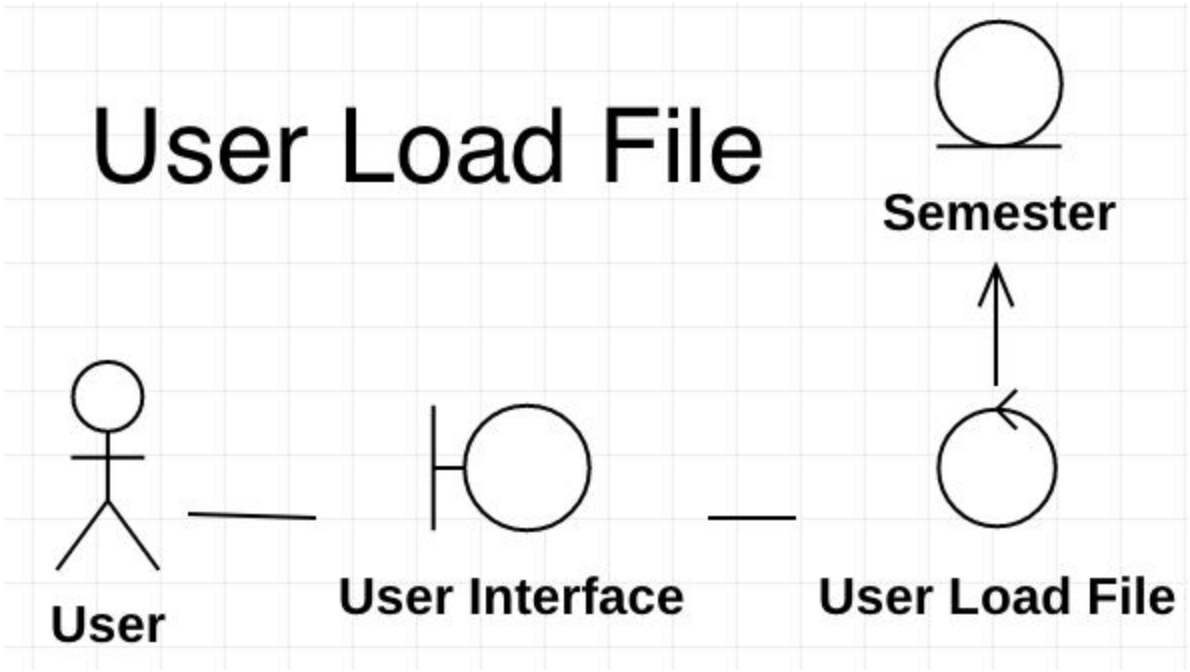
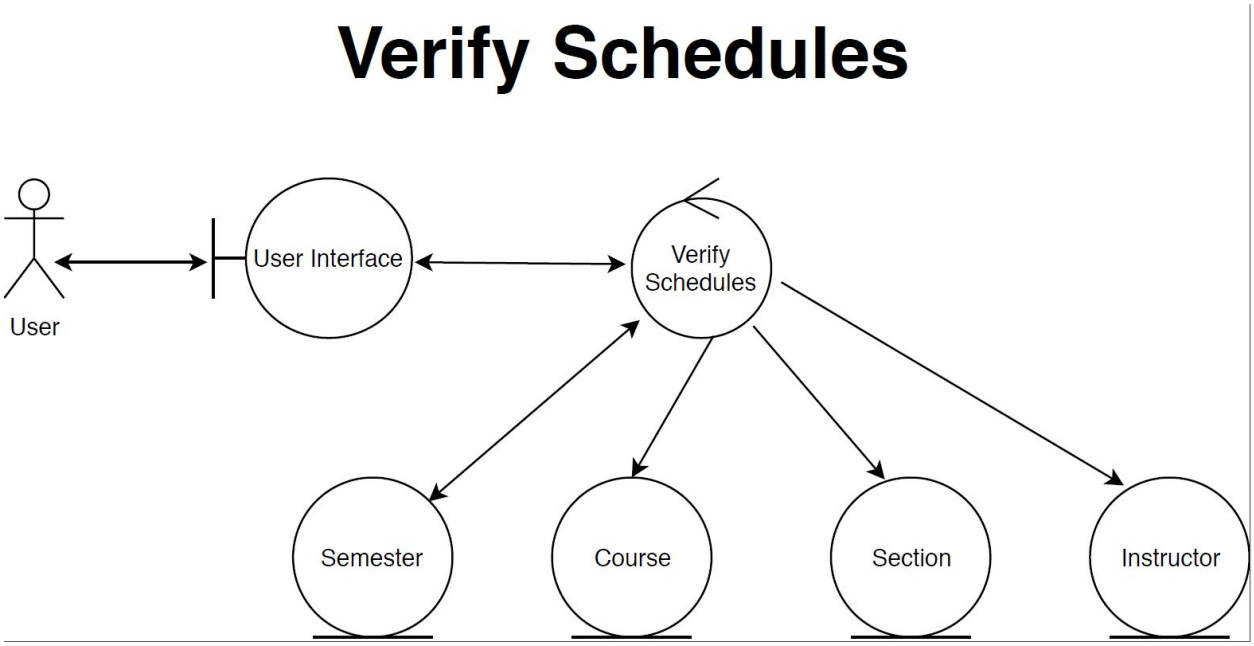
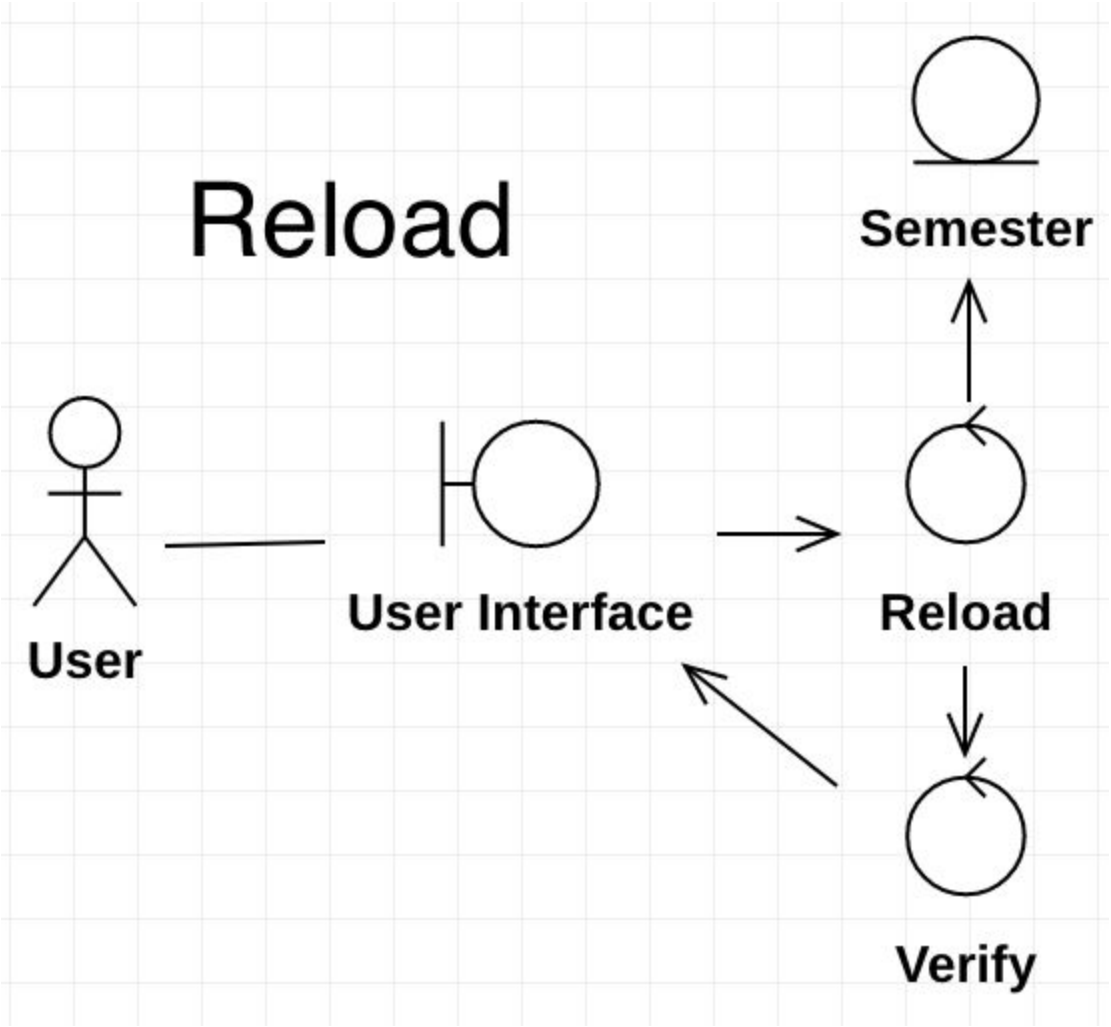


Justifications:
The User Interacts with the user interface, selecting the load file function. This causes the system to load the file into a Semester entity (or schedule). Once the file is loaded, the user interface displays the loaded file's name/path to the user, hence the double arrows between User load File, User Interface, and the User. The User Load File function only creates a Semester, therefore it uses only a directed arrow. (The solid lines with no arrow are double arrows but the program did not contain double arrows)

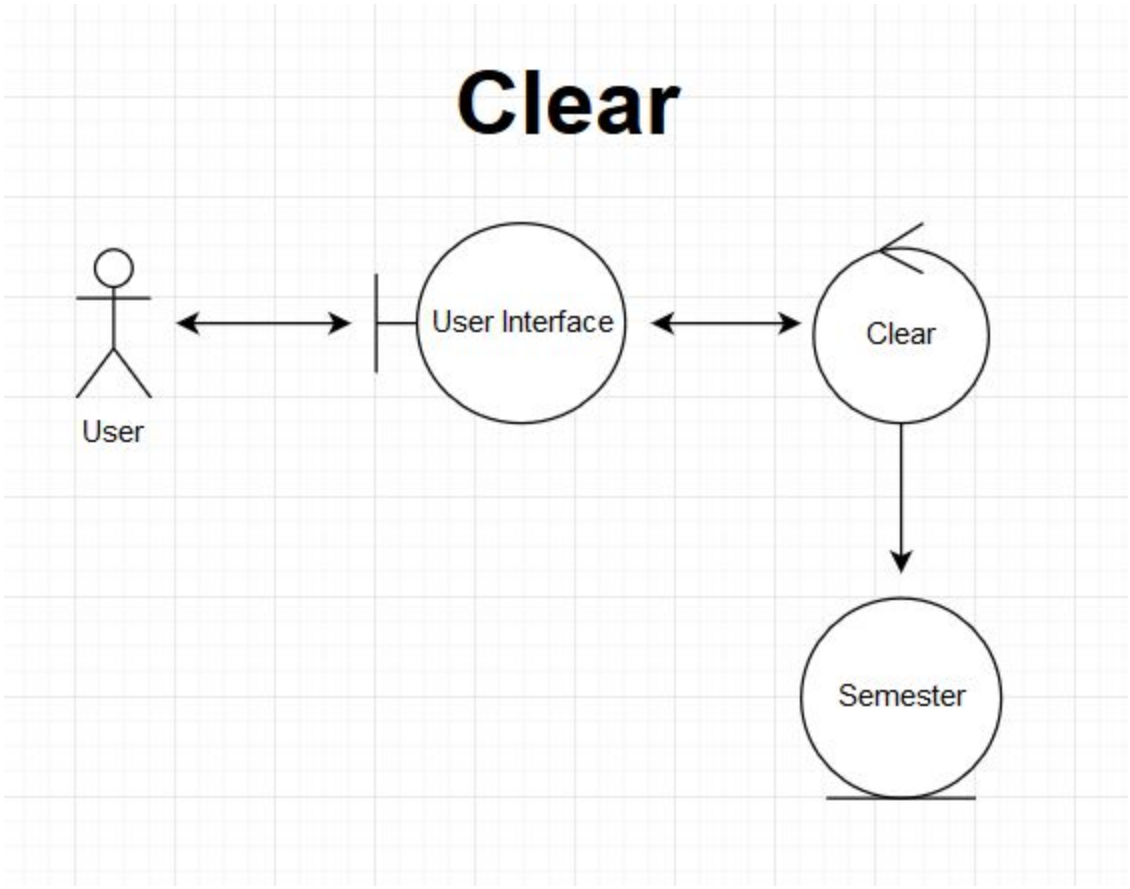


Justifications: The user either provides a filename or selects a file, either way interacting with

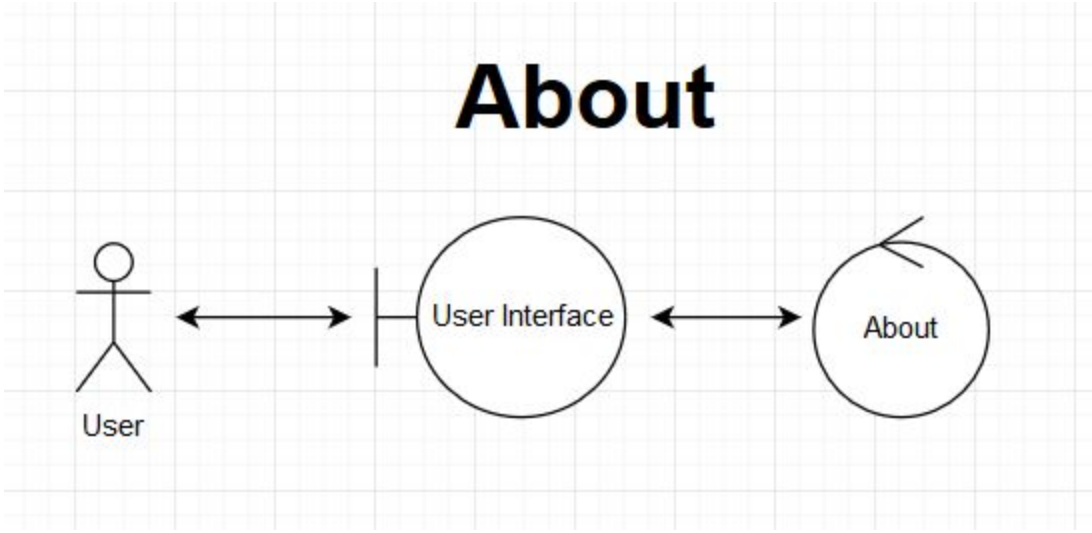
the UI so a double arrow, and then the verify schedules reads in and loads the schedule information creating a Semester, course, sections, and instructors accordingly. The double arrow on Semester is because verify schedules then has to look at the data stored there to verify the two schedules. Then, the control has to show success or failure to the user through the user interface, so a double arrow.



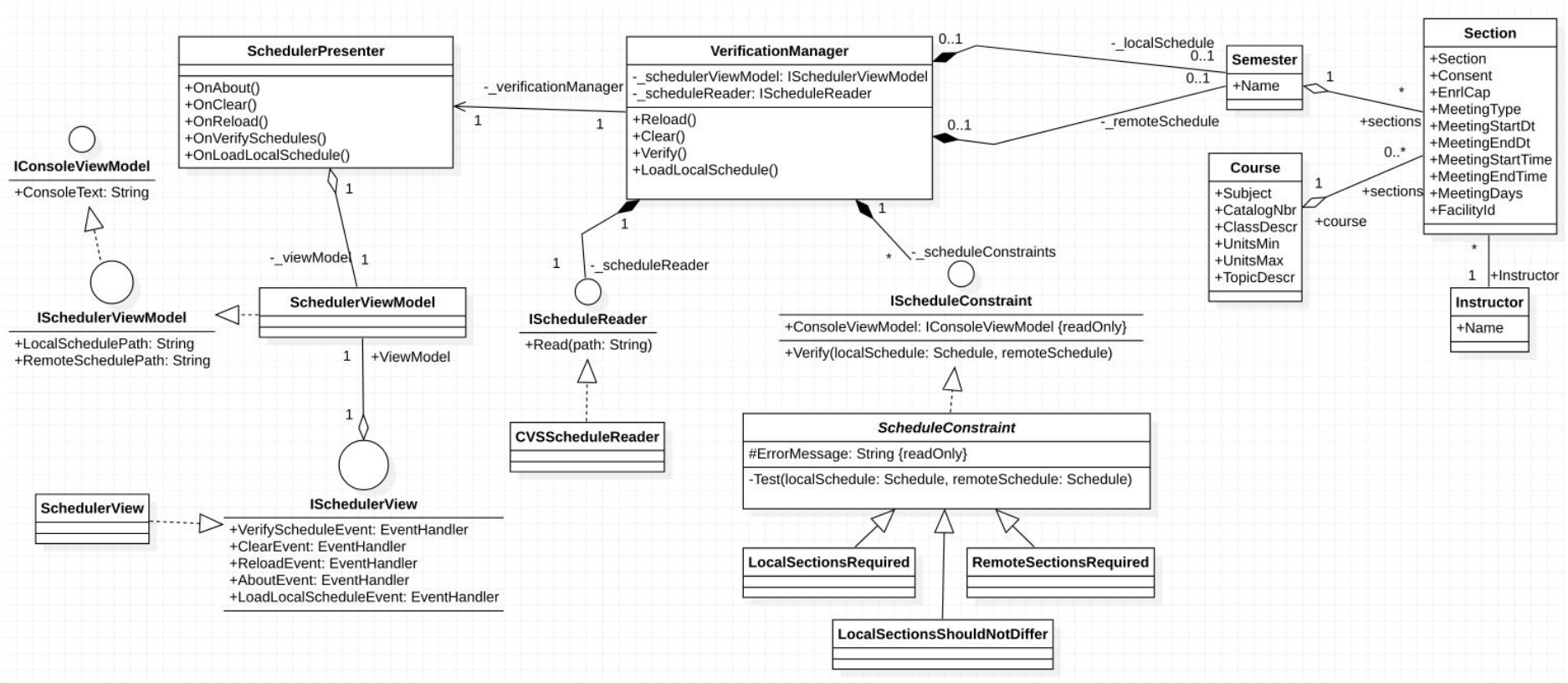
Justifications:
The User interacts with the system via the user interface by selecting the ‘Reload’ function. The user interface then tells the system to begin reloading. During reload, the system removes loaded schedules from memory and loads new ones given the same file paths. Then, still during reload, the system starts the verify function to verify the 2 schedules. Verify then talks to the user interface to display a success or error message to the user depending on the results. (The solid lines with no arrow are double arrows but the program did not contain double arrows)



Justifications: The user has to select clear, and then the Clear control clears any schedules from memory, meaning the Semesters are modified. Then, the control goes back to the UI and the system/UI updates the display, so a double arrow for both.



Justifications: The user selects About function and then the control says to display the about information, which is then displayed to the user through the UI. This means double arrows for both.

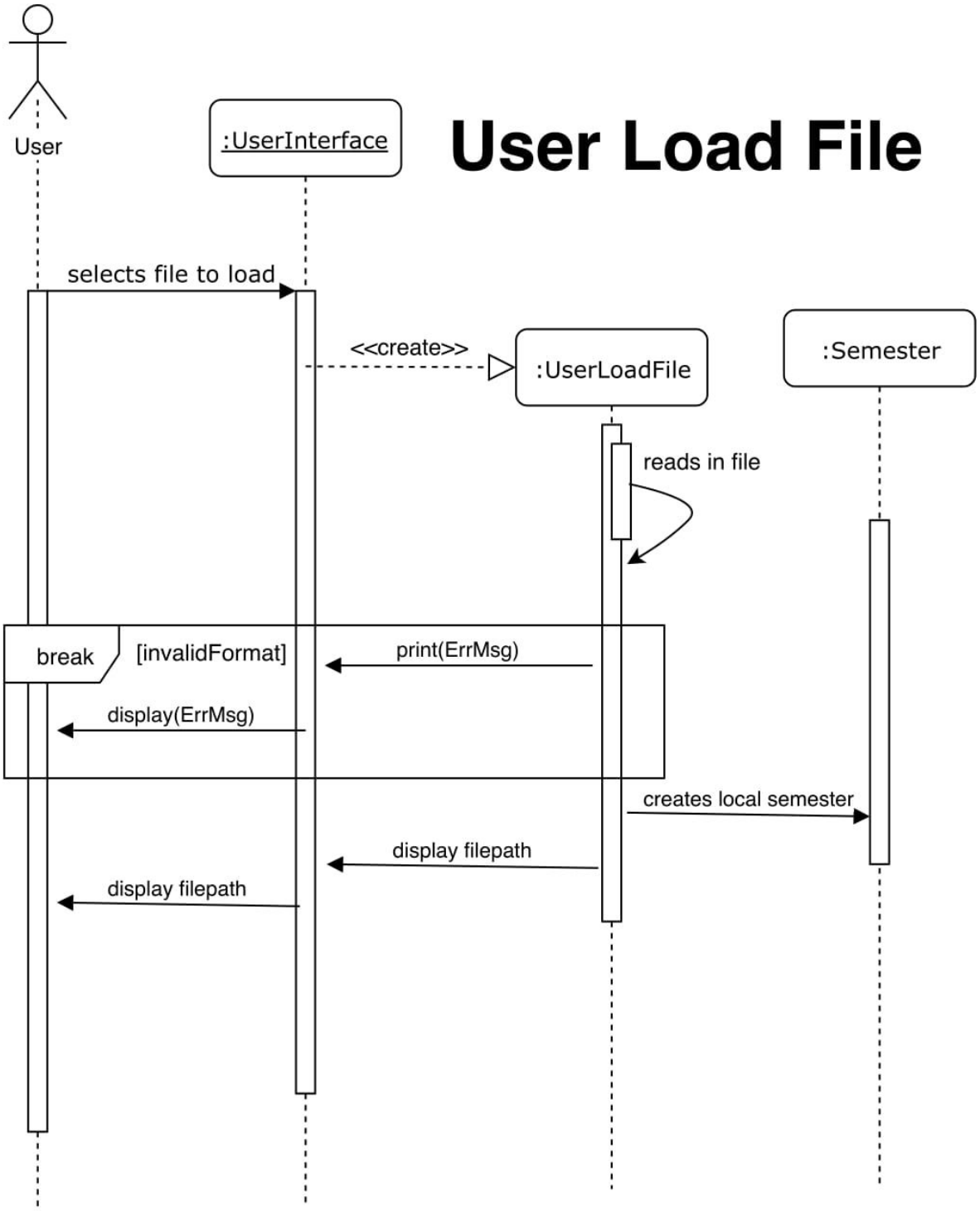


Cale Povilonis Andrew Fulton

Cale Povilonis Andrew Fulton

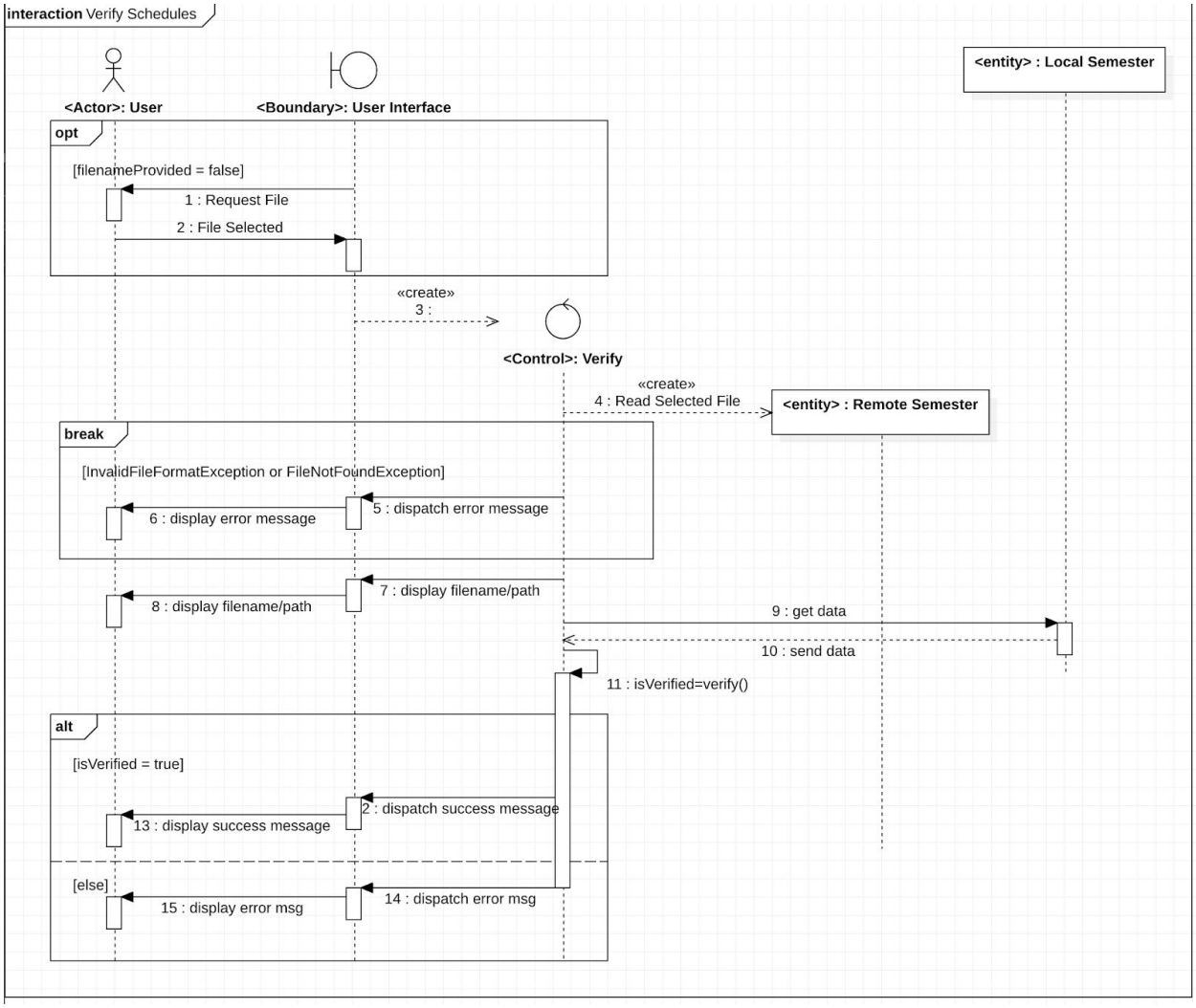
Justifications:
For our User Interface boundary, we decided to use the MVP architectural style. Our SchedulerViewModel contains ConsoleText this text is the text that is displayed to the user. The Local and remote schedule attributes are set by the user from the SchedulerView. The SchedulerPresenter handles all the business logic. It creates a VerificationManager at construction which needs a list of IScheduleConstraints, the IScheduleReader, and the ISchedulerViewModel. The SchedulerPresenter handles the About Use case on its own, simply updating the ConsoleText in the ViewModel. The clear, reload, Verify Schedules, and User Load File use cases are handled partly by the presenter, but ultimately handed off to the VerificationManager.

Mulitiplicity Justifications:
In an MVP architecture, most of the time, a Presenter, View, and ViewModel will all be 1 to 1 in terms of multiplicity. The SchedulerPresenter has a ViewModel, and the View has a ViewModel, but the View and the presenter do not know about each other. The VerificationManager and SchedulerPresenter were made 1 to 1 because there is no reason, at least for this project, for our system to have multiple VerificationManagers. A VerificationManager is composed of 1 (it should actually be aggregation) ScheduleReader to read in the Local and remote schedule files, since it is only reading from 1 type of file in this project's case. Likewise, the IScheduleReader is only associated with 1 VerificationManager because there is only 1 VerificationManager for this project's case. A VerificationManager also has multiple scheduleConstraints because this project has atleast 3 different types of reasons two schedules could not be verified, but could easily have more. These ScheduleConstraints are used to verify the two local and remote schedules against each other in the verify use case. Finally, a VerificationManager could have 0 or 1 Local/Remote Schedules because the schedules may not have been loaded in yet; however, once they are loaded in, the VerificationManager contains 1 of each.



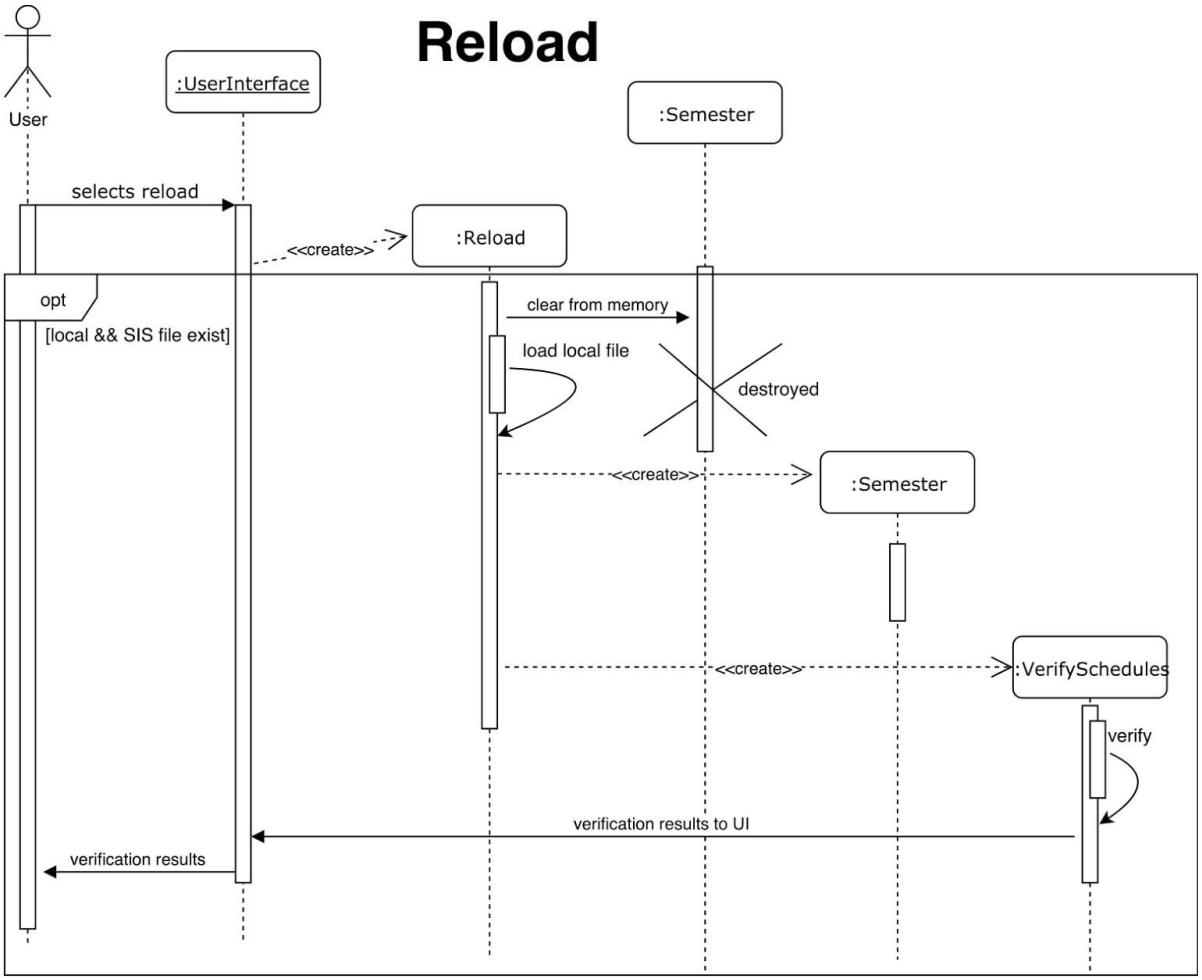
Justifications: The user selects a file to load, and the the user interface ‘creates’ the UserLoadFile control. The file reads in the local file. If the format of the file is invalid, we have a break that displays an error message. If the format is valid however, we create a local semester

based on the read in information and then display the file path to the UI and then to the user.

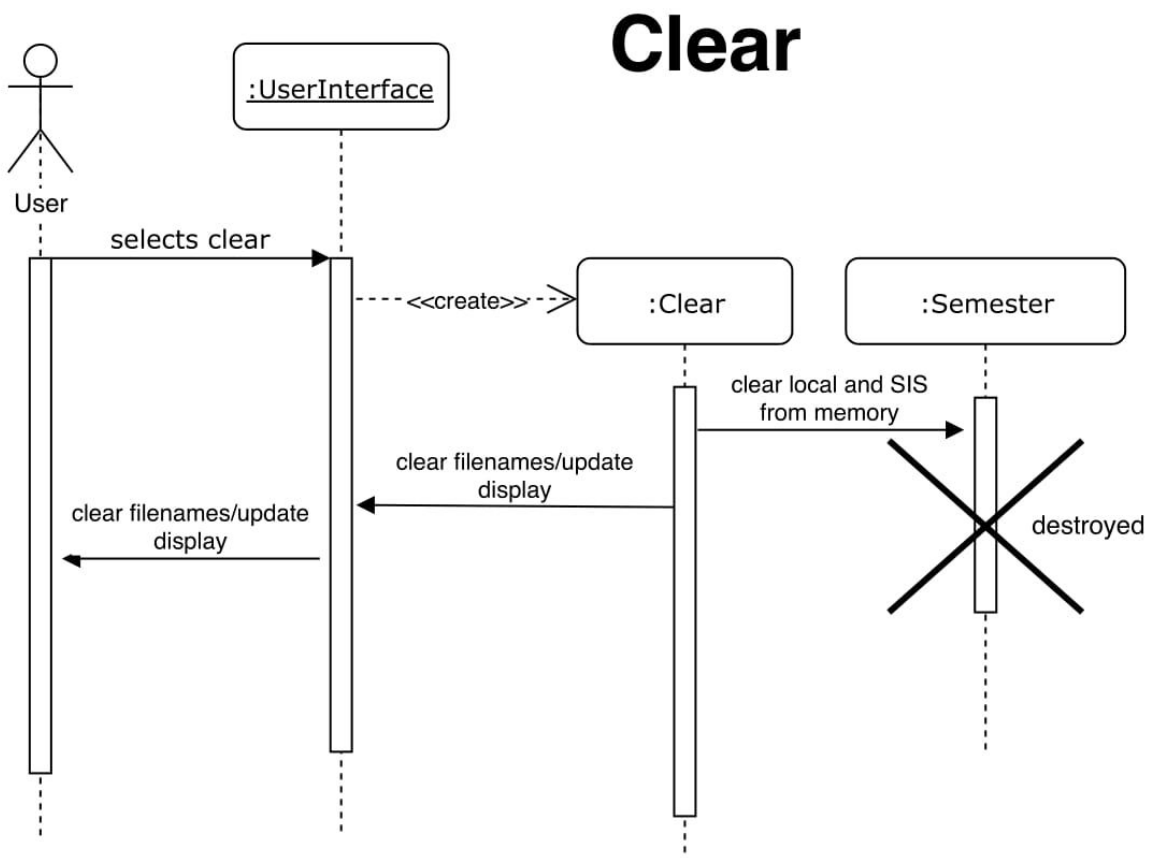


Justifications:

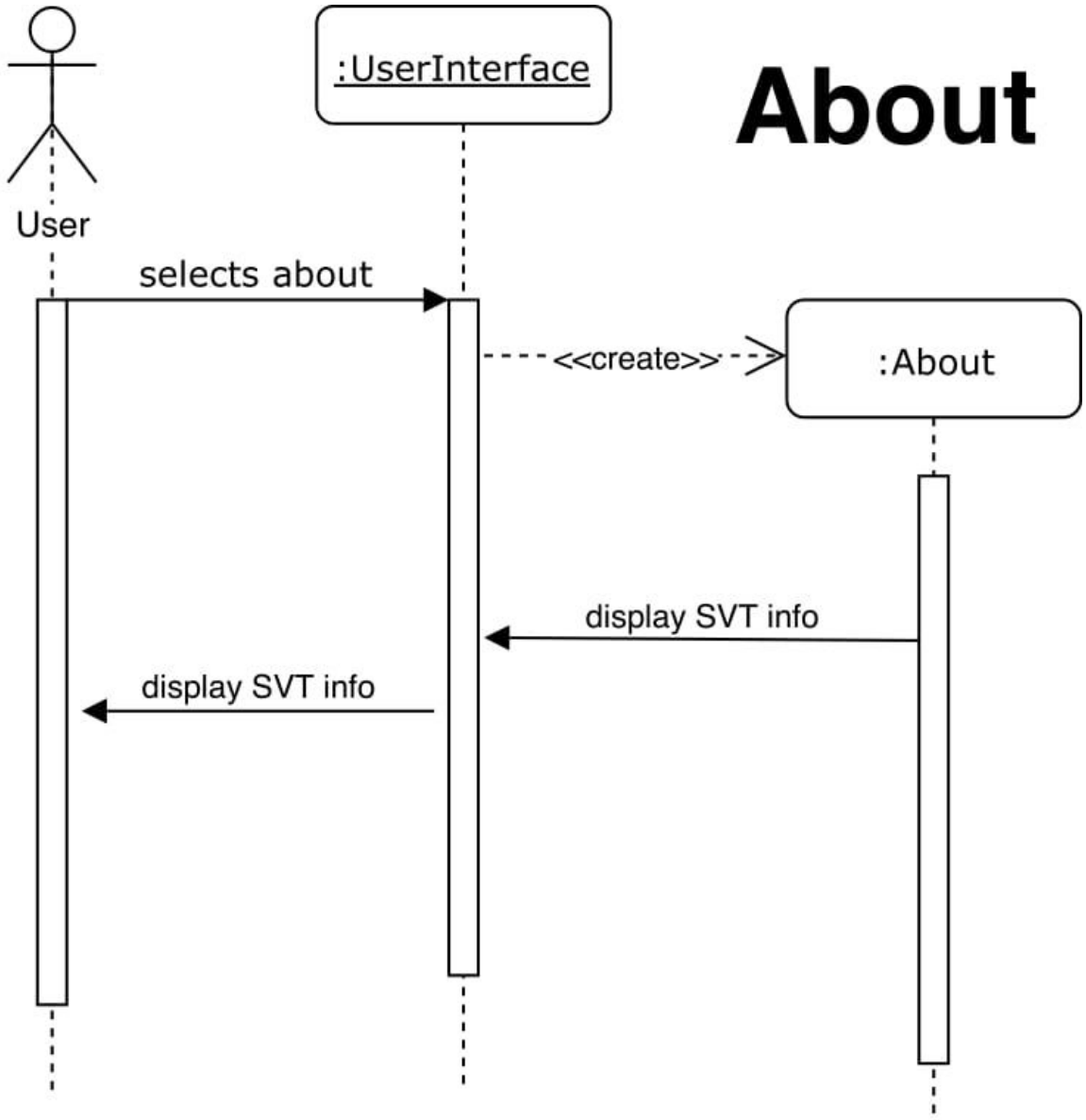
If the filename is not provided, we ask the user for it. Then, we can move on and create the verify control. This will then read the selected KSIS (remote) file. If either an InvalidFileFormatException, or a FIleNotFoundException occur, the user is notified via an error message and the program breaks out of the sequence as it cannot continue without the file. Otherwise, the system will then continue by displaying the remote filename/path to the user. The verify control then gets the local semester data and verifies it against the recently loaded remote semester. If the semesters are successfully verified, we display a success message to the user, otherwise we should display an error message that depicts exactly why the two were unable to be verified.



Justifications: The user selects reload which then goes to the UI boundary. The UI then creates the Reload control. Then, we have a huge opt around the whole rest where if the local and SIS file exist, we will perform the functions, else we don't need to do anything/can't. If the files do exist, however, we will clear them from memory and destroy them. Then, we load in the local file with the valid format. This then creates a semester object for the local file. We then create the VerifySchedules control which is included in this use case. The steps for this sequence are shown in the Verify Schedules use case sequence diagram above so we opted to say 'verify' as this process has already been shown. At the end of the control it updates the display with the verification results which the user then sees.



Justifications: This use case sequence diagram is pretty simple. Firstly, the user selects clear. The UI boundary then creates the Clear control. This control clears or deletes the local and SIS semesters from memory and destroys it. Then, the display is updated and the filenames are cleared.



Justifications: This use case is also pretty simple as well. Firstly, the user selects about. The UI boundary then creates the About control. The about control displays the SVT info which goes to the UI and then to the user.