CIS 422 Project 2:
JTAS (Jaqua Tutoring Appointment Scheduler)
SDS

David Han (dh), Kassandra Morando (km), Kelly Schombert (ks), Brianna Vago (bv), and Mert Yapucuoğlu (my)

February 12, 2022 – v1.0

Table of Contents

1. SDS Revision History

02/12/2022	my	Created the doc, system overview started
02/13/2022	my	System overview mostly done, software architecture started
02/13/2022	dh	Added class diagrams to software architecture section
02/14/2022	my	Software Architecture done, software modules mostly complete
02/14/2022	dh	Added sequence diagrams to operational scenarios section
02/15/2022	my	Finished software modules after adding design rationale
02/15/2022	dh	Added Dynamic models and sequence diagrams for use cases
03/04/2022	my	Changed component names and general updates

2. System Overview

2.1. System Purpose

The system All in a Week's Work, which will be referred to as JTAS from now on, is a software program designed specifically for the John E. Jaqua Academic Center, with the purpose of automating the process of creating tutoring schedules for the large number of tutors and athletes. It will serve to provide two functionalities:

- Create a schedule meeting the requirements of all athletes
- Separate each individual tutor/athlete's schedule on demand.

The chosen method for a scheduling algorithm is to create many schedules with different athlete orderings and utilize a scoring system to determine which schedule has created the most optimal schedule with all required hours of tutoring scheduled.

2.2. Components

The JTAS contains 7 components to store data, execute the algorithm for scheduling, provide basic manager interaction UI, and functions for file management.

2.2.1. **ScheduleSystem**: The center component which initializes and controls the main

flow of the program.

2.2.2. **Scheduler**: The component responsible for making a schedule and managing the information about an individual scheduling variation while randomizing certain aspects of the data to create variation.

2.2.3. **FileI/O**: A module containing the functions that manage file input and output.

2.2.4. **Appointment**: Contains information about each tutoring appointment that is made by the scheduling algorithm.

2.2.5. **Tutor**: The module to define and use the information of a tutor in the system.

2.2.6. **Athlete**: The module to define and use the information of an athlete in the system.

2.2.7. **Manager Interface**: The UI component to manage tkinter windows to provide user interface to the manager.

2.3. Files

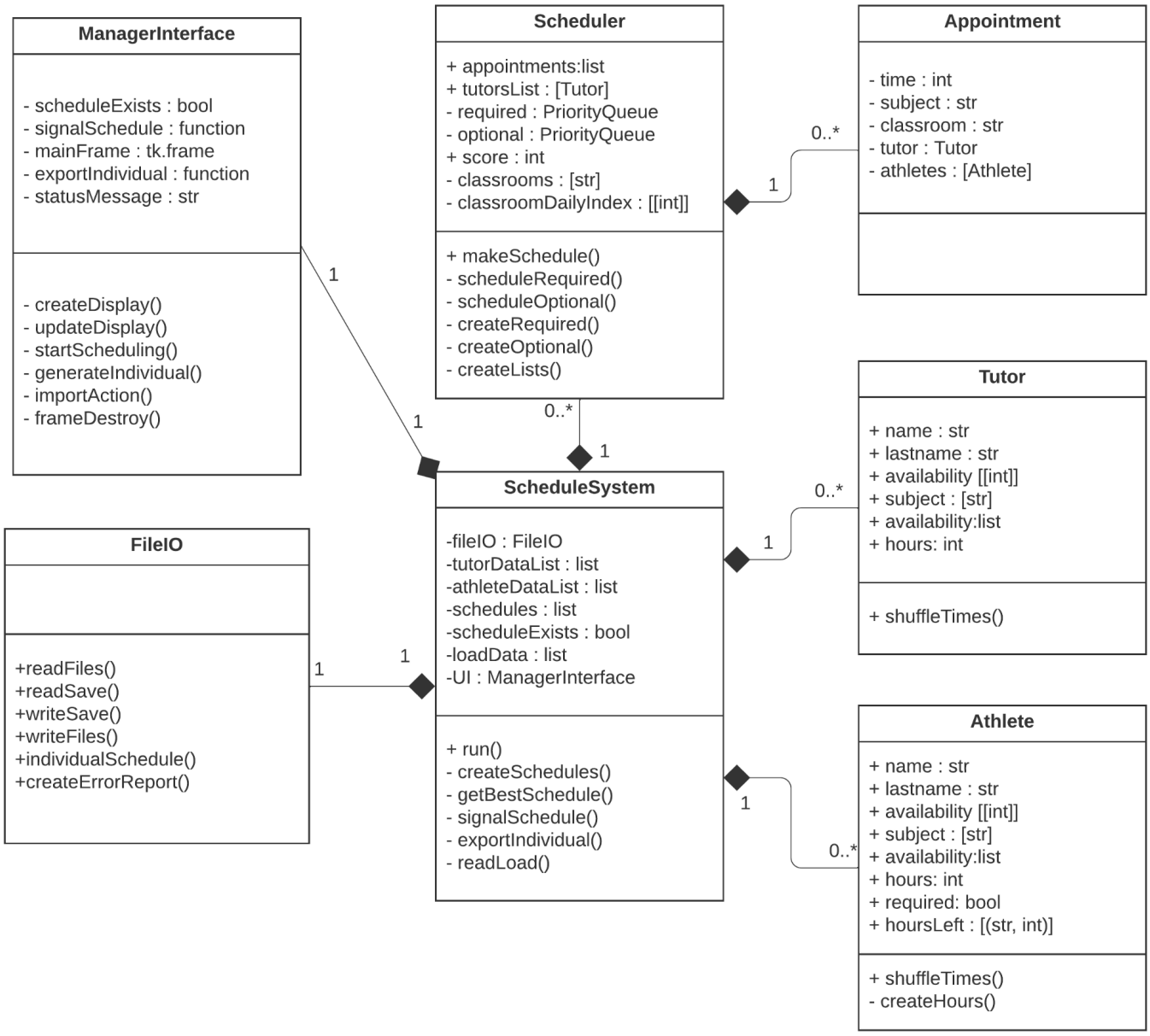
The JTAS system requires two .csv input files in the correct format in order to be able to create a schedule. These files contain:

- Information of all the tutors in the system including availability, subjects, max hours wanted.
- Information of all the athletes in the system including availability, subjects, hours needed, grade, and GPA.

If the scheduling succeeds and a working schedule is created, it will also be stored in the program directory as a .csv file. Another .txt file will be kept storing the appointments as raw that, for info retrieval when the program is run again.

The separate individual schedules for an athlete or tutor will be saved as a .csv in the program directory.

3. Software Architecture



UML Class Diagram for the JTAS

3.1. ScheduleSystem

The ScheduleSystem module is the skeleton of the program. It's purpose is the control the program states and flow depending on the UI interactions and the reaction of the program. It uses FileIO, creates Scheduler instances, interacts with ManagerInterface, and calls the functions of these modules to control the flow.

3.1.1. Component Interactions

- **FileI/O**: The ScheduleSystem calls the functions within the FileI/O to read files to get athlete and tutor data, write and save the final schedule after it has been created, read an existing schedule, and write individual schedules.
- **Scheduler:** ScheduleSystem module creates multiple Scheduler instances and gives each of them the tutor and athlete information. Then it calls the algorithm function in each Scheduler, waits for all of them to be done, and retrieves the score of each one after it is done. It then uses the score to pick which schedule is the best.

3.1.2. Design Rationale

Each program needs a main flow, and in the JTAS system, the execution of the program is driven by the ScheduleSystem module. It is the highest level component in the system, and as such, it initializes scheduler instances, calls their methods to build the schedule, chooses the best schedule, and uses the file read/write functions to interact with the file system.

3.2. Scheduler

The Scheduler component makes an individual schedule after receiving all the information needed to build an individual schedule, which are athlete and tutor information lists. Two priority queues of athletes are created to keep track of who needs how many hours to meet their requirements. The score that evaluates the optimality of a schedule is also contained as a variable.

3.2.1. Component Interactions

- **Tutor and Athlete:** Each row of data in the tutor and athlete file corresponds to an individual. Each scheduler feeds each row of data to either Tutor or Athlete and creates an instance of the class to store the data efficiently. The athletes are then queued into two priority queue structures, their priority defined by the number of hours they still need to fulfill.
- **Appointment:** Each time the scheduling algorithm matches a tutor and athlete at the same time for a tutoring session, the schedule creates an appointment instance and fills it with data to define the appointment made. These are then stored in the list inside that scheduler instance.
- **ScheduleSystem:** A scheduler will return its score back to the ScheduleSystem so that it can choose the best one to present to the user.

3.2.2. Design Rationale

A scheduling process consists of the algorithm, and the information about the tutors, athletes, subjects, classrooms, times, and all the appointments made. In order to contain all this information and also have the main algorithm to build a schedule, this module was created. A priority queue was used because the scheduler needs to know who is the athlete that needs the most hours of tutoring at all times.

3.3. FileI/O

This module consists of functions that read and write to/from the file system and validate the input.

3.3.1. Component Interactions

- **ScheduleSystem:** FileI/O is used by ScheduleSystem to read initial data of tutors and athletes from .csv files, and write the final schedule created into a .csv file. It also reads the most recent schedule on boot-up if it exists, and gives the manager the ability to export individual schedules. If the input files have errors in them,

fileIO is also responsible for creating an errorLog to help user fix their input.

3.3.2. Design Rationale

In order to read file input for the data of tutors and athletes, the program needs to interact with the file system. It also needs to be able to write a resulting schedule once the run is complete. Thus, this module was created to contain all the functions to interact with the file system.

3.4. Appointment

Appointment is a component that encapsulates information about a singular tutoring appointment. Multiple instances of this class are created and used by schedule instances.

3.4.1. Component Interactions

- **Scheduler:** A scheduler creates an instance of an appointment when it makes a tutoring appointment decision.

3.4.2. Design Rationale

A schedule can ultimately be represented as a set of appointments for a limited amount of time. Each schedule object will contain multiple appointment instances to keep track of the decisions it has made.

3.5. Tutor

Tutor is the component responsible for providing ease of access to a tutor's information. contains all the information about a single tutor in the system. When a scheduler instance is created by the ScheduleSystem module, the information given to the schedule is used to create instances of tutors. These tutors are then stored in a list in the schedule instance.

3.5.1. Component Interactions

- **ScheduleSystem:** Is used by the ScheduleSystem module to store data on a tutor.

3.5.2. Design Rationale

Since there is a lot of information that defines each individual tutor, this module's instances will contain them and ease their creation and access.

3.6. Athlete

Athlete is the component similar to Tutor in the way that it defines a single athlete in the system. It is also used to create a list of athletes in the Scheduler module, and its functions are used to randomize order of certain data in the class to provide random scheduling.

3.6.1. Component Interactions

- **Scheduler:** The schedule module creates a priority queue using the hours needed for each athlete, and uses a randomizing method to distinguish those that need the same amount of tutoring hours, so that each schedule instance returns a different scheduling result.

3.6.2. Design Rationale

Same reasons with the tutor module, but for athletes.

3.7. Manager Interface

Manager interface is the UI of the system used by the ScheduleSystem module in order to prompt the user for the input files required by the system.

3.7.1. Component Interactions

- **ScheduleSystem:** The ScheduleSystem initializes the manager interface and calls the corresponding method from it that creates a file input window for the user. It then accepts the file path provided and tries opening the files. If successful, the program continues, if the file open fails or the file is not in the required format, the user is notified and is re-prompted. After a schedule is made or found on a save, the user can provide a student id to export an individual schedule from the main schedule.

3.7.2. Design Rationale

The program needs to be able to accept the files it needs without forcing the user to use the file system to directly move them to the program directory. This is why the manager interface is made to provide a UI where the user can choose input files at the start of program execution.

4. Software Modules

4.1. ScheduleSystem

4.1.1. Role and Primary Function

The ScheduleSystem module contains the main execution flow, and is responsible for being the interaction between modules that create the software. It provides the following functionalities:

- Checking for existing schedule
- Accepting data file paths for new scheduling
- Reading data
- Creating Schedulers
- Retrieve schedules and compare scores
- Save the best schedule as a file
- Save individual schedules as a file
- Create error logs

4.1.2. Module Interface

Check Recent Schedule:

Upon program boot, the ScheduleSystem uses fileIO functions to see an already made schedule exists in the program files. If it exists, it is read and loaded back into the system for individual schedule export.

Accepting Data Files:

After checking save files, the ScheduleSystem module instantiates a Manager Interface. If the user decides to create another schedule, they must specify the locations of the input files containing the data for athletes and tutors. The manager interface provides this functionality, and returns the absolute paths to the files to the ScheduleSystem.

Read Athlete and Tutor Data:

If the user decides to create a schedule and provides the data files, the ScheduleSystem uses the FileIO functions to read and fetch the data contained. If this process fails in any way, the manager interface is used to notify the user and ask for new input.

Create Schedulers:

If the data input is valid, the program creates a certain amount of Scheduler instances. Then the scheduling method is called on all the schedules, making multiple different schedules.

Retrieve and Compare Schedulers:

Once the Scheduler instances finish their algorithms, the ScheduleSystem retrieves their scores and compares them to pick the highest.

Saving the Best Schedule:

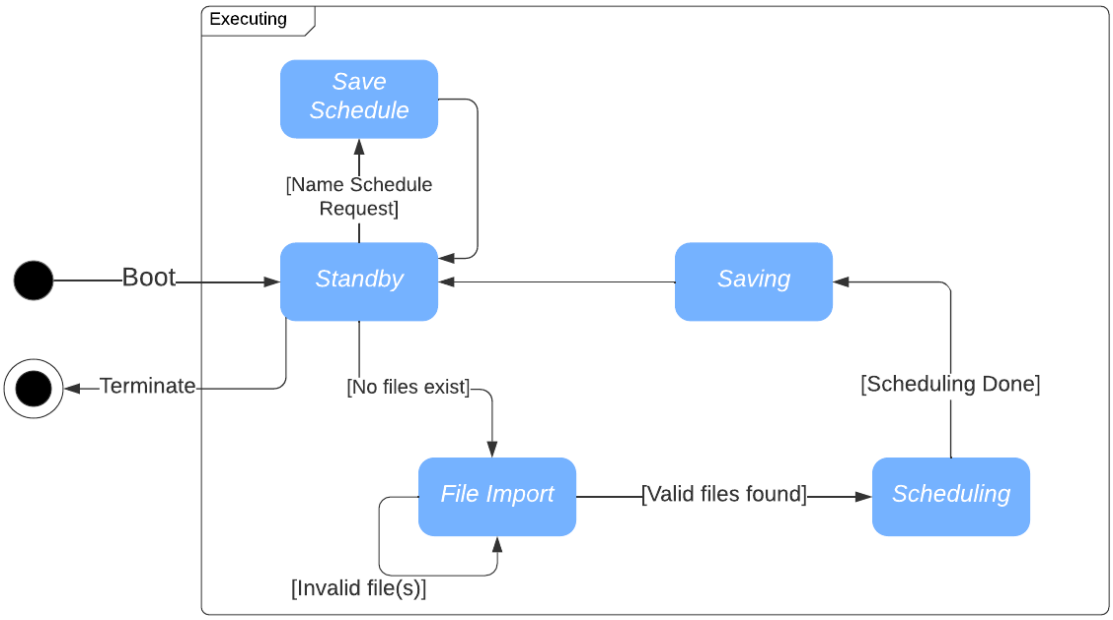
The best Scheduler out of all is saved in two files using the FileIO functions. One file is a .csv file that shows the schedule in a day-of-the–week and time table. The other file is a text file that contains the information about all the appointments that define the schedule.

Saving Individual Schedules:

After the scheduling is done and the saving is complete, the manager can export the schedule for an individual before the program exits or at the beginning of the next program session. If the user chooses this option from the Manager Interface, they must provide a name and lastname. This name is accepted by the ScheduleSystem module, which then looks through the appointments list of the current best schedule, and finds the appointments that include the name and lastname in it and saves them to a list in the ScheduleSystem. Then uses fileIO functions to create the schedule file with these appointments.

4.1.3. Static Model

4.1.4. Dynamic Model



UML State Diagram for the ScheduleSystem Module

4.1.5. Design Rationale

The reason the ScheduleSystem module undertakes many tasks is to make sure that other modules don't have to communicate with multiple modules and thus have to define multiple interfaces. This way they only communicate through the ScheduleSystem, and only the ScheduleSystem must be changed to re-define the interactions. One other reason is that the JTAS has different states of service, and a module must keep track of which one the system is in, thus we found it fitting that the main module of component interaction is the tracker of the state as well.

4.2. Scheduler

4.2.1. Role and Primary Function

The Scheduler module contains tutor, athlete, classroom information, and uses this information with the algorithm to build a schedule. It is responsible for:

- Creating randomized priority queues of athletes
- Making the schedule
- Scoring the decisions taken
- Creating Appointments

4.2.2. Module Interface

Creating Athlete Priority Queues:

Scheduler will create a list of athletes, tutors, and classrooms with the data from ScheduleSystem. The Scheduler module will take the list of athletes, and depending on their school year will divide them into two lists called required and optional. Since all the athletes in the required list have a required 8 hours of tutoring, a unique random decimal will be added to each to distinguish them and create a random ordering. Then both lists will be fed into priority queues that use the hours needed as priority.

Making the Schedule:

After the priority queues have been created, the ScheduleSystem module will call the make method of the scheduler to make the schedule. This means going over the required priority queue and creating appointments to each athlete in order until the queue is empty. Then the optional priority queue is emptied the same way.

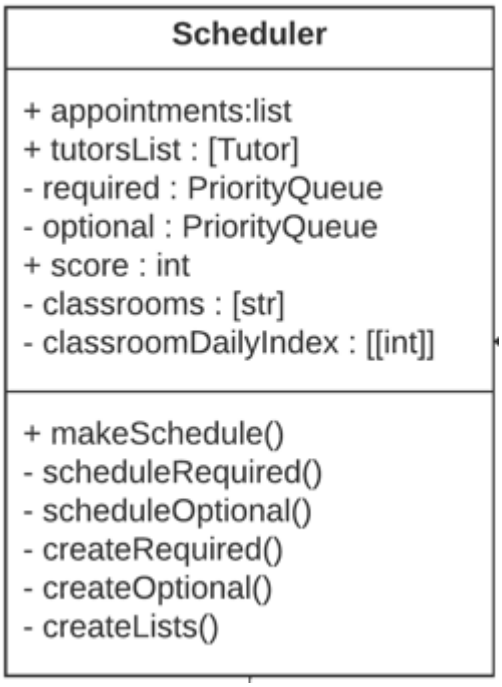
Scoring the Decisions:

Each time an appointment is made for an athlete, the score counter will be increased depending on the time and hours.

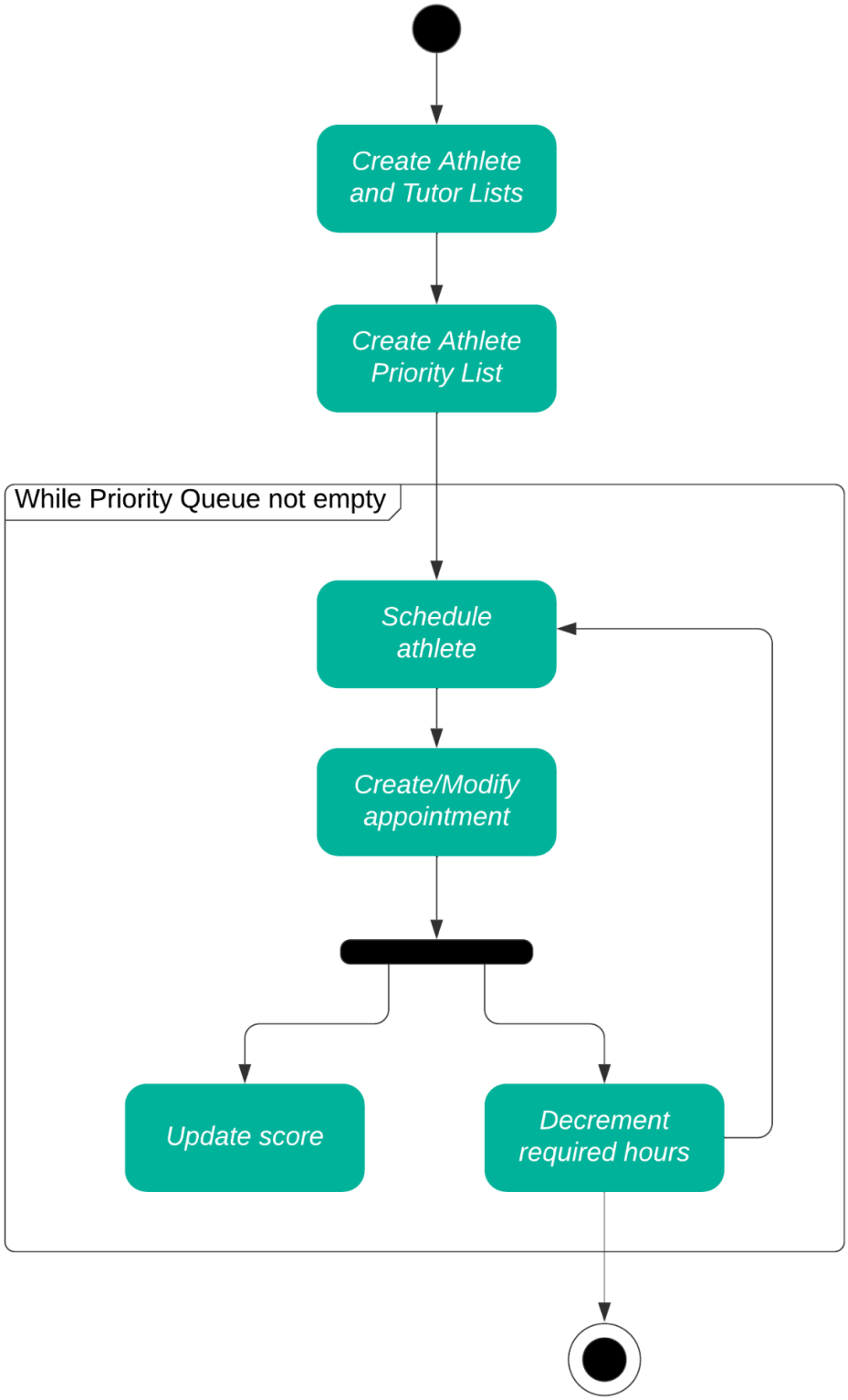
Creating Appointments:

The schedule maker algorithm will loop through the queues of athletes, check the subjects they need, look through the tutors and find a fitting tutor with the same subject and available time, and match them. Upon matching, an appointment instance is created with all the information about the tutoring session, and the appointment is added to the appointment list of the schedule object.

4.2.3. Static Model



4.2.4. Dynamic Model



UML Activity Diagram for Scheduler

4.2.5. Design Rationale

The main purpose of the JTAS is to create a schedule, and for a schedule, a scheduling algorithm must be used. Because the system tries to find the most optimal schedule by calculating many schedules with the same data, a class had to be made to contain the tutor and athlete data and then schedule with that data. This way the system can have multiple schedules that are built at the same time with the algorithm.

4.3. FileI/O

4.3.1. Role and Primary Function

The FileIO is the module with a set of functions that manage the reading and writing required by the software. The files it interacts with are:

- Athlete and Tutor Information Files
- Schedule Save Files
- Individual Scheduler Save File
- Create Error Log File

4.3.2. Module Interface

Athlete and Tutor Information Files:

If the ScheduleSystem module calls the function to read the files, a file path is passed to the fileIO, which opens the file and reads it. If it fits the required format, a list containing list of information about each individual is passed to the ScheduleSystem. If the file does not fit the format, an error is returned.

Scheduler Save File:

If the ScheduleSystem is successful and wants to save the made schedule, the data that defines the schedule is passed to the fileIO function when it is called by the ScheduleSystem. The fileIO takes the parameter and creates a .csv file. It then creates a .txt file that contains a list of appointments in the schedule so

that it can be used later on for individual schedule creation.

Individual Scheduler Save File:

If the ScheduleSystem uses the fileIO function to save an individual's schedule, it creates a separate .csv file to store the schedule.

Creating Error Log:

If the input file process returns and error, the ScheduleSystem will call fileIO functions to create an errorLog.txt for the user to see which lines are faulty.

4.3.3. Static Model

4.3.4. Design Rationale

Taking inputs for the scheduling algorithm is best done with files, .csv per our choice because that is the most commonly used. FileIO contains the set of functions that do the file reading and writing.

4.4. Appointment

4.4.1. Role and Primary Function

An appointment is the component class that structures the information to define a tutoring session. It has one interaction.

- Check if the appointment has an empty spot

4.4.2. Module Interface

Check for Empty Spot:

In the case that an athlete can be added to an already existing appointment, the scheduling algorithm checks to see if the appointment has a spot for 1 more athlete (max:3), and returns a boolean as an answer. If it is true, the athlete is added to the appointment.

4.4.3. Static Model

4.4.4. Dynamic Model

4.4.5. Design Rationale

Because the existing tutoring system can appoint 3 athletes to a tutor in the same tutoring session, the JTAS also needed some data type to keep track of already existing tutoring appointments. The appointment instance will be created multiple times for each of the scheduler instances to keep track of made appointments.

4.5. Tutor

4.5.1. Role and Primary Function

Tutor is the component that encapsulates data about a single tutor in the system. It provides one interface.

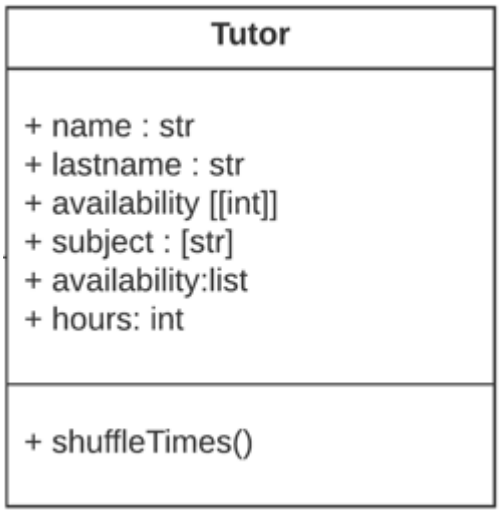
- Check if a tutor is available

4.5.2. Module Interface

Check Availability:

In order to find a fitting tutor, the scheduler algorithm will check each tutor in the list and call the function from the tutor class to check if they are available at that time for that subject. If true, an appointment is made.

4.5.3. Static Model



4.5.4. Dynamic Model

4.5.5. Design Rationale

Because a tutor has 5-6 attributes, some of which are 2-D arrays, a component had to be made to better manage and define stringy and comparison functions.

4.6. Athlete

4.6.1. Role and Primary Function

Athlete is the component that contains the information about a single athlete and randomizes them accordingly to create varying schedules. Like Tutor, it has an interface for availability and subject information.

- Retrieve remaining required hours

- Retrieve availability
- Randomize availability and subject order

4.6.2. Module Interface

Retrieving Remaining Tutoring Hours:

During the loop on the scheduling algorithm, the program will try to match a tutor with the athlete with the highest hour need, and to determine which subject is needed, a method will be called by the scheduler to get the subject list of an athlete.

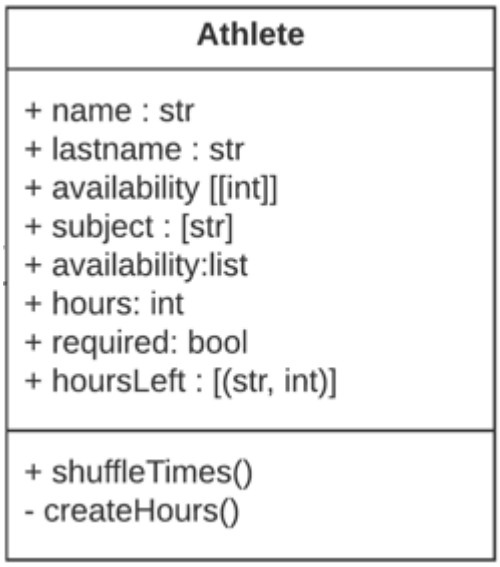
Retrieve Availability:

Even though the schedule instance know which athlete needs the most hours, it needs to ask the athlete object directly to retrieve the next available time. It will call the athlete method to get the next available time for the athlete.

Randomize Availability and Subject Order:

Upon initialization, the order of daily availability is shuffled, along with the subject list, for each athlete. Because athletes are created separately in each scheduler instance, the randomness contributes to different schedules being made.

4.6.3. Static Model



4.6.4. Dynamic Model

4.6.5. Design Rationale

A component to contain the information of a single athlete was necessary in order to quickly find out the availability and requirement of each athlete as the scheduling algorithm proceeded. It is also the main feature of the JTAS that it creates different schedules and chooses the best out of it, and this is ensured by the availability and subject list shuffling done in athletes of each scheduler.

4.7. Manager Interface

4.7.1. Role and Primary Function

The manager interface is the graphical interface of the JTAS. It is always present on the display as long as the program is running. It facilitates user interaction for:

- Accepting input files
- Taking name input for individual schedules
- Displaying the program state

4.7.2. Module Interface

Input File Import:

If the program is being run for the first time, or the user chooses to make a new

schedule, “import files” button present on the display can be pressed. This will bring two consecutive file selection windows for the user to choose athlete data and tutor data files. After this process is done, the file paths are sent to the ScheduleSystem for verification and reading.

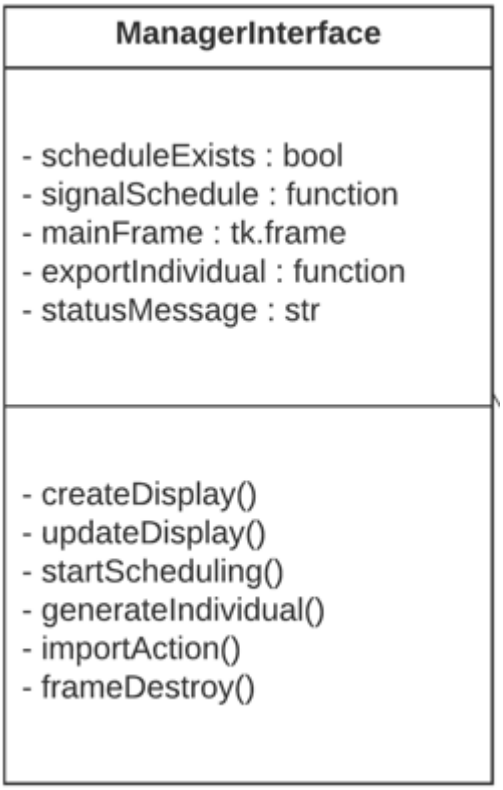
Name Input Field and Individual Schedulers:

If a schedule is made or is saved by a previous run of the program, the manager interface will present a text field for the user to fill to request the schedule for a specific tutor or athlete. After the field is filled and the generate button is pressed, the name is sent to the ScheduleSystem which will create a new .csv file for the individual’s schedule.

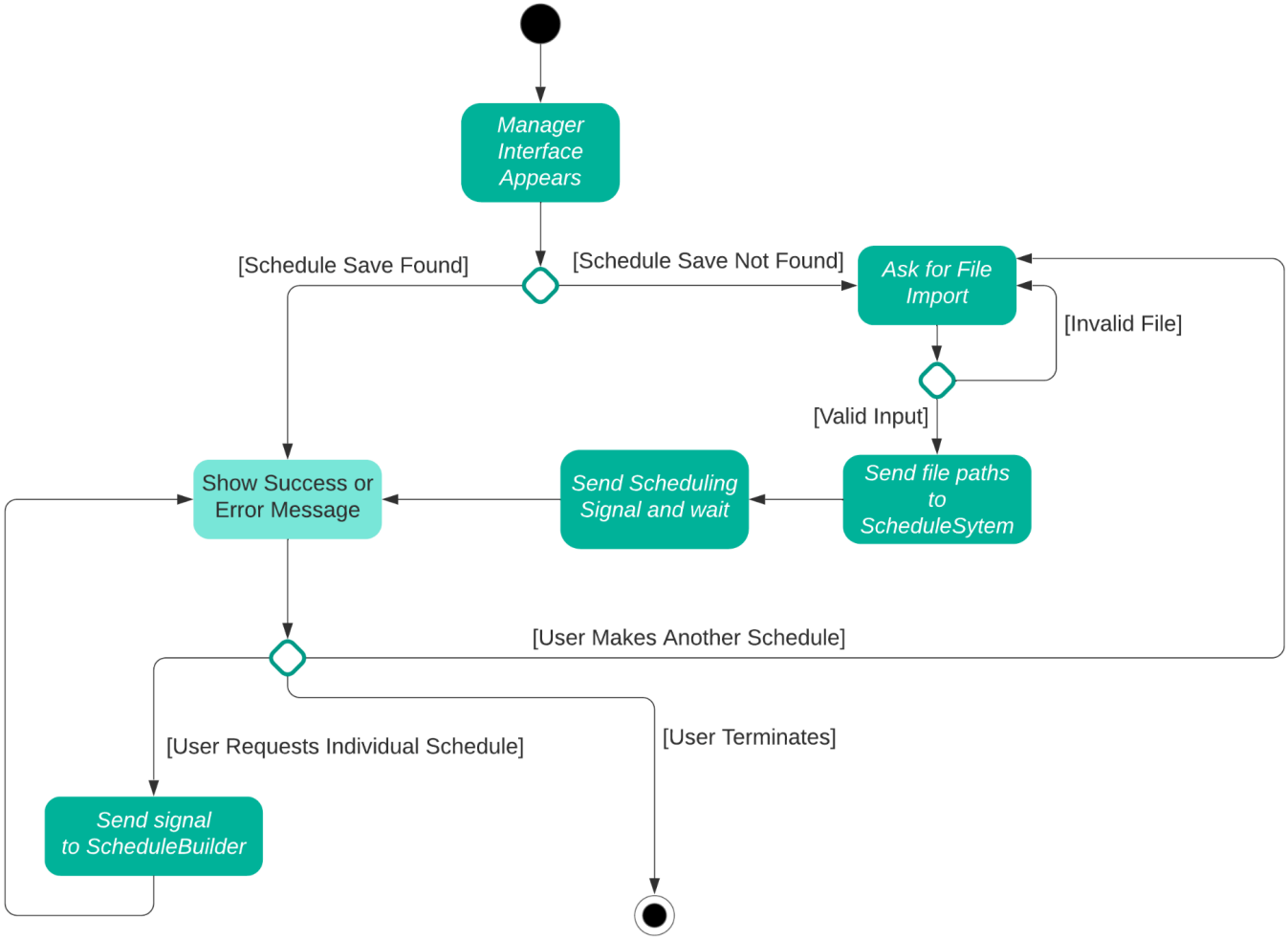
Program State Display:

The program has 4 states, import, scheduling, saving, export-ready. Depending on which one the ScheduleSystem is currently in, the Manager Interface will display an according to message.

4.7.3. Static Model



4.7.4. Dynamic Model



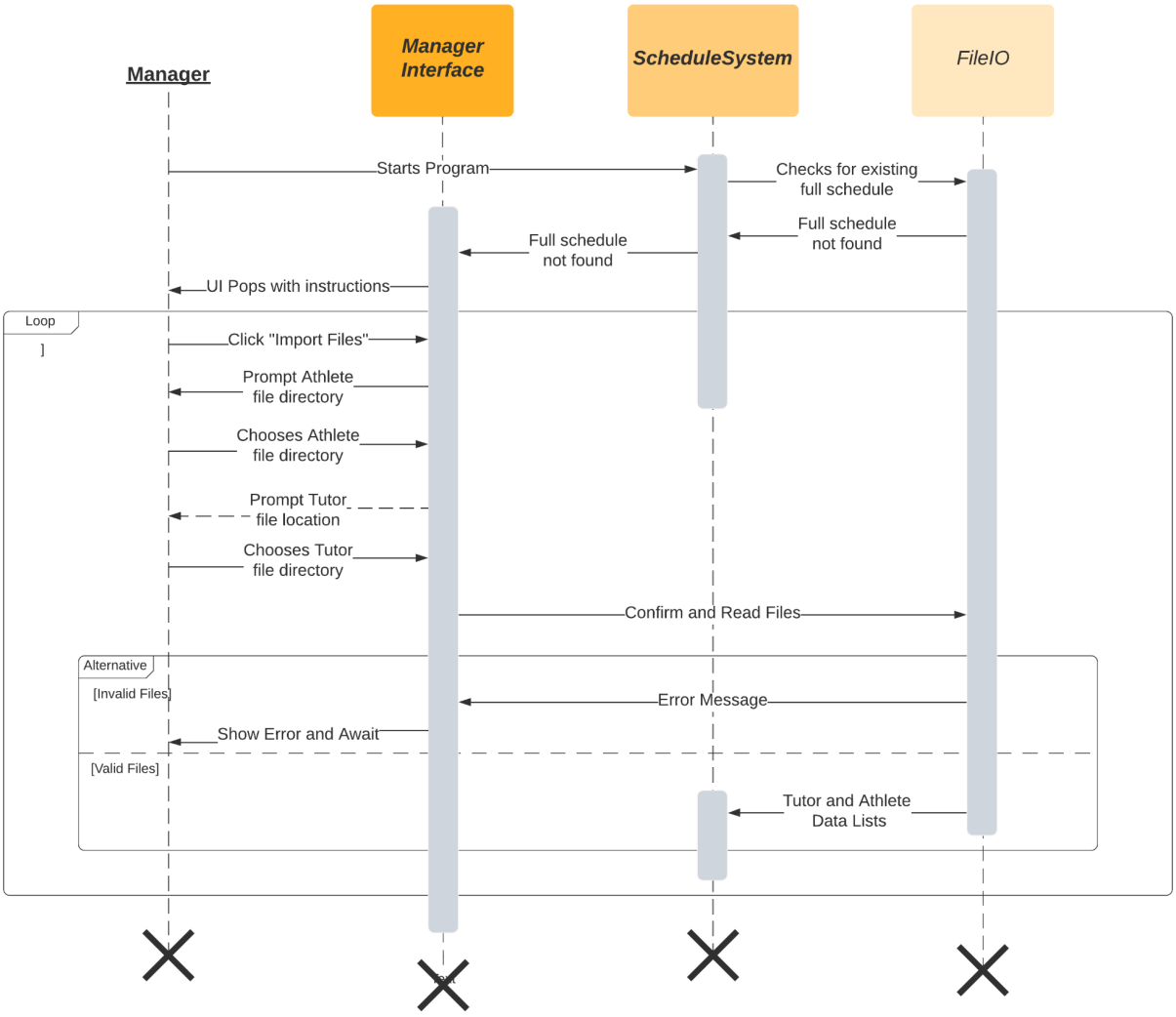
UML Activity diagram of Manager Interface

4.7.5. Design Rationale

The JTAS allows the manager to re-do schedules on-demand and extract individual's schedules with a name input. To provide these functionalities, buttons were needed to be shown to serve as a trigger for the different states. Since the program is expected to take longer than instantaneous execution, a graphical interface to display the current state of the system was also deemed necessary by the developer team.

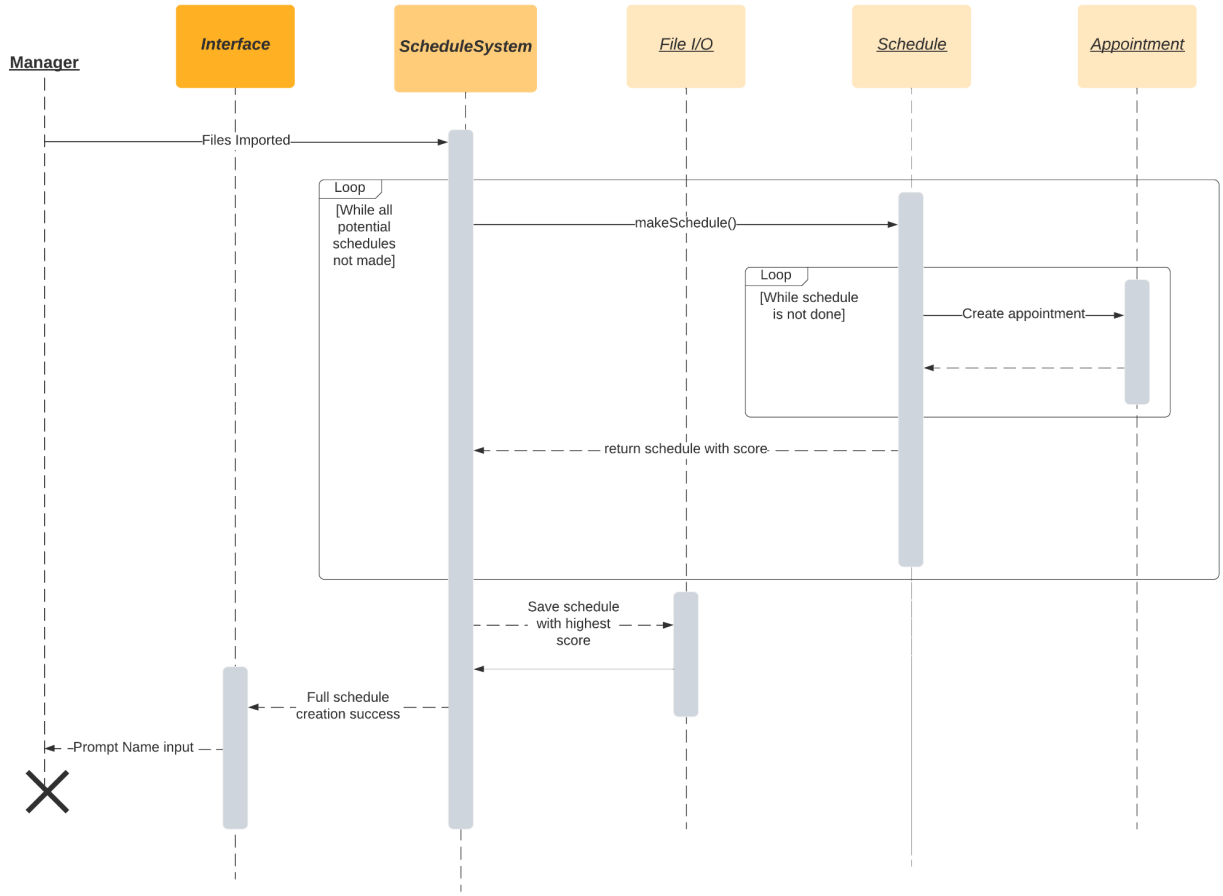
5. Dynamic Models of Operational Scenarios

Use Case 1: Accepting File Imports



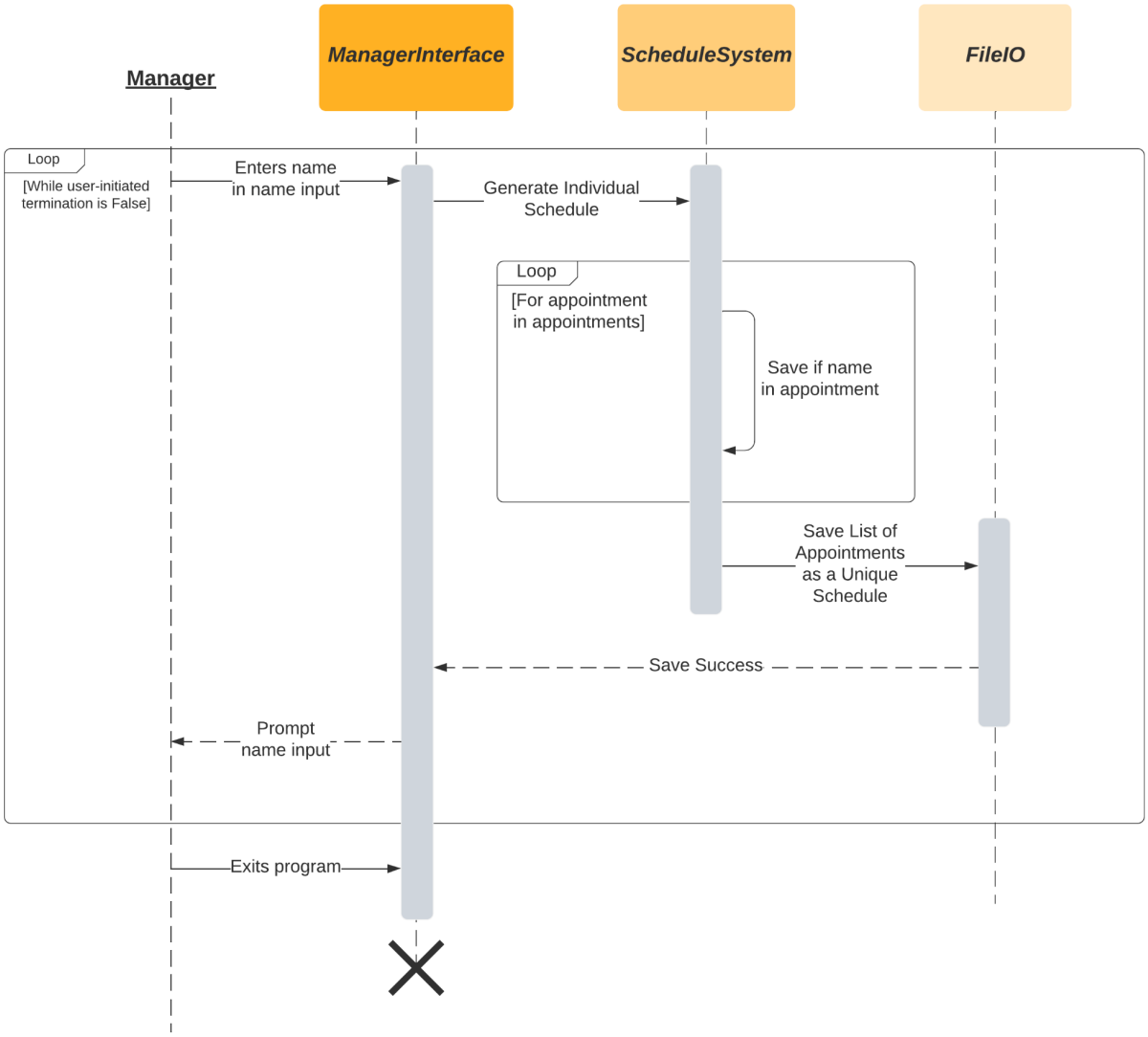
UML Sequence Diagram for File Import via Manager Interface

Use Case 2: Building Schedulers



UML Sequence Diagram for the Scheduler-Making Process

Use Case 3: Exporting Individual Schedulers



UML Sequence Diagram for exporting individual schedules

6. References

SDS Design Template - Professor Anthony Hornof:
https://classes.cs.uoregon.edu/22W/cis422/Handouts/Template-SDS.pdf

Class Website Miscellaneous Links section - Kieras and Fowler UML notation

Page 46 of Sommerville, Software Engineering. “The Rational Unified Process” - for information on static and dynamic models

7. Acknowledgements

The formatting of this document was based on a Software Design Specification template provided by Professor Anthony Hornof.