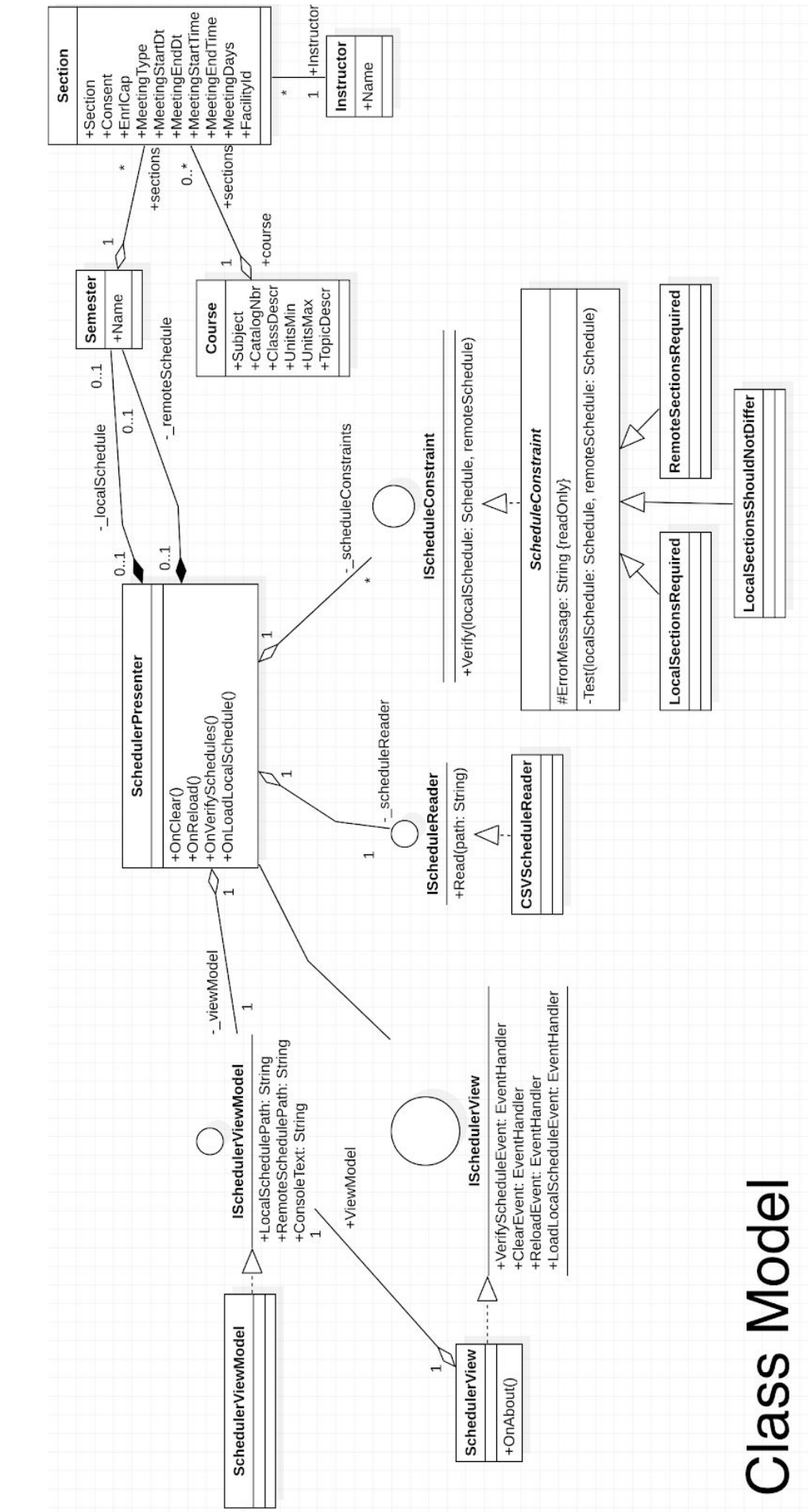
Cale Povilonis Andrew Fulton



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 when the user selects a local or remote file to load. The SchedulerPresenter handles all the business logic. This includes reading both the local and KSIS files using the provided IScheduleReader, verifying them using the given list of IScheduleConstraints, clearing the schedules from memory, reloading them. The SchedulerView handles the About use case on its own, since it contains no business logic. The SchedulerPresenter takes in a ISchedulerViewModel, IScheduleReader, and a list of IScheduleConstraints. This allows the Scheduler presenter to be open for modification because we can easily change the constraints needed to verify the schedules by simply passing in some different constraints. Also, we can change the type of schedule format we are reading in by changing the IScheduleReader we pass in. The SchedulerPresenter requires a ISchedulerViewModel so that it can update data that the view depends on. The presenter is also associated with an ISchedulerView so that it can subscribe to the view's events. All the Schedule constraints inherit from an Abstract class ScheduleConstraint which would implement the Verify function for the IScheduleConstraint so that all ScheduleConstraints don't have to, making it easy to create new Constraints.

Also, sideNote: our SchedulerViewModel would implement INotifyPropertyChanged so that as attributes in our view model change, they are reflected in our view via an implementation of the observer pattern within Microsoft's .Net framework.

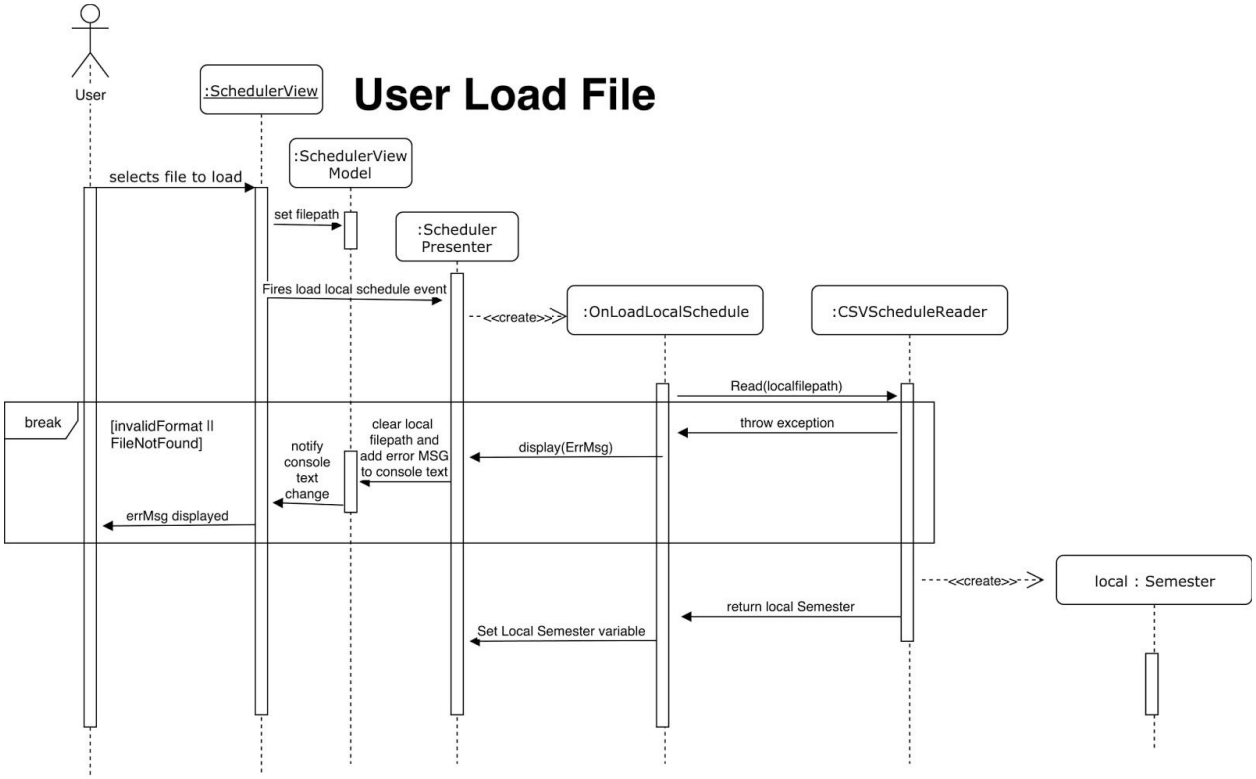
Also, events in the view are fired and they are subscribed to in the presenter. (Observer pattern, but we simply used an already existing implementation)

The only pattern/style we actually ended up using and implementing was Model-View-Presenter and the Façade pattern with the IScheduleConstraint class.

We considered using a factory pattern to create a SchedulerPresenter, since it has to subscribe to a view, but this would be unnecessary as the presenter is created once.

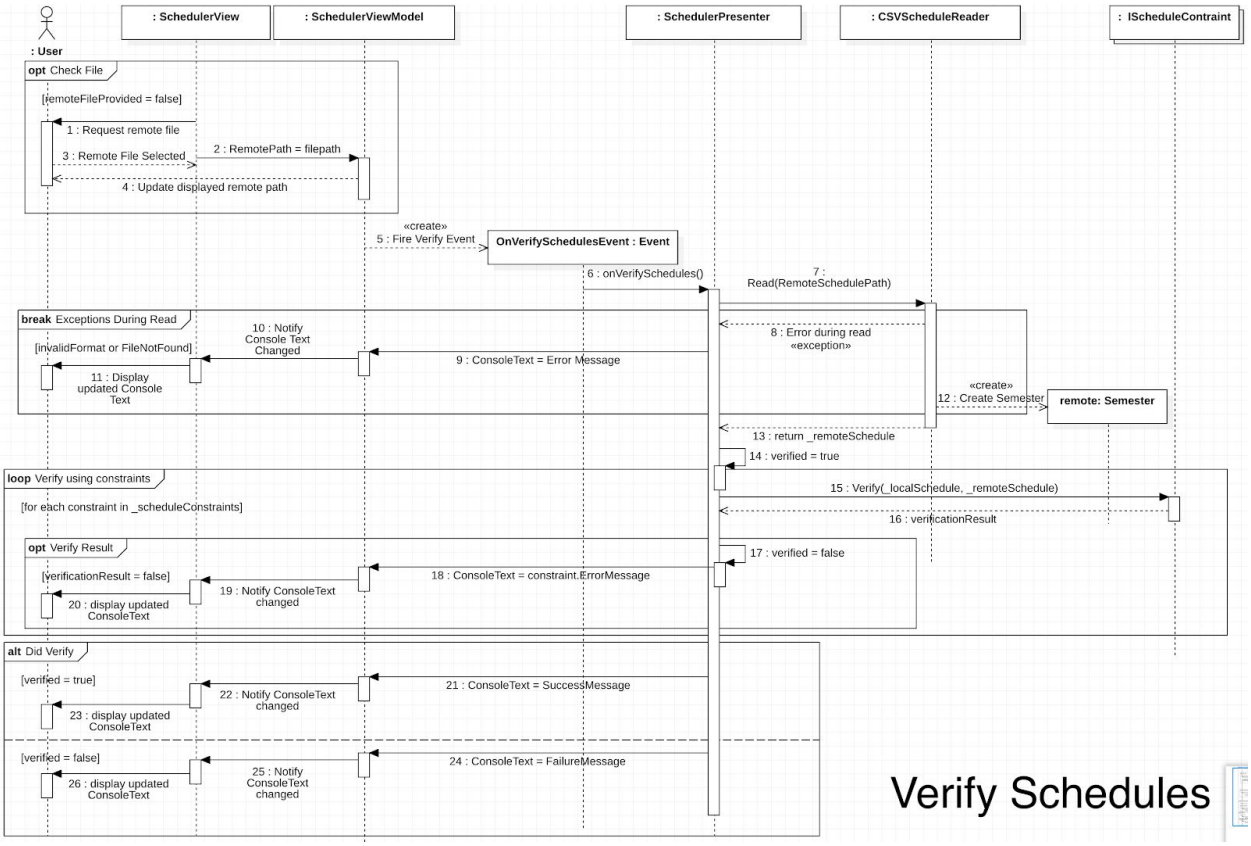
Mulitiplicity Justifications:
In an MVP architecture, most of the time, a Presenter, View, and ViewModel will all be 1 to 1 in terms of multiplicity. The SchedulerPresenter has a ViewModel, and the View has a ViewModel, but the View and the presenter do not know about each other. A SchedulerPresenter is composed of 1 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 SchedulerPresenter because there is only 1 SchedulerPresenter in this project's case. A SchedulerPresenter 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 SchedulePresenter 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 SchedulePresenter contains 1 of each.



Justifications: The user selects the file to load. This then goes the the SchedulerView. Then, the filepath is set thru the SchedularViewModel. After that, the event for OnLoadLocalSchedule is fired which goes to the Presenter and that create the OnLoadLocalSchedule method/control. This then uses the CSVScheduleReader stored in the Presenter and calls its Read() function with the local filepath. If the filepath does not exist or has an invalid format, an exception is thrown in the CSVScheduleReader and is caught by OnLoadLocalSchedule. Then, an error message will be displayed and the filepath will be cleared/set to empty. This will be displayed to the user. If there are not errors, however, the local semester will be created and returned by the Read() function and set as the local semester variable located in the Presenter.

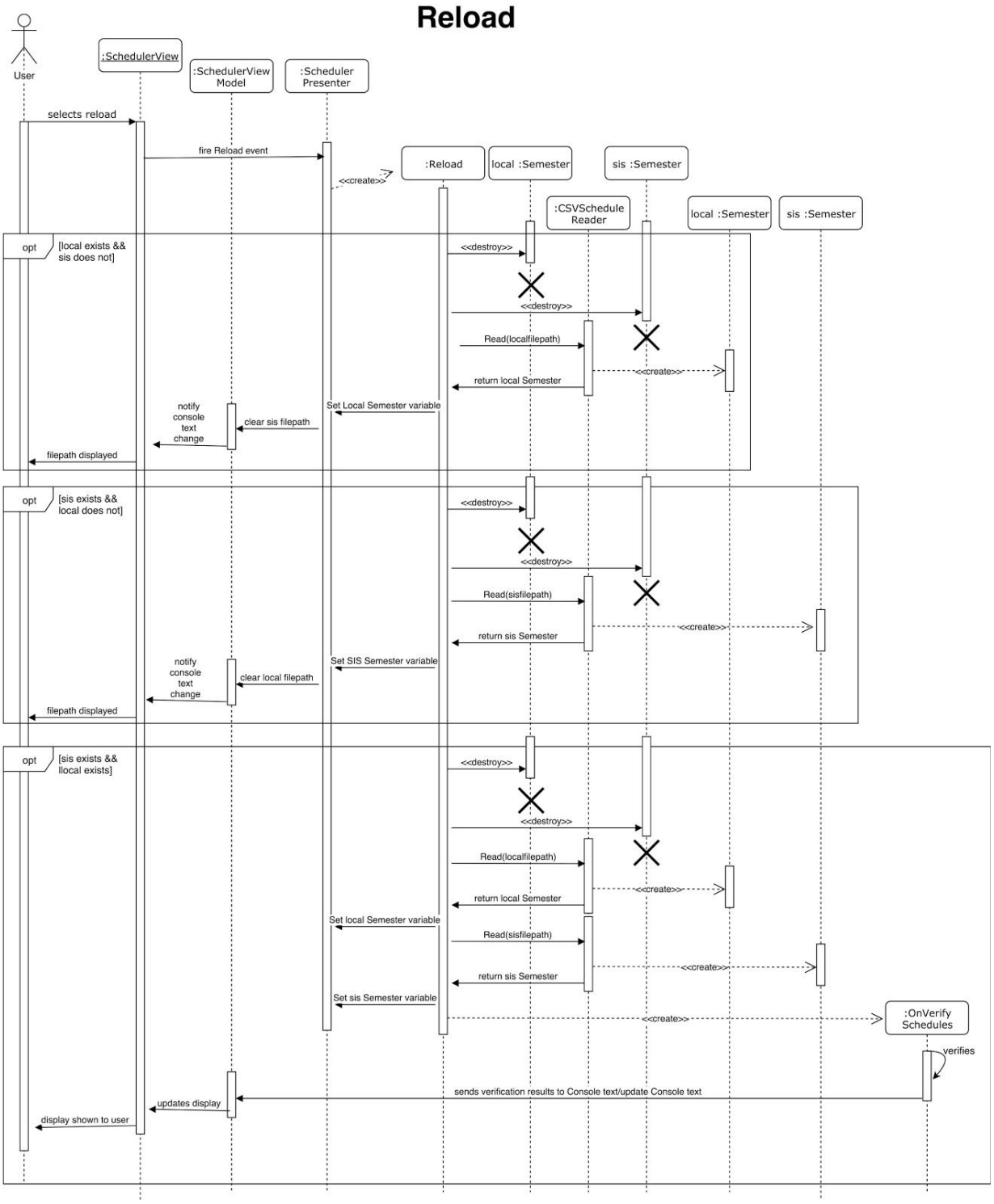
Cale Povilonis Andrew Fulton



Justifications:

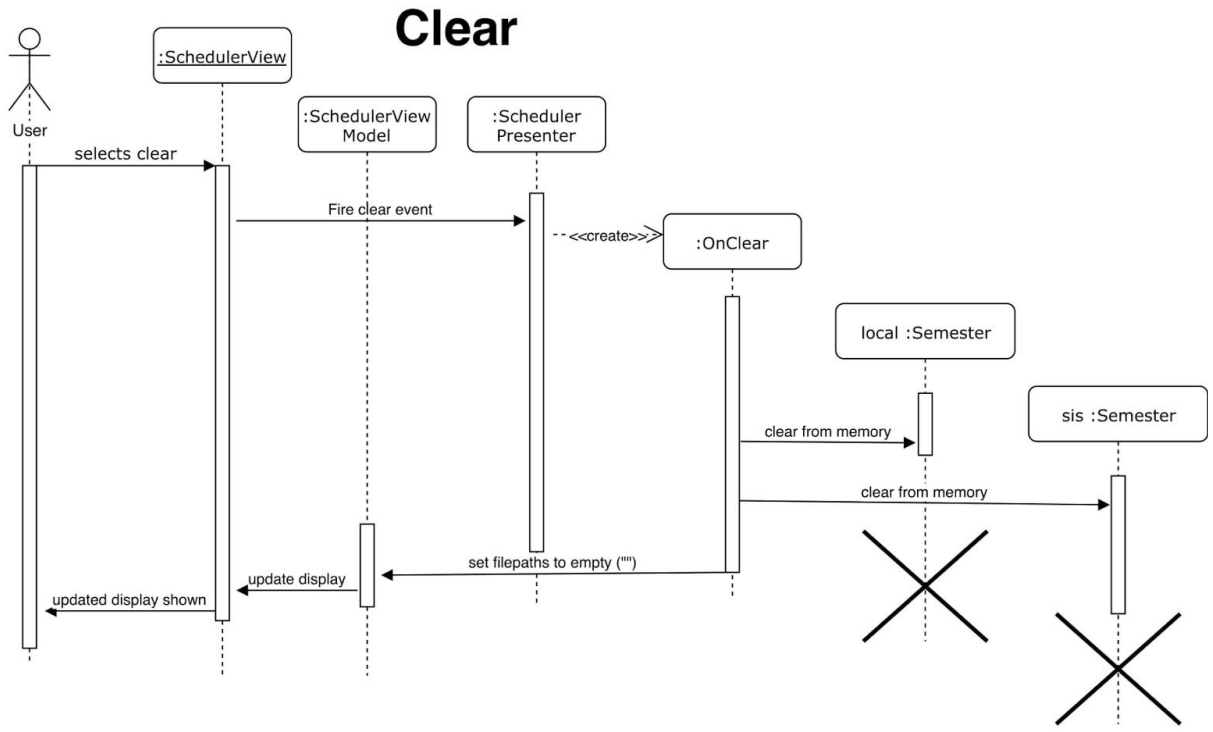
If the filename is not provided, we ask the user for it. Once the filename is provided, we fire the OnVerifySchedulesEvent which is subscribed to in the SchedulePresenter, and handled by the onVerifySchedules() method. This method then reads in the remote schedule using the RemoteSchedulePath. If an en exception occurs during read, we send an error message to the console and display it to the user. Once we have read the remote schedule, we save that in the schedulePresenter as the _remoteSchedule. Next, we set a boolean (called verify) telling us the overall result of the verification process. Now, getting to the actual verification, we loop through each scheduleConstraint given to the SchedulePresenter calling its Verify(..., …) method, passing in the local and remote schedules we loaded in earlier. If the constraint was unsucessful in verification, we send the constraint's error message to the console and set our verified boolean to false. Once we have looped through each constraint… If we successfully verified the 2 schedules, we send a successMessage by updating the ConsoleText in the viewModel which sends a property changed event to the view, updating the displayed console text. If we unsuccessfully verify the 2 schedules, we send a FailureMessage.

Cale Povilonis Andrew Fulton

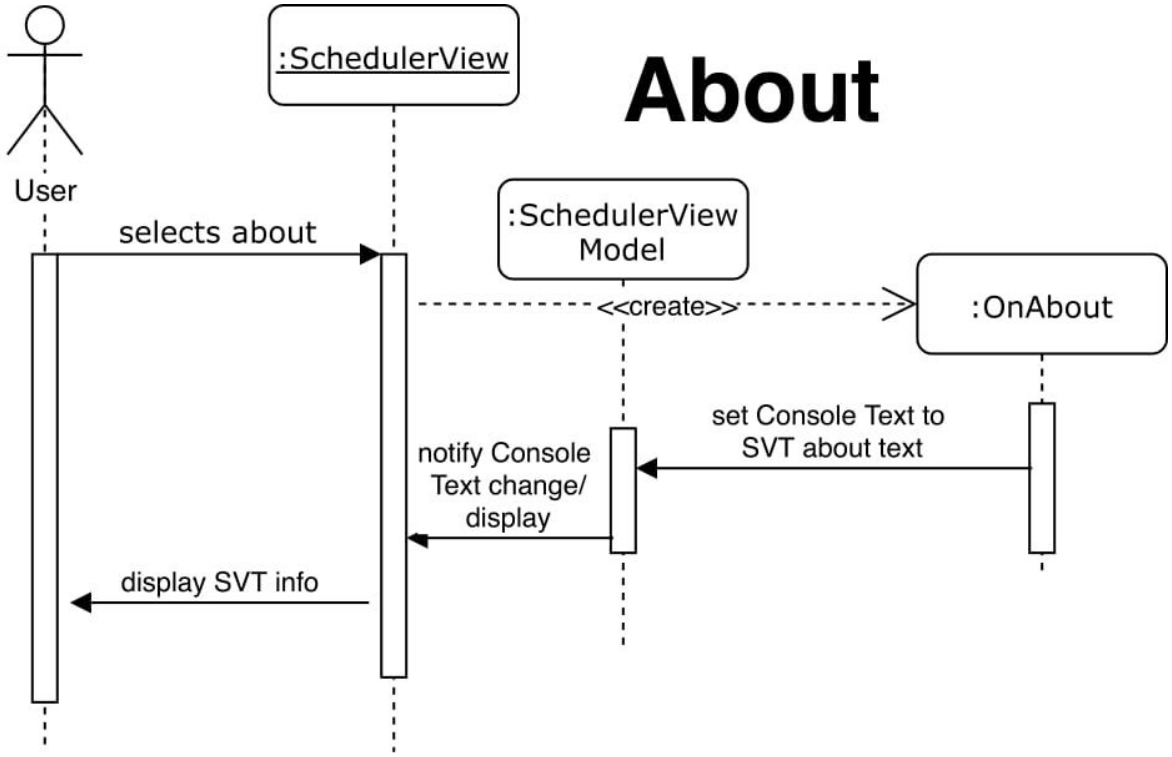


Justifications: To start the user selects reload on the UI which fires the reload event. This has the Presenter create the Reload function/method. Then there are three main opts: if the sis exists and local doesn't, if local exists and sis doesn't, and if both exist. If only one exists, then both of the current semesters will be deleted from memory and the filepath that exists will be

loaded in again thru the CSVScheduleReader and returned and set into the corresponding parameter in Presenter. Then, the filepath of the file that didn't exist will be cleared/set to empty. This change will be displayed and shown to the user. If both files exist, however, then both of the current semesters will be deleted from memory. Then, the local and sis semesters will be loaded in again with the CSVScheduleReader and stored in their corresponding parameters in the Presenter. Then, after both are loaded in again, Reload will create OnVerifySchedules to verify the two reloaded schedules. We have opted not to show that flow again as it has been shown in the VerifySchedules sequence diagram above. After the schedules have been verified, the results will be added to the Console text and displayed to the user.



Justifications: Firstly, the user selects clear. Then the SchedulerView fires the clear event to the Presenter. The Presenter then creates the OnClear function/method. Then, the local and sis semesters will be destroyed/cleared from memory. Finally, both the filepaths will be set the empty and cleared. The display is updated and shown to the user.



Justifications: The user selects about and the SchedulerView creates the OnAbout function/method. We do not have this function inside the presenter as it does not need to be there as all it does is display the SVT about information. This information is static and non-changing and has no business logic. The OnAbout method then sets the Console text to the SVT about info and the display is updated and shown to the user.