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

UMI®

TEACHING ASSISTANT ASSIGNMENT PLANNER

Jiantao He

A Major Report

in

The Department

of

Computer Science

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

April 2002

ABSTRACT

Teaching Assistant Assignment Planner

Jiantao He

Doing TA assignment manually is a tedious and error-prone process, which means it is not only time-consuming but also susceptible to human errors such as schedule conflicts, multiple assignments and overload assignments. Teaching Assistant Assignment Planner (TAAP) --- a software tool implemented using Java in Windows environment --- facilitates the process of doing TA assignment. It is a semi-automatic software system, which helps to promote staff productivity, eliminate human errors and generate status reports.

Both the TAAP design and implementation are presented in this major report. The issues to be addressed are environment setup, requirement analysis, detailed design, final implementation and improvement suggestion. Environment setup identifies the hardware and software requirements along with some useful background information. Requirement analysis defines the database scheme to be used, explores the system functionality and performs the task analysis. Detailed design presents the MVC design pattern, explains how MVC is being applied, and elaborates the design detail in the UML notations. Final implementation only covers critical coding issues such as database access from Java, data representation in GUI and database manipulation. The improvement suggestion sheds the light for the future improvement of TAAP, e.g. the migration to a web-based application.

ACKNOWLEDGEMENTS

I would like to express my grateful thanks to my supervisor Prof. Peter Grogono for the time and effort he has put in, the great patience and support he has shown and the tremendous help and encouragement he has offered since I started the major report.

Many thanks to Ms. Pauline Dubois for her help, advice and feedback on the major report, and to Mr. Stan Swiercz and Ms. Galina Kolesova for setting up the lab environments.

Many thanks to my teammate Mr. Jun Liu for the collaboration and the time we have shared together.

Special thanks to Ms. Halina Monkiewicz for her smile and help that I have received during the years.

Thanks to the Graduate Program Director and all the instructors who have taught and helped me during my studies at Concordia.

TABLE OF CONTENTS

LIST OF FIGURES vii

LIST OF TABLES ix

CHAPTER 1. INTRODUCTION 1

1.1 Background 1

1.2 What is TAAP 2

1.3 Hardware Environment 3

1.4 Software Environment 3

CHAPTER 2. REQUIREMENT ANALYSIS 5

2.1 Database Scheme 5

2.2 Functionality Analysis 9

CHAPTER 3. DETAILED DESIGN 14

3.1 General MVC Model 14

3.2 MVC Model in TAAP 16

3.3 Static Design Description --- UML Class Diagrams 19

3.4 Dynamic Design Description --- UML Sequence Diagrams 30

CHAPTER 4. IMPLEMENTATION 39

4.1 What is SQL 39

4.2 Access to Database from Java 40

4.3 Data Representation in GUI 43

4.4 Data Manipulation in MS Access Database 45

CONCLUSION 53

REFERENCES 56

APPENDIX A. TAAP INITIAL REQUIREMENTS 57

APPENDIX B. "DO TA ASSIGNMENT" SCREENSHOT 63

LIST OF FIGURES

Figure 2.1 Database Abstraction in UML 8

Figure 2.2 Main Use Case Diagram 9

Figure 2.3 "Add Assignments" Diagram 11

Figure 2.2 "Delete Assignments" Diagram 12

Figure 3.1 MVC Triad 15

Figure 3.2 TAAP MVC Model with Back-end Database 17

Figure 3.3 Package Diagram – "Do TA Assignment" Subsystem 19

Figure 3.4 Class Diagram – TaapTaAssignment GUI 23

Figure 3.5 Class Diagram – MVC Core 24

Figure 3.6 Class Diagram – TaAssignment Model 25

Figure 3.7 Class Diagram – UnassignedTaskList View 26

Figure 3.8 Class Diagram – AssignedTaskList View 27

Figure 3.9 Class Diagram – TaList View 28

Figure 3.10 Class Diagram – Assignment View 29

Figure 3.11 Sequence Diagram – Subsystem Initialization 31

Figure 3.12 Sequence Diagram – Unassigned Task List Update (pulling) 32

Figure 3.13 Sequence Diagram – Unassigned Task List Update (pushed) 33

Figure 3.14 Sequence Diagram – Assigned Task List Update (pulling) 33

Figure 3.15 Sequence Diagram – Assigned Task List Update (pushed) 34

Figure 3.16 Sequence Diagram – TA List Update 35

Figure 3.17 Sequence Diagram – Application Information Update.....35

Figure 3.18 Sequence Diagram – Add Assignment.....36

Figure 3.19 Sequence Diagram – Delete Assignment.....37

Figure 4.1 dbInterface Implementation.....43

Figure 4.2 DB Table Representation.....44

LIST OF TABLES

Table 2.1 Database Table - Tasks 6

Table 2.2 Database Table - TAs 7

Table 2.1 Database Table - Applications 7

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

This major report is originated from an actual requirement in the Computer Science Department at Concordia University.

Every year staff in the Computer Science Department need to spend a lot of time in processing TA applications before each new term. There are normally more than a hundred of TA tasks posted every term along with dozens of applicants on the list. Therefore, the total number of applications can easily top hundreds. The information about tasks, applicants and corresponding applications are stored in the database located in a UNIX machine. The basic process of doing TA assignment is to first pick up an unassigned task from the task table, then check the application table to get the information about the applicant(s) who have applied this task, and finally modify the application table to reflect the new assignment. In order to obtain the detailed information (e.g. experience) for an individual applicant, the applicant table has to be consulted constantly during this process.

So far the whole assignment process is still carried out manually by the TA assigners (the role usually taken by the lab coordinators), and no tools have been developed for them to facilitate such a routine task. Obviously doing TA assignment is a tedious, time-consuming and error-prone process. It is susceptible to human errors such

as schedule conflicts (i.e. one applicant gets two tasks with conflicting schedules), multiple assignments (i.e. the same task is assigned to more than one applicant) and overloaded assignments (i.e. some applicants are too overloaded) etc. Without a tool, it is also hard for the assigners to get the status information about how many and what tasks have been assigned and how many and what tasks are yet to be assigned.

1.2 WHAT IS TAAP

"Teaching Assistant Assignment Planner" (TAAP) demonstrated in this major report is a software tool intended to automate the process of doing TA assignment. The automated process helps to promote staff productivity, eliminate human errors and generate status reports. Another goal is to keep the system itself as simple as possible, and make sure the productivity is obtained by focusing on features like user-friendly interface, easy-to-use and easy-to-maintain.

TAAP is a semi-automatic tool to help simplify the process of TA assignment. However, the actual assignment still needs to be done manually. TAAP guarantees better productivity by providing the functionality to assist TA assigners doing their job not only effectively and efficiently, but also correctly. The automatic assignment mechanism (which usually requires a sophisticated algorithm) is not considered here, thus not implemented in TAAP. It might be part of the TAAP implementation for the next phase.

The main functionality implemented in TAAP are categorized as follows:

- Task assignment and deletion
- Schedule conflict check
- Dynamic status update

- TA, task and assignment report

TAAP is a localized database application system with both the client-side application and the back-end database running on the same machine. Therefore it can only be accessed from where TAAP is installed; any attempt for remote access is not possible. TAAP is developed and operated solely on Windows platforms (9x/2000/NT). For the sake of clarity, TAAP will be referred as the client-side application hereafter, while the back-end database will be referred using the database name.

1.3 HARDWARE ENVIRONMENT

- Any Windows-compatible PC's with at least PII 266MHz Processor
- 128MB RAM
- 50MB free storage

For optimal performance, PIII 600MHz+ processor and 256 MB RAM are recommended. 1GB of storage will be enough for any further system expansion.

1.4 SOFTWARE ENVIRONMENT

TAAP is developed using Java [1] in JBuilder 5.0 (Personal Edition with JDK1.3) development environment [2]. The purpose of writing TAAP in Java is to make sure it is highly portable to the platforms other than Windows in the future. Besides the cross-platform characteristics, Java is also a web-ready programming language, which allows us to easily migrate TAAP from a local application system to a web-based system using JSP and Java Applet.

JBuild5.0 has some major advantages over the other Java IDE environments. For example, JBuilder5.0 offers a free-download life-time version, the Personal Edition, which has the similar functionality provided by its expensive licensed professional version. Because of its being widely used in the universities and industries, it is easy for us to find support resources when we have questions and problems during the development. Considering the portability of TAAP, no JBuilder-specific Java features but the standard AWT and Swing have been used.

Microsoft Access [3] is a lightweight and fully functional database, which has been adopted as the back-end database for TAAP because of its availability and easy-to-use. The connection between TAAP and Microsoft Access is made via JDBC using the Windows build-in ODBC driver. (Note: The Windows build-in ODBC driver also supports JDBC.)

CHAPTER 2

REQUIREMENT ANALYSIS

The requirement analysis is based on the TAAP initial requirements (see Appendix A) assembled by Prof. Peter Grogono. Two parts will be presented in this chapter. The database scheme will first be covered in a very detailed manner along with some design considerations. Then in the second part, a series of Use Cases will be exploited and fully explained in order to help us, the designers, complete the detailed design of TAAP.

2.1 DATABASE SCHEME

Although the department database located on a UNIX machine has enough information for us to implement TAAP, the information and the database tables are not well defined and organized, thus susceptible to future changes and adjustments. Trying to stabilize our development and also being advised by Prof. Peter Grogono, three new MS Access tables are customized and used in place of the existing ones in TAAP.

The proper functioning of TAAP depends on the database mentioned here. But the database itself is not part of TAAP, and its configuration and setup shall always be assumed ready and done in the target system before TAAP is loaded.

The three new tables are *Tasks*, *TAs* and *Applications*, which contain the information about posted tasks, TA applicants, and submitted applications respectively. Detailed structures will be presented for each of them as follows.

2.1.1 Table Structures

Ten fields are defined in table *Tasks* (Table 2.1).

Field Name	Type (Size)	Examples	Comments
TaskID	Long Integer	123, 406, ...	Indexed and unique value is required across the table.
Course	Text (4)	COMP, ENCS, ...	
Number	Text (4)	641, 775, 690L, ...	
Session	Integer	1, 2, 3, 4	a.k.a Term ID
Title	Text (25)	Operating System, ...	
Section	Text (2)	AA, CB, AB, ...	
TaskType	Text (3)	TUT, MRK, LAB	Tutor, Marker and Lab tutor are the only valid task types.
Day	Text (1)	M, T, W, J, F	M: Monday T: Tuesday W: Wednesday J: Thursday F: Friday If one task is held more than one day, it must be separated into different tasks.
Start	Date	17:30PM, 10:00AM, ...	Only valid for TUT and LAB.
End	Date	19:30PM, 12:00PM, ...	Only valid for TUT and LAB. Has to be later than 'Start'

Table 2.1 Database Table - *Tasks*

Three fields are defined in table *TAs* (Table 2.2).

Field Name	Type (Size)	Examples	Comments
ID	Text (6)	4338967, 5912375 ...	Student ID, indexed and unique value is required across the table.
Name	Text (20)		
Experience	Text (64)	COMP641-ENCS455-...	Different pairs of

			'Course' plus 'Number' connected via '-'. Having experience in these course in either of TUT, LAB or MRK.

Table 2.2 Database Table - *TAs*

Three fields are defined in table *Applications* (Table 2.3).

Field Name	Type (Size)	Examples	Comments
TaskID	Long Integer	123, 406, ...	Jointly indexed with 'ID' and unique along with 'ID' across the table.
ID	Text (6)	4338967, 5912375 ...	Student ID, jointly indexed with 'TaskID' and unique along with 'TaskID' across the table.
Assigned	Boolean	FALSE, TRUE	Indicate whether assigned or not.

Table 2.3 Database Table - *Applications*

The contents of the three forementioned tables are populated based on the information stored in the department database using SQL programming.

2.1.2 Table Relationship

In order to present a clear picture of how these tables are related to each other, the UML class diagram is employed in the demonstration (Figure 2.1). The entities in Figure 2.1 are not the database tables; they represent the abstraction of records stored in the tables. For example, TA represents the abstraction of TA records stored in the *TAs* table, while Task represents the abstraction of task records stored in the *Tasks* table. Since the information stored in the *Applications* table has two different states, two separate derived entities are used.

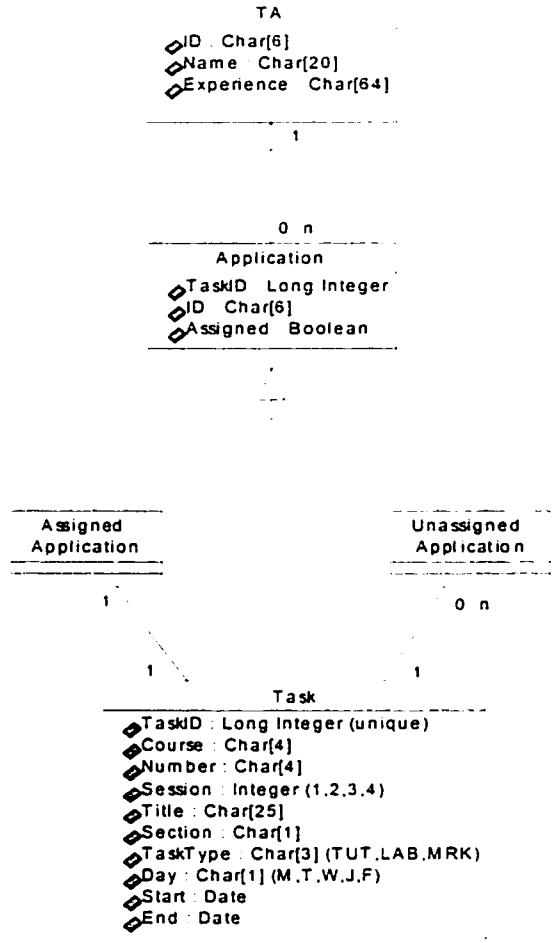

Figure 2.1 Database Abstraction in UML

The state of a task is determined by its corresponding applications. If there exists one and only one application in the assigned state, the task must be assigned; otherwise, the task remains unassigned. Any task can have many unassigned applications (one-to-many mapping), but only one assigned application (one-to-one mapping). The logic illustrated in the diagram assures the multiple assignment never happens in reality.

2.2 FUNCTIONALITY ANALYSIS

Use case diagrams are typically used to specify and characterize the functionality and behavior of a whole application system interacting with one or more external operators. Use case diagrams are the best functional analysis tools, which can provide a clear mapping between the functionality of the target system and the customer's requirements.

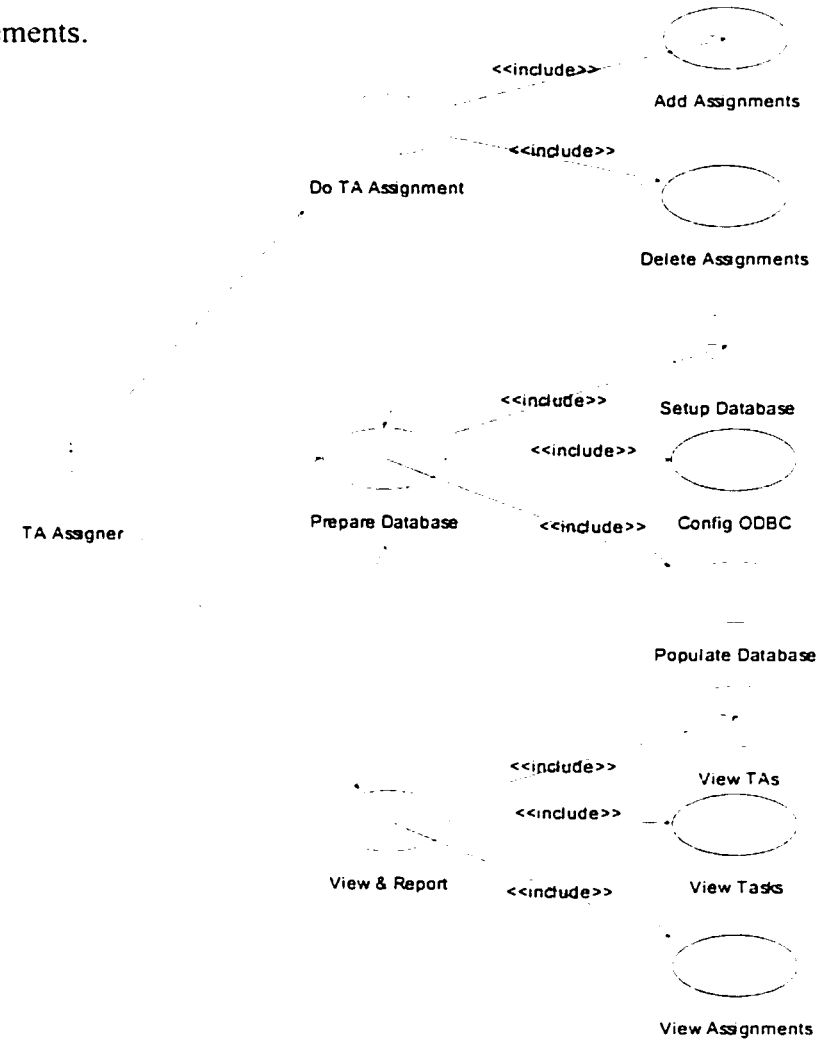

Figure 2.2 Main Use Case Diagram

The main use case in Figure 2.2 demonstrates three high-level functional activities defined in the TAAP system: "Prepare Database", "Do TA Assignment" and "View & Report".

These three functional activities correspond to three major tasks to be conducted by the TA assigners using the TAAP system. Unlike "Do TA Assignment" and "View and Report" tasks, "Prepare Database" doesn't really belong to the functionality of TAAP, but it is essential to the proper functioning of the TAAP system and also part of the TA assigners' job.

"Prepare Database"

"Prepare Database" task contains three sub-tasks:

- Setup Database

Create a new MS Access database and three new tables in it: *Tasks* (Table 2.1), *TAs* (Table 2.2) and *Applications* (Table 2.3).

- Configure ODBC

Configure MS Access ODBC driver and setup ODBC connection to the newly created database.

- Populate Database

Retrieve data from the department database and convert the received data into MS Access database automatically using SQL programming.

"Setup Database" task has to be conducted prior to other two tasks, and these three sub-tasks are one-time work for every new term.

"Do TA Assignment"

"Do TA Assignment" task is the core functionality provided by the TAAP system, and it contains two sub-tasks:

- Add Assignments (Figure 2.3)

When a TA assigner chooses an unassigned task from the unassigned task list, TAAP will prompt a list of TAs who have applied to the selected task. Once the TA is chosen from the TA list, the detailed TA information (i.e. experience, ratio of assigned vs. applied tasks) will be presented in TAAP. After the TA assigner confirms the assignment, TAAP will check the schedule conflicts. If no conflicts, the assignment will be fulfilled; otherwise the conflict information will be popped up.

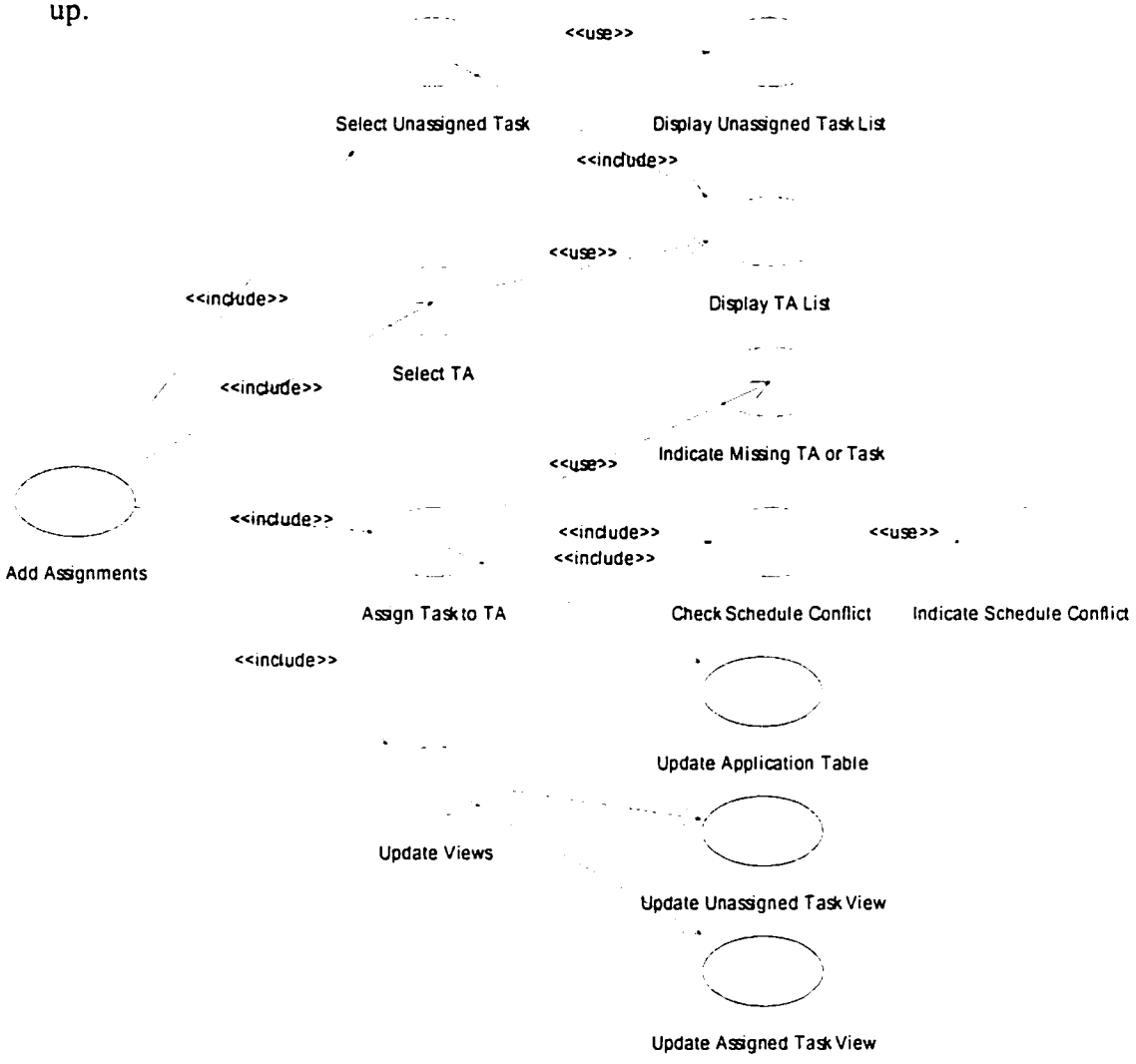

Figure 2.3 "Add Assignments" Use Case

- Delete Assignments (Figure 2.4).

During this process, the TA assigner will choose an assigned task from the assigned task list to delete. TAAP always asks for the confirmation.

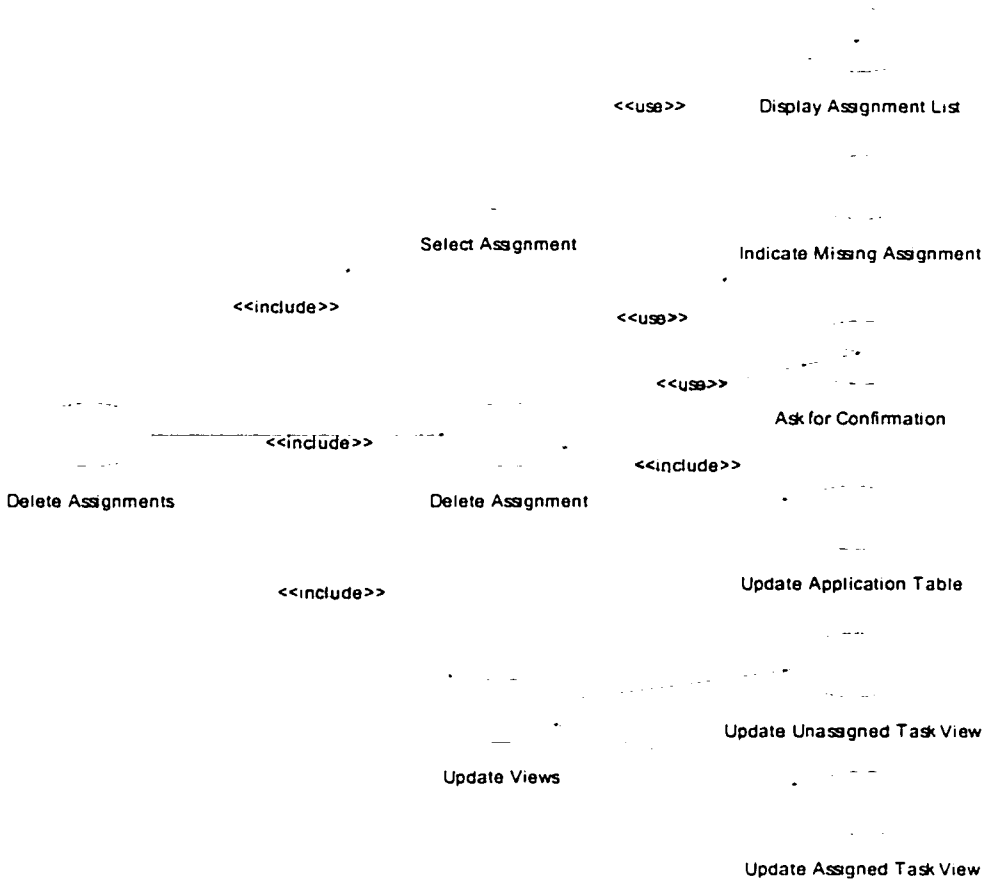

Figure 2.4 "Delete Assignments" Use Case

"View and Report"

The functionality of status report is implemented in the "View and Report" task, which is further divided into three sub-tasks:

- View Tasks

TAAP provides different queries for the posted tasks, and the query criteria comprises the combination of course type, task type and task state. It also provides

the statistical information based on the *Tasks* and *Applications* tables, for example, the number and the percentage for both assigned and unassigned tasks. The status report can be printed out by TAAP.

- View TAs

TAAP lists the TA information and provides the statistical information based on the *TAs* and *Applications* tables, for example, the number and the percentage for both TAs with assigned tasks and TAs with no assigned tasks. The status report can be printed out by TAAP.

- View Assignments

TAAP lists the existing assignment information and provides the statistical information based on the *Tasks* and *Applications* table. This sub-task is similar to "View Tasks" except it only focuses on the assigned tasks.

TAAP is a project accomplished in the team of two, and the development work (including analysis, design and final implementation) is distributed between my teammate Jun Liu and myself. Therefore, only my contribution to the project, which is covered by the "Do TA Assignment" use case (see Appendix B for the screenshot), will be fully expanded and explained in this major report. You may refer to Liu's major report [4] for the detailed development associated with the "View and Report" use case.

CHAPTER 3

DETAILED DESIGN

Both design pattern and detailed design will be covered in this chapter. First, the MVC model will be introduced as the framework to be employed in the TAAP detailed design. The basic concept of MVC model [5] and its advantages will be explained and presented. Also I will demonstrate how to tailor the general MVC model into the design of TAAP system.

Later, the detailed design will be described in two different UML diagrams. Class diagrams will be used as a static description tool to reveal how different classes construct the whole TAAP system. Sequence diagrams will be presented to depict the interaction between each object (i.e. instance of a class) in a dynamic way under every possible scenario. Some design issues will also be discussed in this chapter.

3.1 General MVC Model

MVC (*Model-View-Controller*) framework is the most popular object oriented modeling technique, which was first introduced in SmallTalk (a pure OO language). Nowadays it has extensively been used by all kinds of GUI applications in different programming environments.

In the MVC paradigm the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of objects, each of which is specialized for its own task. The *view* manages the graphical and/or

textual output to the portion of the bitmapped display that is allocated to its application. The *controller* interprets the mouse and keyboard inputs from the user, commanding the *model* and/or the *view* to change as appropriate. The *model* manages the behavior and data of the application domain, responds to requests for information about its state (usually from the *view*), and responds to instructions to change state (usually from the *controller*). The formal separation of these three tasks is an important notion that is particularly suited to any OO programming language where the basic behavior can be embodied in abstract objects: *view*, *controller* and *model* objects. The MVC behavior is then inherited, added to, and modified as necessary to provide a flexible and powerful system.

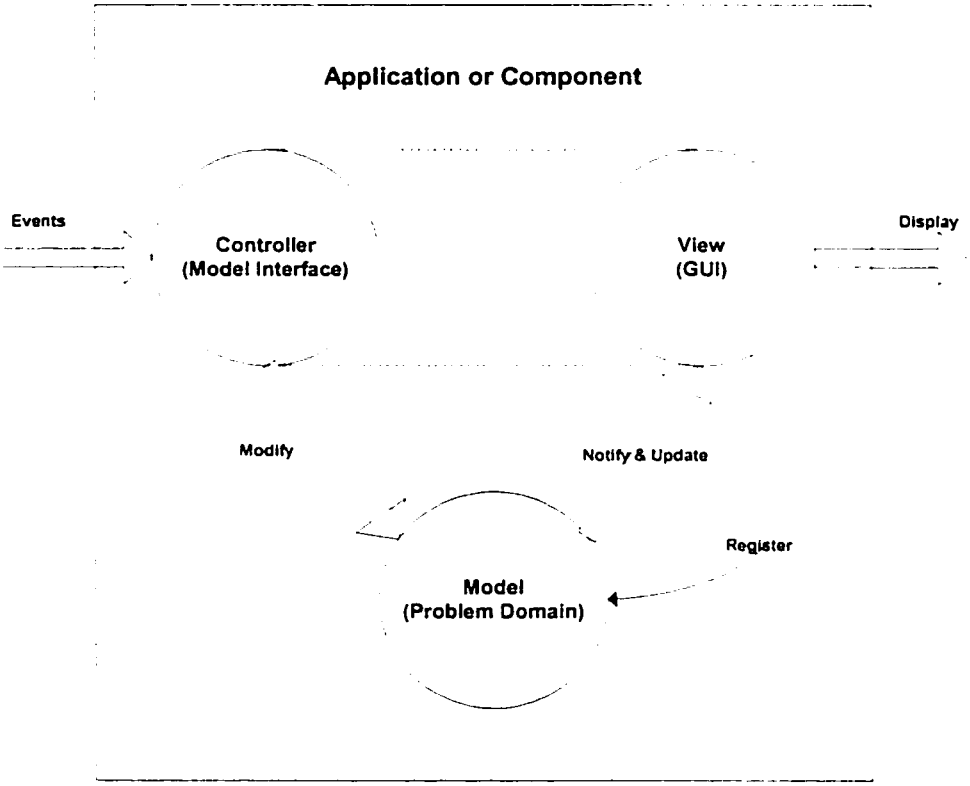

Figure 3.1 MVC Triad

The *model*, the *view* and the *controller* involved in the MVC triad (Figure 3.1) must communicate with each other if an application is to manage a coherent interaction with the user. Communication between a *view* and its associated *controller* is straightforward because *view* and *controller* are specifically designed to work together and sometimes it is hard to separate them. Usually the *controller* is defined as an interface class (with no member attributes) of the *model*, and the manipulation to the *controller* is initiated within its *view*, so in most cases people intend to merge the *controller* directly into its *view*. The *model*, on the other hand, communicates in a subtle and indirect manner --- "*register-and-notify*". The *view* registers itself in the *model*, so whenever the *model* changes it will notify all the *views* registered in it. In Java such kind of communication can be achieved by implementing *Observable* and *Observer* classes.

3.2 MVC Model in TAAP

The interaction between the lab coordinator and the TAAP system was described in Chapter 2. As noted in that chapter, the lab coordinator needs the following information to do TA assignment: unassigned task, assigned task and TA information. All the information will be controlled and manipulated through three different *views: Unassigned Task List, Assigned Task List, and TA List*. Some other feedback such as the completion percentage of assignments, the number of application and assigned task per applicant, and TA experience, will also be provided in these *views* to assist the lab coordinators doing the assignment.

A simplified MVC model (i.e. with combined *controller* and *view*) will be incorporated in the "Do TA Assignment" design. The rationale behind using the

combined *controller* and *view* is because first the *controller* itself is part of system event handling, which belongs to the functionality of operation system (Windows) and Java virtual machine. In addition, the *controller* is always defined as an interface to the *model* without any locally stored information, and it just provides the entry points for the callback functions. By combining *controller* and *view* we can simplify the design and focus on the domain-specific problems and GUI parts. Sometimes the author may simply refer *controller/view* as *view*.

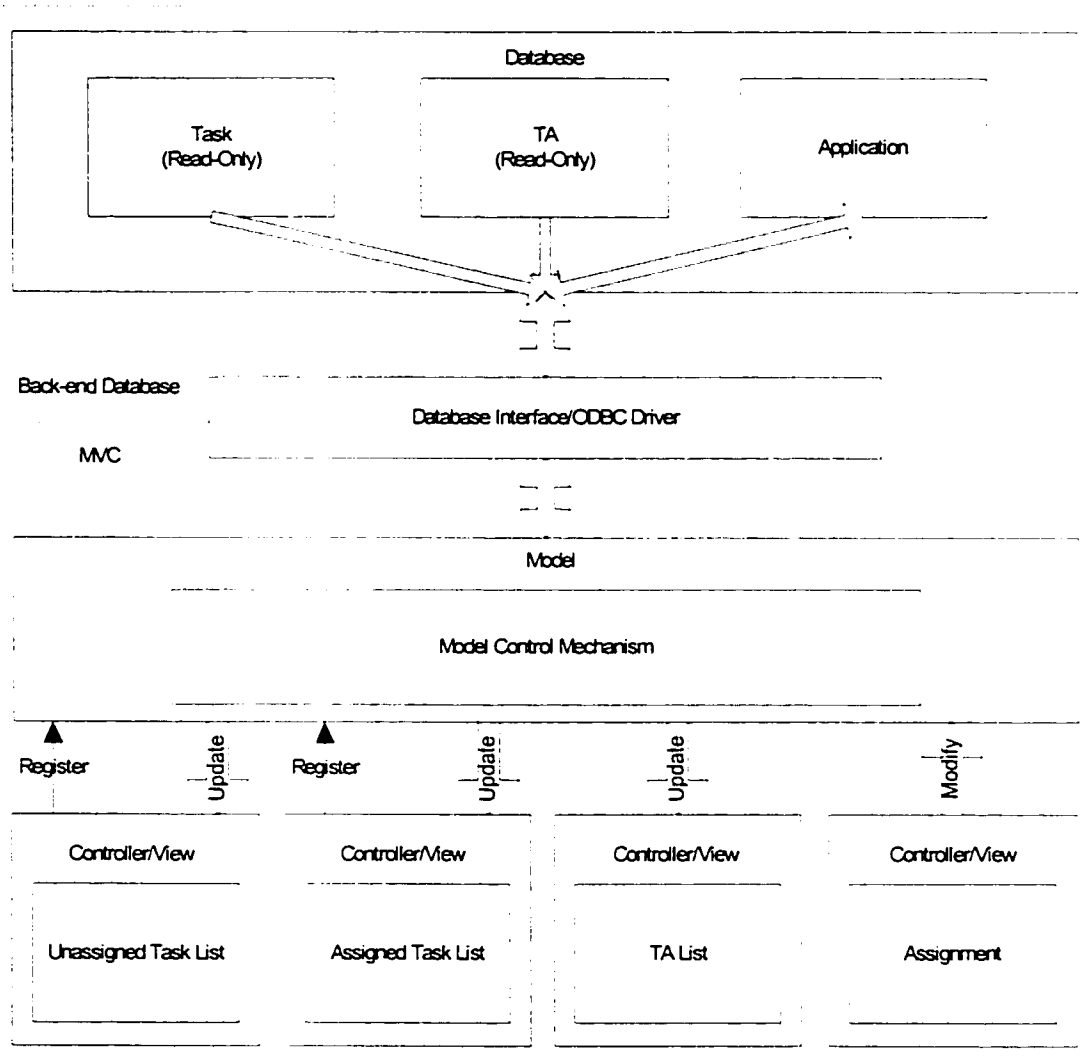

Figure 3.2 TAAP MVC Model with Back-end Database

The three-tier MVC model is presented in Figure 3.2. The "*Unassigned Task List*" *view*, "*Assigned Task List*" *view* and "*TA List*" *view* are also known as information *views*, which are used to provide information to the lab coordinators, so a new *view* called "*Assignment*" *view* is needed to be able to carry out the actual TA assignment. There are totally four *controllers/views* defined in the TAAP system.

The cardinality between *model* and *view* is one-to-many (here "many" equals four). The *model* provides controlling mechanism based on the needs of the actual application, and works as an interface between different *views* and the back-end database through the database interface/ODBC driver. The main task of the *model* is to translate the requests received from its *views*, generate corresponding SQL queries, send the queries to the back-end database, and then update involved *views* in the "pulling" or "pushed" mode. In the "pulling" mode, the *views* ask for the information update by sending the request to the *model*. Contrarily, the information update to the *views* in the "pushed" mode is initiated from the *model* side whenever the *model* senses any change that could affect its involved *views*. The major difference between "pulling" and "pushed" mode is that a *view* working in the "pushed" mode needs to be registered in its corresponding *model*.

The "*Unassigned Task List*" *view* and "*Assigned Task List*" *view* are operating under both "pulling" and "pushed" mode, so we need to register them in the *model* side. *The "TA List*" view is only used in the "pulling" mode; therefore no registration is required. Unlike these three information *views*, there is no information exhibited through the "*Assignment*" *view*. The main purpose of using the "*Assignment*" *view* is to provide a means for the lab coordinators to perform the actual assignment activities (i.e. add or

delete assignments). Any activity in the "*Assignment*" *view* will cause the information update (pushed by the *model*) in the registered *views*. A *view* like the "*Assignment*" *view* is usually called an action *view*.

3.3 STATIC DESIGN DESCRIPTION — UML CLASS DIAGRAMS

The detailed design follows the package-oriented description adopted by UML. The package diagram works as the high level representation of a software system or subsystem, in which every package represents one functional module that is further described using class diagram. The package diagram not only depicts how the whole system is constructed from different modules, but also illustrates the dependency between each module. Figure 3.3 shows the package diagram for the "Do TA Assignment" subsystem.

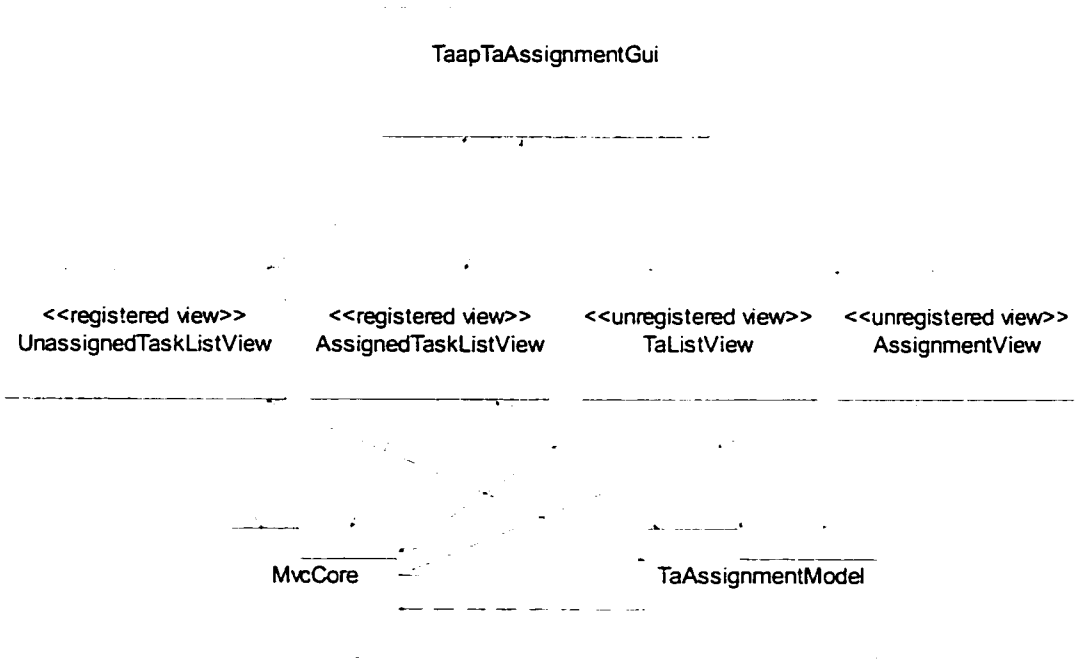

Figure 3.3 Package Diagram - "Do TA Assignment" Subsystem

There are total seven functional modules defined in Figure 3.3. They implement the functionality of TAAP MVC model.

TaapTaAssignmentGui Module

The "TaapTaAssignmentGui" module presents the actual appearance (User Interface) of "Do TA Assignment" subsystem. It responds the end user's activities by generating appropriate system events and visual feedback.

The "TaapTaAssignmentGui" module works very closely with the MVC *views*, and usually is considered as part of the MVC *views*. To separate the real GUI part from the MVC *views* allows us to develop multiple look-and-feel user interfaces by simply rewriting the GUI part without affecting the implementation of MVC model.

MvcCore Module

The "MvcCore" module implements the core part of MVC, i.e. *register-and-notify* mechanism. Such mechanism can also be realized using Java *Observer/Observable* class, which is usually working fine at the programming language level. However, "MvcCore" module gives developers more flexibility to customize their MVC models at the application level, and it is also much easier for them to add new features to the MVC core part.

The implementation of the *view* and *model* modules depends on the "MvcCore" module.

TaAssignmentModel Module

The "TaAssignmentModel" module corresponds to the *model* part of MVC implemented by the "Do TA Assignment" subsystem.

The main part of "TaAssignmentModel" module is to define the resolution scheme based on the domain problem (e.g. how to add and delete assignments? how to get TA and task information? etc.) It translates the requests received from *views* to proper SQL queries, which will be sent to the back-end database; after retrieving the information, it will update the involved *views* in the pulling and/or pushed mode.

UnassignedTaskListView Module

The "UnassignedTaskListView" module processes the unassigned task list, which is displayed through the "TaapTaAssignmentGui" module. The update to this *view* may be either requested by the lab coordinators via a system event (button click) or by the *model* via an application event (*model* change).

AssignedTaskListView Module

The "AssignedTaskListView" module processes the assigned task list, which is displayed through the "TaapTaAssignmentGui" module. The update to this *view* may be either requested by the lab coordinators via a system event (button click) or by the *model* via an application event (model change).

TaListView Module

The "TaListView" module processes the TA list, which is displayed through the "TaapTaAssignmentGui" module. The update to this *view* can only be requested by the lab coordinators via a system event (i.e. selecting an unassigned task from the unassigned task list).

AssignmentView Module

The "AssignmentView" module is the only *view* module that doesn't handle any list information to be displayed to the end users. Also it is the only *view* module that

responds to the "Add Assignments" and "Delete Assignments" requests from the lab coordinators.

Any change made to the *model* has to be done through the "AssignmentView" module.

Please take note that the implementation of "Do TA Assignment" subsystem does not necessarily provide one-to-one mapping between the components of the actual GUI layout and the *view* modules mentioned above. The GUI layout design is based on the human intuition and cognition, while the implementation of *view* modules belongs to the system design domain, which is affected by the developing environments (e.g. platform, target programming language and design pattern).

3.3.1 TaapTaAssignmentGui

In the class diagram Figure 3.4, the definition of "TaapTaAssignmentGui" class and its interfaces with the four different *views* are presented.

The purpose of using the interface classes is to simplify the representation of the class diagram. Without the interface classes, we have to write the complete *view* classes, which will make the class diagram complicated and hard to read. The interface classes are only used as the design representation in this major report; thus they are not part of the final implementation.

It's also for the sake of simplicity that not all attributes and member functions are listed in the "TaapTaAssignmentGui" class. For example, the label, the scroll pane and the layout controls (components) are ignored. Only those, which have direct interaction with *views*, are presented.

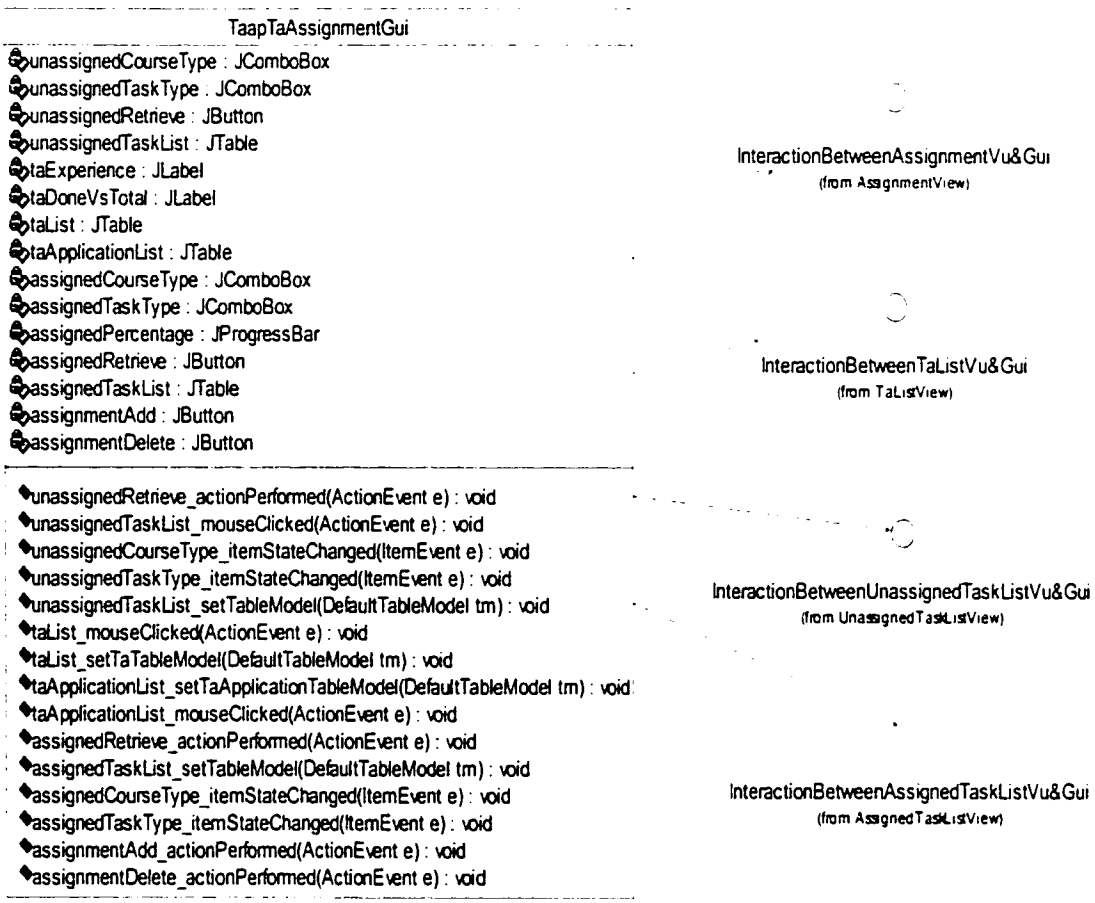

Figure 3.4 Class Diagram – TaapTaAssignment GUI

The "TaapTaAssignmentGui" class diagram implicates three visually-separated areas in the GUI part, "Unassigned Task List" area, "TA List" area and "Assigned Task List" area. The action buttons "Add Assignment" and "Delete Assignment", which correspond to the activities covered by the "AssignmentView" module, are located in the "TA List" and "Assigned Task List" area respectively.

3.3.2 MvcCore

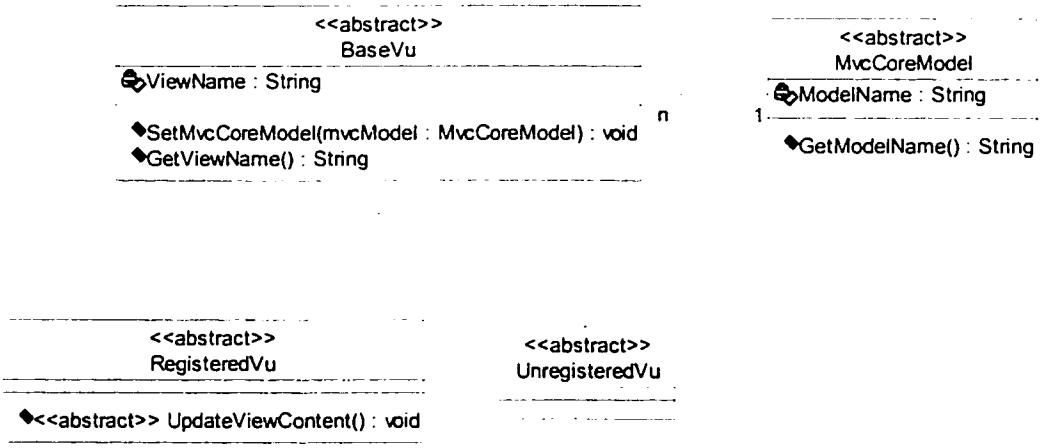

Figure 3.5 Class Diagram – MVC Core

All classes defined in the "MvcCore" class diagram (Figure 3.5) are abstract classes that cannot be instantiated as objects but can only be inherited by the other classes.

There are two types of *views*, unregistered *view* and register *view*. For the *views* to be registered in the MVC *model*, they have to be derived from the "RegisteredVu" class; while for the unregistered *views*, they have to be derived from the "UnregisteredVu" class instead.

"MvcCoreModel" class is served as the base class for the MVC *models*, and all the MVC *models* must inherit from it.

3.3.3 TaAssignmentModel

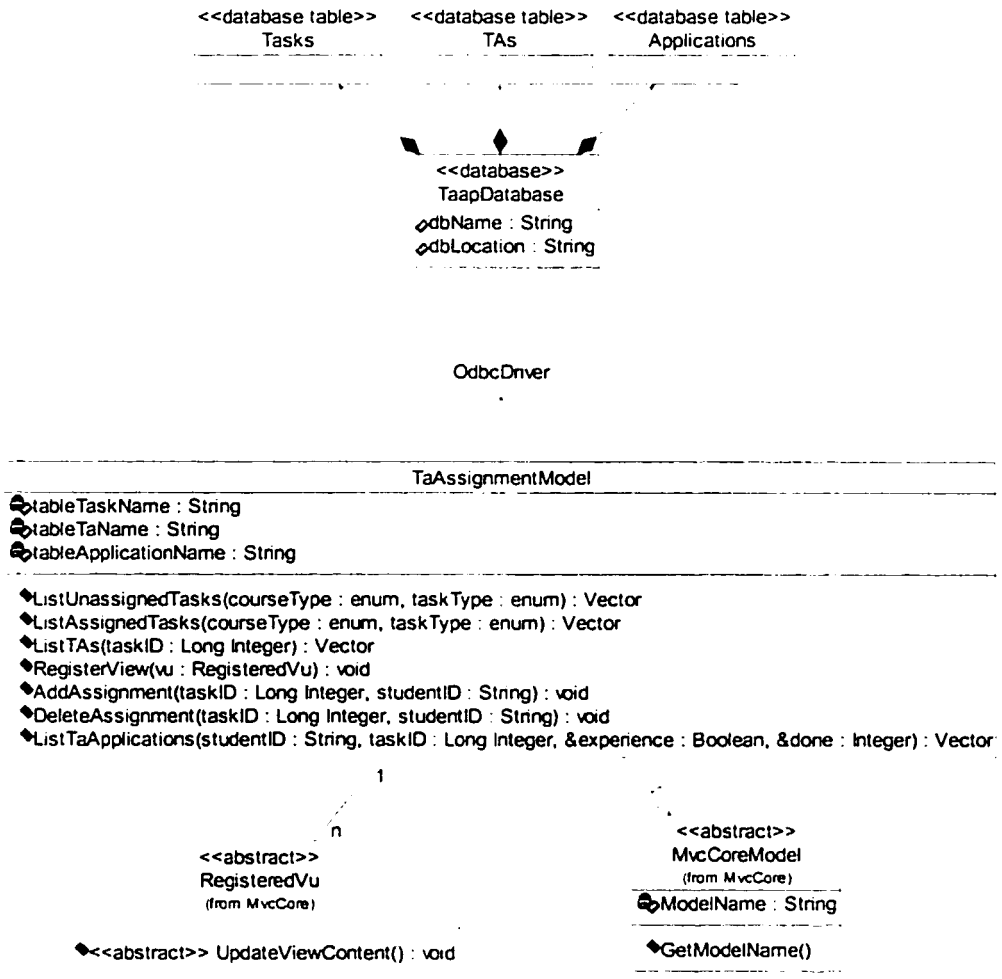

Figure 3.6 Class Diagram – TaAssignment Model

In the class diagram Figure 3.6, "TaAssignmentModel" class provides a clear mapping between the application domain (information stored in the "TaapDatabase") and the "Do TA Assignment" subsystem. The member functions except "RegisterView" defined in the "TaAssignmentModel" class represents the resolution scheme for the

domain problems. The access to the "TaapDatabase" is done using SQL language via ODBC driver.

The "TaAssignmentModel" object can have multiple registered *view* objects, in our case the multiplicity *n* is equal to 2.

3.3.4 UnassignedTaskListView

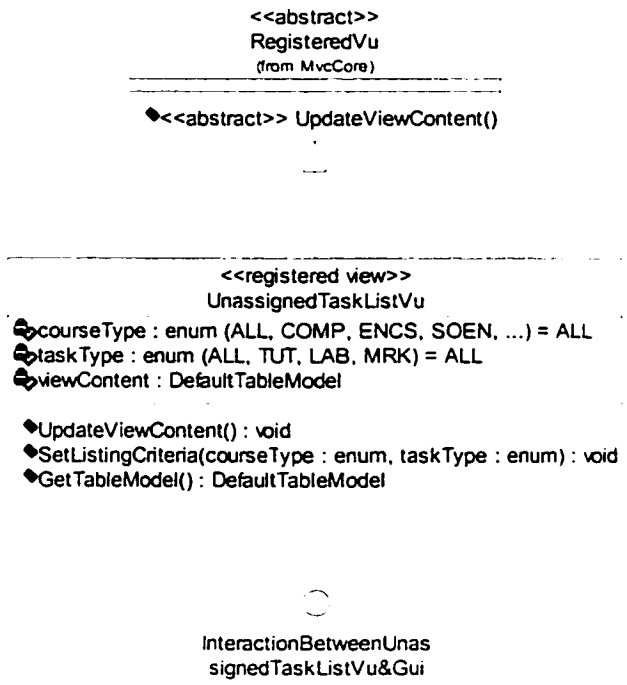

Figure 3.7 Class Diagram – UnassignedTaskList View

In the class diagram Figure 3.7, the "UpdateViewContent" abstract member function defined in the "RegisteredVu" class must be implemented in the "UnassignedTaskListVu" class. This member function will be called either by the

"TaapTaAssignmentGui" class in the "pulling" mode or by the "TaAssignmentModel" class in the "pushed" mode.

The "UnassignedTaskListVu" class uses the Java "DefaultTableModel" component as the storage of the *view* content. The "TaapTaAssignmentGui" class can get the "DefaultTableModel" by calling the "GetTableModel" member function, once it gets the table model, the *view* content will be displayed to the end users in the "Unassigned Task List" area via the Java "JTable" component.

3.3.5 AssignedTaskListVu

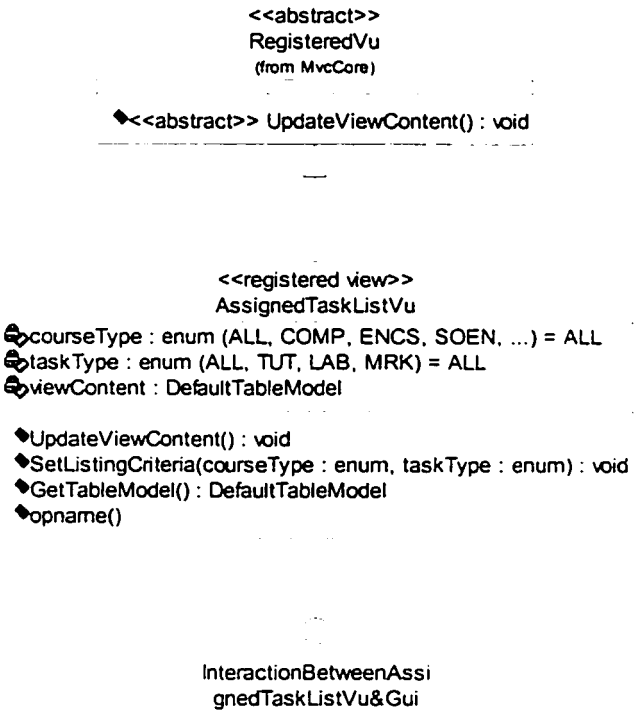

Figure 3.8 Class Diagram – AssignedTaskList View

Everything defined in the class diagram Figure 3.8 is pretty similar to that in the class diagram Figure 3.7. Except that the *view* content of the "AssignedTaskListVu" class stores the "Assigned Task List" information, and will be displayed by the "TaapTaAssignmentGui" class in the "Assigned Task List" area.

3.3.6 TaListView

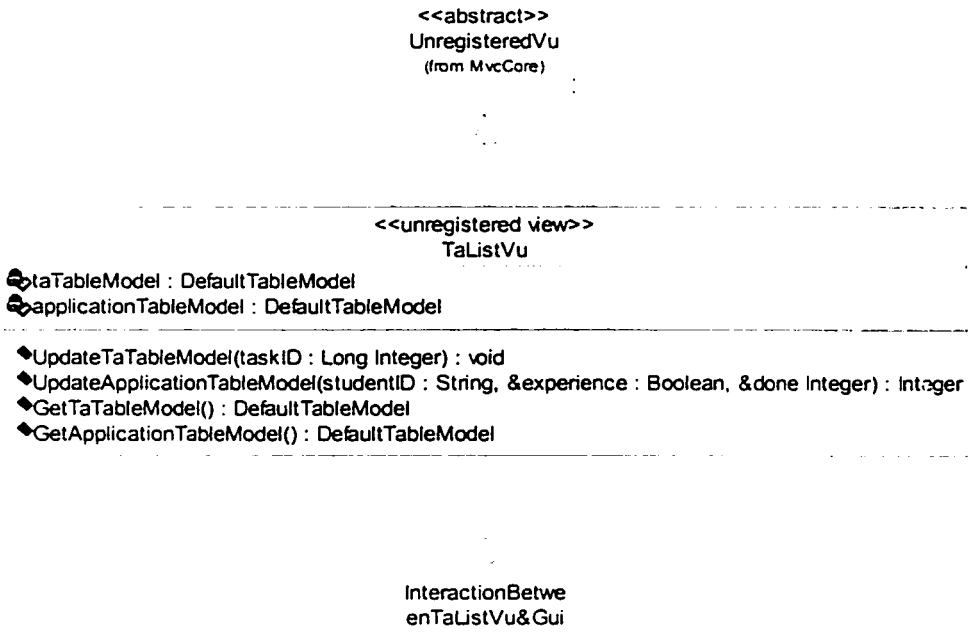

Figure 3.9 Class Diagram – TaList View

In the class diagram Figure 3.9, the "TaListVu" class is an unregistered *view* class. Compared to the registered *views*, the changes to the *view* content of the "TaListVu" class are always initiated from the "TaapTaAssignmentGui" class.

The *view* content of the "TaListVu" class consists of two table models: taTableModel and applicationTableModel. The first table model is used to store the information (ID, Name and Experience) of TAs who have applied a selected unassigned task. Whenever a specific TA is chosen by the lab coordinator, his/her application history will be retrieved and stored in the second table model.

The "TaListVu" class utilizes the information stored in the applicationTableModel to generate some useful data on-the-fly such as whether the applicant has experience relevant for the selected task and the ratio of assigned tasks vs. total applications on the per applicant basis. Such kind of data will be displayed in the "TaList" area through the "TaapTaAssignmentGui" class.

3.3.7 AssignmentView

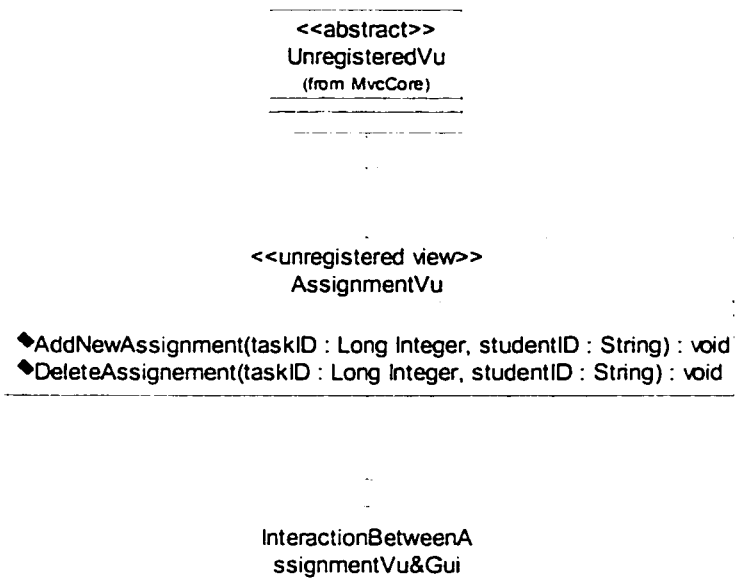

Figure 3.10 Class Diagram – Assignment View

In the class diagram Figure 3.10, the "AssignmentVu" class implements an unregistered *view*. It only sends action requests (received from the "TaapTaAssignmentGui" class) to the *model*, and never retrieves any data from the *model* in either "pulling" or "pushed" mode. Therefore no update member function is needed.

3.4 Dynamic Design Description — UML Sequence Diagrams

Sequence Diagram is an alternate representation of an interaction. It traces the execution of an interaction in time, and is the most used technique for the dynamic design description.

Not all the possible scenarios have been covered in this section. Only the most important ones that represent the core activities in the "Do TA Assignment" subsystem will be presented via sequence diagrams. The interaction is depicted in a timely manner in every sequence diagram using a series of member function calls among the involved objects. Considering the self-explanatory nature of sequence diagram, comments are only added when they are really needed.

3.4.1 "Do TA Assignment" Subsystem Initialization

The initialization of the "Do TA Assignment" subsystem reveals how each object is instantiated and how *views* are registered in the *model*.

The initialization procedure begins when the TAAP system is started by the lab coordinators.

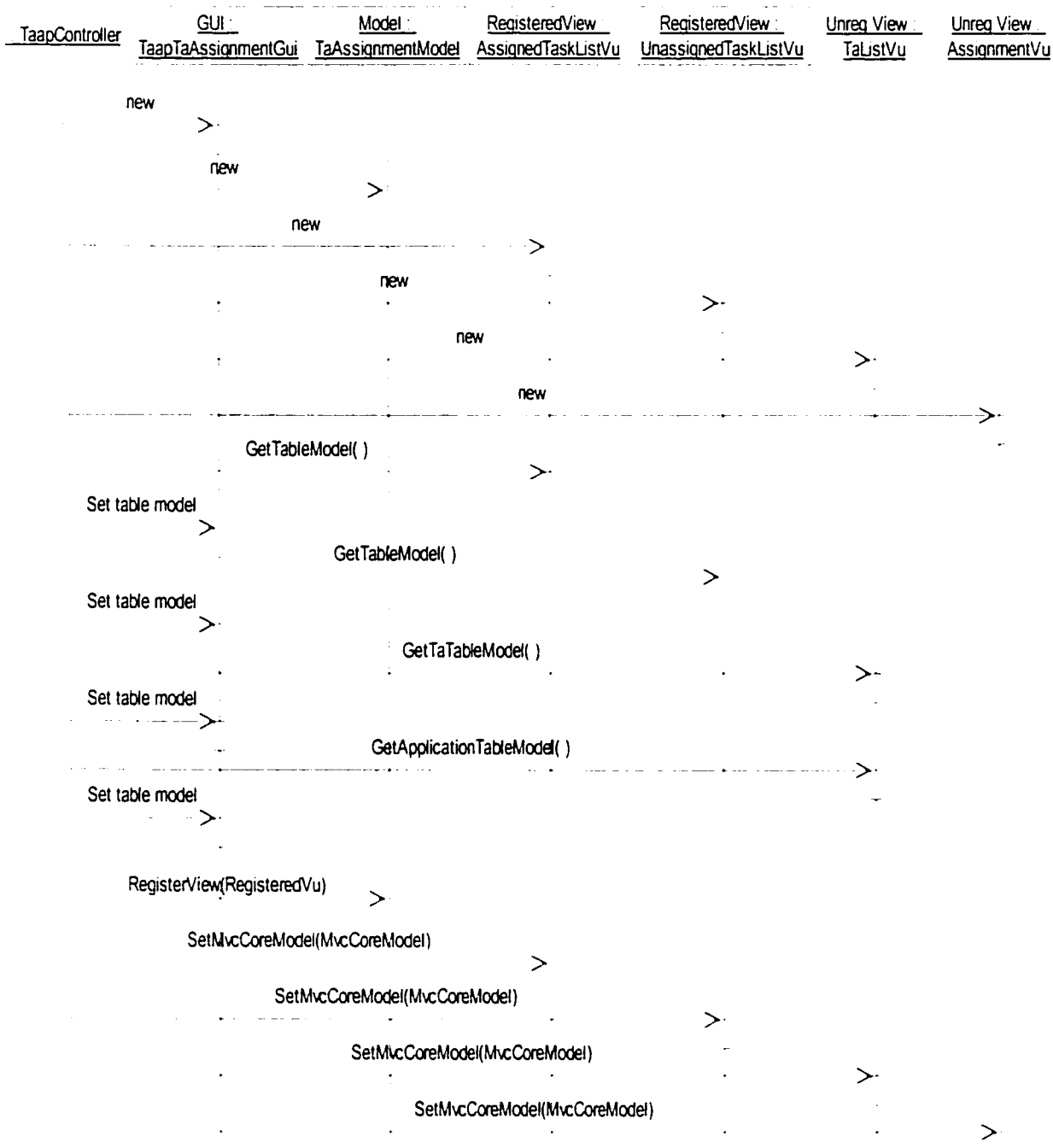

Figure 3.11 Sequence Diagram - Subsystem Initialization

3.4.2 Unassigned Task List Update (pulling)

The update to the "*Unassigned Task List*" *view* in the pulling mode is initiated by the RetrieveButtonClick system event, which is triggered by the lab coordinator when he/she clicks the "Retrieve" button from the "Unassigned Task List" area.

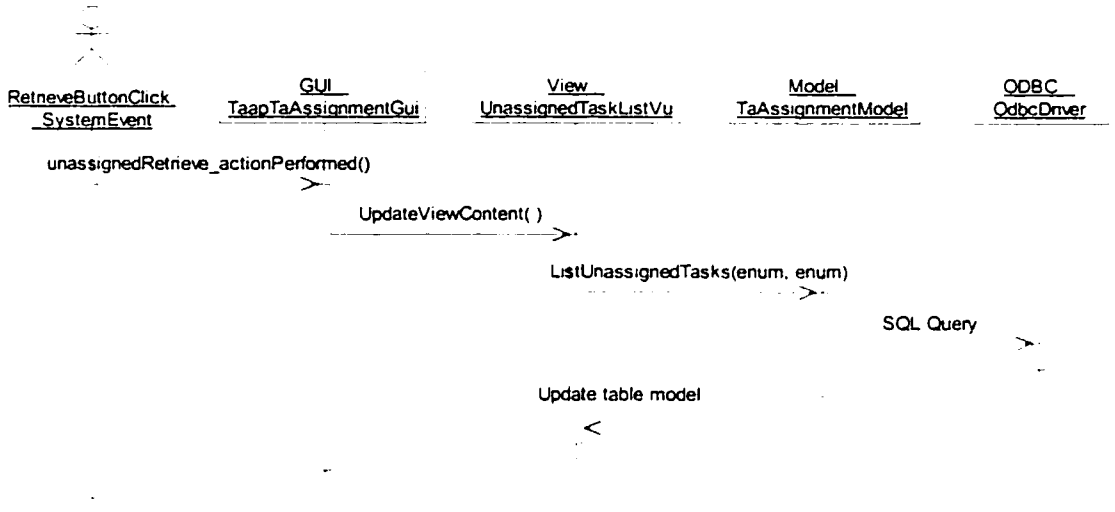

Figure 3.12 Sequence Diagram - Unassigned Task List Update (pulling)

3.4.3 Unassigned Task List Update (pushed)

The update to the "*Unassigned Task List*" *view* in the pushed mode is initiated by the application event, which is triggered by the change made to the *Applications* database table. The change to the *Applications* database table is caused by either adding a new assignment or deleting an existing assignment.

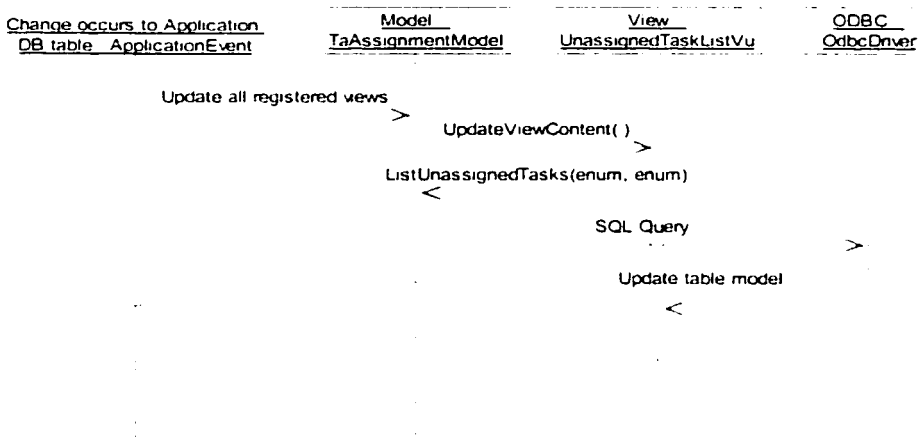

Figure 3.13 Sequence Diagram - Unassigned Task List Update (pushed)

3.4.4 Assigned Task List Update (pulling)

The update to the "*Assigned Task List*" *view* in the pulling mode is initiated by the RetrieveButtonClick system event, which is triggered by the lab coordinator when he/she clicks the "Retrieve button" from the "Assigned Task List" area.

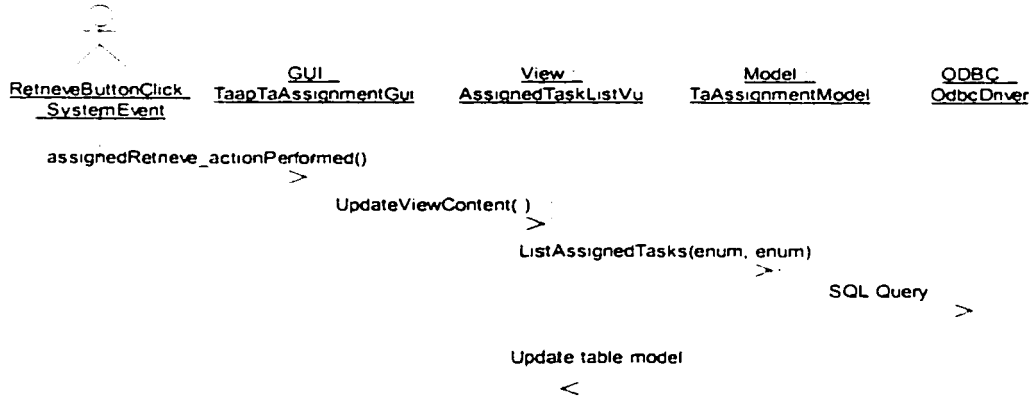

Figure 3.14 Sequence Diagram - Assigned Task List Update (pulling)

3.4.5 Assigned Task List Update (pushed)

The update to the "*Assigned Task List*" *view* in the pushed mode is initiated by the application event, which is triggered by the change made to the *Applications* database table. The change to the *Applications* database table is caused by either adding a new assignment or deleting an existing assignment.

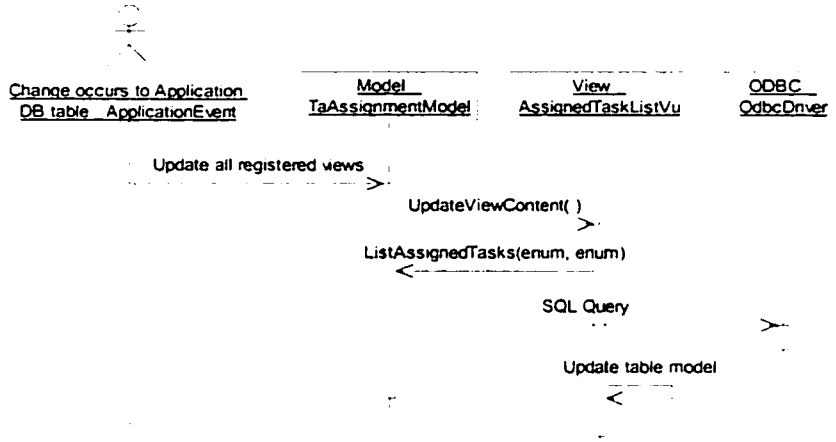

Figure 3.15 Sequence Diagram - Assigned Task List Update (pushed)

3.4.6 TA List Update

The update to the "*TA List*" *view* is initiated by the SelectionChanged system event, which is triggered by the lab coordinator when he/she selects an unassigned task from the "*Unassigned Task List*" *view*.

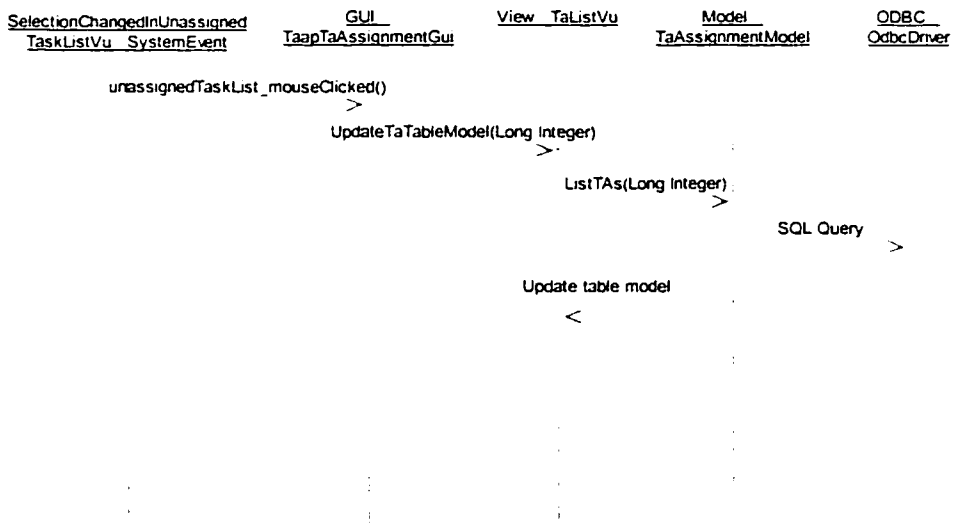

Figure 3.16 Sequence Diagram - TA List Update

3.4.7 Application Information Update

The update to the application information is initiated by the SelectionChanged system event, which is triggered by the lab coordinator when he/she selects a TA from the "*TA List*" *view*.

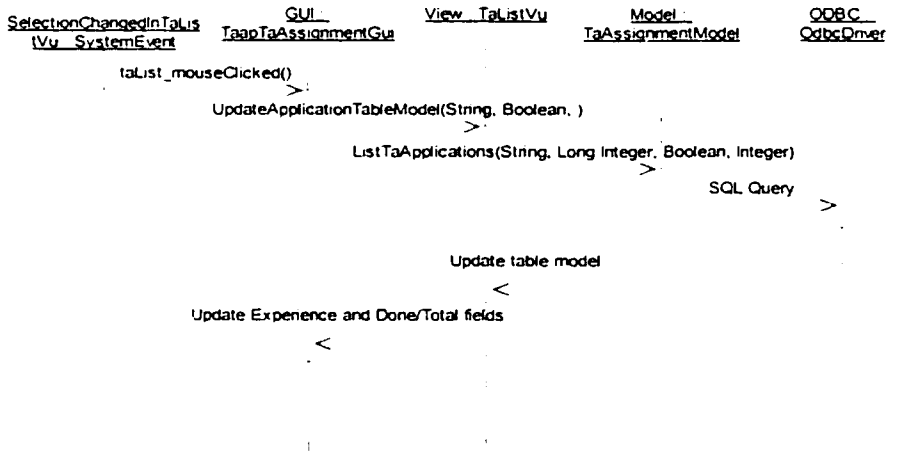

Figure 3.17 Sequence Diagram - Application Information Update

3.4.8 Add Assignment

"Add Assignment" is initiated by the ActionPerformed system event, which is triggered by the lab coordinator after he/she selects an unassigned task from the "*Unassigned Task List*" *view* and a TA from the "*TA List*" *view*, and clicks the "Assign" button from the "TA List" area.

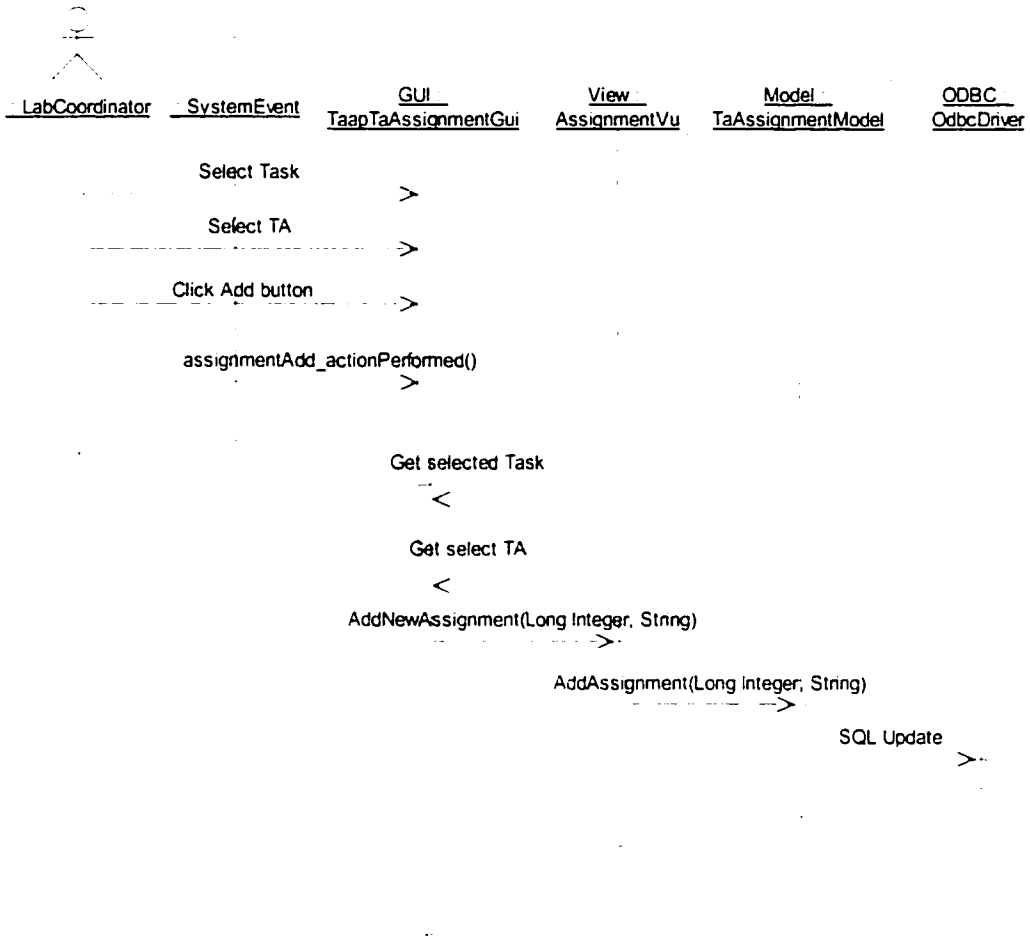

Figure 3.18 Sequence Diagram - Add Assignment

If there is no schedule conflict, the *Applications* database table will be modified to reflect the new assignment and the update to the "*Assigned Task List*" *view* and the "*Unassigned Task List*" *view* will begin (see Figure 3.13 and 3.15). Otherwise, an error message (not shown in the diagram) will be popped up and the assignment won't proceed.

3.4.9 Delete Assignment

"Delete Assignment" is initiated by the ActionPerformed system event, which is triggered by the lab coordinator after he/she selects an assigned task from the "*Assigned Task List*" *view*, and clicks the "Delete" button from the "Assigned Task List" area. Before the deletion can be performed, the lab coordinator will be asked for confirmation.

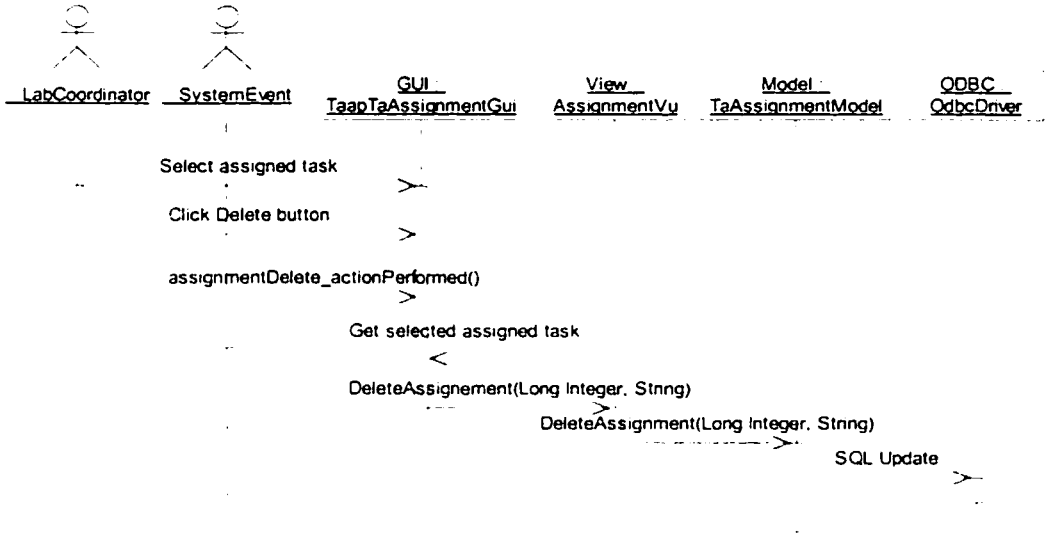

Figure 3.19 Sequence Diagram - Delete Assignment

Once the deletion is confirmed, the *Applications* database table will be modified and the update to the "*Assigned Task List*" *view* and the "*Unassigned Task List*" *view* will begin (see Figure 3.13 and 3.15).

CHAPTER 4

IMPLEMENTATION

The implementation of "Do TA Assignment" subsystem is discussed in this chapter. The focus will be on the issues related to the resolution scheme of the problem domain (which is relevant to the *model* part of MVC), for example, how to access MS Access database from Java program, how to use SQL queries to perform different tasks, etc. This chapter is organized according to the topics to be expanded here.

4.1 WHAT IS SQL

Structured Query Language (SQL) [6] is an ANSI standard programming language for communicating with Database Management Systems (DBMSs). SQL provides a means for programmers to retrieve and update data stored in the different database tables, which are pre-created by the Database Administrator (DBA) during the database setup.

Today almost all the commercial database management systems including MS Access support SQL programming. Because SQL is so widely used in the industries, the support to SQL has gradually been ported in the other programming languages such as C/C++, Visual Basic and Java.

The typical SQL statements include:

- SELECT

Retrieves data (records) from the given database tables based on the user-defined query criteria.

- DELETE

Deletes data (records) satisfying the user-defined query criteria from the specified database table.

- UPDATE

Updates data (records) satisfying the user-defined query criteria from the specified database table.

- INSERT

Inserts data (one record at a time) into the specified database table.

Only SELECT and UPDATE statements will be used to access the MS Access database in the TAAP project.

4.2 Access to Database from Java

JBuilder5.0 Personal Edition is adopted as the development tool in the TAAP project. Unlike its Professional Edition, the Personal Edition doesn't provide the database control components such as dbTable, dbList, dbEdit, dbNavigator and so on. The "dataset" property defined in the database control components implements the encapsulation the access to MS Access database tables. Such kind of encapsulation helps segregate the low-level connection details from the high-level programming implementation, and improve the transparent access to the database tables from Java applications.

The following interface class (Figure 4.1) has been incorporated to simulate the same functionality supported by the "dataset" property.

```
class dbInterface
{
    ...

// Attribute used to maintain the ODBC connection

    public  Connection myConn;
    ...

// Constructor initiates the ODBC connection based on the database configuration
// "try ... catch..." provides the exception handling

    public dbInterface()
    {
        String driverClass = "sun.jdbc.odbc.JdbcOdbcDriver";// ODBC type
        String url = "jdbc:odbc:SystemDB";                  //      Database
  name
        String username = "admin";                          // User name
        String password = "admin";                          // Password

// Start to establish the ODBC connection

        try
        {
            Class.forName(driverClass);
            myConn = DriverManager.getConnection(url, username,
                                                 password);
        }
        catch (Exception e)
        {
            System.err.println("Cannot get connection " + e.getMessage());
        }
    }
    ...
// Function used to execute SQL statements that are passed in thru argument "sql"
// "try ... catch..." provides the exception handling

    public Vector sqlExecutor(String sql)
    {

// Define local variables

        Statement stmt;
        ResultSet results;
        ResultSetMetaData rsmd;
```

```
        int numCols;
        int i;
        Vector retVec = null;
        Vector row = null;

        try
        {

// Create statement entity and execute SQL statement

                stmt = myConn.createStatement();
                stmt.Execute(sql);

// Get query results and table matrix

                results = stmt.getResultSet();
                rsmd = results.getMetaData();
                numCols = rsmd.getColumnCount();

// Process query results and put them into vector

                while (results.next())
                {
                        if (retVec = null) retVec = new Vector;
                        row  = new Vector();
                        for (i=1; i <= numCols; i++)
                        {
                                row.addElement(results.getString(i));
                         }
                        retVec.addElement(newRow);
                }

// Before exit, close both result and statement entities

                results.close();
                stmt.close();
        }
        catch (Exception e)
        {
                System.err.println("SQL execution error  " + e.getMessage());
        }

// Return processed query results in vector.

        return retVec;
```

```
    }
    ...
}
```

Figure 4.1 dbInterface Implementation

Once the object dbInterface is instantiated in the Java source code, the ODBC connection to the MS Access database will be initiated and set up by its constructor. The following code is executed to instantiate a dbInterface object:

```
dbInterface myInterface = new dbInterface();
```

The member function sqlExecutor(String sql) is used to query and update the tables stored in the MS Access database. It takes one input parameter and returns a two-dimension Vector. The input parameter is a string containing one SQL expression. The following two statements are valid function calls:

```
myInterface.sqlExecutor("SELECT * FROM Applications \
                        WHERE Assigned = True");
myInterface.sqlExecutor("UPDATE Application SET Assigned = True \
                        WHERE ID = '4389484' AND TaskID = 123");
```

The return value of sqlExecutor might be NULL if a SQL query returns nothing; for example, the result of SELECT query has an empty set. The sqlExecutor always returns NULL when processing the UPDATE queries.

4.3 Data Representation in GUI

One of the major tasks defined in the "Do TA Assignment" subsystem is to retrieve information (i.e. Unassigned Task List, Assigned Task List and TA List) from

MS Access database and display them in the GUI part. In JBuilder5.0 Professional Edition, the whole process is pretty straightforward and easy to program using dbTable component. To achieve the same goal without the dbTable component, we have to write our own code using the conventional Java table (JTable).

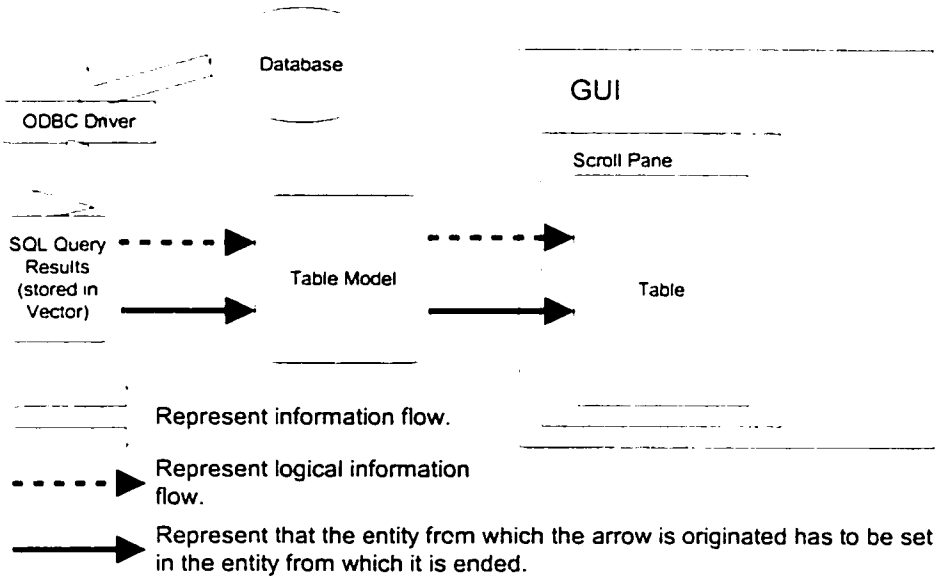

Figure 4.2 DB Table Representation

Figure 4.2 shows the detailed process of how to map the information retrieved from the MS Access database to the GUI interface. The SQL query results are stored in the Vector, which is returned from the sqlExecutor function call. The Vector is the place where the actual data is stored and the logical information flow only indicates the entities through which the data can be displayed in the GUI interface.

The Table Model entity is simply implemented by a DefaultTableModel object in Java, while the Table entity is implemented by a JTable object. The function setModel defined in the JTable class is used to set the DefaultTableModel object in a JTable object; the function setDataVector defined in the DefaultTableModel class is used to set the Vector in a DefaultTableModel object.

To be able to scroll down the information displayed by the JTable object, the JTable object has to be put in a Java Scrollpane object.

4.4 Data Manipulation in MS Access Database

The GUI part of "Do TA Assignment" subsystem provides the means for the lab coordinators to interact with the back-end TAAP database system. The manipulation to the data stored in the MS Access tables is encapsulated in the TaAssignmentModel object and realized via the dbInterface object using SQL queries.

The SQL queries will be elaborated in this section, however, in order to keep the demonstration simple and clear, only the SQL expressions will be present instead of the whole Java source code. All the query results will be displayed in an orderly manner based on the selected data fields.

4.4.1 List Unassigned Tasks

Listing unassigned tasks involves two MS Access tables, *Tasks* and *Applications*, and is a two-step procedure. The first step is to find the list of all the assigned task IDs from *Applications* table; the second step is to get all the unassigned tasks from *Tasks*

table, of which IDs are not in the list obtained from the first step. This is implemented using one SQL query that consists of two SELECT statements.

Considering the different combinations of course and task types (i.e. specific types vs. ALL), there are total four different scenarios.

Scenario One: Course != 'ALL' and TaskType != 'ALL'

SELECT DISTINCT * FROM *Tasks*

WHERE Course = '*courseType*' AND TaskType = '*taskType*' AND

TaskID NOT IN (SELECT TaskID FROM *Applications*

WHERE Assigned = True)

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

Scenario Two: Course = 'ALL' and TaskType != 'ALL'

SELECT DISTINCT * FROM *Tasks*

WHERE TaskType = '*taskType*' AND

TaskID NOT IN (SELECT TaskID FROM *Applications*

WHERE Assigned = True)

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

Scenario Three: Course != 'ALL' and TaskType = 'ALL'

SELECT DISTINCT * FROM *Tasks*

WHERE Course = '*courseType*' AND

TaskID NOT IN (SELECT TaskID FROM *Applications*

WHERE Assigned = True)

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

Scenario Four: Course = 'ALL' and TaskType = 'ALL'

SELECT DISTINCT * FROM *Tasks*

WHERE TaskID NOT IN (SELECT TaskID FROM *Applications*

WHERE Assigned = True)

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

4.4.2 List Assigned Tasks

Like listing unassigned tasks, listing assigned tasks uses the same MS Access tables, *Tasks* and *Applications*, and is a two-step procedure. The first step is to find the list of all the assigned task IDs from *Applications* table; the second step is to get all the assigned tasks from *Tasks* table, of which IDs are in the list obtained from the first step. This is also implemented using one SQL query that consists of two SELECT statements.

Similarly, there are four different scenarios when a lab coordinator wants to list the assigned task information.

Scenario One: Course != 'ALL' and TaskType != 'ALL'

SELECT DISTINCT *Tasks*.*, *TAs*.ID, *TAs*.Name FROM *Tasks*, *TAs*, *Applications*

WHERE Course = '*courseType*' AND TaskType = '*taskType*' AND

Tasks.TaskID = *Applications*.TaskID AND

TAs.ID = *Applications*.ID AND *Applications*.Assigned = True

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

Scenario Two: Course = 'ALL' and TaskType != 'ALL'

SELECT DISTINCT *Tasks*.*, *TAs*.ID, *TAs*.Name FROM *Tasks*, *TAs*, *Applications*

WHERE TaskType = '*taskType*' AND

Tasks.TaskID = *Applications*.TaskID AND

TAs.ID = *Applications*.ID AND *Applications*.Assigned = True

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

Scenario Three: Course != 'ALL' and TaskType = 'ALL'

SELECT DISTINCT *Tasks*.*, *TAs*.ID, *TAs*.Name FROM *Tasks*, *TAs*, *Applications*

WHERE Course = '*courseType*' AND

Tasks.TaskID = *Applications*.TaskID AND

TAs.ID = *Applications*.ID AND *Applications*.Assigned = True

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

Scenario Four: Course = 'ALL' and TaskType = 'ALL'

SELECT DISTINCT *Tasks*.*, *TAs*.ID, *TAs*.Name FROM *Tasks*, *TAs*, *Applications*

WHERE *Tasks*.TaskID = *Applications*.TaskID AND

TAs.ID = *Applications*.ID AND *Applications*.Assigned = True

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

4.4.3 List Applicants Who Apply to the Selected Task

This query contains one SELECT statement referring two MS Access tables: *TAs* and *Applications*. The idea is to first find the list of applications relevant to the given task

from *Applications* table, and then get the list of TAs from *TAs* table using the TA IDs obtained from the list of relevant applications.

SELECT DISTINCT *TAs*.* FROM *TAs*, *Applications*

WHERE *Applications*.TaskID = '*taskID*' AND *TAs*.ID = *Applications*.ID

ORDER BY *TAs*.ID, *TAs*.Name

4.4.4 List Tasks Applied by a Specific TA

This query contains one SELECT statement referring two MS Access tables: *Tasks* and *Applications*. The idea is to first find the list of applications relevant to the specific TA from *Applications* table, and then get the list of Tasks from *Tasks* table using the TaskIDs obtained from the list of relevant applications.

SELECT DISTINCT *Applications*.Assigned, *Tasks*.Course, *Tasks*.Number,
Tasks.Session, *Tasks*.Section FROM *Tasks*, *Applications*

WHERE *Applications*.TaskID = *Tasks*.TaskID AND *Applications*.ID = '*taID*'

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.Section

4.4.5 Get the Total Number of Assigned Tasks

This query is very similar to the one described in the sub-section 4.3.2, except it returns the number of records instead of the list of records. And also it has four different possible scenarios depending on the selection of course and task types.

Scenario One: Course != 'ALL' and TaskType != 'ALL"

SELECT COUNT(*) FROM *Tasks*, *TAs*, *Applications*

WHERE Course = '*courseType*' AND TaskType = '*taskType*' AND

Tasks.TaskID = *Applications*.TaskID AND

TAs.ID = *Applications*.ID AND *Applications*.Assigned = True

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

Scenario Two: Course = 'ALL' and TaskType != 'ALL"

SELECT COUNT(*) FROM *Tasks, TAs, Applications*

WHERE TaskType = '*taskType*' AND

Tasks.TaskID = *Applications*.TaskID AND

TAs.ID = *Applications*.ID AND *Applications*.Assigned = True

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

Scenario Three: Course != 'ALL' and TaskType = 'ALL"

SELECT COUNT(*) FROM *Tasks, TAs, Applications*

WHERE Course = '*courseType*' AND

Tasks.TaskID = *Applications*.TaskID AND

TAs.ID = *Applications*.ID AND *Applications*.Assigned = True

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

Scenario Four: Course = 'ALL' and TaskType = 'ALL"

SELECT COUNT(*) FROM *Tasks, TAs, Applications*

WHERE *Tasks*.TaskID = *Applications*.TaskID AND

TAs.ID = *Applications*.ID AND *Applications*.Assigned = True

ORDER BY *Tasks*.Course, *Tasks*.Number, *Tasks*.Session, *Tasks*.TaskType

4.4.6 Add a New Assignment

Based on the TAAP design, the new assignment is reflected by changing the Assigned status of a specific application from False to True in *Applications* table, and no new record is added during this process. This is one of the two queries using UPDATE statement in the TAAP project.

UPDATE *Applications* SET Assigned = True

WHERE ID = '*taID*' AND TaskID = '*taskID*';

4.4.7 Delete an Existing Assignment

Contrast to adding a new assignment, deleting an existing assignment is done by changing the Assigned status of a specific application from True to False in *Applications* table, and no record is really deleted during this process. This query also uses UPDATE statement.

UPDATE *Applications* SET Assigned = True

WHERE ID = '*taID*' AND TaskID = '*taskID*';

4.4.8 Schedule Conflict Check

The criteria for schedule conflict check is that one TA cannot have two assigned tasks in the same day, of which schedules are interleaved or the interval between them are less than 15 minutes. The schedule conflict check only applies to the non-marker tasks (i.e. tutor and lab).

Two MS Access tables *Tasks* and *Applications* will be used in this query and the Start and End time need to be converted into minutes before any comparison. If the result

of this query is NULL, it means no schedule conflict for the given task (specified by '*taID*'); otherwise the schedule conflict exists between the tasks in the return list and the given one.

SELECT *Tasks*.* FROM *Tasks*, *Applications*

WHERE *Applications*.ID = '*taID*' AND *Tasks*.TaskID = *Applications*.TaskID AND

Tasks.Days = '*day*' AND *Applications*.Assigned = True AND

Tasks.TaskType != '*MRK*' AND

((HOUR(Start)*60+MINUTE(Start) > *start-15* AND

HOUR(Start)*60+MINUTE(Start) < *end+15*) OR

(HOUR(End)*60+MINUTE(End) > *start-15* AND

HOUR(End)*60+MINUTE(End) < *end+15*))

Note: The time of *start* and *end* denoting the schedule of the given task are already converted into minutes.

CONCLUSION

The TAAP system test has been conducted. In order to guarantee the credibility of test results, the TAAP database was populated with enough data to simulate the actual workload in the department. The *Tasks* table contains 184 records converted from the department database for the 2002 winter term. At the time of testing TAAP we didn't have enough information about TA in the department database, so the *TAs* table was generated using some dummy data, and the total number of TAs is 120. The number of tasks that every TA can apply follows the distribution of 20(4)-40(3)-40(2)-20(1), which means each of the first 20 TAs has 4 applied tasks, each of the second 40 TAs has 3 applied tasks and so on. This would result in 300 different applications in the *Applications* table. The mapping between a TA and a task is done using SQL programming based on a random algorithm.

The test results proved to be very well and positive. The TAAP system passed all the test scenarios covered by the initial requirements (Appendix A). No performance problem has been perceived during the test, and the system delay for action responses (i.e. GUI update) is much better than the acceptance threshold. Additionally, the productivity is improved significantly by using the TAAP system. Compared to one or two days for the manual assignment, the 184 tasks can be assigned in around one hour. The TAAP system shows more efficiency especially when doing the assignment modification.

However, the current TAAP system has two drawbacks. Firstly, it is not a distributed system and can only run on the machine where it is installed. If the lab

coordinator uses different computers or more than one person need to work on the TA assignment, the TAAP system has to be physically installed on each computer. In this case we have to do the table populating for every single machine from the department database. To be able to maintain the consistence among the databases residing on the different computers, the sychronization mechanism must be introduced and will become the major overhead for the maintenance. Secondly, the TAAP system has its own customized database because the department database is not well designed and organized. This will become another overhead for the lab coordinator to keep the integrity of the customized database in case of any change made on the department database.

Since the TAAP system is developed using Java technology, it is very easy for us to overcome the first drawback by migrating TAAP to a web-based application. The future TAAP system can be implemented using JSP and Java applets on any web server, and the web browsers will be used as the TAAP client through the Intranet/Internet. In order for TAAP server to identify the login users and make sure the exclusive use of the system or resources, the user registry and notification services have to be included in the new TAAP system. Because of the standard ODBC connection used by TAAP, the machine for database server is not necessarily the same as the one for TAAP system. This will provide a great flexibility for the system administrators when they deploy the TAAP system.

To solve the second problem, the department database has to be redesigned and reorganized. Once it is done, the TAAP system will directly connect to the department database instead of using its own. By doing this, we may need to provide some extra functionality in TAAP for the end users to be able to modify the department database.

Also the mechanism for the automatic integrity check and report has to be implemented in the future system. For example, whenever the schedule for an assigned task is changed, TAAP should be able to report the schedule conflicts by listing the involved tasks and the TA assigned to, and the lab coordinator will use this information to do the adjustment.

REFERENCES

[1] Herbert Schildt, "*Java 2: The Complete Reference*", 2000-December, ISBN 0072130849

[2] Daniel Y. Liang, "*Introduction to Java Programming with JBuild4*", 2001-July, ISBN 0130333646

[3] Kevin Marlowe, "*Using MS Access 97*", 1997-November, ISBN 078914396

[4] Jun Liu, "*Major Report - Teaching Assistant Assignment Planner*", 2002-April

[5] Eric Gamma, et la, "*Design Patterns: Elements of Reusable Object-Oriented Software*", 1994-October, ISBN 0201633612

[6] Wrox Author Team, "*SQL Language Programmer's Reference*", 2000-September, ISBN 186100317X

APPENDIX A

TAAP INITIAL REQUIREMENTS

1 Introduction

Each term, the Department of Computer Science assigns tasks to Teaching Assistants. A Teaching Assistant (TA) performs one or more of several tasks: tutoring, lab instruction, and marking, depending on the requirements of the particular course.

Shortly before the beginning of the term, the following information is available: a list of the tasks that need to be done and, for each task, a list of people who have applied to do it. The Department's Lab Coordinator has to match the tasks to the applicants in such a way that the tasks get done properly and the applicants are happy.

The proposed program would not assign the tasks automatically but, instead, would simplify the task of the Lab Coordinator.

2 Entities and Relationships

An assignment is a link connecting a TA to a task. In this section, we describe the data that is associated with each of these entities.

2.1 Tasks

Each course is offered in zero or more sections. Introductory courses usually have several sections each term. Advanced and graduate courses typically have only one section. Courses that are offered only occasionally may have no sections at all in a particular year.

A task is described by the following data:

Course: A four-letter string. COMP, ENCS.

Number: A three- or four-digit string. 248, 5421.

Session: A single digit indicating the term. 1, 2, 4.

Title: The name of the course. Artificial Intelligence.

Section: The code for a section of the course. AA

Task type: A task associated with this section. TUT, LAB, MRK.

Days: Days on which the tutorial/lab is offered. T, WF.

Start: The start time of the tutorial/lab. 10:15.

Finish: The finishing time of the tutorial/lab. 11:30.

2.2 Teaching Assistant

A TA has some fixed characteristics and some data that changes as the user makes assignments. The fixed (or given) characteristics are:

Name: A string of characters.

Experience: A list of the tasks that the TA can perform.

The variable component of the data is described as follows:

Tasks: A list of the tasks actually performed by the TA.

2.3 Relationships

The essential relationship for this application links a TA to a task and is called an assignment. There are various constraints on the assignment relation:

- A TA cannot perform two tasks at the same time. In practice, there should be at least 15 minutes between the end of one task and the start of the next. However, since no particular time is associated with marking, a TA may both tutor and mark a course.

- Only certain tasks are associated with a particular section; it is not possible to perform other tasks. For example, COMP 248 has a tutorial but no laboratory; hence, a lab instructor is not required.
- The tasks that a TA performs are normally chosen from the Experience list of the TA. However, this constraint can be overidden.
- A task can be performed by at most one TA.

3 Use Cases

This section provides an informal set of requirements for the program by describing typical user behavior in terms of use cases.

- Load database.

The program loads either the TA database or the task database. It may be possible to obtain the task database from the Department's files. However, it may be simpler to set up a TA database and update it each year.

- Modify database.

The user can make modifications to the TA database or the task database.

- Load state.

The current state of the program, including instructors, courses, sections, and assignments, can be restored from a _le. As usual, this can be done by selecting File/Open.

- Save state.

The current state of the program, as described in the previous use case, is saved to a file. As usual, there should be two options: File/Save and File/Save as....

- View TAs.

The program displays all TAs, together with their fixed and variable data. This enables the user to check progress and to see which TAs are still short of work.

- View tasks.

The program displays all tasks. Each task is listed with the TA's name if a TA has been assigned; this field is blank if no TA has been assigned.

- View unassigned tasks.

The program displays all tasks for which no TA has been assigned.

- Suggest TA.

Assume that the program is displaying unassigned tasks, as in the previous use case. If the user selects a task, the program should suggest suitable TAs (that is, the TAs whose Experience field includes this task).

- Add assignment.

The user selects a TA and a task; the program assigns the task to the TA and checks for conflicts. Assume that the program is displaying a section and a list of suggested TAs, as in the previous use case. Clicking on a TA assigns the TA to that task.

- Delete assignment.

The user selects a task to which a TA has been assigned and deletes the TA.

- Report TAs.

The program generates a report of TAs and their workloads, sorted alphabetically by TAs' names.

- Report assignments.

The program generates a report of tasks. The report includes the TA assigned to each task, with blanks if no assignment has been made. Several options are provided:

(a) Sequence:

i. Sort by session, course, section, and task.

ii. Sort by course, term, and section, and task.

(b) Selection:

i. List assigned tasks only.

ii. List unassigned tasks only.

iii. List TAs only.

- Summaries.

The program provides summary data on request. Even better would be to display the summary data on the screen at all times.

(a) Number of TAs.

(b) Number of tasks.

(c) Number of tasks with a TA assigned.

4 Normal Use

The program should be designed so that the following operations (which will probably be typical of normal use) are easy and intuitive. Bracketed numbers refer to use cases.

- [3] Start the program and load a complete \state" (TAs and tasks).

- [7] View unassigned tasks.
- Repeat:
    - [8] Look at the suggested TAs for a particular task.
    - [9] Assign a TA to the task, noting con icts.
    - [10] (Occasionally) delete assignments.
- [4] Save the current state and exit.

APPENDIX B

"DO TA ASSIGNMENT" SCREENSHOT