SCHEDULE VIEWER:
A SCHEDULING TOOL FOR UBC OKANAGAN ADMINISTRATION

by

Jacob Orr

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

BACHELOR OF SCIENCE HONOURS

in

The Irving K. Barber School of Arts and Sciences

(Honours Computer Science Major Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Okanagan)

April 2009

ABSTRACT

Schedule Viewer is an add-in for Microsoft Office Outlook 2007 developed to assist administrative staff and faculty members in viewing a selected subset of academic schedules, which has not been possible until now. Information on times, professors and locations for classes has always been readily available online, but to make productive use of this data staff members have been forced to manually sift through spreadsheets and compile the data themselves. The Schedule Viewer application allows users to enter search criteria in order to select a group of time blocks (classes, labs, etc.) and have these time blocks displayed in a useful graphical way. Time blocks are displayed as appointments in an Outlook calendar which allows users to easily view the layout of academic schedules in relation to their specific query. In addition to viewing existing schedule information, users can create their own Outlook appointments which are then saved in the database for others to work with.

The Schedule Viewer application runs on a three-tier client-server architecture. The client portion of the system involves the Outlook add-in which runs locally on each user's machine. There are two components to the server architecture. The server application is written in Java and runs on a campus server. The server application communicates directly with each client and provides the link from the client to the database. The database is located on another server on campus and stores time block information including times, dates, and locations. All information stored in the database, which was created specifically for Schedule Viewer, is obtained from the UBC Student Services web site [4] using an HTML screen scraping process.

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGEMENTS

Foremost, I owe much gratitude and appreciation to Nicholas Blackwell, co-creator of Faculty Scheduler, the first version of the Schedule Viewer system. Nick and I spent many a late-night coding and debugging this program as our Advanced Database term project in 2008 and his contributions played a vital role in bringing the system to where it is today.

I would also like to thank Dr. Ramon Lawrence, my honours supervisor and Advanced Database professor. Without his help and encouragement I would not have taken the extra steps to achieve my honours status nor would I have gained this valuable experience.

1 INTRODUCTION

1.1 Motivation

Until now viewing and creating class schedules at UBC Okanagan has been unnecessarily difficult and tedious. The process required sifting through many separate spreadsheets and manually entering data multiple times. There was no easy way to check when a group of professors had free time for a meeting or to check for class conflicts within a program. Simple scheduling matters have taken far more time than necessary. Faculty and administrators have been requesting a solution to this problem for some time, yet nothing had been produced to meet these requirements until now.

1.2 Thesis Statement and Contributions

The Schedule Viewer system provides a much needed client application for UBC Okanagan administrators which gives them the ability to easily manage faculty schedules by providing intuitive controls in a visual interface. Administration at UBC Okanagan has been struggling with scheduling organization for some time and Schedule Viewer provides a solution for this problem. The client application is a Microsoft Outlook add-in, which makes use of the existing visual interface. Faculty schedule information is stored in a database and accessed by the client through a server application. Administrators will be able to view multiple schedules by importing time blocks based on specific criteria and use Outlook to view these schedules in various levels of detail. This will greatly simplify schedule-planning at UBC Okanagan, making it easier and more efficient.

2 BACKGROUND

2.1 Terms

DOM – The Document Object Model is a platform and language independent standard for modeling HTML or XML documents.

Eclipse – Eclipse, as used in this project, is a Java programming language development environment. It is computer software used to write the Java code for the server portion of Schedule Viewer.

Garbage Collection – In certain programming languages 'garbage collection' is an automatic memory management method used to reclaim memory allocations that are no longer in use.

Java – "Java is a programming language originally…released in 1995 as a core component of Sun Microsystems' Java Platform." [6]

JDBC – "Java Database Connectivity is an API for the Java programming language and defines how a client may access a database." [5]

Microsoft SQL Server – "Microsoft SQL Server is a relational database management system produced by Microsoft." [7]

SVN – An abbreviation for 'Subversion', SVN is an open source version control system. In other words, SVN manages a set of directories and how they change over time. It is a useful tool for project backup.

2.2 Faculty Scheduler

The initial version of Schedule Viewer was developed as a term project for Computer Science 404, Database Management Systems II. Nicholas Blackwell and I created the program. It was with this version that a large portion of the server framework was created as well as a basic client application.

2.3 Course Data

The course information displayed to users of Schedule Viewer is retrieved from a database created specifically for the Schedule Viewer program that is populated with data from the UBC Student Services web page [4]. This page is periodically updated with new course information when updates are made or a new session is created. The screen scraper included in the Schedule Viewer system can be used to insert any new information from the web into the Schedule Viewer database.

2.4 Microsoft Outlook Interface

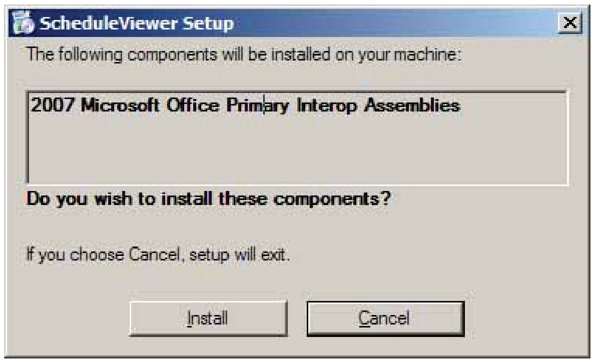
Schedule Viewer makes use of an existing, well developed user interface to provide a graphical experience for users. Using Visual Studios Tools for Office allows the creation of Outlook add-ins which are custom applications that can take advantage of, and customize, the Outlook user interface. Outlook is a widely used email client used by the majority of UBC Okanagan faculty and administrators therefore its use for Schedule Viewer was an easy choice. Specifically, Schedule Viewer makes use of Outlook's calendar functions. Classes are created in Outlook as calendar appointments, which can be viewed in various levels of detail allowing many complex scheduling issues to be resolved in a clean, simple way.

3 CLIENT APPLICATION – USER MANUAL

The client application for the Schedule Viewer system is the Outlook add-in that the user interacts with.

3.1 Installation

Run `setup.exe` from the Schedule Viewer installation directory. When prompted, choose to install the 2007 Microsoft Office Primary Interop Assemblies:



ScheduleViewer Setup

The following components will be installed on your machine:

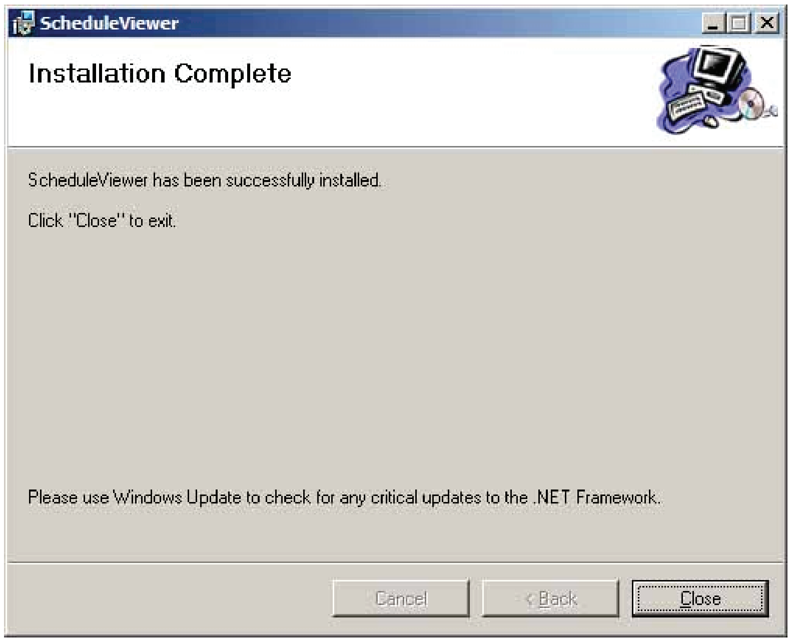
2007 Microsoft Office Primary Interop Assemblies

Do you wish to install these components?

If you choose Cancel, setup will exit.

Figure 1: Installing Microsoft Primary Interop Assemblies

From here, follow the installation instructions through several screens. Upon successful installation the dialog box seen in Figure 2 will be displayed.



ScheduleViewer

Installation Complete

ScheduleViewer has been successfully installed.

Click "Close" to exit.

Please use Windows Update to check for any critical updates to the .NET Framework.

Figure 2: Confirmation of Successful Schedule Viewer Installation

3.2 Startup

Upon Outlook startup, the Schedule Viewer client application attempts to contact the Schedule Viewer server. If a connection cannot be established an error message is displayed and the add-in does nothing until the `Schedule Viewer` button on the Outlook toolbar is pressed. If the button is pressed the client will again attempt to connect to the server. Once a connection is established the Schedule Viewer `Login` form (Figure 3) is displayed.

3.3 Login

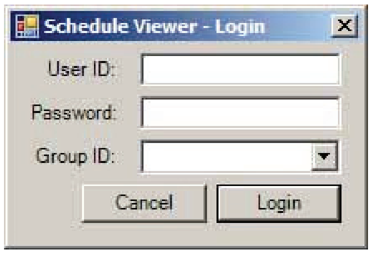


Figure 3: Schedule Viewer Login Form

In order to use Schedule Viewer a user must login with a matching user ID, password and group ID. The user can either enter an existing group ID or select one from the drop down list. If the `Group ID` drop-down arrow is clicked a request will be sent to the server and the `Group ID` combo box will be filled with all group IDs contained in the database. In order for a successful login to occur, the user ID must have permission to access the selected group ID.

Once confirmation is received from the server that a login was successful the login form is closed and a calendar is created in Outlook for use by the Schedule Viewer application. If the Schedule Viewer calendar already exists creation does not occur, instead the existing calendar's contents are scanned and used by the application. This calendar is used for all Schedule Viewer operations.

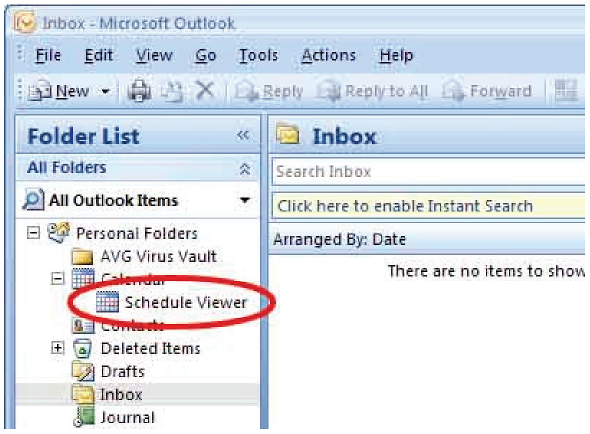


Figure 4: The Schedule Viewer Calendar Item

3.4 Main Form

Once the user is logged in, clicking the `Schedule Viewer` button on the Outlook toolbar (Figure 5) will display a progress bar (Figure 6).

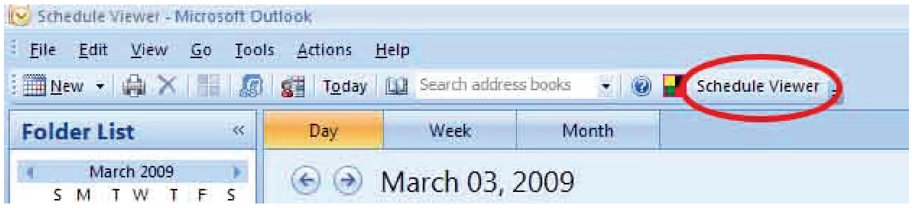


Figure 5: Schedule Viewer Toolbar Button

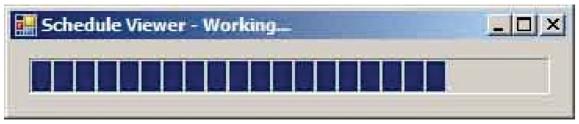
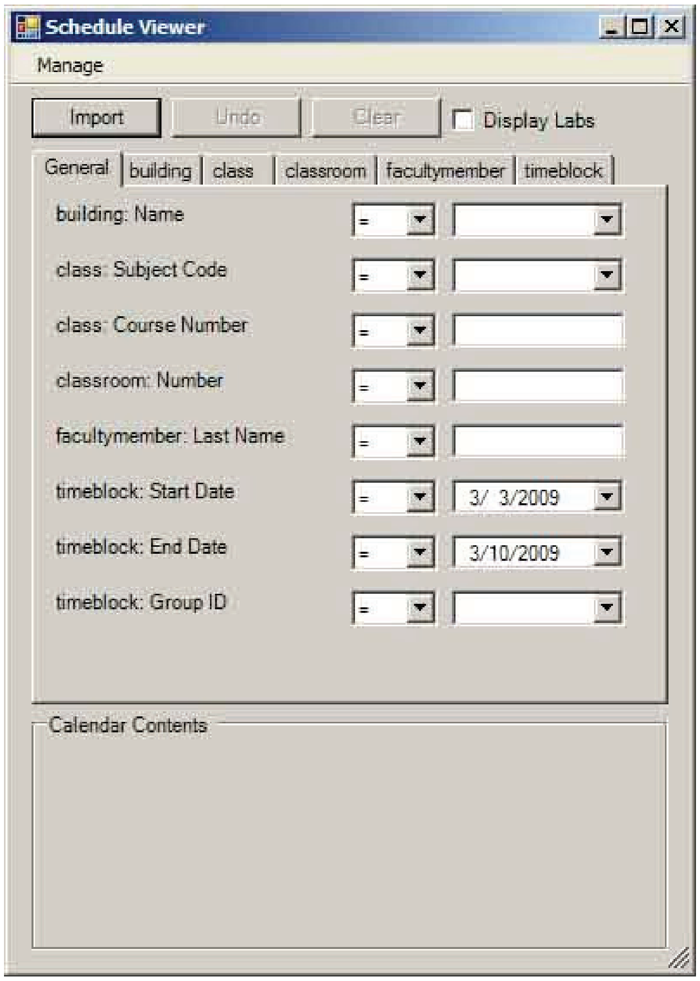


Figure 6: Schedule Viewer Progress Bar

Once Schedule Viewer's main form (Figure 7) is constructed with information from the server, the progress form closes and the main form is displayed.



building: Name	=	
class: Subject Code	=	
class: Course Number	=	
classroom: Number	=	
facultymember: Last Name	=	
timeblock: Start Date	=	3/ 3/2009
timeblock: End Date	=	3/10/2009
timeblock: Group ID	=	

Figure 7: Schedule Viewer Main Form

If the main form is closed, clicking on the `Schedule Viewer` button again will reopen it. The form is displayed immediately. The progress bar is only displayed the first time while the main form is being constructed.

3.5 Tab Layout

In the center of the main form is a tab control (Figure 8) containing one tab for each table in the Schedule Viewer database as well as one `General` tab.

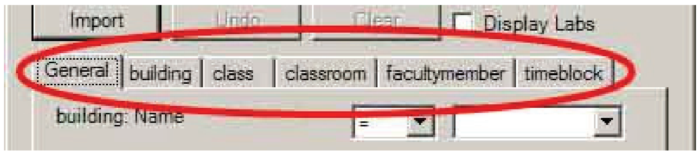
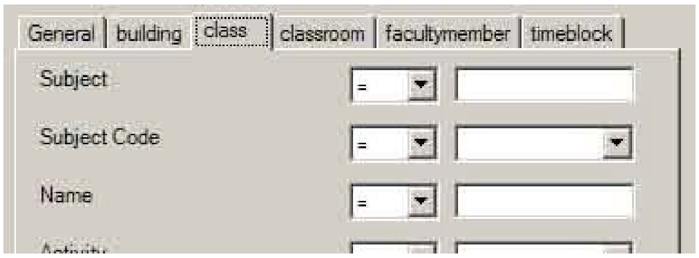


Figure 8: Main Form Tabs

Each table tab contains a list of fields from that table and the general tab contains a list of commonly used fields. Figure 9 illustrates an example of the `class` table's tab.

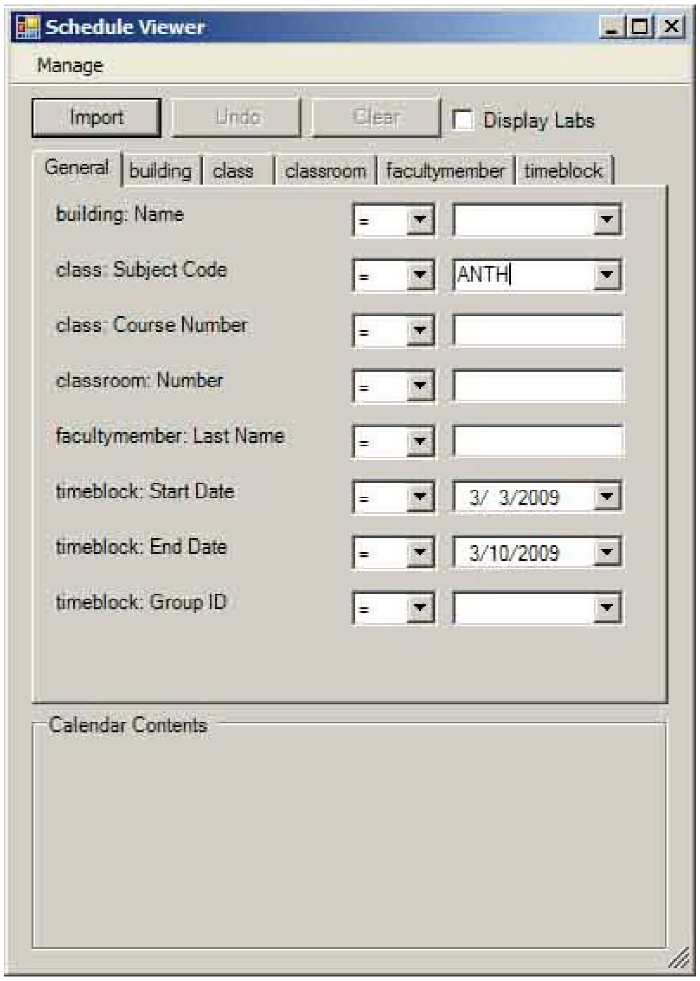


Subject =
Subject Code =
Name =

Figure 9: The Class Tab

3.6 Import

Prior to performing an import a user may enter information into fields from any tabs to limit their search of courses and appointments. In Figure 10, the user performs a search for any classes with the subject code "`ANTH`" occurring within the specified time range.

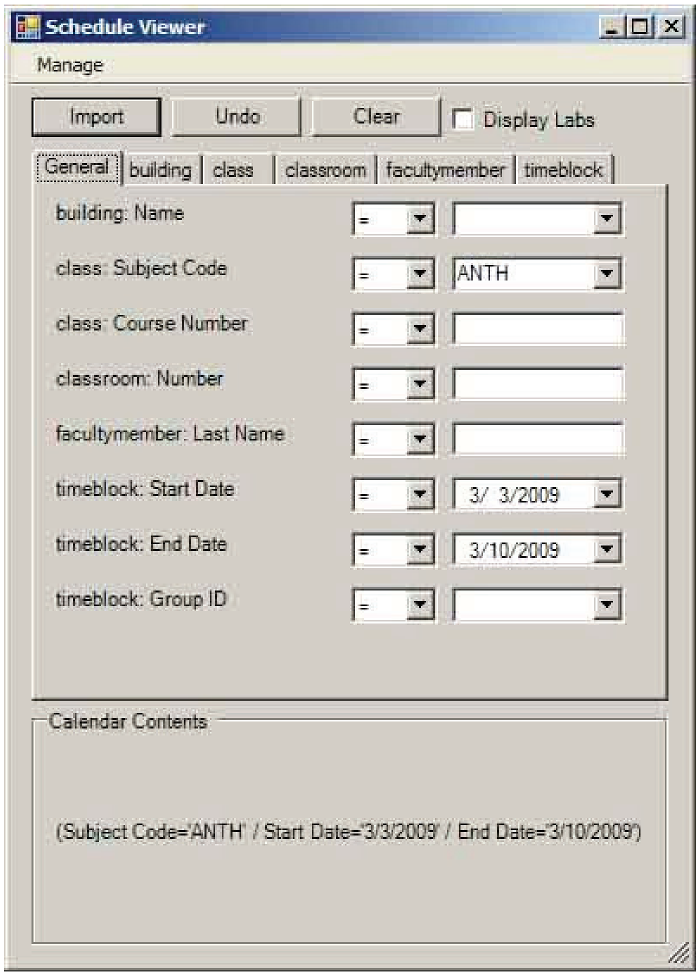


building: Name =
class: Subject Code = ANTH
class: Course Number =
classroom: Number =
facultymember: Last Name =
timeblock: Start Date = 3/ 3/2009
timeblock: End Date = 3/10/2009
timeblock: Group ID =

Figure 10: Course Search for Subject Code "ANTH"

The date range specified during import is required and will return any one-time appointments occurring within the time period as well as any recurring appointments during the time period.

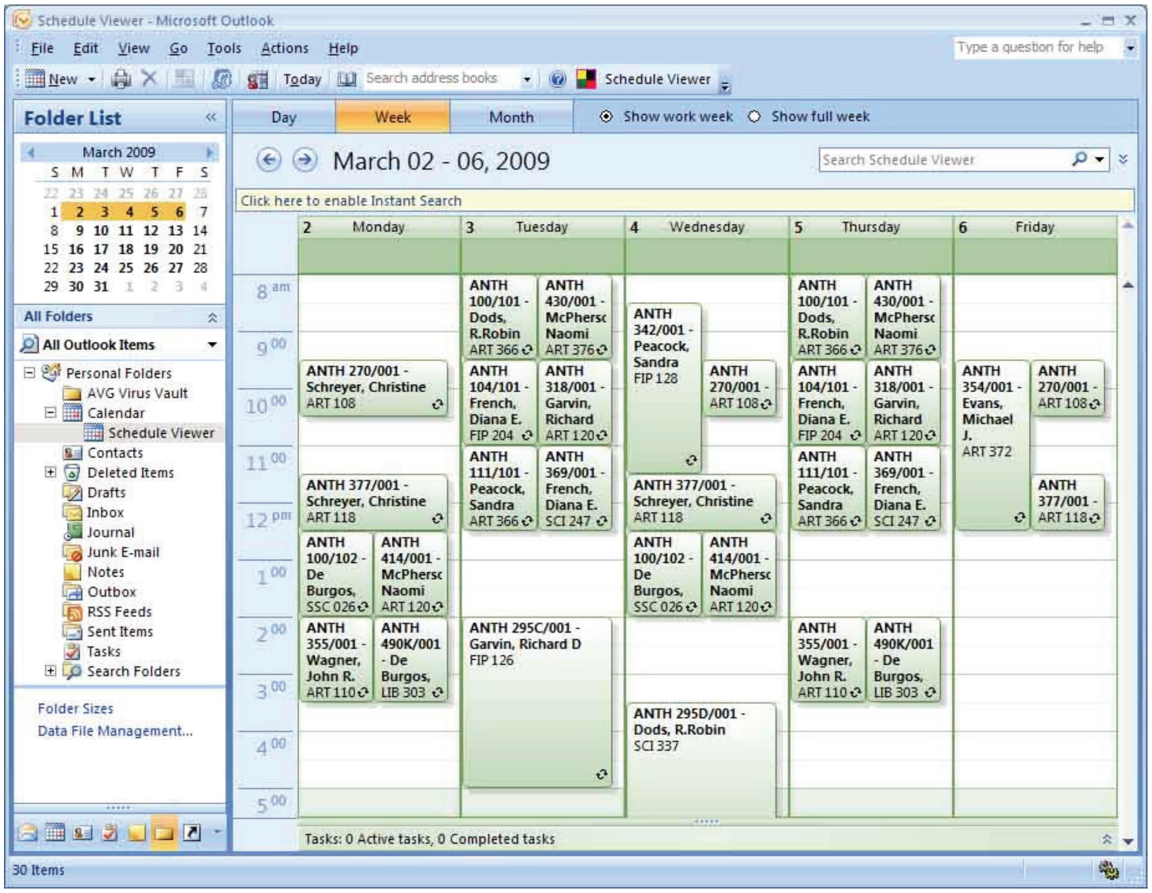
Once the `Import` button is pressed, a query is sent to the server containing search requirements from all filled fields in all tabs. As each result of the search is received by the application an Outlook appointment item is created with information contained in the result and the appointment is added to the Schedule Viewer calendar. A notification is displayed when the operation has been completed and the `Calendar Contents` box at the bottom of the main form is updated. The Schedule Viewer calendar resulting from the import in Figure 11 can be seen in Figure 12.



building: Name	=	
class: Subject Code	=	ANTH
class: Course Number	=	
classroom: Number	=	
facultymember: Last Name	=	
timeblock: Start Date	=	3/ 3/2009
timeblock: End Date	=	3/10/2009
timeblock: Group ID	=	

(Subject Code='ANTH' / Start Date='3/3/2009' / End Date='3/10/2009')

Figure 11: Main Form after Successful Import



March 02 - 06, 2009

2 Monday
ANTH 270/001 - Schreyer, Christine ART 108
ANTH 377/001 - Schreyer, Christine ART 118
ANTH 100/102 - De Burgos, SSC 026
ANTH 414/001 - McPhersc Naomi ART 120
ANTH 355/001 - Wagner, John R. ART 110
ANTH 490K/001 - De Burgos, LIB 303

3 Tuesday
ANTH 100/101 - Dods, R.Robin ART 366
ANTH 430/001 - McPhersc Naomi ART 376
ANTH 104/101 - French, Diana E. FIP 204
ANTH 318/001 - Garvin, Richard ART 120
ANTH 111/101 - Peacock, Sandra ART 366
ANTH 369/001 - French, Diana E. SCI 247
ANTH 295C/001 - Garvin, Richard D FIP 126

4 Wednesday
ANTH 342/001 - Peacock, Sandra FIP 128
ANTH 270/001 - ART 108
ANTH 377/001 - Schreyer, Christine ART 118
ANTH 100/102 - De Burgos, SSC 026
ANTH 414/001 - McPhersc Naomi ART 120
ANTH 295D/001 - Dods, R.Robin SCI 337

5 Thursday
ANTH 100/101 - Dods, R.Robin ART 366
ANTH 430/001 - McPhersc Naomi ART 376
ANTH 104/101 - French, Diana E. FIP 204
ANTH 318/001 - Garvin, Richard ART 120
ANTH 111/101 - Peacock, Sandra ART 366
ANTH 369/001 - French, Diana E. SCI 247
ANTH 355/001 - Wagner, John R. ART 110
ANTH 490K/001 - De Burgos, LIB 303

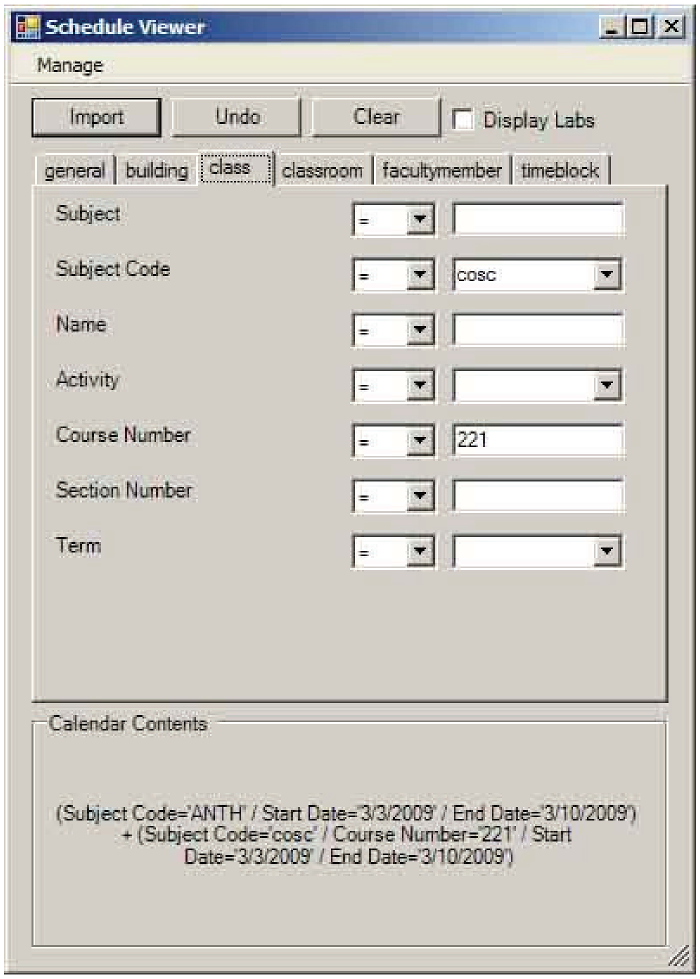
6 Friday
ANTH 354/001 - Evans, Michael J. ART 372
ANTH 270/001 - ART 108
ANTH 377/001 - ART 118

Figure 12: Calendar after Successful Import

Users can search for multiple subject codes at the same time. Entering multiple subject codes, separated by commas, will return search results with all of the listed subject codes providing the results match the rest of the fields. For example, if `Subject Code` is set to "`ANTH,COSC`", all anthropology and computer science courses will be returned. If in addition to `Subject Code` being set to "`ANTH,COSC`" `Course Number` is set to "`430`", a result for ANTH 430 will be returned, but no computer science courses will be returned as there is no COSC 430 course.

3.6.1 Multiple Imports

When multiple imports are performed the results from each import are added to the same Schedule Viewer calendar, resulting in one calendar containing the results of all of the imports. If an import returns a duplicate result of an existing appointment, it is not added. Figure 13 illustrates the main form after another import with information entered in the `class` tab. Note the updated `Calendar Contents` box.



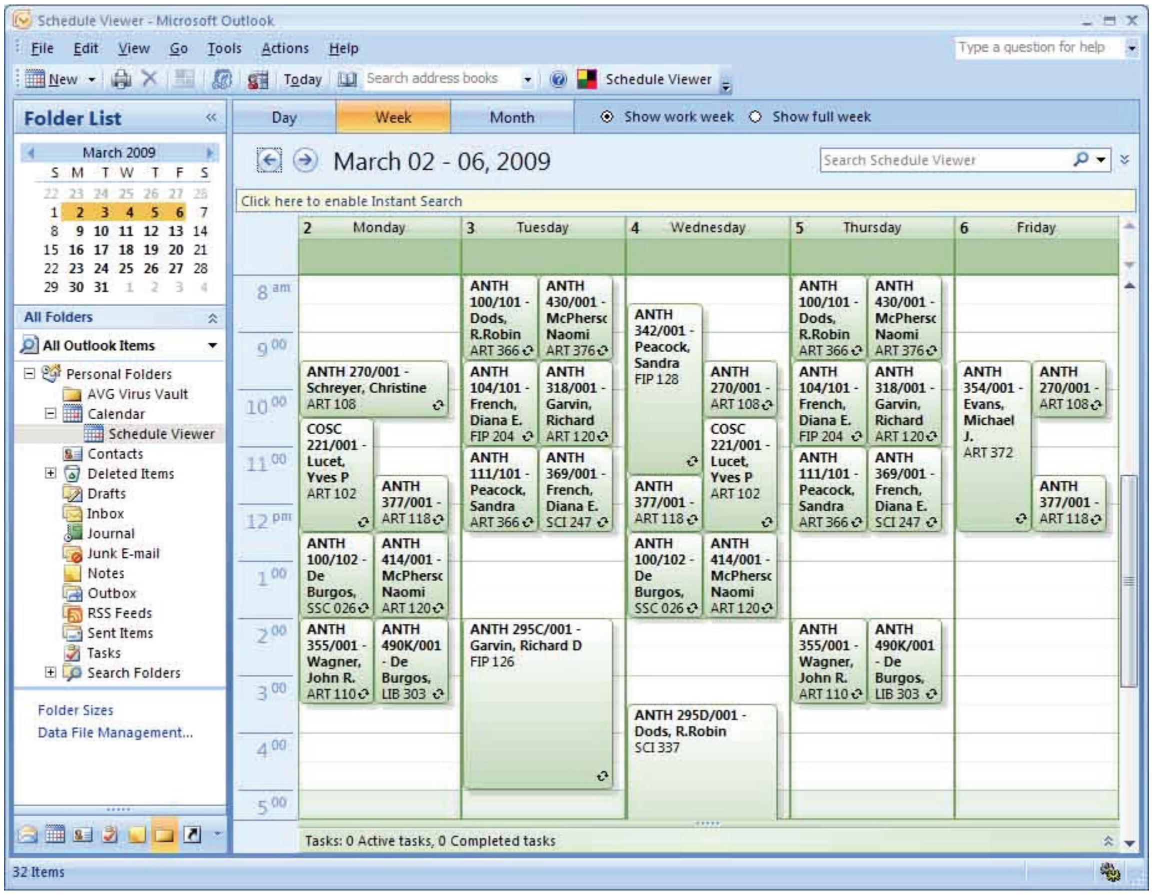
Subject	=	
Subject Code	=	cosc
Name	=	
Activity	=	
Course Number	=	221
Section Number	=	
Term	=	

Calendar Contents

(Subject Code='ANTH' / Start Date='3/3/2009' / End Date='3/10/2009') + (Subject Code='cosc' / Course Number='221' / Start Date='3/3/2009' / End Date='3/10/2009')

Figure 13: Main Form Additional Import

This time the search requirements were in a table specific tab, class, to show that they work the same as entering them on the general tab. The Outlook calendar resulting from the multiple imports can be seen in Figure 14.



March 02 - 06, 2009

2 Monday
- ANTH 270/001 - Schreyer, Christine ART 108
- COSC 221/001 - Lucet, Yves P ART 102
- ANTH 377/001 - ART 118
- ANTH 100/102 - De Burgos, SSC 026
- ANTH 414/001 - McPhersc Naomi ART 120
- ANTH 355/001 - Wagner, John R. ART 110
- ANTH 490K/001 - De Burgos, LIB 303

3 Tuesday
- ANTH 100/101 - Dods, R.Robin ART 366
- ANTH 430/001 - McPhersc Naomi ART 376
- ANTH 104/101 - French, Diana E. FIP 204
- ANTH 318/001 - Garvin, Richard ART 120
- ANTH 111/101 - Peacock, Sandra ART 366
- ANTH 369/001 - French, Diana E. SCI 247
- ANTH 295C/001 - Garvin, Richard D FIP 126

4 Wednesday
- ANTH 342/001 - Peacock, Sandra FIP 128
- ANTH 270/001 - ART 108
- COSC 221/001 - Lucet, Yves P ART 102
- ANTH 377/001 - ART 118
- ANTH 100/102 - De Burgos, SSC 026
- ANTH 414/001 - McPhersc Naomi ART 120
- ANTH 295D/001 - Dods, R.Robin SCI 337

5 Thursday
- ANTH 100/101 - Dods, R.Robin ART 366
- ANTH 430/001 - McPhersc Naomi ART 376
- ANTH 104/101 - French, Diana E. FIP 204
- ANTH 318/001 - Garvin, Richard ART 120
- ANTH 111/101 - Peacock, Sandra ART 366
- ANTH 369/001 - French, Diana E. SCI 247
- ANTH 355/001 - Wagner, John R. ART 110
- ANTH 490K/001 - De Burgos, LIB 303

6 Friday
- ANTH 354/001 - Evans, Michael J. ART 372
- ANTH 270/001 - ART 108
- ANTH 377/001 - ART 118

Tasks: 0 Active tasks, 0 Completed tasks

Figure 14: Multiple Import Calendar

3.6.2 Display Labs

Labs are usually taught by a TA, not the professor who teaches the class. For this reason they are not always required in a schedule if, for example, a schedule for a specific professor is desired. By default labs are not returned from imports. The `Display Labs` checkbox can be checked in order to include labs in import search results.

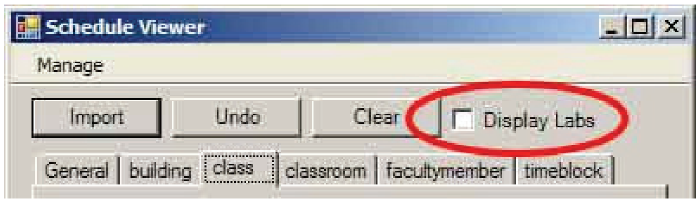


Figure 15: The Display Labs Checkbox

3.7 Undo

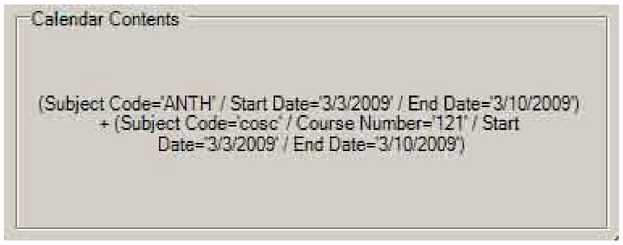
Once a schedule has been successfully imported into the Schedule Viewer calendar, the `Undo` button becomes active. The `Undo` button will delete all of the appointments from the most recent import and update the `Calendar Contents` box. This may take several minutes if the most recent import returned a large number of appointments. Multiple undo operations can be performed until there is no imported data in the Schedule Viewer calendar.

3.8 Clear

The `Clear` button becomes active following a successful import to the Schedule Viewer calendar. Clicking the `Clear` button will delete the Schedule Viewer calendar including any appointments it contained and create a new calendar to be used by the Schedule Viewer application. Caution should be used when performing a clear operation because all appointments in the Schedule Viewer calendar will be lost, including user created appointments. If these appointments were created when the application was not yet connected to the server, they will not be saved in the database and will be permanently lost.

3.9 Calendar Contents Box

The `Calendar Contents` box contains the search restrictions entered by the user for each of the imports that are currently in the Schedule Viewer calendar.


Calendar Contents

(Subject Code='ANTH' / Start Date='3/3/2009' / End Date='3/10/2009') + (Subject Code='cosc' / Course Number='121' / Start Date='3/3/2009' / End Date='3/10/2009')

Figure 16: Calendar Contents Box after Two Imports

Each import is surrounded by parentheses and the search restrictions for each of the imports are separated by forward slashes. Different imports are separated by plus signs.

3.10 New Group

Any user can create new groups. Users can be assigned to these groups by the group's creator or by an administrator. Any user who is part of a group will have permission to view appointments from that group and to create new appointments belonging to that group. In order to create a new group the user must first select the `New Group…` menu item (Figure 17) from the `Manage` menu of the main form. The `Add Group` dialog box will be displayed (Figure 18).

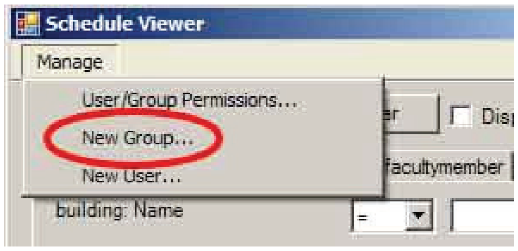


Figure 17: New Group Menu Item

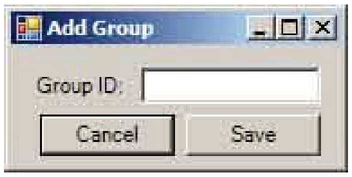


Figure 18: Add Group Form

The only restriction for group ID is that it must be unique and the field must not be left blank. If a user attempts to create a group with a name that is already in use a warning will be displayed and no group will be created. Once a unique name has been entered and the save button has been clicked, the group will be created and a confirmation box will be displayed.

3.11 Web Group

All appointments created during the screen scraping process are part of their own group. This is called the "`web`" group and all users have viewing access to this group. The web group is read-only, so no users can log in using the web group and no appointments can be added or changed as the web group.

3.12 New User

In addition to group creation, an administrator user has the option to create a new user. An administrator can access this functionality by selecting the `New User…` option (Figure 19) from the `Manage` menu on the main form; the `New User` form will then be displayed (Figure 20).

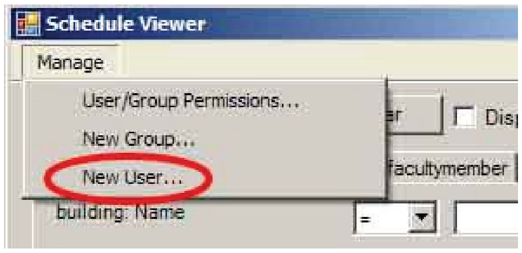


Figure 19: New User Menu Item

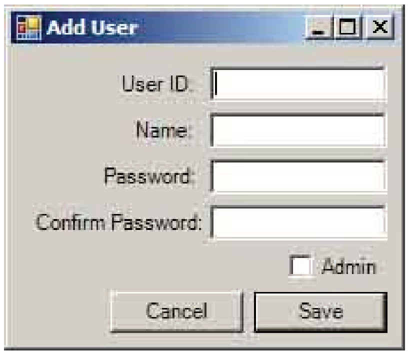


Figure 20: Add User Form

Similarly to group ID, the user ID cannot be left blank and must be unique. On this form the administrator enters the user ID and, optionally, the user's name and password. If the `Admin` check box is checked the user created will also be an administrator. When a user is created a group is also created with a group ID matching the new user ID. Therefore the new user ID cannot already be used as a user ID or a group ID. The new group's owner is set to the new user. The new user is then given access to the new group as well as to the web group.

3.13 User/Group Permissions

Any user can access the `User/Group Permissions` form where selected permissions can be modified. The permissions form allows a user to add users to or remove users from their groups. Administrators have access to all groups. Selecting the `User/Group Permissions…` option (Figure 21) from the `Manage` menu of the main form, a user will be shown the `User/Group Permissions` form (Figure 22).

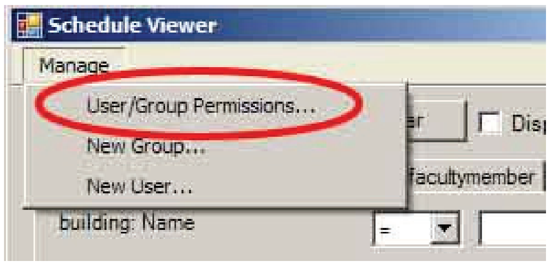


Figure 21: User/Group Permissions Menu Item

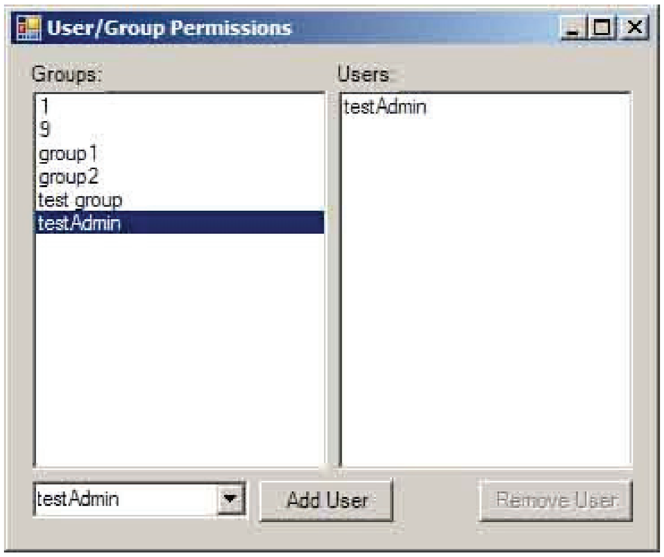


Figure 22: User/Group Permissions Form

On this form, for a normal user, the `Groups` list box will contain a list of all groups owned by the user. For an administrator it will contain all groups. When a group is selected all of its members will be displayed in the `Users` box. If any of the users are selected the `Remove User` button will become activated and if clicked will remove the selected user from the selected group.

The drop down box in the bottom left of the permissions form contains a list of all users. When the add user button is clicked the user in the drop down box will be added to the selected group and displayed in the users list box.

3.14 Add Appointment

The Schedule Viewer calendar functions as a normal Outlook Calendar and users are able to add or change appointments within it. As long as a user is logged in to the Schedule Viewer system, any appointments created in the Schedule Viewer calendar will be uploaded to the server and marked as belonging to the group ID of the user currently logged in. This appointment will then be available to import by any users in the same group.

Simple recurring appointments are supported by Schedule Viewer and if created in the Schedule Viewer calendar, will be saved correctly. A simple recurring appointment means that it is a weekly recurrence and that an end date is specified. If the appointment does not conform to these specifications it is not saved or sent to the server. One-time appointments have no such restrictions and will be saved properly.

3.15 Change Appointment

If a user changes an appointment in the Schedule Viewer calendar, the modifications will be sent to the server as long as the current user is logged in with the group ID of the appointment being changed. If a user logged in with a different group ID attempts to change an appointment, the changes will be saved in Outlook, but will not be sent to the server. On any future imports the changes will not be seen. The same restrictions for recurring appointments mentioned above in the Add Appointment Section apply to appointment changes.

4 CLIENT APPLICATION – TECHNICAL DETAILS

The Schedule Viewer add-in for Microsoft Outlook was written in Microsoft Visual C# using Microsoft Visual Studio 2005.

4.1 System Architecture

The Schedule Viewer system uses a three-tier architecture with clients, a server and a database. In this case all three are located on separate machines with the database on `cssql.ok.ubc.ca`, the server on `cs-suse-4.ok.ubc.ca` and the clients on any machines employed by users. Figure 23 illustrates the architecture of the Schedule Viewer system.

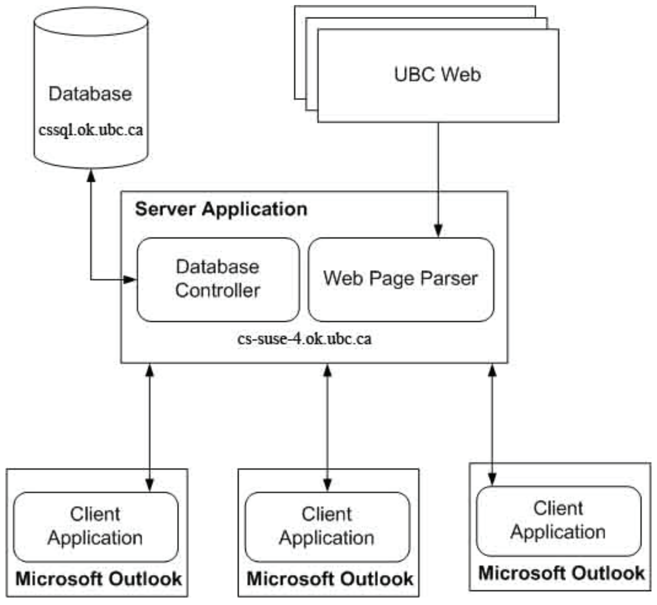


Figure 23: Illustration of Schedule Viewer's Three-Tier, Client-Server Architecture

The UBC Web and Web Page Parser items are separate from normal system execution. Both come into effect during the screen scraping process used to fill the database with class information. This process will be discussed in Section 7.

4.2 Server Communication

Communication between the Schedule Viewer client and server occurs synchronously over a TCP/IP connection. This means that when the Schedule Viewer client sends data to the server it will wait for a response before sending anything further. As an example, consider a client sending a request for all classes with subject code “`ANTH`”. The server determines a list of classes that match the criteria and it sends each class as separate messages to the client. The client receives each one of these responses separately and uses them to create appointments in the Schedule Viewer calendar. The client will receive an “`END`” response when the server is finished sending messages. Upon receiving the “`END`” response the client readies itself to send further requests. Figure 24 illustrates this communication process.

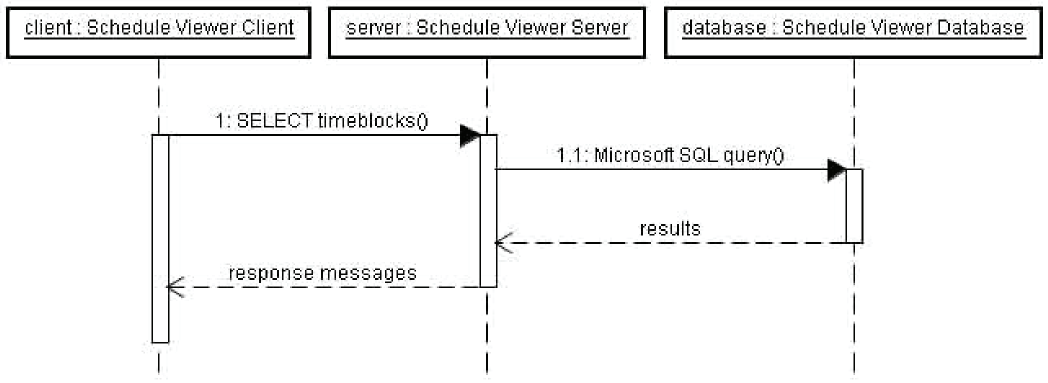

client : Schedule Viewer Client
server : Schedule Viewer Server
database : Schedule Viewer Database
1: SELECT timeblocks()
1.1: Microsoft SQL query()
results
response messages

Figure 24: Schedule Viewer System Communication Sequence Diagram

4.3 Message Syntax

All communication between client and server follows a strict syntax. This allows the meaning of each request and response to be determined. The sub-points below describe the syntax used for all client requests to the server and their expected responses. Any italicized words in the request are placeholders and only meant to illustrate where actual values would be placed.

4.3.1 Login Attempt

`LOGIN` *`userId password groupId`*

The login attempt request will return one of four responses to the client. If either the username does not exist or the password is incorrect "`BAD_LOGIN`" will be returned. If the user ID does not have access to the group ID specified "`PERMISSIONS_CONFLICT`" will be returned. If the login is successful the server will return either "`U`" or "`A`" depending on whether the user ID is a normal user or an administrator, respectively.

4.3.2 Map Tables

`MAP tables`

The `MAP tables` request is sent to the server the first time the client opens the Schedule Viewer main form. It returns each field from each table in a number of separate messages. Returns have the form: `tableName fieldName fieldFriendlyName fieldType`. The field friendly name is the text users will see when looking at data regarding that field. Field type is the Microsoft SQL data type of that field.

4.3.3 New Group

`NEW group` *`userId groupId`*

"`NAME_CONFLICT`" is returned if the group ID is already in use. If the group creation is successful "`OK`" is returned and the new group's owner ID will be set to the given user ID.

4.3.4 New User

`NEW user` *`userId password name type`*

For a `NEW user` request, if the user ID already exists as a user ID or a group ID, "`NAME_CONFLICT`" is returned. If the user creation is successful, "`OK`" is returned. The type field will either be "`U`" if the new user is a normal user or "`A`" if the new user is an administrator.

4.3.5 Get Groups

`GET groups` *`userId userType`*

Each message returned for a `GET groups` request will have the form: `groupId`. If the user type option is "`U`", for a normal user, then a message will be returned to the client for each group that belongs to the given user ID. For an administrator, user type "`A`", all groups in the database will be returned.

4.3.6 Get Users

`GET users`

A `GET users` request will return messages of the form: `userId`. One message will be returned for each user ID in the database.

4.3.7 Get Permissions

`GET permissions` *`userId userType`*

The `GET permissions` request is used to get information about which users have access to which groups. Normal users will receive messages for each user in the groups they own and an administrator will receive messages for users in all groups. Each return message takes the form: `groupId userId` where the user ID has access to the group ID.

4.3.8 Grant Permissions

`GRANT permissions` *`userId groupId`*

The `GRANT permissions` operation is used to give group access to a user. The server will return "`USER_DNE`" if the user ID specified does not exist in the database. Successful execution of this operation will return "`OK`" and the given user ID will now be able to log in as the given group ID. The given user ID will now be able to log in, view and create appointments as the given group ID.

4.3.9 Revoke Permissions

`REVOKE permissions` *`userId groupId`*

Upon a successful `REVOKE permissions` request the given user ID will be removed from the given group and "`OK`" will be returned to the client. The given user ID will no longer be able to use the given group ID.

4.3.10 Add One-time Appointment

`ADD timeblocks` *`subject location body startTime endTime groupId date(with start time)`*

An `ADD timeblocks` request will return the database ID of the new record created in the timeblock table. This ID is stored in the appointment created in the Schedule Viewer calendar. The subject, location and body fields match the same fields in an Outlook appointment.

4.3.11 Add Recurring Appointment

`ADD recurrence` *`subject location body startTime endTime groupId occurrenceStart(date) occurrenceEnd(date) occurrencePattern`*

The response for the `ADD recurrence` request is also the ID of the new record. An example of an occurrence pattern is “`MON WED FRI`” where the recurrence occurs every Monday, Wednesday and Friday. The other options are “`SUN`, `TUE`, `THU`” and “`SAT`”.

4.3.12 Update One-time Appointment

`UPDATE timeblocks` *`timeblockId subject location body startTime endtime groupId date(with start time)`*

A successful `UPDATE timeblocks` will return “`OK`”.

4.3.13 Update Recurring Appointment

`UPDATE recurrence` *`timeblockId subject location body startTime endTime groupId occurrenceStart(date) occurrenceEnd(date) occurrencePattern`*

An `UPDATE recurrence` request will return “`OK`”on a successful update. A description of the occurrence pattern field can be seen in Section 4.3.11.

4.3.14 Select Time Blocks

`SELECT timeblocks WHERE` *`userId fieldName=literal fieldName2=literal2`* …

A `SELECT timeblocks` query will return multiple times. Only timeblocks for groups the given user ID has access to will be returned. One message will be sent for each record matching the given criteria. Each return will have the form: `timeblockId startTime endTime occurrencePattern occurrenceStart occurrenceEnd subject location body`. The occurrence pattern field is the same as discussed in Section 4.3.11.

4.3.15 Get Details

`GET details` *`fieldname`*

A `GET details` request will return one message for each unique value of the requested field. Each return will have the form: `fieldValue`.

4.4 Client-Server Communication Examples

The following section demonstrates how some of the client-server requests and responses are used together. Examples below are taken from server output.

4.4.1 Login Example

```
Client: sent 'LOGIN test test test%group'
Executing: SELECT uType FROM [user] WHERE uId='test' uPassword='test'
Executing: SELECT * FROM [user_group] WHERE uId='test' gId='test group'
Sent Client: U
```

This string of communication shows a client loggingin with user ID "`test`," password "`test`" and group ID "`test group`." The server checks that the user ID and password match and that the user has access to the requested group ID. Both checks were successful so "`U`," for normal user, is returned to the client.

4.4.2 Building Main Form Example

```
Client: sent 'MAP tables'
Sent Client: building   bName           Name            varchar
Sent Client: building   bAddress        Address         varchar
Sent Client: class      cSubject        Subject         varchar
Sent Client: class      cSubjectCode    Subject%Code    varchar
…
Client: received 23 results
Client: received successfully
```

In order to build Schedule Viewer's main form the client sends a `MAP tables` request to the server. Upon receiving the request the server sends back one message for each field in each table. A truncated version of the communication is shown above.

4.4.3 Select Time Blocks Example

```
Client:    sent    'SELECT    timeblocks    WHERE    tbOccurrenceEnd='3/27/2009'
      tbOccurrenceStart='3/20/2009' cSubjectCode='anth''

Executing:   SELECT   tbId,   tbStartTime,   tbEndTime,   tbOccurrencePattern,
      tbOccurrenceStart,  tbOccurrenceEnd,  tbSubject,  tbLocation,  tbBody,
      tbGroupId    FROM    class,    timeblock    WHERE    cTimeBlock=tbId    AND
      ((tbOccurrenceStart>='3/20/2009'  AND  tbOccurrenceEnd<='3/27/2009')  OR
      (tbOccurrenceStart>='3/20/2009'  AND  tbOccurrenceStart<='3/27/2009')  OR
      (tbOccurrenceStart<='3/20/2009'  AND  tbOccurrenceEnd>='3/27/2009')  OR
      (tbOccurrenceEnd>='3/20/2009'  AND  tbOccurrenceEnd<='3/27/2009'))  AND
      cSubjectCode='anth'

Sent Client: 3 8:00 9:30 Tue%Thu 2009-01-05%00:00:00.0 2009-04-08%23:59:59.0
      ANTH%100/101%-%Dods,%R.Robin           ART%366           Anthropology%-
      %Lecture$Introduction%to%Cultural%Anthropology$ANTH%100/101$ART%366$Dod
      s,%R.Robin web
…

Client: received 17 results
Client: received successfully
```

The `SELECT timeblocks` command is the backbone of the Schedule Viewer system. The above communication exchange illustrates a simple request for classes with `Subject Code` equal to “`anth`” which occur during the specified date range. The executed query expands the date range quite dramatically to take care of time period conversions between client and server. The date range issue is discussed in Section 5.6.3. One of the 17 results is displayed in this example. The rest share a similar form with varying data.

4.5 Escape Characters

All communication between server and client takes the form of messages made up of individual fields separated by spaces. This allows the server and client applications to determine the contents of the messages reliably. In order to avoid unwanted spaces in a message, any spaces in the fields themselves are replaced with the percent character. For example, the user ID “`john smith`” would become “`john%smith`”. This allows the applications to send messages with the only spaces being the ones separating the fields. Upon receiving a message the value of the fields is determined by replacing percent signs with spaces. If any percent signs already exist in the fields they are escaped with a backslash so that they are not converted into spaces in the translation. An example of this is “`100%`” being converted into “`100\%`”. When a message is received the escaped characters are replaced with normal percent signs.

Another issue with web communication is that new line characters cannot be directly sent in messages. To deal with this the Schedule Viewer system uses a similar escape character method. Just like spaces, new lines have their own escape character which takes the form of a dollar sign. The escaping method is the same as described above for spaces.

4.6 Storing Persistent Data

When a user opens Outlook and logs into Schedule Viewer, the client application begins to run. The application runs until termination upon Outlook shutdown. The next time the user starts Outlook, they will still have access to the same Schedule Viewer calendar unless it was cleared or deleted before the previous Outlook shutdown. The Schedule Viewer calendar will still contain any imported or user created appointments from the previous session. When Schedule Viewer starts with an existing calendar it needs some way to determine what the data in the calendar means in order for the application to work correctly.

To maintain data consistency between Outlook and Schedule Viewer persistent storage was necessary. In order to avoid unnecessary complexity for the user, Schedule Viewer makes use of existing data fields in Outlook to store the time block ID and the group ID of appointments as well as the calendar contents string for the existing calendar. The time block ID and group ID for each appointment is stored in the `mileage` field of the Outlook appointment object. This field is otherwise unused and not visible to the user. The `description` text field of the Outlook calendar object is used for the calendar contents string. On a successful login these components of existing data are read by the Schedule Viewer application and stored as application data for the session.

As a note, the calendar contents of an existing calendar are treated as one single import even if they originally consisted of multiple imports. For this reason, when the existing contents are the last set of imports, clicking the undo button will remove everything from the calendar in one operation. Any additional imports will be treated normally.

5 SERVER APPLICATION

5.1 Installation

The Schedule Viewer server application consists of two parts. The first is the `ScheduleViewer.jar` file which is a Java Archive file that is an executable version of the Schedule Viewer server code. `ServerAccess.jar` is the second part of the server application and is a simple command line program which provides access to the server program while it is running in the background. In order to install the Schedule Viewer server application the two files mentioned above must be copied into a directory on the server machine and the machine itself must have a port open for clients to connect to. Currently, the server application is running on `cs-suse-4.ok.ubc.ca` on port `4449`.

5.2 Startup and Shutdown

Of the two files mentioned above, only `ScheduleViewer.jar` is needed to start the server application. Running the following string of commands will result in a running Schedule Viewer server.

```
% java -jar ScheduleViewer.jar
% <ctrl>+z
% bg
```

The first of these commands executes the application in a Java Virtual Machine. If this command is executed with the `-withinput` flag it will start the server version which accepts input from and prints output to the console. To run the server in the background the `-withinput` flag cannot be included. Next, the application is stopped with the second command listed and finally the `bg` command causes the stopped application to begin running in the background. At this point, the server application will continue running in the background, available for clients to connect to, until it is shutdown as described below.

Executing the `ServerAccess.jar` as shown below will allow a shutdown of the server.

```
% java -jar ServerAccess.jar [address [port]]
```

If no port or address is specified, a default of `cs-suse-4.ok.ubc.ca:4449` will be used. After connecting to the server application, the access program will prompt for a command. Anything entered here will be sent to the server as an admin command. To shutdown the server the command `stop` can be sent. The server will execute a shutdown and inform the client of its completion. If the server does not recognize a command sent it will respond to the client with the message, "`Unrecognized command.`" The server access program can be exited by typing the command `exit` which will not be sent to the server.

5.3 Database Map

During server start up and after a database connection is established, a data structure is constructed to represent the layout of the Schedule Viewer database. The process starts by requesting the meta data of the database and using this to create a list of database `Table` data structures. Then, for each table, a `SimpleNode` is created. The `SimpleNode` class is used to represent a node and its position in a network, graph or tree using its edges and their connections.

Once the nodes in the database have been established they are linked using foreign key relationships. `SimpleUndirectedEdge` objects are created for the relationships and a join string is added to each for future query building. The join strings are the string that would be used in the where clause of an SQL query. For example, to join the `timeblock` table and the `class` table using the fields `tbId` and `cTimeblock`, respectively, the join string would be "`tbId=cTimeblock`".

The `DatabaseMap` is then used by the server application during query processing in order to determine the best path through the database depending on the data requested. This process is discussed later in Section 5.6.

5.4 Client Communication

Once the database map has been successfully constructed, a socket server is started. This server listens on the server's open port for incoming client connections. When communication from a client is detected, a socket thread is started, and this thread deals with all client-server exchanges. A separate thread is started for each client that connects to the Schedule Viewer server. When a command is received from a client the socket thread does one of two things. First, if the command is prefixed with "`admin:`" anything that follows is executed in the socket thread as an admin command. These are commands that would be received from the `ServerAccess.jar` application mentioned in Section 5.2. If a command does not have the admin prefix, it is forwarded to the `DatabaseInterpreter` class which handles request processing. The database interpreter processes the request and returns a list of results to the socket thread. Each result in the list is sent separately to the client and an end notification is sent upon completion. The socket thread awaits a confirmation from the client then begins listening for any future requests.

5.5 Request Processing

It is the job of the `DatabaseInterpreter` class to determine the meaning of commands sent from clients and to query the database for results pertaining to those commands. The requests sent from the client have been listed and described in Section 4.2. The following subsections will describe what actions the server takes on receiving these requests.

5.5.1 Login

Upon receiving a `LOGIN` command the database interpreter executes a query to determine if the given user ID/password combination exists. If it does, another check is performed to certify that the user has access to the requested group. If both of these checks are successful, the user type is returned which is interpreted by the client as a successful login.

5.5.2 Map Tables

The database interpreter uses its database map to satisfy a `MAP tables` request. For each table, the interpreter iterates through each field and, using the database map, builds a string containing the table name, field name, field type, and friendly name for the field. Each of these strings is added to the list of results and the list is returned to the socket thread. All table information is stored in the database map from its creation on startup so no database access is needed for a `MAP tables` request.

5.5.3 New Group

A check is performed upon a `NEW group` request to determine if the desired group ID is already in use. If the ID is free, the group is created with the given user ID as its owner. Finally, the same user ID is given access to the new group.

5.5.4 New User

Similar to a `NEW group` request, the `NEW user` request also starts with a check for existing names. The requested user ID cannot already be used as a user ID or a group ID since a group will be created with an ID matching the new user ID. If the check passes a new user is created. In addition, a group is created with the same ID, the user is given access to this group and the user is given access to the web group.

5.5.5 Get Groups

For the `GET groups` command, the `group` table is queried for group ID's. If the user type of the request is a normal user, only groups belonging to that user are requested; otherwise, all groups are requested. Each group ID is added to the result list before the list is returned to the socket thread.

5.5.6 Get Users

The `GET users` command requests a list of all user ID's from the database. Each user ID is then added to the result list and the list is returned to the socket thread.

5.5.7 Get Permissions

This request queries the `user_group` table for user ID/group ID pairs illustrating which users have access to which groups. If the user type for the request is a normal user, only permissions pertaining to groups owned by the user are requested. For administrators, all permissions are requested. In either case, each user ID/group ID string is added to the result list which is then returned to the socket thread.

5.5.8 Grant Permissions

When a `GRANT permissions` request is received, the database interpreter first checks that the given user ID exists in the database. If the user does exist, a record is inserted into the `user_group` table for the given user ID/group ID combo.

5.5.9 Revoke Permissions

On a `REVOKE permissions` command, the database interpreter deletes the given user ID/group ID record from the `user_group` table.

5.5.10 Add Time Blocks

An insert statement is created for an `ADD timeblocks` request. This insert creates a new record in the `timeblock` table which contains all of the information about the appointment sent from the client. The end date field is entered as "null" and the occurrence pattern field is filled with a blank string. The command,

```
SELECT @@IDENTITY AS NewID
```

is appended to the end of the insert statement and causes the query to return the auto generated ID for the new record. This ID is then added to the list of results to be sent back to the client.

5.5.11 Add Recurrence

The `ADD recurrence` request is processed the same as the `ADD timeblocks` request in Section 5.5.10 except the end date and occurrence pattern fields are also added to the insert.

5.5.12 Update Time Blocks

The `UPDATE timeblocks` request is executed by setting all `timeblock` records for the given ID to the given values from the client. If the given group ID is the web group ID, the time block is not updated and a blank list of results if returned. On a successful update, `OK` is added to the list of results to be returned to the client.

5.5.13 Update Recurrence

The only difference between `UPDATE timeblocks` and `UPDATE recurrence` is that the end date and recurrence pattern fields contain data.

5.5.14 Select Timeblocks

`SELECT timeblocks` is the most complex request of the Schedule Viewer server commands. From the client request, the server builds two lists. The first of these is a list of requests and the second is a list of conditions. For our use, the only request will be `timeblocks`. The list of conditions will contain all of the field conditions for the request (i.e. "`cSubjectCode=COSC`"). Once the lists are created they are passed to the `createDatabaseQuery` method in the interpreter's database map.

In the `createDatabaseQuery` method the list of conditions is scanned and the database tables needed to satisfy the conditions are determined. A list of these tables is then used by the database map to determine the best path through the database for the conditions specified. The path returned consists of an array of two lists. The first list contains the tables that will be traversed in the query and the second list contains the linking conditions for the tables required. For example if the `class` and `timeblock` tables need to be traversed to reach all of the necessary data the first list will consist of "`class, timeblock`" and the second list will have the link between the two, "`cTimeBlock=tbId`". The list of user conditions is then added to the second parameter list.

These parameters are used to build the Microsoft SQL query, and the query is returned to the database interpreter. The interpreter executes the query and adds each returned record to the list of results to be sent to the client.

5.5.15 Get Details

The `GET details` request is provided only with a field name on which to return details. The interpreter uses the database map to determine which table the field is from and then builds a query to return all unique values of that field. Each of these values is added to the result list to be sent to the client.

5.6 Query Building

The bulk of the query building process is described in Section 5.5.14. This section contains details regarding query building which have not yet been covered.

5.6.1 Determining the Best Path

Section 5.5.14 mentioned that the database map determines the best path to traverse a list of tables needed to complete a query. The details of finding this path have not yet been discussed.

The `bestPath` method is used to determine the shortest path through the database which covers all tables necessary for the query. The method starts with a loop that iterates once for each table in the database. During each iteration the method attempts to create a path starting from one table in the database to each of the tables needed for the query. Future iterations repeat the process until the method has used each of the tables in the database as a starting point. After the iterations have finished, the shortest path is the one returned by the method.

5.6.2 Shortest Path

The best path algorithm described in the previous section makes use of another algorithm to determine the shortest path between two tables. This shortest path algorithm recursively performs a depth-first search of the database in order to find the path with the least number of tables from the starting table to the ending table. The `shortestPath` method accepts three arguments: start table, end table and a list of tables to be skipped in the search. This skip list is necessary to avoid loops in the recursion since it is the only way for future recurrences to check which tables have already been traversed. The `shortestPath` method starts by adding the current start node to the skip list. It then checks whether the start table is the same as the end table. This is the base case which means the algorithm is complete. If the base case has been encountered, the method returns a new path containing only the end node.

If the base case has not been reached each table connected to the start table by a foreign key is explored. For each of the tables connected to the start table another call to `shortestPath` is made using the new table as the start table, the same end table and the new skip list. This call to `shortestPath` will return a path to the end node if one exists. If the returned path is shorter than the shortest path so far it will be saved and the recurrence will continue. This process will return the shortest path from the start table to the end table to the `bestPath` method.

5.6.3 Date Ranges

All classes and recurring appointments created by users occur over a date range. The range is inclusive with a start date and an end date. When importing appointments, the date range is treated differently. By default the import date range is set for one week starting on the current day. Any appointments of classes that are occurring during the import time period will be returned. This logic requires four different cases to be checked.

1) The appointment starts after the given start date and ends before the given end date. The entire appointment occurs during the date range.
2) The appointment starts after the given start date but before the given end date. At least the start of the appointment is within the date range.
3) The appointment starts before the given start date and ends after the given end date. The appointment occurs during the entire date range.
4) The appointment ends after the given start date but before the given end date. At least the tail end of the appointment is in the date range.

All the above logical operations are treated as inclusive. If an appointment meets all other import requirements and if any one of the above cases is true, the appointment will be returned by the import. Below is an image to illustrate the four cases visually.

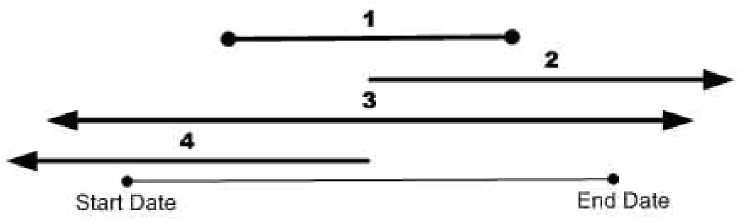


Figure 25: Import Date Range Cases

5.7 Database Communication

The third tier of the Schedule Viewer architecture is the database. Database abstraction from the client is achieved by allowing communication from the clients to only directly reach the server application. All communication with the database is executed through the server which allows more control over database access. The Schedule Viewer server uses the JDBC driver to communicate over the network with the Microsoft SQL Schedule Viewer database. Using the JDBC driver, the server connects to the database which is located on a separate machine. The driver does allows a connection to a database on the same machine as the server if desired.

5.8 Server Errors

Server errors are a part of any development process, and it is also possible for them to occur after release. Even with testing it is possible for bugs to get by and appear in a released application. Many possible errors in the Schedule Viewer server have been guarded against and will not interrupt overall operation of the server application. Most errors will display a notification on the server if it is running in

`-withinput` mode and may cause the client to function incorrectly. These errors will only affect the user which the error occurs for. Other users will be unaffected.

It is possible for a critical error to occur in the server application. If this occurs users will lose connection to the server and will no longer be able to connect to it. In such an event the Java Virtual Machine will usually create an error log for the crash. This error log will be located in the same directory as the `ScheduleViewer.jar` file. To resolve a situation similar to this the Java process running on the server machine should first be killed. This will avoid socket binding issues if the crashed server application is still holding on to the socket. At this point the server can be started again normally using the `ServerAccess.jar` file. Instructions for this operation are located in Section 5.2.

6 DATABASE

As mentioned, Schedule Viewer operates using a Microsoft SQL database located on `cssql.ok.ubc.ca`.

6.1 ER Diagram

Below, a detailed representation of the Schedule Viewer database is shown. Each of the titled boxes is a table in the database with a list of its fields and their types. The relationships between each of the tables are also shown with a descriptive label and a numerical representation of the relationship's multiplicity.

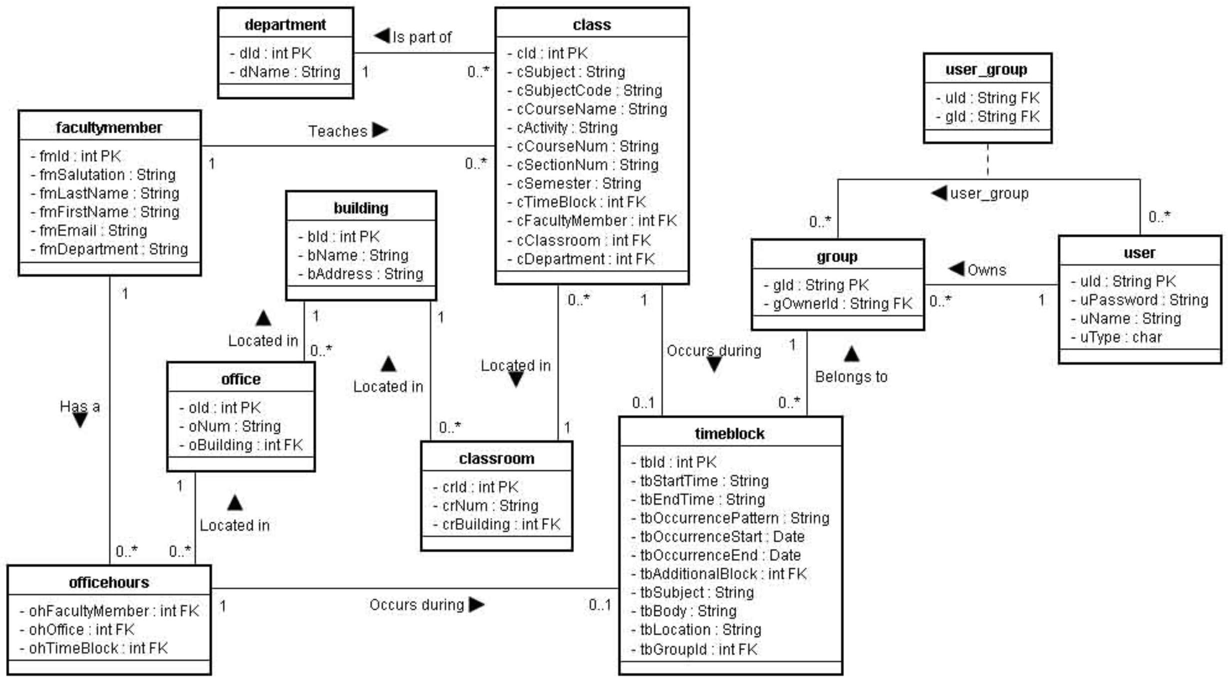


Figure 26: Schedule Viewer Database ER Diagram

6.2 Naming Conventions

The Schedule Viewer database follows a naming convention which allows some abstraction in the server application. Each table contains an ID field, any foreign key fields have the name of the table they refer to and finally all fields are preceded with one or two letters abbreviating the name of the table they belong to. These conventions apply to all tables used to store class data and differ slightly for the user and group tables used by the Schedule Viewer system.

6.3 Database Recreation

It was necessary at several points during the Schedule Viewer project to delete the existing database and create a new one in its place. Running the file `DatabaseWipeRecreate.java` located in the `screenScraper` package of the Schedule Viewer server code will execute several operations resulting in a new database with no data. First all records in the database are deleted, constraints on tables are dropped and the tables themselves are dropped. At this point everything in the database is gone. Next, new tables are created and their foreign key constraints are added. At this point the new database is ready to be used. This method will erase everything from the database and no existing classes or user created appointments will remain.

7 SCREEN SCRAPING

7.1 Why Screen Scrape?

The Schedule Viewer system was built to allow school administrators to view class and faculty schedule information. The data pertaining to these schedules needs to come from somewhere. The required data does exist on a UBC database, but access to that database was not granted to this project for security reasons. Instead, the Schedule Viewer system makes use of the publicly available course schedule information from the UBC Student Services web site [4]. The Schedule Viewer screen scraper makes use of HTML parsing technology to pull a large amount of data from the internet and dump it in the Schedule Viewer database. Compared to manually recording data from the web, the screen scraping process is exponentially faster. For this reason screen scraping was a great choice to provide a relatively quick method of gathering the required data. The Schedule Viewer screen scraper can be run again at any point in order to retrieve new or updated course data from the Student Services web site.

7.2 Bi-Yearly Updates

UBC Okanagan course information is separated into two sessions: winter and summer. The screen scraper should be run at least once for each of these sessions. The winter session may require more executions since course data for the second semester may change closer to its start. The screen scraper code to be executed is called `ScreenScrapeAndUpdateDatabase.java` and within it are two variables which need to be set depending on the session being imported. The `year` variable needs to be set to the year for the start of the session. For example, January 2009 is the 2008 winter session so the year should be set to 2008. The second variable needed to be set is `session`. This should be set to either "winter" or "summer" depending on the session for which data is being imported.

7.3 Screen Scraping Process

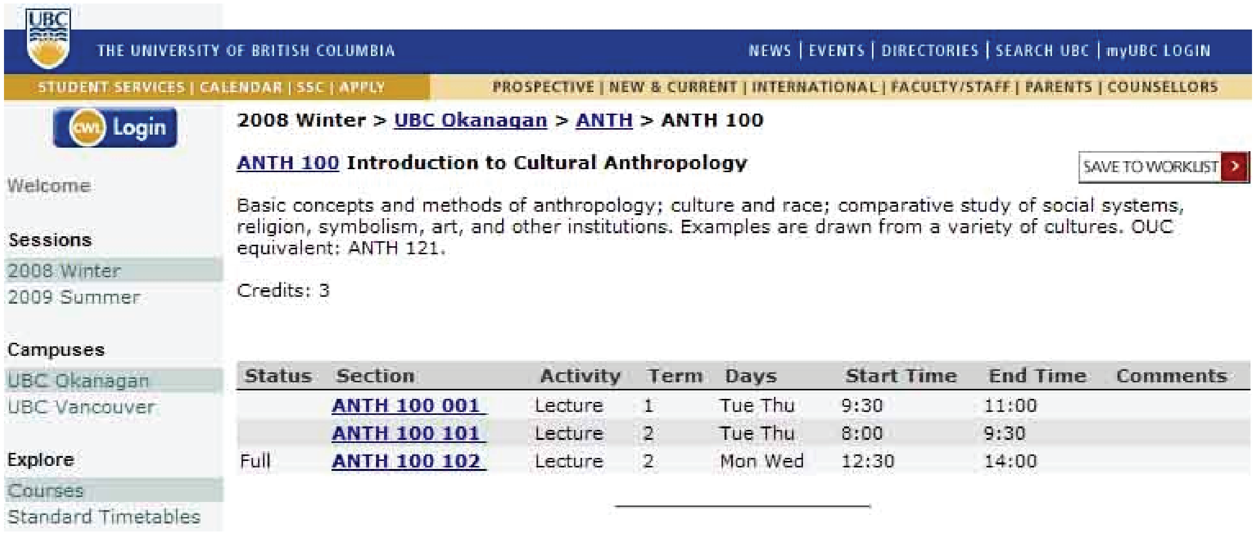
Schedule Viewer makes use of Java's XPath functionality as well as the `Tidy.java` package [2]. `Tidy.java` is used by the Schedule Viewer screen scraper to attempt to clean up the HTML code retrieved from the Student Services web site. Specifically, `Tidy.java` is used to parse each page's DOM or Document Object Model. Using this syntactically strict version of the web page, XPath is then used to parse specific information out of the web page data. This data is then put together as data structures within the code and finally used to build SQL queries which are run by the Schedule Viewer database to insert new information and update existing data.

7.3.1 XPath Syntax

XPath uses a detailed syntax system in order to parse data from XML or HTML documents. W3Schools provides a great tutorial [3] with a description of the XPath system and numerous helpful examples.

7.4 Screen Scraper Example

Figure 27 shows an example page from the UBC Student Services website. The illustration is an example of how a user's internet browser displays the page. The browser uses the source code from Figure 28 to produce this display. When the screen scraper requests web information from the Student Services site it receives source code like the example source code seen in Figure 28. The screen scraper then parses out the required data and either stores it in the Schedule Viewer database, or uses it to retrieve further information from the site. In the example shown the course data that the screen scraper would be interested in for Anthropology 100 Section 001 can be seen.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA
NEWS | EVENTS | DIRECTORIES | SEARCH UBC | myUBC LOGIN
STUDENT SERVICES | CALENDAR | SSC | APPLY
PROSPECTIVE | NEW & CURRENT | INTERNATIONAL | FACULTY/STAFF | PARENTS | COUNSELLORS
CWL Login
Welcome
Sessions
2008 Winter
2009 Summer
Campuses
UBC Okanagan
UBC Vancouver
Explore
Courses
Standard Timetables
SAVE TO WORKLIST

2008 Winter > UBC Okanagan > ANTH > ANTH 100

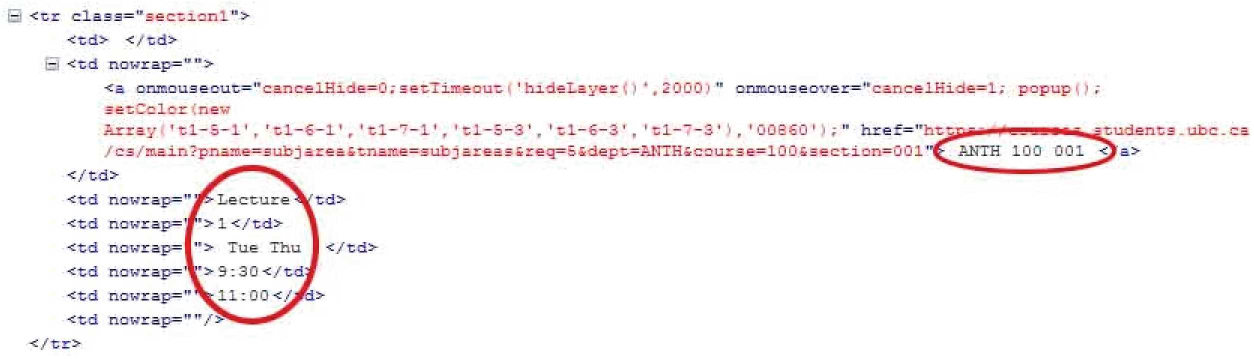
ANTH 100 Introduction to Cultural Anthropology

Basic concepts and methods of anthropology; culture and race; comparative study of social systems, religion, symbolism, art, and other institutions. Examples are drawn from a variety of cultures. OUC equivalent: ANTH 121.

Credits: 3

Status	Section	Activity	Term	Days	Start Time	End Time	Comments
	ANTH 100 001	Lecture	1	Tue Thu	9:30	11:00	
	ANTH 100 101	Lecture	2	Tue Thu	8:00	9:30	
Full	ANTH 100 102	Lecture	2	Mon Wed	12:30	14:00	

Figure 27: UBC Student Services Web Page



```
⊟ <tr class="section1">
      <td> </td>
   ⊟ <td nowrap="">
         <a onmouseout="cancelHide=0;setTimeout('hideLayer()',2000)" onmouseover="cancelHide=1; popup();
         setColor(new
         Array('t1-5-1','t1-6-1','t1-7-1','t1-5-3','t1-6-3','t1-7-3'),'00860');" href="https://courses.students.ubc.ca
         /cs/main?pname=subjarea&tname=subjareas&req=5&dept=ANTH&course=100&section=001"> ANTH 100 001 </a>
      </td>
      <td nowrap="">Lecture</td>
      <td nowrap="">1</td>
      <td nowrap=""> Tue Thu </td>
      <td nowrap="">9:30</td>
      <td nowrap="">11:00</td>
      <td nowrap=""/>
   </tr>
```

Figure 28: UBC Student Services Page Source

8 RESULTS AND USER FEEDBACK

One of the benefits of working on a project that will have real users is that beta users are easy to find. Jan Paseska and Patricia Lasserre were both kind enough to spend some time using Schedule Viewer while it was still in development and provide feedback on their experiences. This user response helped to pinpoint bugs in the software as well as to discover changes and additions that would improve the user experience. Here are Jan's thoughts on Schedule Viewer:

> *"The Schedule Viewer program makes the viewing of multiple schedules a simple task. It is a huge timesaver for someone in my position who is always juggling schedules and times in order to schedule meetings, speakers and other activities. The choices given for determining the schedules are practical and allow for different uses of the program. All in all I think it's a practical program which will be very useful to support staff."*

9 DISCUSSION AND CONCLUSIONS

9.1 Discussion

During my time working on Schedule Viewer I had the chance to work with a lot of different technologies. Most of these tools helped greatly in the production of this software and made my life as a developer a lot easier. Learning new technologies meant that I also ran into many challenges. What follows are some of the more interesting and useful things I discovered.

9.1.1 Visual Studio 2005 Tortoise SVN Toolbar

Using SVN to maintain a backup of development code is a necessity for software development. Losing existing code in the middle of development with no backup to recover from can be disastrous. Eclipse, used to develop the Java Schedule Viewer server code, has a plug in to make SVN backups convenient and easy. In contrast, Visual Studio 2005, used to write code for the C# Schedule Viewer client, has no such plug in. Tortoise SVN can be used to manually backup the C# code, but there is no easy way to do this from within the Visual Studio 2005 program. I found a handy tutorial [1] with instructions on how to create a Tortoise SVN toolbar for Visual Studio 2005 which allows SVN backups to be executed conveniently from within the program.

9.1.2 Visual Studio 2005 Application Deployment/Tortoise SVN Issue

After deployment of the Schedule Viewer client application I encountered a problem with committing the Visual Studio 2005 project files. Tortoise SVN would fail saying it was unable to access the files in the project directories. I discovered that after building the ScheduleViewerSetup project in Visual Studio 2005, folders within the project directory would become read-only, causing Tortoise SVN to fail. This problem was solved by navigating through Windows Explorer to the top level project folder for the Schedule Viewer client and turning off the read-only status of it and all subfolders.

9.1.3 C# Garbage Collection

Garbage collection in Visual Studio 2005 using C# caused some very tricky bugs during the development of Schedule Viewer. The first bug that appeared was the toolbar button for the Schedule Viewer client would become unresponsive. It would work the first time it was clicked but after that it would not do anything. After much time and frustration I discovered this was being caused by garbage collection. I had declared the variable for the toolbar button locally in a method. Once this method had finished running there was no longer a reference to the button and so it was removed from the memory. I solved this issue by declaring the toolbar button globally in the class so its reference was not lost.

Another similar issue was with items in the calendar changing. When appointments are changed or created an event is triggered which causes code to run dealing with the event. The problem caused by garbage collection was that these events would be triggered several times then they would stop registering. Changes would be made, but the code for the event would not get run. Finally, I discovered storing a reference to the calendar’s items rather than just to the calendar would keep the events active. This was because the items were a separate data structure that was being dealt with by garbage collection independently from the calendar itself.

9.1.4 Microsoft SQL Reserved Words

Problems were encountered upon adding the `user` table to the Schedule Viewer database. References to the new table would cause errors and not return any results. I determined that this was caused by "user" being a reserved word in Microsoft SQL. Not wanting to change the intuitive name I had selected for the table I searched for a way around this. I learned that adding square brackets to a table name will cause Microsoft SQL to use the literal value rather than trying to evaluate a reserved word expression. To be safe, I enclosed all references to table names within the Schedule Viewer server with square brackets. This will help avoid any future problems.

9.2 Conclusions

Many challenges were encountered and overcome during the development of the Schedule Viewer system and a lot was learned. Creating an entire three-tier software system gave exposure to many issues that were interesting to solve and some that were not yet discovered in previous work. Starting with existing software from the COSC 404 final project allowed the chance to fix problems and add features that time had not allowed in the past. It also provided the opportunity to produce the size and caliber of product that was completed. Working on a project that is useful for and needed by real users was a great experience. It provided real world experience that is extremely valuable in a schooling experience. The Schedule Viewer project produced a valuable tool for UBC Okanagan administrators that will help them work with scheduling information in an easy, intuitive way.

9.3 Future Work

Several features were discovered during the development of the Schedule Viewer system which were valuable, but did not fit into the time constraints of this project. The first of these is the ability to add appointments with more details. At this point, adding appointments only creates an entry in the `timeblock` table with almost all of the user entered information in the subject, location and body fields. This does not allow users to enter a new class that can be searched for by subject code, room number, or most of the other specific fields on the main form. Another form could be created that allows entry into each of these fields. This information could then be stored in the database in its correct fields and could be searched for more specifically. An editing feature to change this specific information would also need to be included.

When creating a schedule for a new year, administration would benefit from Schedule Viewer features to help keep things organized. It is possible at this point, through the use of groups, to create a set of appointments which form a working copy of future schedules. It would be useful if a feature was created to allow a spreadsheet to be built from information on these working copy courses. This spreadsheet would be of the correct form to be submitted for school approval. Once approved, the ability to change the appointment's groups to a finalized status or group would also be beneficial.

Encryption is a feature that would prove useful to the Schedule Viewer system as it is now. At this point there is no encryption performed. All data is transmitted in plain text between client and server. As well, queries between the server and database are not encrypted. Finally, the information stored in the database is also in plain text. This poses some security threat to user data and could be remedied with some additional work.

REFERENCES

[1] Hogue, D. (2006, 08 16). *TortoiseSVN in Visual Studio* . Retrieved March 14, 2009, from blog.vorpal.cc, David Hogue's Blog: http://blog.vorpal.cc/category/development/tortoisesvn-in-visual-studio.html

[2] Raggett, D., & Quick, A. (1998-2000). *Tidy.java*. Retrieved March 14, 2009, from org.w3c.tidy.Tidy: http://www.docjar.org/docs/api/org/w3c/tidy/Tidy.html

[3] Refsnes Data. (1999-2009). *XPath Tutorial*. Retrieved March 14, 2009, from W3Schools: http://www.w3schools.com/XPath/default.asp

[4] The University of British Columbia. (1999-2007). *The University of British Columbia - Course Schedule*. Retrieved 2008/2009, from UBC Student Services: https://courses.students.ubc.ca/cs/main

[5] Wikipedia Foundation, Inc. (2009). *Java Database Connectivity - Wikipedia, the free encyclopedia.* Retrieved March 19, 2009, from http://en.wikipedia.org/wiki/Jdbc

[6] Wikipedia Foundation, Inc. (2009). *Java (programming language) - Wikipedia, the free encyclopedia.* Retrieved March 19, 2009, from http://en.wikipedia.org/wiki/Java_(programming_language)

[7] Wikipedia Foundation, Inc. (2009). *Microsoft SQL Server - Wikipedia, the free encyclopedia.* Retrieved March 19, 2009, from http://en.wikipedia.org/wiki/MS_SQL