McMaster University

CAS 4ZP6

Team 9

Capstone Project 2013/2014

Porter Simulation

Design Revision 1

Authors:
Vitaliy Kondratiev - 0945220
Nathan Johrendt - 0950519
Tyler Lyn - 0948978
Mark Gammie - 0964156

Supervisor:
Dr. Douglas Down

April 16, 2014

CONTENTS

1 Revision History

Revision #	Author	Date	Comment
1	Vitaliy Kondratiev, Nathan Johrendt, Tyler Lyn, Mark Gammie	January 11, 2014	Revision 0 Added to repository
2	Vitaliy Kondratiev, Mark Gammie	January 12, 2014	Added diagrams to Latex
3	Tyler Lyn	January 12, 2014	Added dispatch module
4	Vitaliy Kondratiev, Nathan Johrendt	January 13, 2014	Added GUI Component to document
5	Mark Gammie	January 13, 2014	Updated Dependency Diagram to contain algorithms and data structures. Updated dependency diagram with a legend, logging comment and fixed the porter dispatcher dependency
6	Tyler Lyn, Mark Gammie	January 13, 2014	Several Modules Updated
7	Nathan Johrendt	January 13, 2014	Added Appendix
8	Mark Gammie	January 14, 2014	Completed Design Overview
9	Mark Gammie	April 16, 2014	Updated Design Overview. Updated Dependency Diagram. Updated several modules.

2 Executive Summary

2.1 Introduction

This document outlines the design decisions, style and methodology for the project of Porter Simulation to be completed for Hamilton Health Sciences. A modular design methodology has been chosen as our design principle. This document is based on IEEE Draft Standard for Software Design Descriptions (IEEE P1016/D5.0).

2.2 Purpose

The purpose of this document is to outline the design of each component and how they interface between each other. This document will aid the developers in the development process as well as any future maintenance required.

2.3 Design Overview

Our solution is a discrete event simulation that is implemented using the Python library SimPy. The simulation has three main sections named UI, Export and Core.

The UI section contains the modules for providing the user interface. The UI is responsible for taking input from the user as well as preventing them from entering erroneous data/ ensuring integrity of the data before releasing it to the simulation core..

The Export section contains a module for reporting statistics that are generated by the simulation. The module will be used after the simulation has completed to write the generated data to a csv file that will be interpreted later by some Visual Basic scripts. These scripts will be used to generate predetermined graphs for the user to view.

The Core section is where the simulation actually occurs. The module Simulation Core is responsible for initializing the Simulation State, Job List Builder and Dispatcher modules. The Job List Builder module will create a list of jobs using the data contained in the file filled with historical data. This list of jobs will be submitted to the Simulation State to be stored.

Once the simulation is started, the Dispatcher module is responsible for receiving the jobs as they become available and assigning them intelligently to any eligible porters. The porters that operate on the tasks follow a finite state machine that creates timeouts based on the time it would take for each state that the porter is transitioning into. There may be cases when all the porters are busy and the amount of assigned jobs piles up, or the porters are stuck waiting on jobs to become available. Once the simulation terminates, the export of the statistical data is finalized.

3 Implementation Material

3.1 Language of Implementation

Visual Basic in Excel will be used to interface GUI to Simulation Core module. Python Version 2.7.5 will be used for Simulation Core module and all other modules.

3.2 Supporting Technology and Frameworks

Simulation will be built on the SimPy 3.0.2 library
https://simpy.readthedocs.org/en/latest/. GUI will be built in Excel 2010.

4 Process Diagram

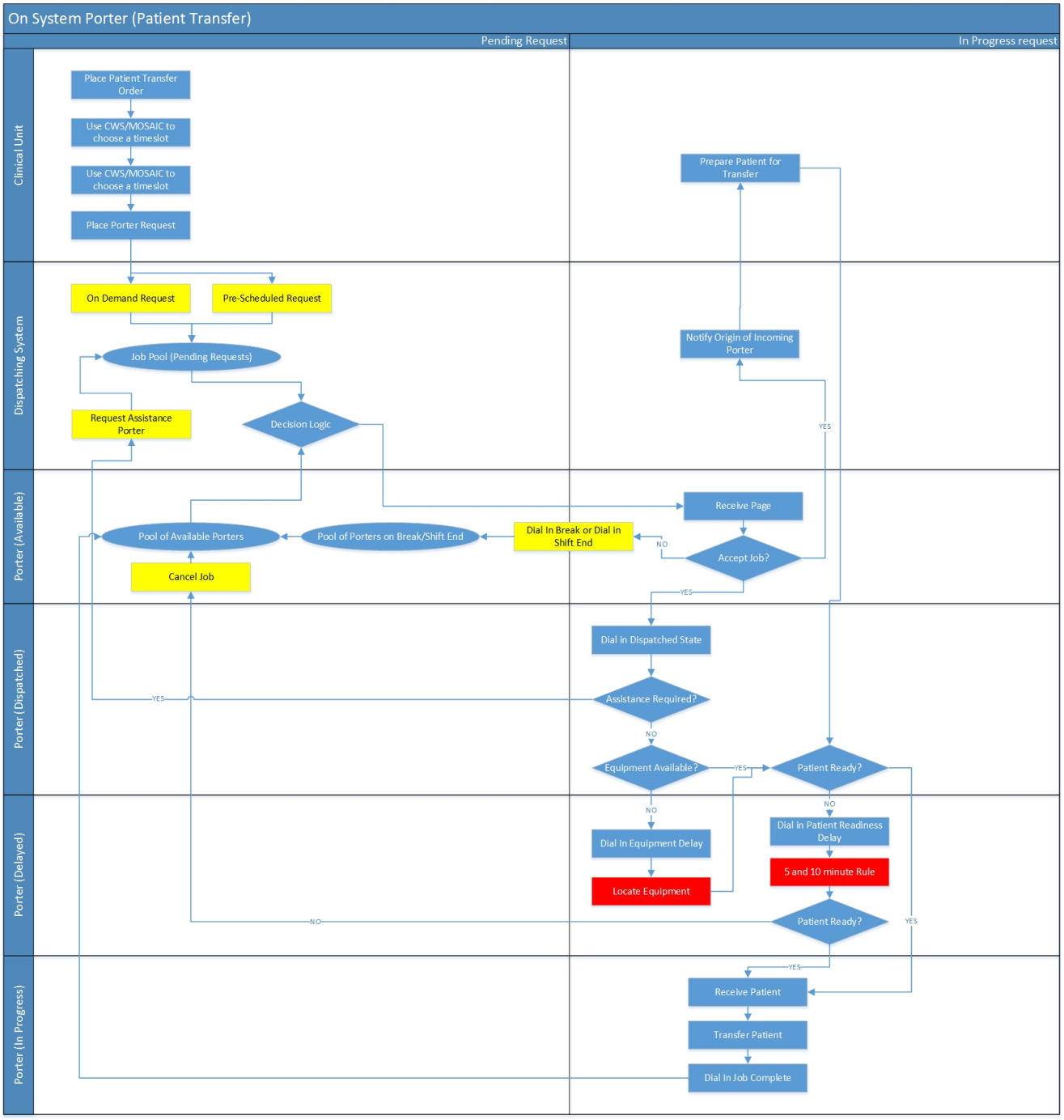

Figure 4.1: Process Diagram

5 Dependency Diagram

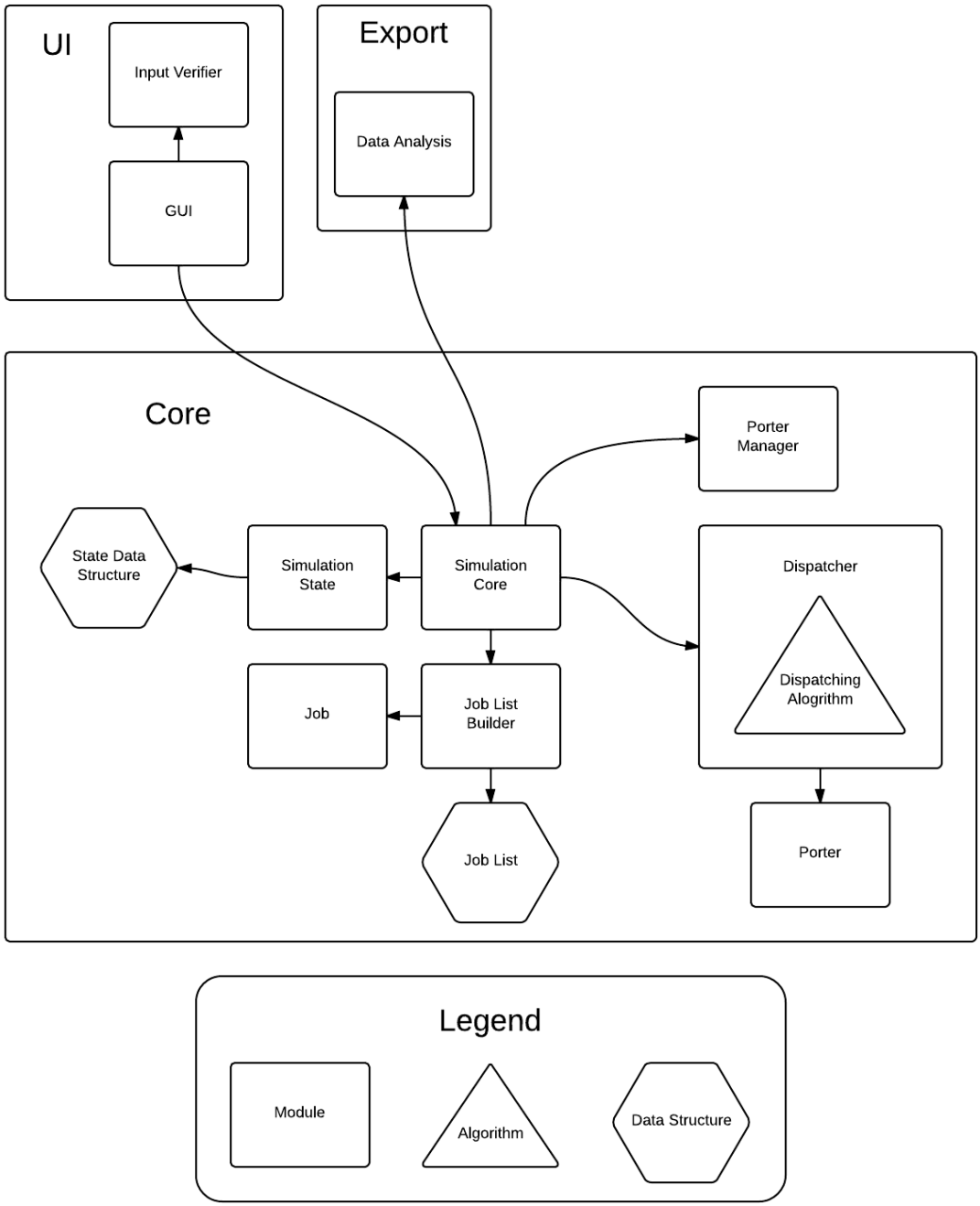

Figure 5.1: Dependency Diagram

6 Decomposition Description

6.1 Core - Simulation Core

Type: Module

Purpose: This module calls the required functions to create the Job List, initialize the simulation state and initialize the simulation processes.

Function: This module takes in configuration data from the GUI module and uses it to initiate the required modules Job List Builder, Porter Manager and Dispatcher, and passes their resulting data into the Simulation State.

Interface:
The interface to this module is a dictionary containing the information for the configuration of the simulation.

Process Steps: This module is passed the configuration data from GUI module. With the configuration data it constructs the Job List, Porter Manager, Dispatcher which will be stored in the Simulation State. Once that has been done, the module will run the simulation for the configured amount of time. At this point, the simulation has completed and the module finishes by running the Statistical Data Export module.

Data: None

Error Handling: Catch all on the simulation loop to report pertinent errors and to prevent unexpected termination.

Requirement Reference: 9.1.3

6.2 Core - Simulation State

Type: Module

Purpose: This module's purpose is to be an interface between the simulation and its simulation state data structure.

Function: This module will contain several objects that will be used by the many modules that are in the simulation.

Interface:
env:

- A reference to the simpy env object.

porterManager:

- A reference to the porterManger object.

dispatcher:

- A reference to the dispatcher object.

jobList:

- A reference to the jobList object.

maxDelayReason:

- A dictionary used to look up the max delay for each delay reason.

Process Steps: This module handles the simulation state data structure and allows the rest of the modules to interact with it. If modules need information about the simulation as a whole, they will communicate with the Simulation State module in order to satisfy their needs.

Data: Simulation state data structure

Error Handling: Catch any null values in the simulation state data structure

Requirement Reference: None

6.3 CORE - JOB

Type: Module

Purpose: Provide information about a job to the porter.

Function: The task is declared with an origin, destination, priority, appointment, delay information and state change time values. This information is necessary to allowing a porter to complete a job.

Interface:
autoUpdateProcess(simState, au):

- A process that increases the job's priority as it waits to be assigned to a porter based off of the au input.

Process Steps: Jobs will be initialized within the Job List Builder. An initialized job will be able to provide the Dispatcher all the required information to assign jobs. Once a job is given to a porter, the porter will then be able to start and complete a transport job.

Data: Transport Job Information

Error Handling: Catch any null values returned by the task's properties

Requirement Reference: None

6.4 CORE - JOB LIST BUILDER

Type: Module

Purpose: Produces the list of jobs for the Simulation Core and Dispatcher modules to process

Function: This module parses the file that contains historical job data and creates list of jobs that are ordered based of their creation time.

Interface:
runImport(fileName):

- Imports the data from historical job data file and constructs and returns the jobList along with the dispatchTable.

addDispatchTime(origin, dispatchTime)

- Adds a dispatch time to the dispatchTable

Process Steps: The Job List Builder will first receive the historical data file's file name from the Simulation Core. This file name will allow the Job List Builder to create jobs based off of historical data. Secondly, the Job List Builder will generate jobs and insert them into the jobList data structure. The jobList is responsible for the storage, ordering and releasing of jobs into the dispatcher.

Data: jobList

Error Handling: Catch any null tasks

Requirement Reference: 9.1.1

6.5 CORE - PORTER

Type: Module

Purpose: To complete jobs provided by the dispatcher

Function: Completes the transport jobs assigned by the dispatcher. Unless a job is cancelled the porter will traverse through four states ('pending', 'dispatched', 'inprogress', 'complete')

Interface:
work(simState):

- Input the Simulation State
- Finite state machine that causes the Porter to ultimately be locked to the job for the duration of its execution. Will also cancel a job and reschedule it if the porter is waiting too long on the patient. The function will also update state values of the job for storage of the generated statistics.

Process Steps: The module is responsible for representing Porters in the simulation and their ability to perform jobs and perform those jobs according to . This is where the majority of statistically valuable information is generated.

Data: Stores internal data relating to its' current state.

Error Handling: None

Requirement Reference: None

6.6 CORE - DISPATCHER

Type: Module

Purpose: To organize pending jobs based on a weighted-value and assign them to porters

Function: This module orders pending jobs based off of a Dispatch Value which is computed using several parameters (Weighted Job Priority and Appointment Factor). The pending job with the greatest Dispatch Value will be assigned to the closest available porter. Once the job is assigned to the porter the job will be considered a dispatched job.

Interface:
assignJob(Task):

- Assigns the job with the greatest Dispatch Value to the closest available porter.

getWeightedJobPriority(TaskOrigin, TaskDestination):

- Input the origin and destination of a pending job
- Output a value based on the priority of the pending job

getAppointmentFactor(Task):

- Input a pending job
- Update the value for a job depending on if it was pre-scheduled or on-demand.

getDispatchValue(Task):

- Input a pending job
- Compute the DispatchValue for a job: (WeightedJobPriority * AppointmentFactor)

Process Steps: All pending jobs are assessed and given a dispatch value (DV) based on the weighting and values of specified dispatch parameters.

These weights and values are determined using either the location of an available porter or the priority of a pending job.

All of the pending jobs are then ordered from greatest dispatch value to the least. When there is an available porter the pending job with the greatest dispatch value is given to the closest porter.

Data:

- Pending jobs

Error Handling: None

Requirement Reference: None

6.7 CORE - PORTER MANAGER

Function: This module populates a series of tasks following either a distribution or statistical data.

Interface:
importPorterSched(self, porterSchedule)

- Input the porter schedule and initialize all of the porters.

getPendingPorters(self, creationTime):

- Input the creationTime of a job. Create a list of porters that are available depending on the creationTime.

Process Steps: The porterManager handles the creation of porters and the availability of porters at a certain time. Based on the current time in the simulation the proterManager will provide the simulation with the porters who can accept jobs.

Data: Porter Schedule

Error Handling: None

Requirement Reference: None

6.8 EXPORT - STATISTICAL DATA EXPORT

Type: Module

Purpose: Format and output the raw data from the simulation.

Function: This module will gather all of the raw data from the simulation and format it so that it can be exported into a readable excel file.

Interface: publishData(simulationData)

- This function will format the simulation's raw data, create an excel file and populate the file with the formatted data

Process Steps: All data that is to be exported must be added to this module. Once the simulation's raw data is gathered the module will publish the data. The publishing will generate an excel file and outline the results from the simulation.

Data: simulationData

Error Handling: Produce warning message if data is missing during the publishing.

Requirement Reference: 9.1.4

6.9 EXPORT - STATISTICAL DATA ANALYSIS

Type: Module

Purpose: Outputs an excel dashboard file containing all the statistical analysis

Function: This module will take the excel output and use it construct excel charts and graphs

Interface: runMacros(outputFile)

Process Steps: Record the output data into itself allowing for inside functions to manage the calculations. Create a copy of itself at the user specified location

Data: outputData

Error Handling: Produce warning message if errors occurs during process

Requirement Reference: 9.1.5

6.10 UI - Basic Settings - General

Type: User Interface

Purpose: Allows the user to change the basic-general settings of the simulation

Function: Interface will allow entry of basic variables while preventing erroneous entry

Interface:

(i) **Simulation Duration:** Number of days the software will simulate events.

(ii) **Porter Wait Times:** Maximum number of minutes a porter waits for a delay (patient, equipment, nurse) before cancelling the job

(iii) **Job Flow:** The rate of event generation during the course of the simulation.

(iv) **Start Day:** Define the first day the simulation generates events from.

Process Steps: Not Available

Data: Not Available

Error Handling: The interface prevents the user from violating constraints

Simulation Duration: Value 0 to 7

Porter Wait Times: Value 0 to 60

Job Flow: Drop down defined by interface Low, Normal, High

Start Day: Drop down defined by interface Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday

Requirement Reference: 11.1.1, 11.1.2, 11.1.3, 11.1.4

6.11 UI - Basic Settings - File Locations

Type: User Interface

Purpose: Allows the user to choose the directory paths for required import/export files

Function: Interface will allow entry of directory paths by manual input or file browse dialogs

Interface:

(i) **Statistical Data Source:** Determines which source file will be used to provide the simulation with statistical distribution information

(ii) **Schedule Data Source:** Determines which source file will be used to provide scheduling information to be used during simulation

(iii) **Output Location:** Determines the Location to store output data

Process Steps: Not Available

Data: Not Available

Error Handling: The interface prompts the user with an error message upon entry of erroneous data

Statistical Data Source: File exists. File extension is ".csv"

Schedule Data Source: File Exists. File extension is ".csv"

Output Location: Directory exists

Requirement Reference: 11.1.1, 11.1.2, 11.1.3, 11.1.4

6.12 UI - Basic Settings - Buttons/Output Console

Type: User Interface

Purpose: Allows the user to activate simulation, reset all data and open file dialogs

Function: Interface connects GUI elements to actions

Interface:

(i) **Simulate Button:** Runs all the integrity checks and initializes the simulation run

(ii) **Reset All Button:** Resets all data to initial values

(iii) **Output Console:** Prints any logging statements

(iv) **File/Folder Browse:** Allows to browse user directory visually (supported by operating system)

Process Steps: Not Available

Data: Not Available

Error Handling: Interface prevents user from entering erroneous data

Output Console: Console is read only

Requirement Reference: 11.1.1, 11.1.2, 11.1.3, 11.1.4

6.13 UI - Advanced Settings - Dispatcher/Random Seed

Type: User Interface

Purpose: To provide the user with the ability to modify dispatcher values

Function: Allows the user to modify dispatcher values

Interface:

(i) **Appointment Factor:** Gives jobs that are scheduled a high priority than those who are generated on-demand by hospital staff.

(ii) **Automatic Job Priority Values:** To ensure jobs with a low priority are completed it is important that after a specified amount of time a job's priority is increased.

(iii) **Weighted Job List:** A list with each job's priority.

(iv) **Dispatch Value:** The value assigned to each job in the dispatcher (Job Priority * Appointment Factor)

(v) **Random Seed:** allows the user to enter a specific random seed instead of true random

Process Steps: Not Available

Data: Not Available

Error Handling: Interface prevents user from entering erroneous data

Appointment Factor: Value 0.0 to 3.0

Automatic Job Priority Values: Values 0.0 to 99.99

Weighted Job List: Values 0.0 to 99.99

Random Seed: Value 0 to 999999999

Requirement Reference: 11.1.1, 11.1.2, 11.1.3, 11.1.4

6.14 UI - SCHEDULE VERIFIER

Type: Module

Purpose: Confirms the integrity of the schedule

Function: Parses the schedule file and confirms the validity of each file

Interface:

(i) **scheduleChecker(file):** checks integrity of the schedule as a whole
(ii) **verifyShiftID(element):** checks integrity of each shift id
(iii) **verifyStartTime(element):** checks integrity of each start time
(iv) **verifyEndTime(element):** checks integrity of each end time
(v) **verifyPorterID(element):** checks integrity of each porter id
(vi) **verifyDays(element):** checks integrity of each day entry

Process Steps: scheduleChecker opens the schedule file and verifies each element of the file with a subroutine

Data: schedule data file

Error Handling: Module gathers and returns all the errors
scheduleChecker(file): returns the returns from all the subroutines (returns the cell and place of error)
verifyShiftID(element): string of character [A-Za-z0-9]
verifyStartTime(element): is date
verifyEndTime(element): is date
verifyPorterID(element): string of character [A-Za-z0-9]
verifyDays(element): value 0 - 6

Requirement Reference: None

6.15 UI - SCHEDULE PARSER

Type: Module

Purpose: Converts the schedule to an easier to read format by Porter Manager Module

Function: Processes all the elements and reconstructs them into a data structure

Interface:

(i) **scheduleParser(file):** checks integrity of the schedule as a whole

Process Steps: scheduleParser opens the schedule file and converts it to a python dictionary format

Data: schedule data file

Error Handling: No error if Schedule Verifier has been executed

Requirement Reference: None

7 ANTICIPATED CHANGES

None

8 APPENDICIES

8.1 FIGURE APPENDIX

(a) Figure 4.1: Process Diagram.jpg - the process flow diagram details the process modeled by the simulation.

(b) Figure 5.1: Dependency Diagram.png - layout of the module, algorithm and data structure dependencies.

8.2 ALGORITHM APPENDIX

(a) Dispatching Algorithm - main dispatcher algorithm used to assign jobs to porters based on a number of factors.