8);
        con.append(c);
    }
    } catch ( IOException e){
    }

    content = new String(con.toString());
    this.first();
}

//================================================================================
// numberOfrecord();
public int numberOfrecord(){
    number_of_record=(int)((content.length()-head)/record_length);
    return number_of_record;
}

//================================================================================
// currentRecord();
public int currentRecord(){
    return current_record;
}

//================================================================================
// get field contents
public String eventid(){
    if( eventid.equals(null)){
        return (new String("null"));
    }else{
        return eventid.trim();
    }
}
public String subeventid(){
    if( subeventid.equals(null)){
        return (new String("null"));
    }else{
        return subeventid.trim();
    }
}

public String subeventname(){
    if( subeventname.equals(null)){
        return (new String("null"));
    }else{
        return subeventname.trim();
    }
}

public String facilityid(){
    if( facilityid.equals(null)){
        return (new String("null"));
    }else{
        return facilityid.trim();
    }
}

public String begin_date(){
    if( begin_date.equals(null)){
        return new String("null");
    }else{
        return begin_date.trim();
    }
}

public String end_date(){
    if( end_date.equals(null)){
        return new String("null");
    }else{
        return end_date.trim();
    }
}

// function to set records;
private void set_record(int num){
    current_record=num;
    current_pos=head+record_length*(num-1);
    if(num>=1 && num<=this.numberofrecord()){ // 1. eventid
eventid=content.substring(current_pos,current_pos+10);
current_pos=current_pos+10;

    // 2. subeventid
    subeventid=content.substring(current_pos,current_pos+10);
current_pos=current_pos+10;

    // 3. subeventname
    subeventname=content.substring(current_pos,current_pos+60);
current_pos=current_pos+60;

    // 7. facilityid
    current_pos=current_pos+22;
facilityid=content.substring(current_pos,current_pos+10);
current_pos=current_pos+10;

// 8. begin_date
begin_date=content.substring(current_pos,current_pos+8);
current_pos=current_pos+8;

// 9. end_date
end_date=content.substring(current_pos,current_pos+8);
current_pos=current_pos+8;
}
}

//===============================================
// get first record

public void first(){
    if(this.numberofrecord()>=1){
        set_record(1);
    }
}

//===============================================
// get next record

public void next(){
    if(current_record<this.numberofrecord()){
        current_record=current_record+1;
        set_record(current_record);
    }
}

//===============================================
// get previous record

public void previous(){
    if(current_record>1){
        current_record=current_record-1;
        set_record(current_record);
    }
}

//===============================================
// get previous record

public void last(){
    if(this.numberofrecord()>=1){
        current_record=this.numberofrecord();
        set_record(current_record);
    }
}

//===============================================
// End of subevent_datasource class
//===============================================

97
public class facility_datasource {

    private String facilityid;       //field # 1 char *10
    private String name;            //field # 2 char *60
    private String addr1;           // # 6 char *60
    private String addr2;           // # 7 char *60
    private String city;            // # 8 char *30
    private String state;           // # 9 char *20
    private int number_of_record;
    private int head=1323;
    private int record_length=384+1+200;
    private int current_record;
    private int current_pos;
    private String content;

    //constructor
    facility_datasource(){

        String s;
        StringBuffer con=new StringBuffer();

        s="http://www.eng.auburn.edu/~lifengl/facility.dbf";
        URL url=null;
        try {

            // Code continues here...
        }
        catch (Exception e) {

            // Exception handling
        }
    }
url = new URL(s);
    ) catch (MalformedURLException e){
        System.exit(0);
    }

    URLConnection connection=null;
    try {
        connection=url.openConnection();
    } catch (IOException e){
        System.exit(0);
    }

    InputStream in = null;
    try {
        in = new DataInputStream(new BufferedInputStream{
            connection.getInputStream()));
    } catch (IOException e) {
        System.exit(0);
    }

    byte ch[] = new byte[1];
    try {
        for (;;) {
            ch[0] = in.readByte();
            String c = new String(ch, 0);
            con.append(c);
        }
    } catch (IOException e) {
    }

    content = new String(con.toString());
    this.first();

  }

  //==================================================================================
  // numberOfrecord();
  public int numberOfrecord(){
      number_of_record=(int)((content.length()-head)/record_length);
      return number_of_record;
  }

  //==================================================================================
  // currentRecord();
  public int currentRecord(){
      return current_record;
  }

  //==================================================================================
  // get field contents
  public String facilityid(){
      if( facilityid.equals(null)){
          return (new String("null"));
      }else{
          return facilityid.trim();
      }
  }

  public String name(){

if( name.equals(null)){
    return (new String("null"));
}else{
    return name.trim();
}
}

public String addr1(){
    if( addr1.equals(null)){
        return (new String("null"));
    }else{
        return addr1.trim();
    }
}

public String addr2(){
    if( addr2.equals(null)){
        return new String("null");
    }else{
        return addr2.trim();
    }
}

public String city(){
    if( city.equals(null)){
        return new String("null");
    }else{
        return city.trim();
    }
}

public String state(){
    if( state.equals(null)){
        return new String("null");
    }else{
        return state.trim();
    }
}

//=-----------------------------
// function to set records;

private void set_record(int num){
    current_record=num;
    current_pos=head+record_length*(num-1);
    if(num>1 & & num<=this.numberofrecord()){
        // 1. facilityid
        facilityid=content.substring(current_pos,current_pos+10);
        current_pos=current_pos+10;

        // 2. name
        name=content.substring(current_pos,current_pos+60);
        current_pos=current_pos+60;

        // 6. addr1
        current_pos=current_pos+33;
        addr1=content.substring(current_pos,current_pos+60);
        current_pos=current_pos+60;

        // 7. addr2
        current_pos=current_pos+60;
        addr2=content.substring(current_pos,current_pos+60);
        current_pos=current_pos+60;
}

}
// 8. city
    city=content.substring(current_pos,current_pos+30);
    current_pos=current_pos+30;

// 9. state
    state=content.substring(current_pos,current_pos+20);
}
}

//========================================================================================================
// get first record
public void first(){
    if(this.numberOfrecord()>=1){
        set_record(1);
    }
}

//========================================================================================================
// get next record
public void next(){
    if(current_record<this.numberOfrecord()){
        current_record=current_record+1;
        set_record(current_record);
    }
}

//========================================================================================================
// get previous record
public void previous(){
    if(current_record>1){
        current_record=current_record-1;
        set_record(current_record);
    }
}

//========================================================================================================
// get previous record
public void last(){
    if(this.numberOfrecord()>=1){
        current_record=this.numberOfrecord();
        set_record(current_record);
    }
}

//========================================================================================================
// End of facility_datasource class
//========================================================================================================
(6) fees_datasource.java

// 1. fee_datasource class
// The class "fee_datasource" allows us to access the "FoxPro 2.5" database "Fee", which is stored in \WM server "http://www.eng.auburn.edu/~lifengl" or somewhere else.

// 2. Some public methods for "subevent_dataasource" class
// (1) fee_datasource() // constructor
// (2) int numberOfrecord() //return # of records in database
// (3) void first() // get first record
// (4) void last() // get last record
// (5) void next() // get next record
// (6) void previous() // get previous record
// (7) String eventid() // get content of filed "EVENTID"
// (8) String subeventid() // get content of field "SUBEVENTID"
// (9) String feename() // get content of field "FEE NAME"
// (9) String fee() // get content of field "FEE"

import java.net.*;
import java.applet.**;
import java.lang.*;
import java.io.*;

public class fees_datasource {

private String eventid; //field # 1 char *10
private String subeventid; //field # 2 char *10
private String fee; // # 4 num *12
private String feename; // # 5 char *40

private int numberOf_record;
private int head=514;
private int record_length=134+1;
private int current_record;
private int current_pos;
private String content;

//constructor
fees_datasource(){

String s;
StringBuffer con=new StringBuffer();

s="http://www.eng.auburn.edu/~lifengl/fees.dbf";
URL url=null;
try {
  url= new URL(s);
} catch (MalformedURLException e){
  System.exit(0);
}
URLConnection connection=null;
try {
    connection=url.openConnection();
} catch (IOException e){
    System.exit(0);
}

DataInputStream in = null;
try {
    in = new DataInputStream(new BufferedInputStream(
        connection.getInputStream()));
} catch (IOException e) {
    System.exit(0);
}

byte ch[]=new byte[1];

try {
    for (;;) {
        ch[0]=in.readByte();
        String c = new String(ch,0);
        con.append(c);
    }
} catch (IOException e){
}

content= new String(con.toString());
this.first();

//*****************************************************************************
// numberOfrecord();

public int numberOfrecord(){
    number_of_record=(int)((content.length()-head)/record_length);
    return number_of_record;
}

//*****************************************************************************
// currentRecord();

public int currentRecord(){
    return current_record;
}

//*****************************************************************************
// get field contents

public String eventid(){
    if( eventid.equals(null)){
        return (new String("null"));
    }else{
        return eventid.trim();
    }
}

public String subeventid(){
    if( subeventid.equals(null)){
        return (new String("null"));
    }else{
        return subeventid.trim();
    }

public String feename(){
    if( feename.equals(null)){
        return (new String("null"));
    }else{
        return feename.trim();
    }
}

public String fee(){
    if( fee.equals(null)){
        return new String("null");
    }else{
        return fee.trim();
    }
}

//===============================================
// function to set records;
private void set_record(int num){
    current_record=num;
    current_pos=head+record_length*(num-1);
    if(num>=1 && num<=this.numberofrecord()){ // 1. eventid
        eventid=content.substring(current_pos,current_pos+10);
        current_pos=current_pos+10;

        // 2. subeventid
        subeventid=content.substring(current_pos,current_pos+10);
        current_pos=current_pos+10;

        // 4. fee
        fee=content.substring(current_pos,current_pos+12);
        current_pos=current_pos+12;

        // 5. feename
        feename=content.substring(current_pos,current_pos+40);
    }
}

//===============================================
// get first record
public void first(){
    if(this.numberofrecord()>=1){
        set_record(1);
    }
}

//===============================================
// get next record
public void next(){
    if(current_record<this.numberofrecord()){
        current_record=current_record+1;
        set_record(current_record);
    }

104
}  
// get previous record  
public void previous(){  
    if(current_record>1){  
        current_record=current_record-1;  
        set_record(current_record);  
    }  
}  

// get previous record  
public void last(){  
    if(this.numberOfRecord()>1){  
        current_record=this.numberOfRecord();  
        set_record(current_record);  
    }  
}

// End of fees_datasource class  
//=================================
import java.awt.TextArea;

public class help {
    private String how_help=null;
    private String about_help=null;

    help(){
        how_help = new String("\n\nEESV On-Line Registration System\n\n\ntLast modified date: April 8, 1996.\n\n\ntDirections to use the EESV On-Line Registration System\n\ntNew Course of Engineering Extension Service at Auburn University (EESV) actually is stored in WWW server at www.eng.auburn.edu'. In order to connected the new course database to check what are new course offered recently. You can click the one of two push button with caption of 'Push Me To Connect To Database' on the first 'Welcome...' HTML Page. The new course information will pop up after you click 'Connect...'

button.\n\ntNew Course of Engineering Extension Service at Auburn University (EESV) actually is stored in WWW server at www.eng.auburn.edu'. It will be update frequently to make sure clients can obtain the most recent new course.\n\ntCurrent time on the on-line new course database?\n\ntAfter you finish the searching course(s) and wish to close or disconnect the new course database. You can\n\nt Click the push button with the caption of 'Push Me To Disconnect Database'. After you click it, the new course\n\ntwindow will disappear. You can accomplish this 'close\n\ntfunction by choosing the 'Close' menu item under 'File'\n\ntMenu on the top of new course window.\n\ntWhere is the new course database stored?\n\ntNew Course of Engineering Extension Service at Auburn University (EESV) actually is stored in WWW server at www.eng.auburn.edu'. It will be update frequently to make sure clients can obtain the most recent new course.\n\ntOnce the new course window is displayed on the screen, it is arranged in three parts. The upper part is for displaying the event information. (Event ID, Event Name, Event starting and ending dates). Event can be rotated by the buttons 'First', 'Previous', 'Next' and 'Last'. \n\ntWhen the new course window first displayed, it automatically shows the first new event record.\n\ntButton 'First' -- Show first event record in database.\n\ntButton 'Previous' -- Show previous event record in database.\n\ntIgnored if it goes to first record.\n\ntButton 'Next' -- Show next event record in database.\n\ntIgnored if it goes to last record.\n\ntButton 'Last' -- Show last event record in database.\n\nThe same functionality can be achieved by choosing corresponding menu items under 'Event' menu.\n\nt--First Event.\n\nt--Previous Event.\n\nt--Next Event.\n\nt--Last Event.\n\n\n\tThe middle part of new course window is designed for displaying the subevent information associated with the above event.\n\ntButton 'First' -- Show first subevent record in database.\n\ntButton 'Previous' -- Show previous subevent record in database.\n\ntIgnored if it goes to first record.\n\ntButton 'Next' -- Show next subevent record in database.\n\ntIgnored if it goes to last
button 'Last' -- Show last subevent record in database. The same functionality can be achieved by choosing corresponding menu items under 'Subvent' menu. --First subvent. --Previous subvent. --Next subvent. --Last subvent. The button part of new course window is designed for displaying course fee information associated with the tab above subvent. It can also be rotated by 'Previous', 'Next'. Where the subvent is changed, it automatically displays first fee information associated with the above subvent. --Show previous fee record in database. Ignores if it goes to first record. --Show next fee record in database. Ignores if it goes to last record. The same functionality can be achieved by choosing corresponding menu items under 'Fee' menu.

--Previous Fee. --Next Fee.

How to register a course?

If you want to register and get more information about EESSV, choose the menu item under 'Register'. The new input HTML form will show up for you. You can input your requirements and click 'Send Message' to submit your text. The EESSV on-line registration system will send your text input to EESSV registerer as an e-mail in text format. You will be expected to received the confirmation letter and course package.

Order to know how to use this system efficiently, check on-line help information frequently. This system may be modified slightly without notification. The on-line help will be updated correspondingly.

How to report system problems?

If you encounter any system problems or have any suggestions, please send an e-mail to Feng Li at mail: Feng.Li@eng.auburn.edu or call (334) 884-5801 to Welcome to any comments.

about_help = new String("About the 'EESSV On-Line Registration System' This application is written in Java Beta 1.0 as a part of thesis work of Feng Li who is graduate student of Computer Science and Engineering at Auburn University. The application was developed under the supervision and direction of Dr. Carlisle, who is professor in the Computer Science and Engineering at Auburn University. The application is designed for Engineering Extension Service at Auburn University (EESV) to allow clients to check and register course(s) through the internet (WWW). The current version is 1.0. April ,1996. If you have any questions or suggestions, please send an e-mail to author at Feng.Li@eng.auburn.edu.

public void set_how_help(TextArea tal){
    if( tal!= null & how_help!=null){
        tal.setText(how_help);
    }
}

public void set_about_help(TextArea ta2){
    if(ta2 != null & about_help !=null){
        ta2.setText(about_help);
    }
}