INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Teaching Assignment Planner

Wei Qi Zhang

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

March 2002

ABSTRACT

THE TEACHING ASSIGNMENT PLANNER

Wei Qi Zhang

The focus of this thesis is the development, design and implementation of a Teaching Assignment Planner project.

It can, with some constraints, be used in organizing teaching assignment. It offers suggestions about course assignment based on information about the instructor teaching experience. With the two views in the main window of Teaching Assignment Planner, the interface conveniently performs assignments and displays reports.

The application consists of instructor, course and section management subsystems, assignment views and reports, database management and ODBC connection. The application manages the TAP database developed with the Microsoft Access.

The design and implementation of the database is based on Microsoft MFC framework and Access database. The application is implemented in VC++ version 6.0, which provides a powerful set of developing and debugging tools in an integrated environment for developing object-oriented application.

ACKNOWLEDGMENT

I would like to give special thanks to Dr. Peter Grogono for his kind support and encouragement in getting me started, for his valuable suggestions and instructions while in the process of writing this thesis. It could not have been completed without his help.

Table Of Contents

List Of Figures

Chapter 1 Introduction

The Teaching Assignment Planner is an application that can be used to building schedules for university teachers. It allows the management of data such as instructor experience and workload, as well as all the sections' schedules. Constraints are automatically checked when a new assignment is entered. An update is then given based on the new assignment. If the current assignment meets some constraints, an error message is displayed and no changes will occur.

The current status of all assigned and unassigned courses, sections, and the current coordination are all reported in the report list along some statistics provided in the summary list.

1.1 What is a Teaching Assignment Planner?

The Teaching Assignment Planner stores the information about instructors, courses, sections and all assignments in the TAP database, which is a relational database, built with Microsoft Access and is managed by an ODBC connection.

The goal of this application is to provide assistance in managing the teaching assignment plan. It can provide some suggestions for assignments or give out some options based on the information already stored in the database.

The Teaching Assignment Planner consists of course, section and instructor management subsystems as well as views and reports. It manages the basic data of courses, instructors, sections, and the assignment relationship between them. Each

subsystem consists of a set of functions to load, update and store data. Any addition and cancellation of assignments is performed in these subsystems. The three subsystems perform all data management based on the records of the TAP database. These records, which can be added, deleted and updated, are managed in the database table. However, new data isn't stored in the database until confirmed. Records can be searched based on the information in the subsystem.

When entering a new assignment, the program will automatically check for constraints and notify the user of any found on conflicts. The following are some constraints:

- Not more than two different coordinators can be assigned to one course.
- Each section requires, at least, one instructor with adequate experience.
- Each course has zero or some sections, which must be scheduled in different terms, days, and must have different starting and finishing times.
- The instructor's teaching time should be well planned so that two sections connect occurring at the same time and they are not close one to another.
- One or two coordinators assigned to some course, which may or may not be assigned to a section.

Depending on which constraint is met, the Teaching Assignment Planner will either continue or abort a new assignment. If for example, an instructor is assigned as a coordinator for a course, even if the instructor's workload exceeds the pre-assigned workload value the assignment will be confirmed. Whereas other assignments will abort if, for example, an instructor is assigned to a section, which takes places at the same time as another section to which the instructor has already been assigned. The

assigned and unassigned list of course to section, section to instructor, coordinator to a course, unassigned course, unassigned section, workload and instructor's teaching experience as well as the summary of some statistics are provided in reports which also give an overview of all the information.

1.2 How the Teaching Assignment Planner works

Data operations are divided into two parts. Ordinary operations, such as the database connection (opening) and disconnection (closing) are done automatically. Other operations, such as updating the data, making new assignments and storing new data are only executed once confirmed and updated in the database. [11][12]

The Teaching Assignment Planner automatically loads all the data in the TAP database whenever the application is launched. The event list window displays the state of database operations. The information message box is launched if the Teaching Assignment Planner can't connect to the database or if some records in the database table aren't accessible.

Subsystem operations are activated whenever the user changes data by means of dialog box functions. A message box informs the user of any operational errors. For each updated data item, a default value is provided by the subsystems. With the copy and paste functions, usable within different dialogs in the same subsystem, adding and deleting data to update the operation's record, is made easier. Subsystems also have commit and rollback functions to protect all the data. The data stored in the database

remains unaffected until the commit operation is performed. On the other hand, data changes in the views are immediately stored in the database without any confirmation. Whenever a new assignment is entered, the Teaching Assignment Planner verifies if there are any constraints. The verified elements are: term, date, assignment's starting and finishing times, assigned workload, actual workload of the instructor, and the number of assigned coordinators for a course. If a conflict is found, a message box will inform the user. If, for example, the instructor subsystem finds a time conflict between two sections taught by one instructor, a warning will be given.

Subsystem operations are performed in the dialog windows with different functions like Find, Add, Update, and Delete, which are some of the usual ones. Additional functions manage information such as teaching experience, teaching assignment, and coordinator assignment, and can be found in each individual subsystem.

The instructor subsystem manages the information about instructors such as the instructor's name, ID, experience, course coordination list, the number of courses that should be taught, and the assigned and actual workloads. The operations performed in this subsystem find available instructors, to add new instructors, delete or update any instructor data in the database (such as instructor experience), teaching and coordination assignments.

The course subsystem manages all the information about courses such as course code, number, title, credits, sections and information about the need of coordinators.

Operations performed in this subsystem are to find, add, delete, and update course information.

The section subsystem manages the information about the sections like section ID, session, code, teaching days, and start and finish times. The operations performed in this subsystem are to find, add sections, and update the information of sections.

The views operation allows the user to have a visual idea of all assignments. Information such as which course is assigned to which section, course coordinator assignments, teaching assignments, and suggestions for instructor teaching assignments, are provided in this operation. Colored icons identify the state of each item.

There are two separated views in the application frame. The left view is used mainly to display data, such as sections left to be assigned. The right view displays possible options for this new assignment such as instructor information. The right view also displays reports and summaries of statistics. Assignments usually end in a cooperation of the two views. New and cancelled operations are therefore resumed in the right view.

Views also provide reports on different combinations of courses, sections, instructors and coordinators. These reports provide summaries and data details such as the number of instructors, courses, courses requiring coordination, courses with an

assigned coordinator. sections. and sections with an assigned instructor. Other reports list courses and sections including their assigned coordinator and instructors assigned to each section.

A message box is used to notify the user of an internal event. or of the state of the application's execution. All message boxes display the information with standard Window operating system icons.

The information message boxes can be divided into three categories.

Stop: Notifies the user that an error was detected by the application. therefore the current operation aborts.

Information: Notifies the user that an error or event was detected by the application so the current operation may or may not be performed properly. It may also just abort.

Selection: The user can decide whether to continue or to cancel the current operation before the procedure goes on.

Colored icons are displayed and help identify the state of the item presently in the view. These colored icons can be divided into 6 categories.

Red: If the instructor's workload is exceeded.

If the section remains unassigned (with no instructor).

If the course needing a coordinator remains unassigned.

Green: If the instructor's workload is just right.

If the instructor has been assigned for the section.

If (a) coordinator(s) has (have) been assigned for a course.

Yellow: If the instructor's workload is incomplete.

If the section has not yet been assigned to an instructor.

If the course has not yet been assigned to a section.

Blue: If the instructor has teaching experience in this course

Gray: If the instructor is assigned to the section.

Purple: If the instructor is assigned as coordinator of the course.

The Teaching Assignment Planner manages all the data stored in the TAP database, a relational database built in the Microsoft Access database, connected by ODBC. This architecture benefits from the flexibility and efficiency of ODBC API [10] and framework MFC [9]. Classes can hence be managed in association with the records of the TAP database.

1.3 Organization of this Report

This report is organized as follows:

Chapter 1 introduces the Teaching Assignment Planner application, which can be used to assist the assignment of University courses, instructors and coordinators.

In Chapter 2, gives a breakdown of the development of the Teaching Assignment Planner database done by analyzing some tools used to build this application.

Chapter 3 reviews the design issues that were considered to implement this project. The architectural design of the Teaching Assignment Planner is explained and an E-R model of the TAP database given.

Chapter 4 explains the implementation of this project and details are given in the form of codes, figures and tables.

Chapter 5 is the conclusion on developing this project.

The Appendix presents menu structure that can help the user to understand how to use the application.

Finally, the reference indicates helpful documents, which were used.

Chapter 2 Background

In this chapter, the basic development of the Teaching Assignment Planner project is presented. The design, implementation and strategies to deal with some of the constraints occurring in the teaching, coordinating and section assignments are also discussed. Finally, some issues in relation with the project's environment and development tools provided or available in the platform are also discussed.

2.1 Platform and development tools

The first issue in designing and implementing Teaching Assignment Planner was to find an appropriate platform. [2][3] The application for the Teaching Assignment Planner must: (1) enable the user to access data in the database, (2) make logical reasoning, (3) verify if there are constraints caused by new assignments, give out the final result on the screen, and store it in the database. All functions implemented should be interfaced with the end-user, and with the database.

The Windows operating system is the best platform to develop and implement this project [4] [5]. Here are the reasons why the Windows operating system was chosen.

1. **The Microsoft Access database is available.**

The Access database is the most popular and powerful database management system (DBMS) provided by Microsoft. Access is a relational database, which can be used for developing the TAP database.

2. **ODBC is very flexible and powerful.**

ODBC is the database portion of the Microsoft Windows Open Services Architecture (WOSA), an interface that allows Windows-based desktop applications to connect to multiple computing environments without rewriting the application for each platform.

ODBC is a call-level interface that allows applications to access data in any database for which there is an ODBC driver. Using ODBC, database applications can access any database for which the end-user has an ODBC driver. ODBC provides an API that allows the application to be independent of the source database management system (DBMS). The Access driver is already installed in ODBC when published by Microsoft.

The following are components of ODBC:

- ODBC API

A library of function calls, a set of error codes, and a standard Structured Query Language SQL syntax for accessing data on DBMSs.

- ODBC Driver Manager

A dynamic-link library (ODBC32.DLL), it loads ODBC database drivers on behalf of an application. This DLL is transparent to application.

- ODBC database drivers

One or more DLLs that process ODBC function calls for specific DBMSs.

- ODBC Cursor Library

A dynamic-link library (ODBCCR32.DLL), it resides between the ODBC Driver Manager and the drivers and allows scrolling through the data.

- ODBC Administrator

A tool used to configure a DBMS and to make it available as a data source for an application.

An application achieves independence from DBMSs by working through an ODBC driver written specifically for a DBMS rather than working directly with the DBMS. The driver translates the calls into commands its DBMS can use, simplifying the developer's work, and making it available for a wide range of data sources.

The database classes any data source that has an ODBC driver. This might, for example, include a relational database, an Indexed Sequential Access Method (ISAM) database, a Microsoft Excel spreadsheet, or a text file. The ODBC drivers manage the connections to the data source, and SQL is used to select records from the database.

3. MFC provides the classes to manage the database and create user interfaces.

Some classes are listed below:

CDatabase:

A CDatabase provides improved support for transactions. Its objects represent the connection to a data source, which allows the applications to operate on it. Data source is a specific instance of data hosted by a database management system (DBMS) on of which is Microsoft Access is one of the available DBMS. Applications can have more than one CDatabase objects active at a time.

With the class of CDatabase, commit or rollback functions are available and enable transactions.

CRecordSet:

A CRecordset object represents a set of records selected from a data source. Known as "recordsets," CRecordset objects are typically used in two forms: dynasets and snapshots. A dynaset stays synchronized with data updates made by other users. A snapshot is a static view of the data. Each form represents a set of

records fixed at the time the recordset is opened. When an application is scrolled to a record in a dynaset, it reflects changes subsequently made to the record, either by other users or by other recordsets in the same application. CRecordSet encapsulates a set of records selected from a data source. Recordsets enable scrolling from record to record, updating records (adding, editing, and deleting records), qualifying a selection with a filter, sorting the selection, and parameterizing the selection with information obtained or calculated at run time.

CRecordView:

A CRecordView provides a form view directly connected to a recordset object. The dialog data exchange (DDX) mechanism exchanges data between the recordset and the controls of the record view. Like all form views, a record view is based on a dialog template resource. Record views also allow moving from one record to another in the recordset, updating records, and closing the associated recordset when the record view closes.

CDBException:

A CDBException object represents exceptions arising from the database classes. The class includes two public data members, which applications can use to determine the cause of the exception or to describe it by means of a text message. CDBException objects are constructed by member functions of the database classes. An exception results from failures in data access processing. This class serves the same purpose as other exception classes and in the same way as the class library.

CFieldExchange:

The CFieldExchange class supports the record field exchange (RFX) and the bulk record field exchange (Bulk RFX) routines used by the database classes. This class is used if data exchange routines are needed for custom data types or when bulk row searching is being implemented. Otherwise, this class is not used directly. RFX and Bulk RFX exchange data between field data members of a recordset object and the corresponding fields (table columns) of the current record on the data source.

4. A great number of interface controls or components can be available in MFC classes.

The MFC is a Microsoft product used as a framework to develop Window-based desktop applications. There are many classes available to implement GUI. Some of them are listed below:

CDialog:

The CDialog class is the base class used to display dialog boxes on the screen. Two types of Dialog boxes exist: modal and modeless. The user must close a modal dialog box before the application can continue whereas a modeless dialog box allows the user to display a dialog box and to return to another task without canceling or removing it.

A dialog box, like any other window, receives messages from Windows. In a dialog box, applications can handle messages from the dialog box's control.

CPropertySheet:

An object of class CPropertySheet represents property sheets, otherwise known as tab dialog boxes. A property sheet consists of a CPropertySheet object and of one or more CPropertypage objects. A property sheet is displayed, by the framework, as a window with a set of tab indexes by means of which the user can select the current page, and an area for it.

CPropertyPage:

An object of class CPropertyPage represents individual pages of a property sheet. As with standard dialog boxes, it can classify each page in property sheet of the CPropertyPage. [1]

2.2 Constraints Consideration

Before active data in the database is updated by a new assignment and displayed on the screen, constraints should be checked in a logical manner. If the assignment meets any constraints, the operation should be aborted or canceled. Following are a list of aspects that should be considered so to avoid constraints.

1. IF more than one section is assigned to the same instructor, a different time is required for each assignment. Also, an appropriate period of time should be left between sections.
2. It is suggested that a course section should be assigned to an instructor with adequate teaching experience.

3. The workload of an instructor can increase by three points if a new section is assigned and can decrease by three points if an assigned section is cancelled.
4. If a teacher is assigned as a coordinator to a course, the workload differentiates by only one point (increase/decrease).
5. As an instructor's workload shouldn't exceed 16 points and the total workload should be recalculated before the new assignment is activated in the database. If the instructor's workload exceeds 14 points, the user should get notified.
6. Not more than two coordinators can be assigned to one course, if (a) coordinator(s) is (are) needed for the course.
7. If two coordinators are needed by one course, they should be two different instructors.
8. Courses can be assigned as teaching sections in one, two or three terms or not at all. In any case, the course information should be saved in the database.
9. A course can be assigned to one or more sections. However, only one instructor can be assigned to a section.
10. One section can be taught once or twice each week but at different times.

2.3 Components in The User Interface

The interface of the Teaching Assignment Planner consists mainly of Perportysheet, dialog, message box, and list view, all used in different subsystems and for different uses. Next are all the components of the interfaces:

1. Menu

The menu includes items, which allow the user to perform different operations. Any selection will cause an event passing through the message map mechanism of MFC by Windows operating system.

There are three main categories of operations included in the menu.

- **Table operations**. The table operations include all operations of the course table, the section table, and the instructor table. These three subsystems functions manage the data in the TAP database. Each subsystem can update, add or delete data in these tables.
- **Views operations**. There are two views in the main frame of the application of the Teaching Assignment Planner: left view and right view. These two views are driven from the CListViews class. The left view provides mainly the view functions which display some basic data retrieved from the database. The right view provides cooperation with the left view for assignment functions. It also displays some lists and reports data.
- **Reports operations**. This type of operations reports data displayed on the right view. Reports provide an overview of the application's managed data and some numbers of courses, sections and instructors.

2. Dialog box.

The dialog boxes are used for interactions between the end-user and the applications. The subsystems use the dialog to get the user's data key in and to display the data from the database to the user. In these dialog boxes there are some controllers, which

are buttons, list boxes, or edit windows. Dialog boxes and controllers offer user-friendly interface and functional performance for the user's operations.

3. Message box.

The message box is used to notify a user of an event that occurred in the application. The three categories of message boxes are described in Chapter 1. Usually the information of a message box informs the user of a wrong doing in the current operation.

4. List views.

The list views provide help to the user in assignment operations by allowing the selection of different items on different views. Also, it displays lists and reports with flexible formats (column width, position and row arrangements). Displaying the results of statistic is hence, more flexible and effective.

5. Icons.

The icons are used to identify the state of an item in the list view. There are 6 different icon colors used to identify the state of a course, a section, or an instructor. The description of these icons is made in the Chapter 1.

2.4 Summary of Survey

Further analysis and study on database related applications and topics [13][14][15] were made. [1][2][3] The database applications should all have at least three parts, or, layers, which cooperate together to perform the functions provided by the application [16][17]. For the Teaching Assignment Planner project, the three layers built as application architecture are the following.

- **Database management layer.**

  This layer is in charge of managing data in the database. It is used as an interface between the database and the application document layer, which manages the data that will be saved in the database or which has been retrieved from it. The CRecordSet and CDatabase derived classes are in this layer.

- **The document/Data management layer.**

  This layer is in change of managing and re-organizing data into logical categories. It interfaces both the database management layer and the user interface layer. The Cdocument class, which will be introduced in Chapters 3 and 4, belongs to this layer.

- **The user interface layer.**

  This layer interfaces applications with end users. It is in charge of displaying data to the user, in the window. The displayed data depends on the document layer, and on user activities in interfaced components, which communicate with the application. The user interface is divided into two parts in the Teaching Assignment Planner, one part is dialog box, used in the subsystem, and the other is ListViews, used to report and make some assignments. Both can display the data and get user action to the application.

Chapter 3 Design of the Teaching Assignment Planner

The design of the Teaching Assignment Planner consists of system architecture, database design and application architecture. Figures and tables are used in this chapter to explain the design of the Teaching Assignment Planner. All design issues are solved in the system architecture, the database design, and the object-oriented design of application classes.

3.1 System Architecture

The system consists of three parts: application, ODBC, and database.

In this project, the DBMS of Access, developed by Microsoft, is connected at the low level to the ODBC. The TAP database is built in Access and the E-R model explains the details about it.

Open Database Connectivity (ODBC) was Microsoft's first cohesive effort to design an API that could be used by C programmers. ODBC provides its own flavor of SQL. The application passes ODBC SQL to ODBC, which translates the ODBC SQL into the flavor of SQL appropriate for the used DBMS. [11] [12] [10]

ODBC is a proven interface, which makes it possible for an application to access relational data from a variety of database management systems, and therefore ODBC is used to build many database applications.

The ODBC architecture consists of a top-level driver manager (odbc32.dll) and many DBMS-specific DLLs, known as drivers. The driver manager loads and unloads the native drivers, receives, requests from the application, and manages the subsequent driver actions. The driver handles the communication between the driver manager and the data source.

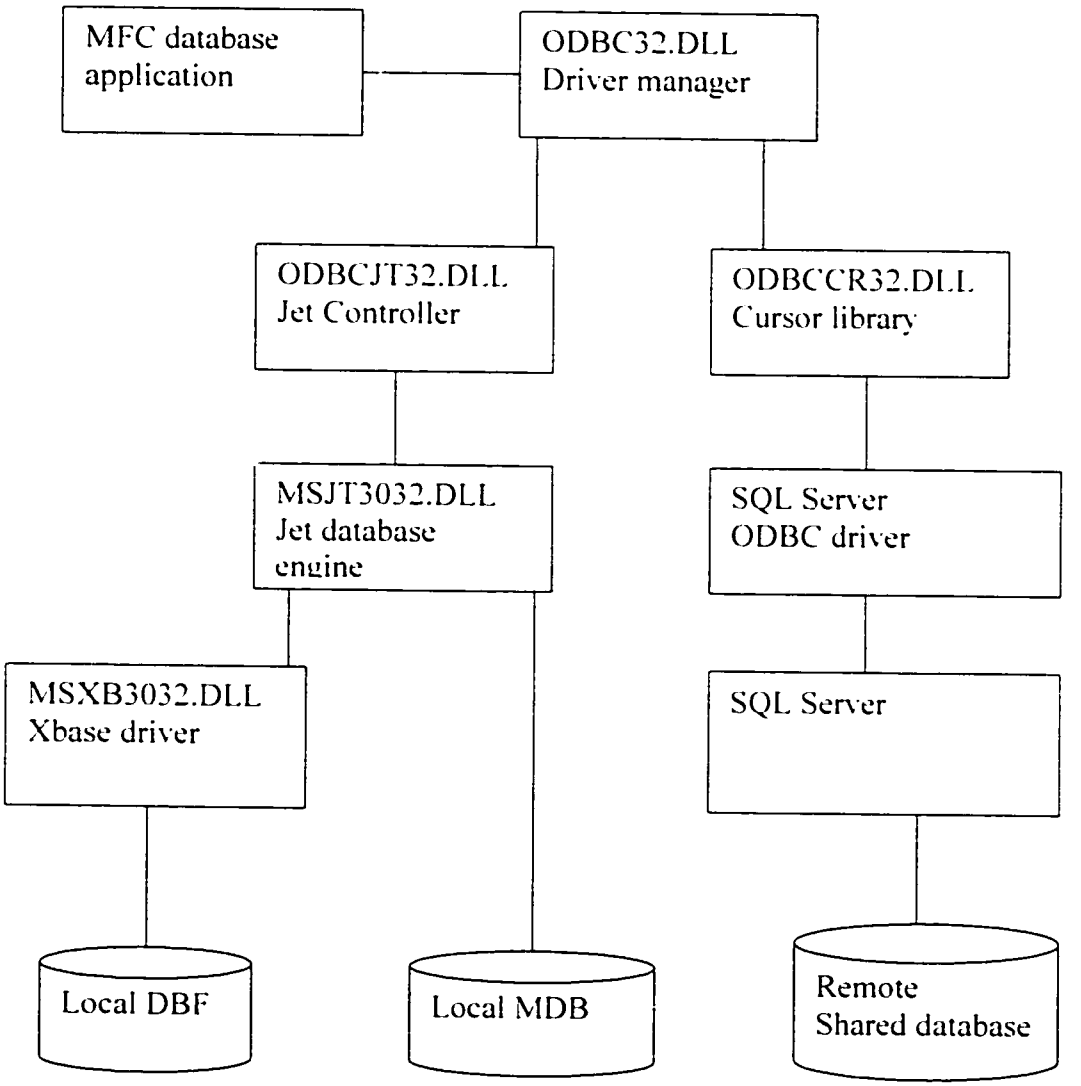

Figure 1: The architecture of ODBC

In the application part, MFC (Microsoft Foundation Classes) provides two wrapper classes for the ODBC API. The CDatabase class represents the database source, and the CRecordSet class represents the data itself. All the details about the results from these two classes will be discussed in the following chapters.

3.2 Database Design

After analyzing the requirements for the Teaching Assignment Planner, three entities and four relations were involved in this application design [13] and are listed below:

Entities

1. Course:

   It records all the information about a course, such as the code, the number, the title, the credits and, if needed, the coordinators. The course code and number are two attributes in the combination of the primary key.

2. Section:

   It records all information about a section, such as the session, the session code, the starting time, etc. The primary key for this entity is Section_ID built with the combinations of course code and number, session and session code.

3. Instructor:

   It records all information about an instructor, such as the name, the instructor_ID, the workload, etc. The primary key of this entity is the instructor ID.

Relations:

1. Teaching experience:

   It records the teaching experience for each instructor, presented by course code and number. The instructor's ID represents the instructor. All the attributes in this table are primary key.

2. Coordination assignment:

   It records the coordination assignments to a course for each instructor. The course code and number represent the course, the instructor's ID represents the coordinator and all attributes in this table are primary key elements.

3. Section assignment:

   It records all the sections assigned to each course. The course code and number represent the course, the section is represented by the section_ID and once again, all the attributes in this table are primary key.

4. Teaching assignment:

   It records all the courses assigned to instructors. The course code and number represent the course, the instructor's ID represents the instructor and all the attributes in this table are primary key.

3.2.1 E-R model

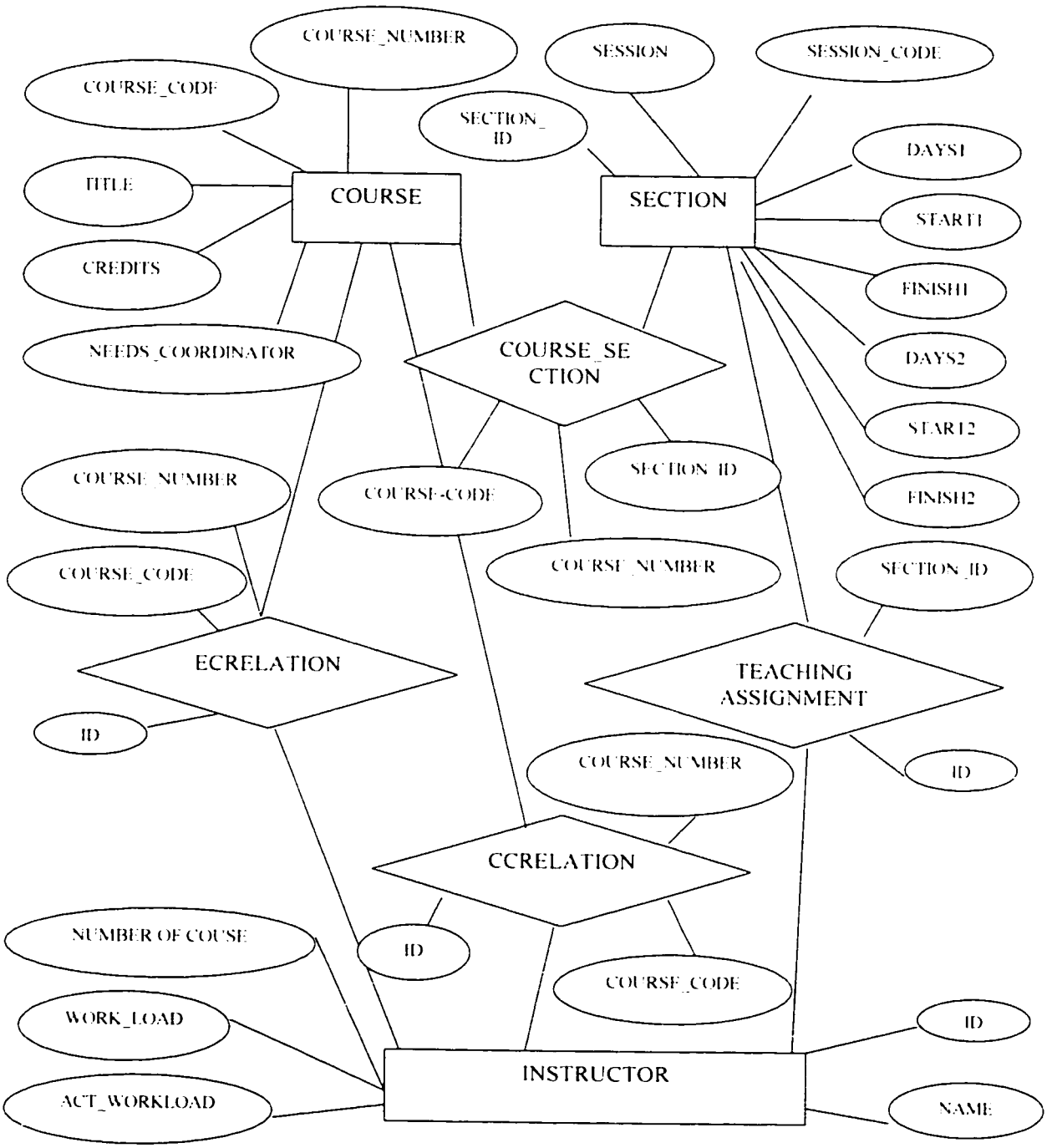

Figure 2: The E-R model of TAP

3.2.2 The schema of COURSE

COURSE (COURSE_CODE, COURSE_NUMBER, TITLE, CREDITS, NEEDS_COORDINATOR)

Description:

FIELD	DATA TYPE	KEY	EXAMPLE
COURSE CODE	Varchar(4)	PRIMARY	COMP
COURSE NUMBER	Varchar(4)		6451
TITLE	Varchar(50)		DATA DESIGN
CREDITS	Varchar(8)		4.0
NEED_COORDINATOR	BOOL		YES

- The Primary key is the combination of COURSE_CODE and COURSE_NUMBER.
- This table is in relationship with the sections.

  The table defines some courses, which may not be assigned to any sections in the current term, or not in any terms. Some courses may only be assigned to one section, and others may be assigned to more than one. The relationship between course and section is 1 to many (0 – N). The relation of COURSESECTION defines the relationship between course and section.
- This table is in relationship with instructors and defines their experience teaching this course.

  Instructors have their course teaching experience presented in a list of courses, defined in this course table. Normally, instructors have experience in teaching some courses, and therefore some courses are taught by only one instructor. Finally, it is also possible that no instructor has the experience of one specific course. The teaching experience of instructors varies so the relationship between course and instructor on teaching experience is many to many (N – M). The relation of ECRELATION defines the relationship between teaching experience and course.

- This table is in relationship with instructor and the role of coordinator if assigned so. Defined in this table are the courses where instructors act as coordinators.

  The relationship between course and instructor as coordinator is (N – M) because one course can have more than one coordinator and each instructor can coordinate more than one course. The relation of Coordination defines the relationship between the instructor and the course. CCRELATION is the relation name.

3.2.3 The schema of SECTION

SECTION(SECTION_ID, SESSION, SESSION_CODE, DAYS, START, FINISH)

Description:

SECTION(SECTION_ID, SESSION, SESSION_CODE, DAYS, START, FINISH)

Description:

FIELD	DATA TYPE	KEY	EXAMPLE
SECTION_ID	Varchar(11)	PRIMARY	COMP64512AX
SESSION	Varchar(1)		2
SESSION_CODE	Varchar(2)		AX
DAYS1	Varchar(1)		M for Monday
START1	Varchar(5)		10:20
FINISH1	Varchar(5)		12:20
DAYS2	Varchar(1)		F for Friday
START2	Varchar(5)		10:20
FINISH2	Varchar(5)		12:20

This schema defines the table of section. which describes the information needed for sections assigned to the course. defined in the course table.

- The primary key is SECTION_ID.
- This table is in relationship with course.

Each section defined in this table can be assigned with only one course defined in the course table. The relationship between course and section is 1 to many. because one section can only be assigned to one course. but one course can have more than one section.

The table COURSESECTION defines the relationship between course and section.

- This table is in relationship with the instructor.

Only one instructor can teach a section but each instructor can teach more than one section. The relationship between instructor and section is 1 to many (1 – N).

The relation TEACHINGASSIGNMENT defines the relationship between section and instructor.

- There are some constraints on this table.

More than one section can be assigned to the course defined in the course table but the sections must be defined as different sessions or in the same session. but with different session codes. That means that in one term. one course can be assigned to more than one section with different session codes.

The session should only exist if the course exists. If no course is defined in the course table. no session should be defined in the section table for that course.

3.2.4 The schema of INSTRUCTOR

INSTRUCTOR(ID, NAME, NUMBER_COURSE, WORKLOAD, ACTUAL_WORKLOAD)

Description:

FIELD	DATA TYPE	KEY	EXAMPLE
ID	Varchar(8)	PRIMARY	00000001
NAME	Varchar(50)		Soheila Kersuova
NUMBER OF COURSE	Int		4
WORK_LOAD	Int		16
ACT_WORKLOAD	Int		10

This schema defines the table of instructor.

- The primary key is instructor ID.
- This table is in relationship with the course and defines the teaching experience and coordination of an instructor.

One instructor can have teaching experience for more than one course. More than one instructor can have teaching experience for the same course. The relationship between teaching experience of a course and instructor is N-M.

As an instructor can be coordinator for more than one course and that more than one instructor can be coordinator for one course. the coordination relationship between course and instructor is N-M.

- This table is in relationship with section and teaching assignment.

Only one instructor can be assigned to teach each section but one teacher can be assigned to teach more than one section. The relationship of teaching assignment between instructor and section is 1 to many.

3.2.5 The schema of COURSESECTION

TEACHING_ASSIGNMENT(SECTION_ID, ID)

Description:

FIELD	DATA TYPE	KEY	EXAMPLE
COURSE_CODE	Varchar(4)	PRIMARY	COMP
COURSE_NUMBER	Varchar(4)	PRIMARY	6401
SECTION_ID	Varchar(11)	PRIMARY	COMP6401IA

This schema defines the relationship between course and section (1 – N). One course can be assigned to many sections but one section can only be assigned to one course.

All attributes are Primary key. Details are from the scheme of COURSE and INSTRUCTOR.

3.2.6 The schema of TEACHINGASSIGNMENT

TEACHING_ASSIGNMENT(SECTION_ID, ID)

Description:

FIELD	DATA TYPE	KEY	EXAMPLE
SECTION_ID	Varchar(11)	PRIMARY	COMP64512AX
ID	Varchar(8)		10000011

This schema defines the relationship between the instructor and a section (1 – N). One instructor can teach more than one section in the same term but each section can only be taught by one instructor.

Primary key is SECTION_ID. Other details are found in the scheme of COURSE. INSTRUCTOR.

3.2.7 The schema of ECRELATION

ECRELATION(ID, COURSE_CODE, COURSE_NUMBER)

Description:

FIELD	DATA TYPE	KEY	EXAMPLE
ID	Varchar(8)	PRIMARY	10000001
COURSE_CODE	Varchar(4)	PRIMARY	COMP
COURSE_NUMBER	Varchar(4)	PRIMARY	6451

This relation defines the relationship of teaching experience between instructor and course (N-N).

All the attributes in the table are Primary key. Other details are found in the scheme of COURSE. INSTRUCTOR.

3.2.8 The schema of CCRELATION

CCRELATION(ID, COURSE_CODE, COURSE_NUMBER)

Description:

FIELD	DATA TYPE	KEY	EXAMPLE
ID	Varchar(8)	PRIMARY	10000001
COURSE_CODE	Varchar(4)	PRIMARY	COMP
COURSE_NUMBER	Varchar(4)	PRIMARY	6451

This relation defines the relationship of coordination between instructor and course (N-N).

All the attributes in the table are Primary key. Other details are given in the scheme of COURSE, INSTRUCTOR.

3.2.9 Data integrity

To keep data integrity, the TAP database is equivalent to the Access Referential Integrity. Referential Integrity is a system of rules, which makes sure that the relationship between rows in related tables are valid and that user cannot accidentally delete or change related data. Data changes in the database only occur if they respect the Referential Integrity system.

The following rules are observed with referential integrity.

- User cannot enter a value in the foreign key column of the related table if that value doesn't exist in the primary key or in the related table except a null. For example, a user cannot assign a course to an instructor if that course does not exist in the course table, or if the instructor does not exist in the instructor table. But a user can enter an instructor or a course individually into the database before assigning a course to an instructor.
- User cannot delete a row from a primary key table if there are rows matching in a related table.
- User cannot change a primary key value in the primary key table if that row has related rows.

For more secure operations and for data protection, TAP also follows the cascading update and cascading deletes.

For relationships in which referential integrity is enforced, TAP automatically follows cascade update and cascade delete related records. Setting these options allow delete and update operations, which are normally prevented by referential integrity rules. When the user deletes records or change primary key values in a primary table, TAP makes the necessary changes to related tables in order to preserve referential integrity.

TAP automatically updates the primary key in all related records any time a user makes some changes because of cascade update and cascade delete, which define the

relationship. For example, if a user changes an instructor's ID in the Instructor table, the instructor ID field in the teaching experience table is automatically updated for each course taught by this instructor so that the relationship doesn't break. TAP cascades updates without displaying any message to the users.

When a user deletes records in the primary table, TAP automatically deletes related records in related table. For example, if user deletes a course record from the course table, the teaching experience records of that course are automatically deleted from the teaching experience table (this includes records in the SECTION_CCRELATION table related to the COURSE records).

3.3 Application architecture

3.3.1 Document/View architecture

The document/view architecture is the basic programming model of MFC for building a window-based application. Document/view implementation in the class library separates data from its display and from most user operations on the data. In this model, an MFC document object reads and writes data into permanent storage. The document may also provide an interface to the data wherever it resides (such as in a database). A separate view object manages data display, from rendering the data in a window to user selection

and editing of data. This view displays data from the document and communicates it back to the document for any changes.

Document is associated with views. A document template creates the frame windows that frame the views and is responsible for creating and managing documents of the same type.

The CDocument class provides the basic functionality for programmer-defined document classes. A document represents the unit of data that the user typically opens with the Open command of the File menu and saves with the Save command of the File menu. In the Teaching assignment Planner, the document class manages all data access to the TAP database for views and reports. The open and close database source is automatically done when the user selects these functions. All changes to the data are managed through the document class. The view uses this interface to access and update the data.

The CListView class provides the basic functionality for programmer-defined view classes. A view is attached to a document and acts as an intermediary between the document and the user. It renders an image of the document on the screen and interprets user input as operations upon the document. The CListview provides a lot of functions for displaying lists and reports. Some member functions are basically available for the

user to click on the screen and the items. Inheritance of the base class CListView, are the two derived classes (CTAPView and CSubCtrolList) developed to display all the information and to get user actions applied to the items, so that the item operation benefits the user operation on the view.

MFC's document/view architecture makes it easy to support multiple views, single document interface, splitter windows, and other valuable user-interface features. At the heart of document/view of the application are five key classes:

- CTAPDoc:

  This class derives from CDocument and is used to store or control the program's data. CTAPDoc gets two pointers to point to the CTAPView object and CSubCtrlList object.

- CTAPView:

  This class derives from CListView and is used to display a document's data and manages user interaction with items operation. It is different from other views, because the item unit is the basic operation element on the screen. CTAPView can activate three other objects, CInstructorTableFontSheet, CCourseTableFontSheet and CSectionTableFontSheet. These three objects serve as interfaces to the user and hold CProportypePage objects to implement three subsystems.

- CSubCtrlList:

This class derives from CListView and is also used to display the document's data and to manage user interaction with items operation. Additional functions distinguish this view from CTAPView. However, for assignment operation, cooperation exists with the object of CTAPView for view and report display.

CFrameWnd:

CFrameWnd, is a class used to provide the frame which contains the two views. The CFrameWnd class provides the functionality of a Windows single document interface (SDI) overlapped or with a pop-up frame window, along with members for managing the window. If a CFrameWnd object contains views and documents, the framework would indirectly create them. The CDocTemplate object orchestrates the creation of the frame, the creation of the containing views, and the connection of the views to the appropriate document. The parameters of the CDocTemplate constructor specify the CRuntimeClass of the three involved classes (document, frame, and view). New frames are dynamically created by the framework using a RuntimeClass object.

A CFrameWnd contains default implementations to perform the following functions needed by a typical Windows application:

- A CFrameWnd frame window keeps track of a currently active view independent of the active window or of the current input focus.

- Command messages and many common frame-notification messages, including those handled by the OnSetFocus, OnHScroll, and OnVScroll functions of CWnd, are brought by a CFrameWnd frame window to the currently active view.

- The currently active view can determine the caption of the frame window. Turning the FWS_ADDTOTITLE style bit of the frame window off can disable this feature.

- A CFrameWnd frame window manages the positioning of the control bars, the views, and other resulting windows inside the frame window's client area. A frame window also updates toolbars and other control-bar buttons and also has default implementations of commands for putting the toolbar and status bar on and off.

- A CFrameWnd frame window manages the main menu bar. When a pop-up menu is displayed, the frame window uses the UPDATE_COMMAND_UI mechanism to determine which menu items should be enabled, disabled, or checked. When the user selects a menu item, the frame window updates the status bar with the message string for that command.

- A CFrameWnd frame window has an optional accelerator table that automatically translates keyboard accelerators.

- A CFrameWnd frame window has an optional ID help set with a LoadFrame used for context-sensitive help. The frame window is the main orchestrator of semi modal states such as context-sensitive help (SHIFT+F1) and print-preview modes.

- CDocTemplate:

  CDocTemplate is an abstract base class, which defines the basic functionality for document templates. The application usually creates one or more document templates in the implementation of the **InitInstance** function. A document template defines the relationships among three types of classes:

  - A document class, with an application deriving from CDocument.

  - A view class, displaying data from the document class listed above.

  - A frame window class, containing the view. For a single document interface (SDI) application, the class is derived from CFrameWnd.

  - The document template stores pointers to the CRuntimeClass objects for the document, view, and frame window classes. These CRuntimeClass objects are specified when constructing a document template.

The document template contains the ID of the resources used with the document type. The document template also has strings containing additional information about its document type. These include the name of the document type and the file extension. [9]

In a running application, these objects cooperatively respond to user actions, bound together by commands and other messages. A single application object manages one document template. Document template creates and manages one or more documents. The user views and manipulates a document through a view contained inside a frame window.

3.3.2 Splitter Window and List View

A splitter window is used in the Teaching Assignment Planner to display the views and reports and to perform some operations in relation to an assignment. The splitter window benefits the user by splitting the window into two or more scrollable panes. A splitter control in the window frame next to the scroll bars allows the user to adjust the relative sizes of the panes. Each pane is a view on the same document.

A view can be attached to only one document, but a document can have multiple views attached to it. For example the document is displayed in a splitter window, an application can support different types of views for a given document type. These different types of views can be placed in separate frame windows or in separate panes of a single frame window if a splitter window is used.

A view may be responsible handling several different types of input, such as keyboard input, mouse input or input via drag-and-drop, as well as commands from menus.

toolbars, or scroll bars. A view receives commands forwarded by its frame window. If the view does not handle a given command, it forwards the command to its associated document. Like all command targets, a view handles messages via a message map. [8]

The Teaching Assignment Planner uses static windows with views created by different classes, and starts with a window split into two panes, each with a different purpose. The left pane displays view's results or the data to be assigned and the right pane displays reports and cooperation assignments.

The CView class provides the basic functionality for user-defined view classes. CListView is derived from Cview and makes it easy to integrate a control list in the MFC document/view architecture, encapsulating the control as much as CEditView encapsulates an edit control. This control fills the entire surface area of an MFC view.

3.3.3 CRecordSet inheritance

In the MFC, several classes are provided to manage the database. One row of stored data in the database table is called record. A set of record can be retrieved through Structured Query Language (SQL). After a set of record gets into the memory, or controlled by the application, update, delete, and add functions become available. For each record set, FMC provides the CRecordSet class to manage it.

A CRecordset object represents a set of records selected from a database. CRecordset objects are typically used in two forms: dynasets and snapshots. A dynaset stays synchronized with data updates made by other users. A snapshot is a static view of the data. Each form represents a set of records fixed at the time the recordset is opened, but when scrolled to a record in a dynaset, it reflects the changes subsequently made to the record, either by other users or by other recordsets in the application.

The application takes an application-specific recordset class from CRecordset to work with it. Recordsets select records from a data source, and the application can then:

- Scroll through the records.
- Update the records and specify a locking mode.
- Filter the recordset to constrain which records it selects from those available on the data source.
- Sort the recordset.
- Give parameters to the recordset to customize its selection with information, which remains unknown until run time.

The derived class in the application is used to be connected to the database and to build a recordset object, giving the constructor a pointer to CDatabase object. The user may then select the recordset's Open member function if the application specifies whether the object is a dynaset or a snapshot. Selecting the Open function pulls out data from the

data source. After the recordset object is opened. member functions and data members can be used to scroll through the records and to operate on them.

To refresh records that may have been changed or added since the Open selection. the object's Requery member function can be selected. The Close member function is used to destroy the object when the application is finished with it.

In a derived CRecordset class. record field exchange (RFX) or bulk record field exchange (Bulk RFX) is used to support reading and updating record fields.

Listed below are the classes derived from CRecordSet for the Teaching Assignment Planner.

- CCourseSet

  The object of this class manages the records retrieved from the Course table database. It manages the operations of adding. updating and deleting course records.

- CSectionSet

  The object of this class manages the records retrieved from the Section table database. It manages the operations of adding. updating and deleting sections records.

- CInstructorSet

  The object of this class manages the records retrieved from the Instructor table database. It manages the operations of adding. updating and deleting records of instructors. instructors' teaching experience. coordinating and teaching assignment.

- CandCRelationSet

The object of this class manages the records retrieved from the CCRelation table database. It manages the operations of adding, updating and deleting coordination records.

- CEandCRelationSet

The object of this class manages the records retrieved from the ECRelation table database. It manages the operations of adding, updating and deleting teaching experience records.

- CCourseSectionSet

The object of this class manages the records retrieved from the CourseSection table database. It manages the operations of adding, updating and deleting records of section assignment.

- CCTeachingAssignmentSet

The object of this class manages the records retrieved from the Teaching Assignment table database. It manages the operations of adding, updating and deleting records of teaching assignment.

Details will be discussed in Chapter 4.

Chapter 4 Implementation of the Teaching Assignment Planner

The design of the Teaching Assignment Planner includes a database presented as a window-base related application, which inherits some classes from MFC. The TAP database runs on the Window Microsoft Access operating system as DBMS and therefore the implementation directly and clearly inherits from classes in MFC. The application interface is shown below:

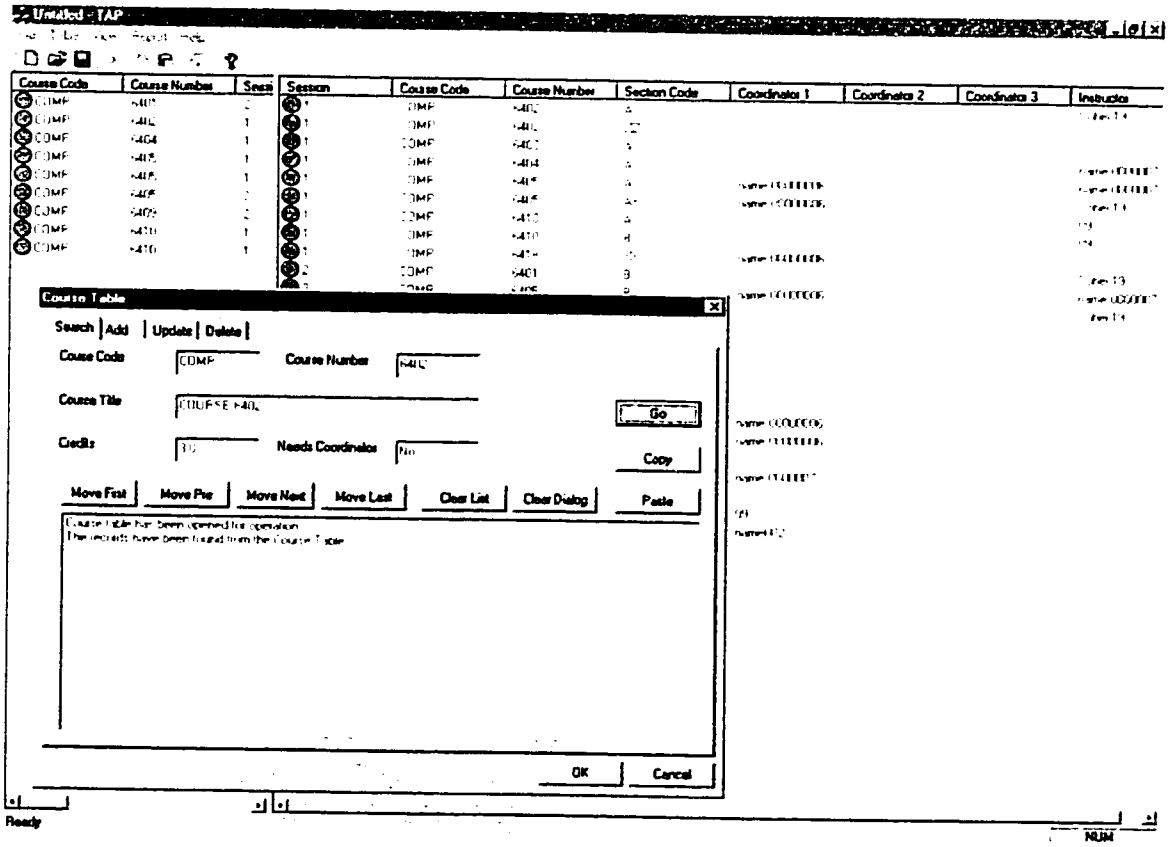

Figure 3: The interface of theTeaching Assignment Planner

4.1 The Application Architecture

4.1.1 Programming Language and tools

The Microsoft Visual C++ is the best language.

VC++ includes:

- VC++ IDE provides programmer with a framework that they can extend using other C++ sources and tools.
- VC++ incorporates ANSI C, necessary for Object -Oriented programming.
- MFC class wizard includes and integrates development environment (IDE) to create a main frame and to develop the application.
- All the components to create menus, toolbars, icon edit and other elements are included in the VC++ IDE.
- The MFC library is used to develop applications. [7]

4.1.2 The Application Framework

MFC provides reusable codes in the form of pre-written C++ classes. The application uses a class library, and applies the classes to specific problems, which the application has to solve. The MFC libraries provide such utility classes, it also implement an application framework. In addition to the functionality normally presented by simple class libraries, this framework offers a backbone for applications. The application

framework built into MFC manifests itself in classes and data structures designed to help the creation of an application and its fundamental features. [8]

The application framework helps initializing and running application with the capability to respond to messages sent by Windows at runtime. and to adequately terminates the application. It also allows the implementation of OLE. as well as other large-scale application feathers such as print preview. When using the MFC framework to develop an application. some steps must be followed. Parameters must be assigned in each step to configure the application. The frame of an application is available after each step of the setting.

The details about MFC framework can be found in. [1][7][8]

The following figure is the interface of the VC++ environment to develop this application with MFC frame.

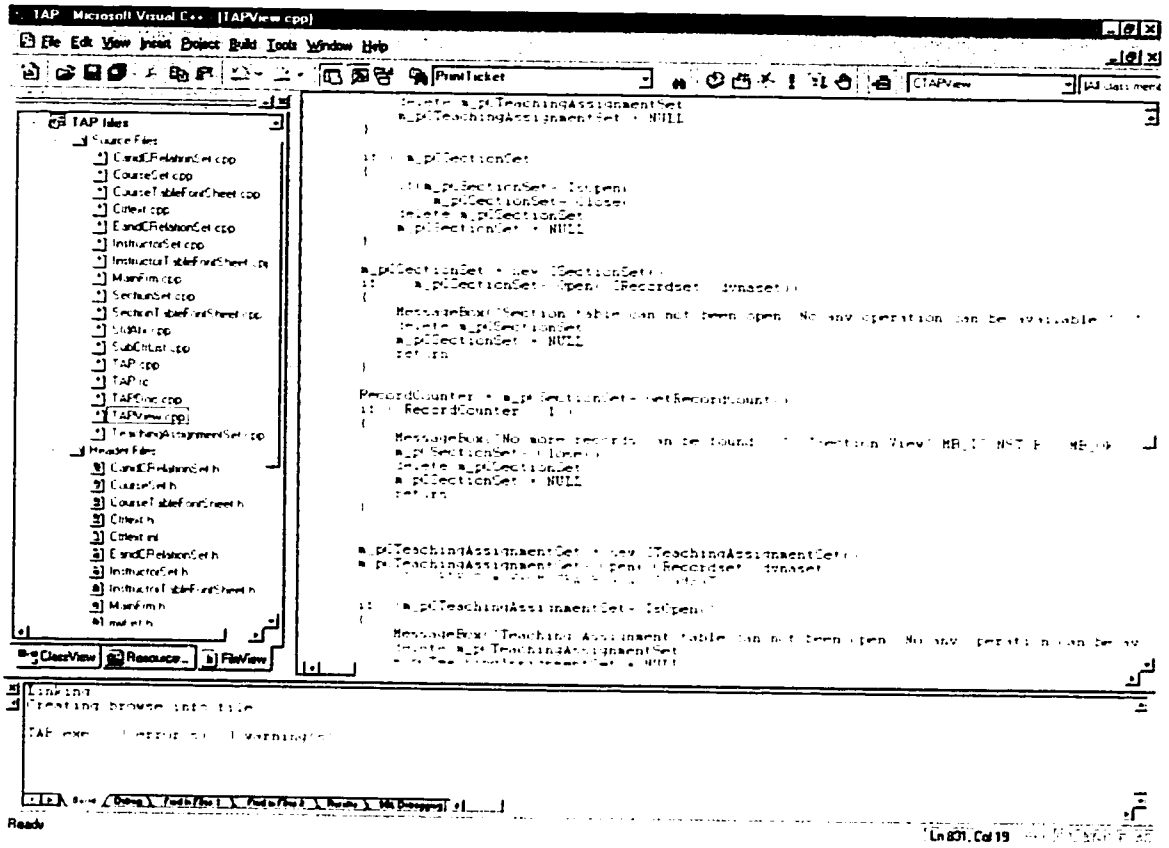

Figure 4: MFC framework interface

4.1.3 Message Map

Message Map is a way to associate Windows and user messages with the functions handling them. By implementing message maps, MFC avoids the problems associated with those brutishly large vtables.

Message maps are easier to use than the "Switch" statements. Message maps clearly indicate that a given message is being handled by a certain function within an MFC class or within the user defined class deriving from CCmdTarget class of MFC. Because this information directly comes from the message map, the particular function responsible for responding to a specific message as well as the intent in a central location is clearly

given. Any MFC class derived from CCmdTarget (including CCmdTarget class) accepts a message map. [8]

In the Teaching Assignment Planner, some classes can accept the message from message map. These are CTAPDoc, CTAPView, CSubCtrList, CSectionTableFontSheet, CInstructorTableFontSheet, CcourseTableFontSheet and all CPerportyPage derived classes included in the CSectionTableFontSheet, CInstructorTableSheet, CCourseTableFontSheet. As an example, the message map built in the CTAPView class included in the TAPView.C file is showing below:

```
BEGIN_MESSAGE_MAP(CTAPView, CListView)
        //{{AFX_MSG_MAP(CTAPView)
       ON_COMMAND(ID_TEST_LIST, OnTestList)
       ON_NOTIFY_REFLECT(NM_DBLCLK, OnDblclk)
       ON_NOTIFY_REFLECT(NM_RCLICK, OnRclick)
       ON_COMMAND(ID_VIEW_INSTRUCTOR, OnViewInstructor)
       ON_COMMAND(ID_VIEW_SECTION, OnViewSection)
       ON_COMMAND(ID_VIEW_UNASSIGNEDSECTION, OnViewUnassignedsection)
       ON_COMMAND(ID_VIEW_CLEAR, OnViewClear)
       ON_COMMAND(ID_VIEW_SUGGEST_INSTRUCTOR, OnViewSuggestInstructor)
       ON_COMMAND(ID_VIEW_ADD_ASSIGNMENT, OnViewAddAssignment)
       ON_COMMAND(ID_VIEW_ADD_COORDINATION_ASSIGNMENT,
OnViewAddCoordinationAssignment)
        //}}AFX_MSG_MAP
        Standard printing commands
       ON_COMMAND(ID_FILE_PRINT, CListView::OnFilePrint)
       ON_COMMAND(ID_FILE_PRINT_DIRECT, CListView::OnFilePrint)
       ON_COMMAND(ID_FILE_PRINT_PREVIEW, CListView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

To start a message map in the application and in a class, some special macros should be used, which normally are "BEGIN_MESSAGE_MAP()" and "END_MESSAGE_MAP()". Other macros are also used to indicate exactly what message will be mapped.

CCcmdTarge takes care of implementing the message map. which searches and links codes. while CWinThread has the code. which sends the message and gets it routed to the appropriate area in the corresponding CCmdTarget-derived object. There are three main categories of messages:

1. Windows messages

   These primarily include messages beginning with the **WM_** prefix. except the **WM_COMMAND**. Windows messages are handled by windows and views and often have parameters used to determine how to handle the message.

2. Control notifications

   Includes **WM_COMMAND** notification messages between basic (parent) and added (child) windows. For example. an edit control sends its parent a **WM_COMMAND** message containing the **EN_CHANGE** control-notification code if the user took action. which may have altered text in the edit control. The window message handler responds to the notification message by retrieving the text in the control.

   The framework routes control-notification messages such as other **WM_** messages. The only exception is the **BN_CLICKED** control-notification message sent by buttons the user clicked. This message is treated specifically as a command message and routed like other commands.

3. Command messages

This includes **WM_COMMAND** notification messages from user-interface objects such as: menus, toolbar buttons, and accelerator keys. The framework processes commands differently from other messages, and they can be handled by many kinds of objects. [9]

4.2 Document/View Implementation

The Teaching Assignment Planner application is designed in such a way a request of the application's document class is created for each document object with two views displaying data.

4.2.1 The clone of application

The clone functionality allows the user to create as many instances of the application as needed. The clone function can be activated by clicking the clone menu item included in the View menu. Below are the results after executing the clone functions three times. Each application executes different functions.

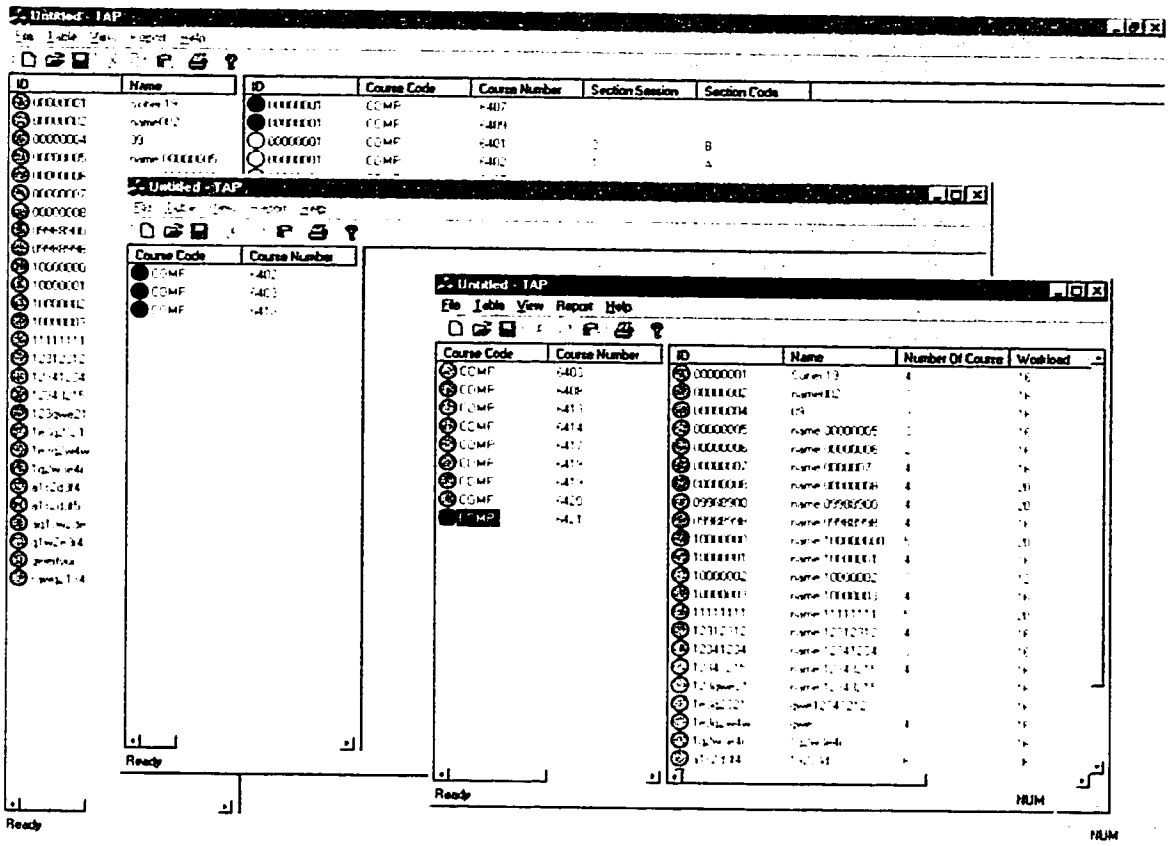

Figure 5: The applications layout after executing the clone function three times.

In the Document/View architecture, more than one request for a particular view may be associated with a given document and more than one view can be associated with the same document. This is the inherent power of the document/view architecture. As the application is running, a clone of it can be created or destroyed. This feature allows the user to view and compare data from the same data source. To implement the clone function, one member function should be added into the mainframe class. The following is the piece of code implementing the clone function in the Teaching Assignment Planner.

```
void CMainFrame::OnViewClone()
{
        STARTUPINFO si;
```

```
    PROCESS_INFORMATION pi;

    // Initialize the STARTUPINFO structure.
    memset(&si, 0, sizeof(si));
    si.cb = sizeof(si);

    CreateProcess(
            NULL,    // pointer to name of executable module
            (LPTSTR) AfxGetApp()->m_pszAppName,  // pointer to command line string
            NULL,    // pointer to process security attributes
            NULL,    // pointer to thread security attributes
            FALSE,   // handle inheritance flag
            0,       // creation flags
            NULL,    // pointer to new environment block
            NULL,    // pointer to current directory name
            &si,     // pointer to STARTUPINFO
            &pi      // pointer to PROCESS_INFORMATION
        );
}
```

4.2.2 Splitter Windows

In a splitter window, the window can be split into two or more scrollable panes. A splitter control (or "split box") in the window frame next to the scroll bars allows the user to adjust the relative sizes of the panes. Each pane is a view of the same document.

In the Teaching Assignment Planner project, there are two views in the main frame, combined with PropertySheet objects, which give the user a friendly interface. The code added to create the splitter windows in a member function of the mainframe class is listed below:

```
BOOL CMainFrame::OnCreateClient( LPCREATESTRUCT lpcs,
        CCreateContext* pContext)
{
        // create a splitter with 1 row, 2 columns
        if (!m_wndSplitter.CreateStatic(this, 1, 2))
        {
                TRACE0("Failed to CreateStaticSplitter\n");
                return FALSE;
        }
```

```
    // add the first splitter pane - the default view in column 0
    if (!m_wndSplitter.CreateView(0, 0,
            pContext->m_pNewViewClass, CSize(200, 50), pContext))
    {
            TRACE0("Failed to create first pane\n");
            return FALSE;
    }

    // add the second splitter pane - an input view in column 1
    if (!m_wndSplitter.CreateView(0, 1,
            RUNTIME_CLASS(CSubCtrList), CSize(0, 0), pContext))
    {
            TRACE0("Failed to create second pane\n");
            return FALSE;
    }

    // activate the input view
    SetActiveView((CView*)m_wndSplitter.GetPane(0,0));

    return TRUE;
}
```

The figure below shows the layout of the interface with two views sharing the same frame.

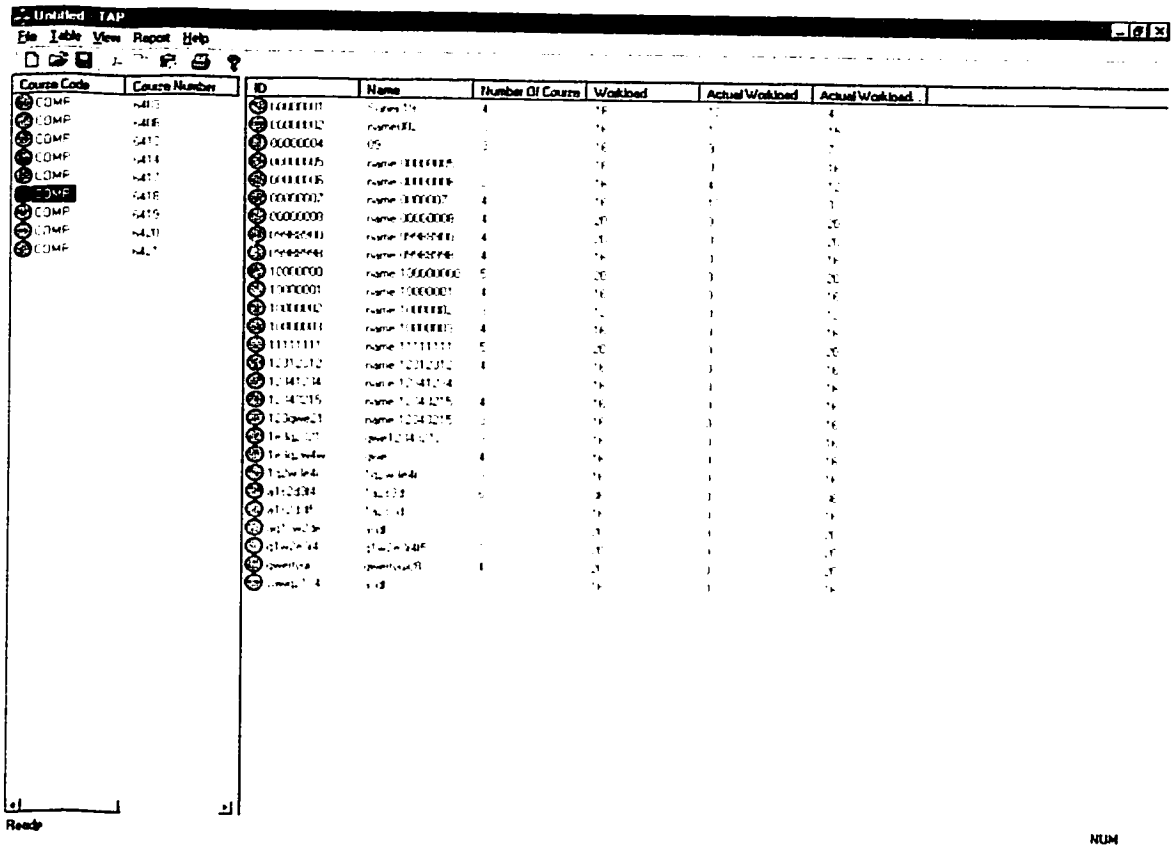

Figure 6: The splitter Windows

4.2.3 The association of Document/View classes

To support the Document/View architecture, the application should be built in such a way that classes separate and cooperate together to manage all functions in the application. The figure below shows which classes support a simple SDI with two views used in the application of the Teaching Assignment Planner implemented around MFC.

Each object of a class in the higher level creates the object of a class in the lower level (sold line). The pointer is used to access the data members or member functions in each object (dashed line).

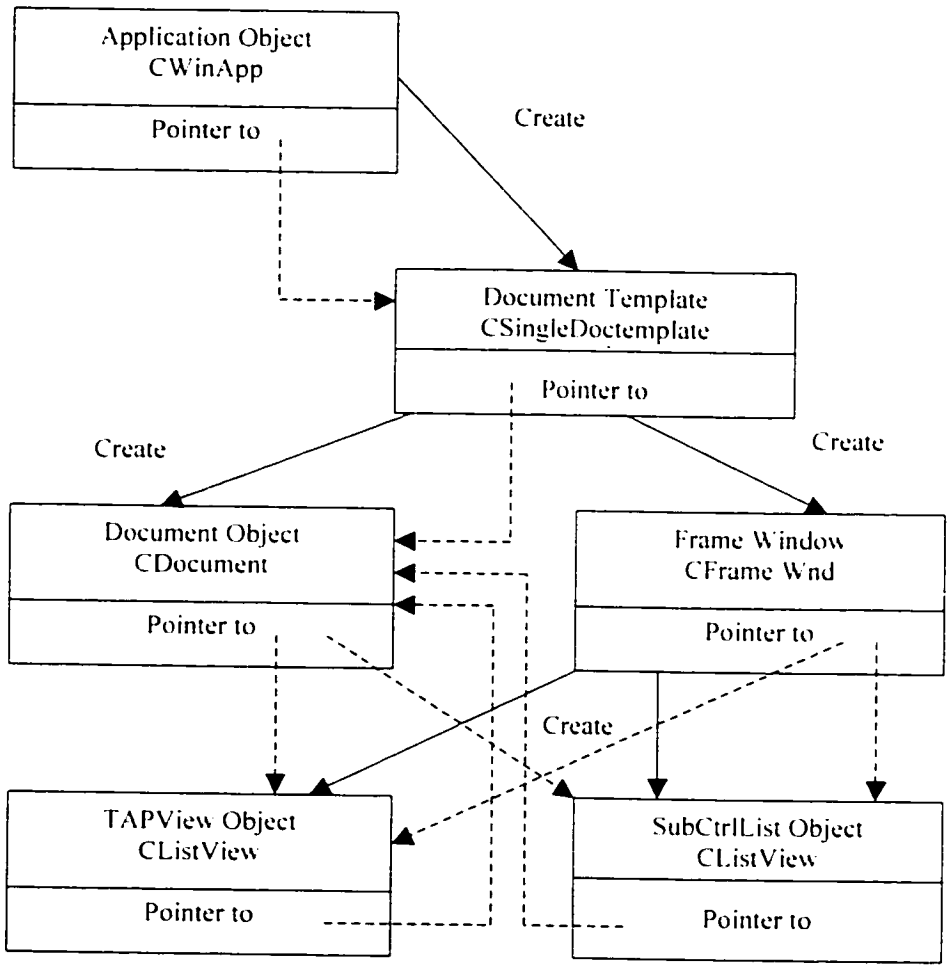

Figure 7: The associated objects in the Document/View architecture

4.3 Main classes introduction

A set of all classes is introduced in this section to clarify the architecture of the Teaching Assignment Planner application. Some main functions are introduced in each section to explain the main feature of that class.

4.3.1 The CTAPApp class and CMainFrame

The CTAPApp class derives from the MFC CWinApp class, which is the base class from which an application takes a Windows application object. An application object provides member functions to initialize and to run the application (and each instance of it).

The application using the Microsoft Foundation classes can only contain one object derived from CWinApp. This object is constructed when other C++ global objects are constructed and is already available when Windows uses the WinMain function, supplied by the Microsoft Foundation Class Library. CTAPApp derives from the CWinApp object at the global level.

When the application derives an application class from CWinApp, it overrides the InitInstance member function to create the application's main window object.

To derive a class and implement an InitInstande function by an application the following steps must be taken:

```
// CTAPApp initialization

BOOL CTAPApp::InitInstance()
{
        AfxEnableControlContainer();

        // Standard initialization
        // If you are not using these features and wish to reduce the size
        // of your final executable, you should remove from the following
        // the specific initialization routines you do not need.

#ifdef _AFXDLL
        Enable3dControls();                     // Call this when using MFC in a shared DLL
#else
        Enable3dControlsStatic();   // Call this when linking to MFC statically
#endif

        // Change the registry key under which our settings are stored.
        // TODO: You should modify this string to be something appropriate
        // such as the name of your company or organization.
        SetRegistryKey(_T("Local AppWizard-Generated Applications"));

        LoadStdProfileSettings();  // Load standard INI file options (including MRU)

        // Register the application's document templates.  Document templates
        // serve as the connection between documents, frame windows and views.

        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CTAPDoc),
                RUNTIME_CLASS(CMainFrame),       // main SDI frame window
                RUNTIME_CLASS(CTAPView));
        AddDocTemplate(pDocTemplate);

        // Parse command line for standard shell commands, DDE, file open
        CCommandLineInfo cmdInfo;
        ParseCommandLine(cmdInfo);

        // Dispatch commands specified on the command line
        if (!ProcessShellCommand(cmdInfo))
                return FALSE;

        // The one and only window has been initialized, so show and update it.
        m_pMainWnd->ShowWindow(SW_SHOW);
        m_pMainWnd->UpdateWindow();

        return TRUE;
}
```

This class cooperates with the CMainFrame class to control all the views created in the application. With the CMainFrame class, the clone application can be created and splitter views can be built in the CMainFrame class with the following steps:

```
BOOL CMainFrame::OnCreateClient( LPCREATESTRUCT lpcs,
        CCreateContext* pContext)
{
          create a splitter with 1 row, 2 columns
        if (!m_wndSplitter.CreateStatic(this, 1, 2))
        {
                TRACE0("Failed to CreateStaticSplitter n");
                return FALSE;
        }

          add the first splitter pane - the default view in column 0
        if (!m_wndSplitter.CreateView(0, 0,
                pContext->m_pNewViewClass, CSize(200, 50), pContext))
        {
                TRACE0("Failed to create first pane n");
                return FALSE;
        }

          add the second splitter pane - an input view in column 1
        if (!m_wndSplitter.CreateView(0, 1,
                RUNTIME_CLASS(CSubCtrList), CSize(0, 0), pContext))
        {
                TRACE0("Failed to create second pane n");
                return FALSE;
        }

          activate the input view
        SetActiveView((CView*)m_wndSplitter.GetPane(0,0));

        return TRUE;
}
```

4.3.2 The CTAPView and CSubCtrList

These are two views built to display data on the screen and to perform or cancel assignment operations. The pointers point to the CTAPDoc class to get data and call some functions implemented in CTAPDoc class.

The CTAPView class derives from CListView class. It implements and overrides some functions.

```
    //{{AFX_MSG(CTAPView)
    afx_msg void OnTestList();
    afx_msg void OnDblclk(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnRclick(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnViewInstructor();
    afx_msg void OnViewSection();
    afx_msg void OnViewUnassignedsection();
    afx_msg void OnViewClear();
    afx_msg void OnViewSuggestInstructor();
    afx_msg void OnViewAddAssignment();
    afx_msg void OnViewAddCoordinationAssignment();
    //}}AFX_MSG
```

Some message handlers are built in the class to respond to the user action on the screen. Initially at the class, some parameters are assigned to control the view properties.

```
BOOL CTAPView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs
    cs.style &= ~(LVS_LIST | LVS_ICON | LVS_SMALLICON);
    cs.style |= LVS_REPORT;
    cs.style |= LVS_SINGLESEL;
    return CListView::PreCreateWindow(cs);
}
```

To see this view as a List view with small icons, the view must be initialized as report type and single selection item.

The message map is below:

```
BEGIN_MESSAGE_MAP(CTAPView, CListView)
        //{{AFX_MSG_MAP(CTAPView)
        ON_COMMAND(ID_TEST_LIST, OnTestList)
        ON_NOTIFY_REFLECT(NM_DBLCLK, OnDblclk)
        ON_NOTIFY_REFLECT(NM_RCLICK, OnRclick)
        ON_COMMAND(ID_VIEW_INSTRUCTOR, OnViewInstructor)
        ON_COMMAND(ID_VIEW_SECTION, OnViewSection)
        ON_COMMAND(ID_VIEW_UNASSIGNEDSECTION, OnViewUnassignedsection)
        ON_COMMAND(ID_VIEW_CLEAR, OnViewClear)
        ON_COMMAND(ID_VIEW_SUGGEST_INSTRUCTOR, OnViewSuggestInstructor)
        ON_COMMAND(ID_VIEW_ADD_ASSIGNMENT, OnViewAddAssignment)
        ON_COMMAND(ID_VIEW_ADD_COORDINATION_ASSIGNMENT,
OnViewAddCoordinationAssignment)
        //}}AFX_MSG_MAP
        Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CListView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CListView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CListView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

All messages created in the view are sent to the message map and handled by this class. Based on the same principle, the CSubCtrlList class has a similar structure.

The main member functions of the CTAPView class are listed below:

- OnDblclk(): this function is a message's handler which responds to the action on the screen double clicking. The event will cause the NM_DBLCLK to activate, the corresponding reaction of the application will take place based on the following statement.

  ```
  ON_NOTIFY_REFLECT(NM_DBLCLK, OnDblclk)
  ```

  Then the OnDblclk() will be executed, to identify which function should be launched by this action from the user. To deal with the double clicking, many switch codes in the function have to be dealt with.

```
switch ( m_WhichFunctionForLDBCLK ){

        case 1:   Suggest instructor for unassigned section

                  ... ...

                  Result = GetDocument()->m_pCSubCtrList-> SuggestInstructorForSection(
                  buf[4], buf[0]);

                  if ( Result == 1 )

                        MessageBox("Database errores foundn. No any operation can be
                        available.", "Suggest and Add instructor".MB_ICONSTOP
                        MB_OK);

                  break;

        case 2:   Delete one assignement section

                  ... ...

                  Result = GetDocument()->DeleteOneAssignedSection( buf[0], str, m_nItem);

                  if ( Result == 1 )

                        MessageBox("Database errores foundn. No any operation can be
                        available.", "Delete Assigned Section".MB_ICONSTOP MB_OK);

                  break;

        case 3:   Add a coordinator to a course

                  ... ...

                  Result = GetDocument()->m_pCSubCtrList->
                  AddCoordinatorOnCourseSeletion( buf[0]);

                  if ( Result == 1 )

                        MessageBox("Database errores foundn. No any operation can be
                        available.", "Delete Assigned Section".MB_ICONSTOP MB_OK);

                  break;

        case 4:   Delete one coordinator from a course

                  ... ...

                  Result = GetDocument()->DeleteOneCoordination( buf[0], buf[1], str ,
                  m_nItem);

                  if ( Result == 1 )

                        MessageBox("Database errores foundn. No any operation can be
                        available.", "Delete Coordination".MB_ICONSTOP MB_OK);

                  Break;
```

- OnViewInstructor(): this function is the message's handler to display a view of instructors. When the user clicks on the Instructor menu item under the View menu, the message ID_VIEW_INSTRUCTOR will be launched and go into the message map. After being launched, the message arrives at the CTAPview message map. The corresponding statement is:

  ON_COMMAND(ID_VIEW_INSTRUCTOR, OnViewInstructor).

4.3.3 The CTAPDoc class

The CTAPDoc class is derived from the CDocument class, which provides the basic functionality for user-defined document classes.

Users access CTAPDoc via the two views associated with the object explained before. A view renders an image of the document in a frame window and interprets user input as operations to the document. CTAPDoc document class has two views associated with it.

CTAPDoc is a part of the framework's standard command routing and consequently receives commands from standard user-interface components. It receives commands forwarded by the active view of each CTAPView or CSubCtrlList. If it doesn't handle a given command, it forwards it to the document template managing it.

The functions overridden by the CTAPDoc are listed below:

```
    //{{AFX_VIRTUAL(CTAPDoc)
public:
virtual BOOL OnNewDocument();
virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL
```

The message map of CTAPDoc is listed below:

```
BEGIN_MESSAGE_MAP(CTAPDoc, CDocument)
        //{{AFX_MSG_MAP(CTAPDoc)
        ON_COMMAND(ID_REPORT_TEST, OnReportTest)
        ON_COMMAND(ID_EDIT_COURSE_TABLE, OnEditCourseTable)
        ON_COMMAND(ID_EDIT_SECTION_TABLE, OnEditSectionTable)
        ON_COMMAND(ID_EDIT_INSTRUCTOR_TABLE, OnEditInstructorTable)
        ON_COMMAND(ID_VIEW_CLEAR_ALL_VIEWS, OnViewClearAllViews)
        ON_COMMAND(ID_DELETE_TEACHING_ASSIGNMENT, OnDeleteTeachingAssignment)
        ON_COMMAND(ID_DELETE_COORDINATION_ASSIGNMENT,
OnDeleteCoordinationAssignment)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

The main functions of the class with associated view functions are explained below:

- SuggestInstructorForUnassignedSection:

  This function gives the user all unassigned sections and instructor's data from the TAP database, then the CTAPView displays all the unassigned sections for the user to select. The user selects an unassigned section by double clicking on the unassigned section's first item, the course code with the red colored icon (displayed on the left view). CSubCtrlist view then displays all the experienced instructors who can be assigned to teach this section. The colored icon which is the first item in the instructor data indicates the instructor's actual workload (displayed on the right view). The user selects an instructor to be assigned to this section by double clicking the instructor's first item. The application will

automatically check for any conflicts for the current assignment. If a conflict occurs, an information box will be launched and this assignment will be canceled.

- AddInstructorForCoordinator:

  This function gets all courses and instructor's data from the TAP database. Then, the course's data is displayed on the CTAPView for the user to make a selection. The user selects a course by double clicking the course's first item with the green icon. CSubCtrlList view will then display all the instructors who can be assigned to coordinate this course. The colored icon, which is the first item of the instructor's data, indicates the actual workload assigned to the instructor. The user then selects an instructor to be assigned as coordinator of this course by double clicking the instructor's first item. The application then checks the actual workload to see whether it is exceeds the normal 14 points. An information box is launched if the workload is exceeded. The user can chose between two options. If OK is selected, the assignment is done; otherwise the application aborts the current assignment.

- UpdateInstructorWorkload:

  This function is used to update the instructor's workload value. This function isn't directly launched by the user through an action on the view, but by other functions, which take care of the maintenance of the database. For example, when the user assigns a course to a coordinator, the coordinator's workload should be updated to indicate the newly assigned workload.

- DeleteOneAssignedSection:

  This function deletes an instructor's teaching assignment. After this operation is completed. the updateInstructorWorkload function should be selected to update the actual workload of that instructor.

4.3.4 The CCourseSet class

The CCourseSet class derives from the CRecordSet class. A CRecordset object represents a set of records selected from a data source. CRecordset objects are typically used in two forms: dynasets and snapshots. CCourseSet stays synchronized with data updates made by other objects of RecordSet. Each class in the Teaching Assignment Planner deriving form the CRecordset represents a set of records fixed at the time when the corresponded table was opened.

The main functions of this class are:

Open:

The application selects this member function to run the query defined by the object. The object connection to the data source depends on how the application constructs the CCourseSet before selecting the Open function. If the application sends a CDatabase object to the CCourseSet constructor. which has not been

connected to the data source. this member function uses GetDefaultConnect to attempt opening the database object. hence making the commit and rollback functions available.

Committing an operation means saving current updated changes in the database. All the updated data will be saved in the database after the last commit operation. To rollback an operation means to abort all updated data and not making any changes in the database up to the last commit operation. The rollback operation gives the user the opportunity to cancel all changes affecting the database. so that the database is secure. Commit and Rollback operations are only available in the subsystems through dialog boxes and aren't provided on the views operations.

The user can select items on the views and can finish the assignment without selecting the Commit and Rollback functions since updated data is saved in the database immediately after items from the views are selected or deleted.

- MoveFirst:

  This function allows making the first record in the first position of the current record set. Regardless if bulk row searching was implemented. this will always make the first record in the CCourseSet available in first position.

- MoveNext:

This function member allows making the first record in the next position of the current record set. MoveNext simply moves to the next record, if existent.

- MovePrev:

  This function member allows making the first record in the previous position of the current record set. MovePrev simply moves to the previous record, if it exists.

- MoveLast:

  This function member allows putting the first record in the last position of the current record set. MoveLast simply moves it to the last record in the recordset.

- Close:

  This function member closes the CCourseSet. The ODBC HSTMT and all the memory of the framework allocated for the CCourseSet are cancelled. Usually after the Close function is selected, the application deletes the C++ CCourseSet object if it was allocated with a new C++ operator.

- Addnew:

  This function member allows adding a new record to the table. The application must connect to the Requery member function to see the added record. The record's fields are initially Null. (In database terminology, Null means "having no value" and is not the same as **NULL** in C++.) To complete the operation, the

application must then call the Update member function. Update saves the changes to the data source.

- Edit:

  This function member allows changing the current record. After the application selects Edit, the application can change the field data members by directly resetting their values. The operation is completed when the application subsequently selects the Update member function to save changes on the data source.

- Delete:

  This function member allows deleting the current record. After a successful deletion, the CCourseSet's field data members are set to a Null value, and the application must explicitly call one of the Move functions in order to move off the deleted record. Once the application moves off the deleted record, it is not possible to return to it. If the data source supports transactions, the application can make the Delete selection part of a transaction.

- Update:

  This function member is used after selecting the AddnNew or Edit function member and is required to complete these operations. As for the Delete function, Update is not required.

4.3.5 The CCourseTableFontSheet class

The CCourseTableFontSheet class derives from the CPropertySheet class included in MFC. The objects deriving from the CPropertySheet represent property sheets, otherwise known as tab dialog boxes. A property sheet consists of a CPropertySheet object and one or more CPropertyPage objects. A property sheet is displayed by the framework as a window with a set of index tabs, from which the user selects the current page as well as the new area for it.

Even though CPropertySheet is not derived from CDialog, managing a CPropertySheet object is similar to managing a CDialog object. For example, the creation of CCourseTableFontSheet requires a two-part construction: creates the object, and then calls DoModal for a modal property sheet. Below is a part of the code used in the CTAPDoc to create CCourseTableFontSheet.

```
void CTAPDoc::OnEditCourseTable()
{
        m_CourseFontSheet.SetTitle("Course Table");
        if( m_CourseFontSheet.DoModal() != IDOK )
        {
                m_CourseFontSheet.Cancel();
        }
}
```

For each CPropertyPage class included in the CPropertySheet creation and deletion, the class derived from the CPropertySheet should take care. The objects of the

CPropertyPage should be added into the sheet to activate them. Here are some parts of the code in the CCourseTableFontSheet to create the CcoursePages objects.

```
m_pCourse_page1 = new CCoursePage1(this);
m_pCourse_page2 = new CCoursePage2(this);
m_pCourse_page3 = new CCoursePage3(this);
m_pCourse_page4 = new CCoursePage4(this);
AddPage(m_pCourse_page1);
AddPage(m_pCourse_page2);
AddPage(m_pCourse_page3);
AddPage(m_pCourse_page4);
```

After CCoursePage objects are created and added into the CCourseTableFontSheet object, all functions can be selected from other objects or interfaces, just like normal dialog box object.

The following part of a code from CCoursePage1 shows how the message map works:

```
BEGIN_MESSAGE_MAP(CCoursePage1, CPropertyPage)
        //{{AFX_MSG_MAP(CCoursePage1)
                NOTE: the ClassWizard will add message map macros here
    ON_BN_CLICKED(IDC_COURSE_FIND_COPY, OnCourseFindCopy)
    ON_BN_CLICKED(IDC_COURSE_FIND_PASTE, OnCourseFindPaste)
    ON_BN_CLICKED(IDC_COURSE_FIND_FIRST, OnCourseFindFirst)
    ON_BN_CLICKED(IDC_COURSE_FIND_GO, OnCourseFindGo)
    ON_BN_CLICKED(IDC_COURSE_FIND_PRE, OnCourseFindPre)
    ON_BN_CLICKED(IDC_COURSE_FIND_NEXT, OnCourseFindNext)
```

```
        ON_BN_CLICKED(IDC_COURSE_FIND_LAST, OnCourseFindLast)
        ON_BN_CLICKED(IDC_COURSE_FIND_CLEARDIALOG, OnCourseFindClearDailog)
        ON_BN_CLICKED(IDC_COURSE_FIND_CLEAR_LIST, OnCourseFindClearList)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Chapter 5 Conclusion

In this report, we discussed how the Teaching Assignment Planner was been implemented using an object-oriented programming environment (VC++ and MFC) . The architecture is the most important part. [2] Detailed explanations of design and implementation can only be done after the architecture design phase. By using ODBC and the framework MFC, a desktop database related application can be developed easily and quickly. A good design allows the creation of codes, and, since code are reusable for classes, the application easily reuses the existing classes to build new ones. With this framework, the application can easily be integrated into an existing architecture suitable for the application to solve problems.

The advantages in using framework are listed below:

- The architecture is clear and stable.

  There are available architectures to develop applications using MFC, whether SDI or MDI. The association between all document, frame and views classes is already done. All the member functions of the basic class are available to the resulting class. The application finds the solutions to the specific problems.

- The components of GUI in the interface of an application are standard.

  The framework provides a lot of components based on the Windows standards. By selecting these objects users identify them and allow the user to start using the application. The message maps provide a useful way to handle all interface events making the application handling easy for the user.

- Reusable codes and inherence makes the development period shorter.

  It takes a longer time to develop an application from scratch. Taking the advantage of existing classes or codes asks fewer efforts to develop an application. It also brings fewer bugs, so the total amount of efforts put into designing and developing a new application is smaller but the performance of the application is enhanced.

The disadvantages when using a framework in the object-oriented programming are:

- The extra time spent studying the framework when starting the design and development of an application.

  Steps as well as some parameters have to be followed to build an application using a framework. Getting familiar with these settings takes time and efforts, especially for beginners. The understanding of the FMC message map and inherence takes time to study, practice and to get familiar with. MFC is a huge topic of class inherence. As it includes many different classes to develop applications, and as so many classes are handled it's harder to study all the information in such a short time.

Appendix

Application interface and Menu structure

- Main interface of the application

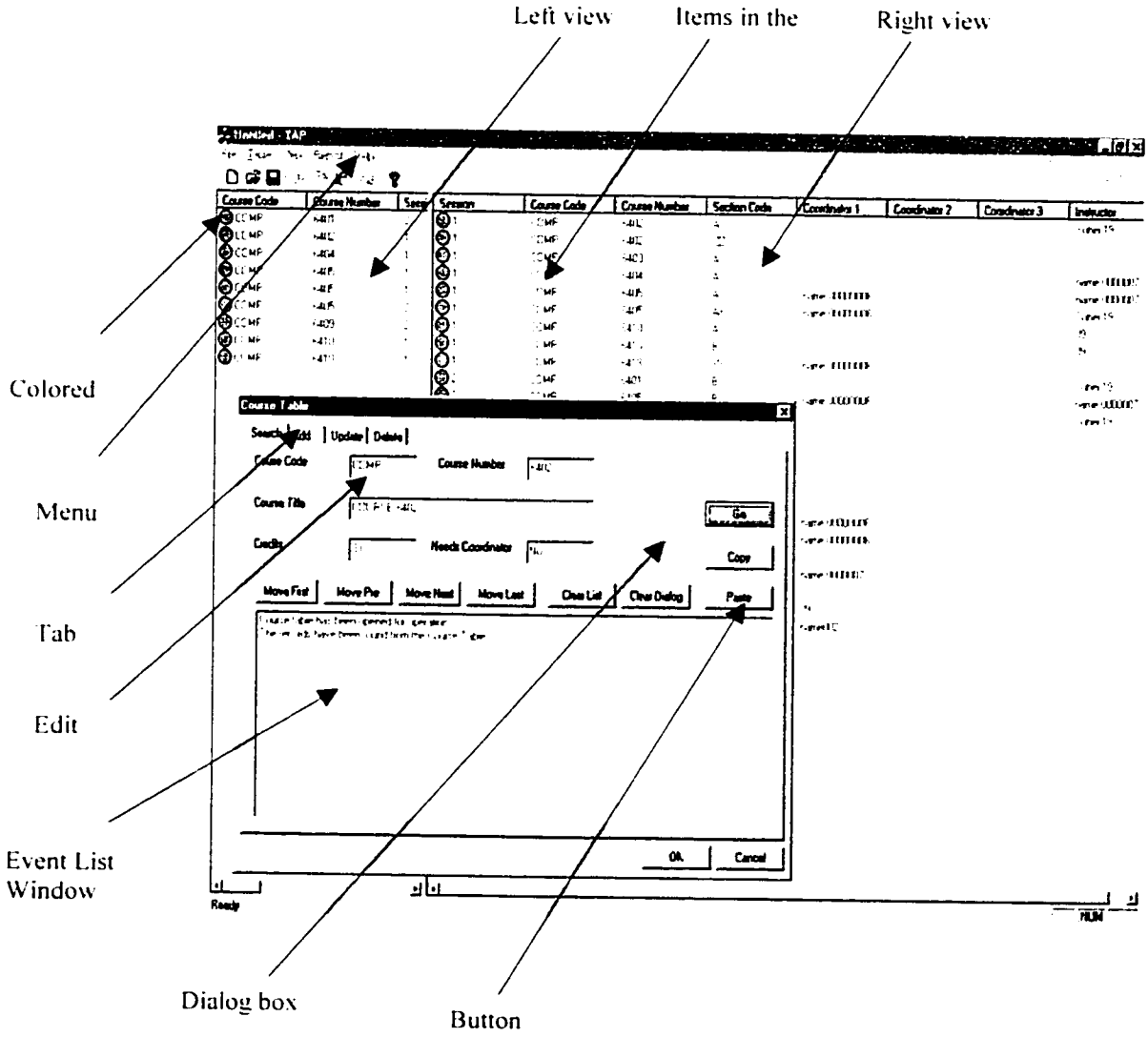

Figure 8: Components of the application layout

- Normal operation in Windows by interface component. Each function can be selected from the menu.

- The Course. Section and Instructor subsystems are activated from the menu item of Table.

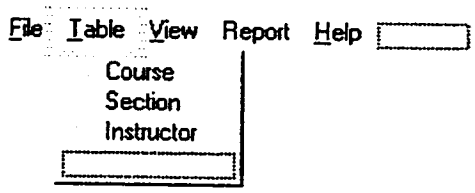

Figure 9: The menu Table with items

- All view functions are activated form the View menu item.

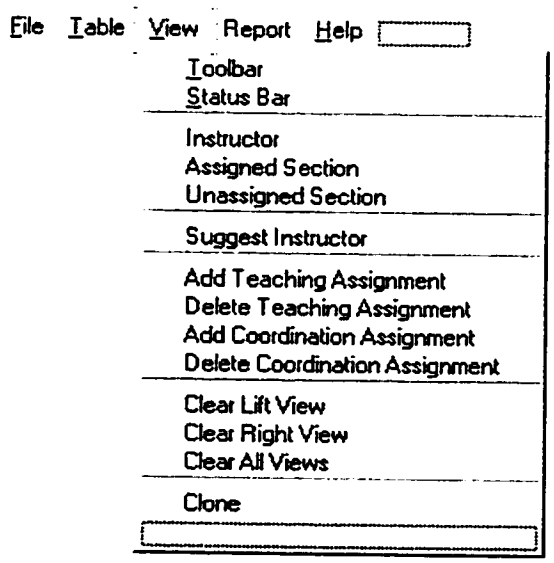

Figure 10: The View menu with items

- All report functions are activated form the menu item of report.

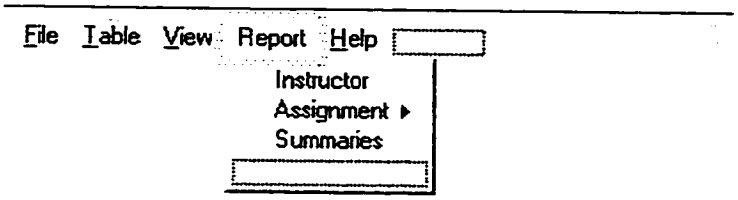

Figure 11: The menu Report with items

- Course subsystem dialog box layout.

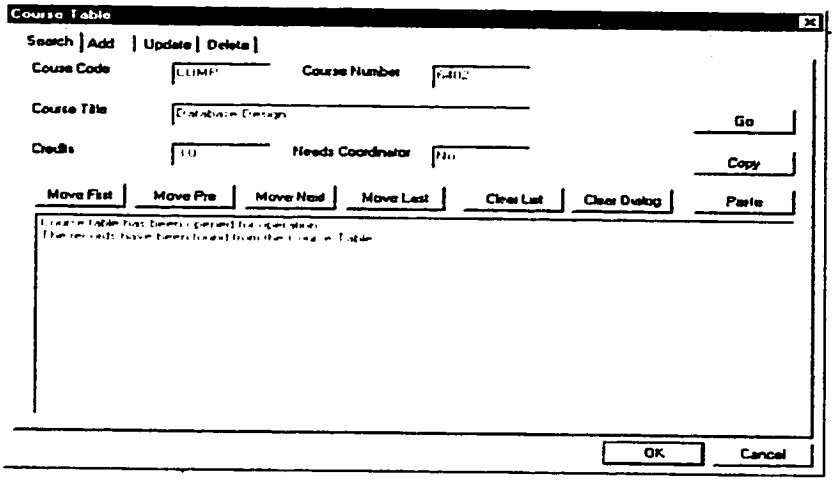

Figure 12 : Course subsystem dialog box layout

- Section subsystem dialog box layout.

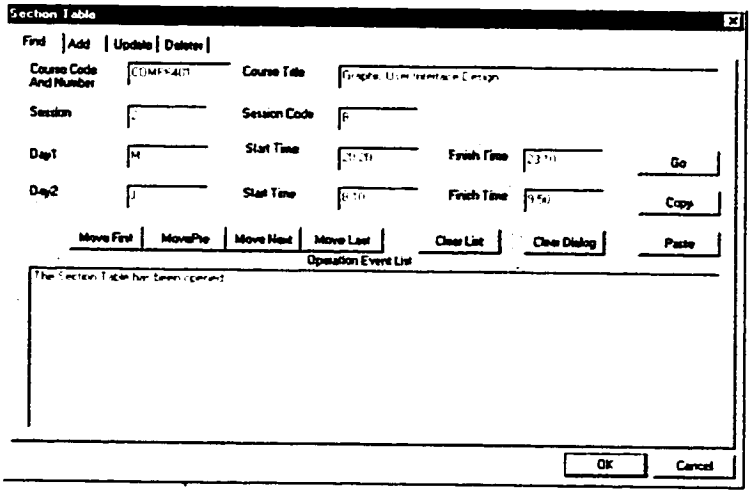

Figure 13: Section subsystem dialog box layout

- Instructor subsystem dialog box layout.

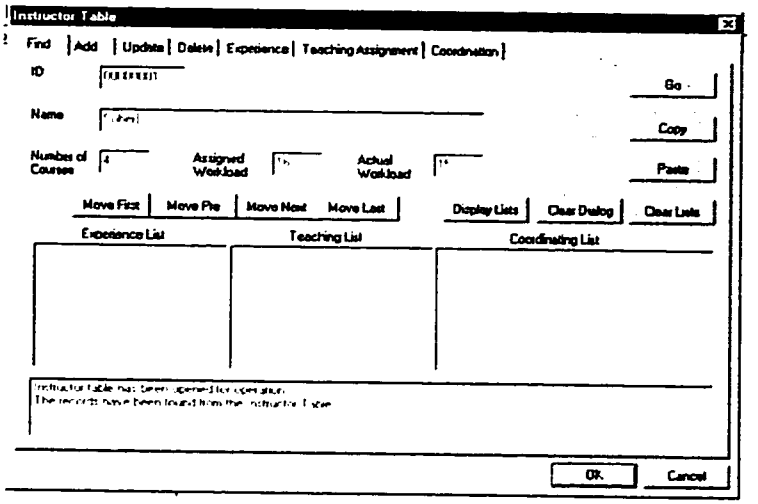

Figure 14: Instructor subsystem dialog box layout

References

1. *Object oriented development with C++*. By Kjell Nielsen 1997 International Thomson computer press
2. *A practical introduction to software design with C++*. By Steven P. Reiss
3. *Object-oriented design with C++*. By Ken Barclay John savage
4. *Object-Oriented Software Design and Construction with C++*. By Dennis Kafura
5. *Object-Oriented Design with Applications*. By Grady Booch
6. *An Introduction to Object-Oriented Design in C++*. By Jo Ellen Perry Harold Dd.Levin
7. *Design Patterns: Elements of Reusable Object-Oriented Software*. By Addison-Wesley
8. *Programming Microsoft Visual C++*. By David Kruglinsk: George Shepherd and Scot Wingo. Microsoft Press 1998
9. *Professional MFC with Visual C++ 6*. By Mike Blaszczak. Wrox Press. 1999
10. *MSDN Library Visual Studio 6.0*. By Microsoft
11. *C++ unleashed*. By Jesse Liberty with Vishwajit Akecha. Steve Haines. Steven Mitchell. Alexander Nicholov. Charles Pace. Meghraj Thakkar. Michael J. Tobler. Donals Xie. Steve Zagieboylo. Sams press 1999
12. *Teach Yourself Database Programming with Visual C++ 6* in 21 days. By ? Sams Press. 1999.
13. Database and expert system application /4th international conference. DEXA '93. Prague. Czech Republic. September 6-8. 1993

14. Database and expert system application /5th international conference. DEXA '94. Athens. Greece. September 7-9. 1994. proceedings/Dimitris Karagiannis. ed.

15. Database and expert system application /8th international conference. DEXA '97. Toulouse. France. September 1-5. 1997 Abdelkader Hameurlain. A Min Tjoa

16. Database modeling & Design. By Toby J. Teorey 1999

17. *Database Processing: Fundamentals. design. and implementation.* By Kroenke. David 1995