ClinicFlow

Design Document

Maxim Vasiliev #400043983
Susie Yu #000955758
Karl Knopf #001437217
Weilin Hu #001150873
Yunfeng Li #001335650

April 1 2017

Contents

1 Introduction **1**
- 1.1 Purpose . . . 1
- 1.2 Description . . . 1
- 1.3 Scope . . . 1

2 Overview **2**
- 2.1 Scheduled Deliverable and Development Time . . . 2
- 2.2 Revision history . . . 2

3 Overall Structure **3**

4 System Architecture **3**
- 4.1 MVC Architecture . . . 3
- 4.2 Controls . . . 5
- 4.3 Simulation Engine . . . 5
  - 4.3.1 Data Analysis . . . 6
- 4.4 Clinic Create . . . 9
- 4.5 Database . . . 9
- 4.6 Display . . . 9
- 4.7 Visualization . . . 10

5 Display: User Interface Design **10**
- 5.1 Overview . . . 10
- 5.2 Navigation flow . . . 10
- 5.3 Login . . . 11
- 5.4 Main Menu . . . 11
- 5.5 Patient Information . . . 11
- 5.6 Clinic Information . . . 12
- 5.7 Simulate Clinic . . . 13

6 Simulation Engine **14**
- 6.1 Design Description . . . 14
- 6.2 Simpy . . . 14
- 6.3 SimulationEngine.py . . . 15
  - 6.3.1 SimulationEngine(clinicFile,employeeFile,patientFile,outFile) 15
  - 6.3.2 Simulation(env,clinic,workerSchedule,patientSchedule,outfile) 15
  - 6.3.3 workerRun(env,worker,clinic,resources) . . . 16

6.3.4 patientRun(env, patient,clinic,resources) 16
6.4 Clinic.py 17
6.4.1 Clinic 17
6.5 ClinicStation.py 17
6.5.1 init (self,newName,prereqs,newMax,newMin,varType,avg,dev) 17
6.5.2 getRandomness(self) 18
6.6 HealthCareSchedule.py 18
6.7 HealthCareWorker.py 18
6.7.1 init (self,newName,breakTimes,workStation) 18
6.7.2 breakTime(self,currentTime) 19
6.8 PatientSchedule.py 19
6.8.1 schedule(self,fileName) 19
6.9 Patient.py 20

7 Clinic Create **20**
7.1 Design Description 20
7.1.1 ClinicCreate.py 21
7.1.2 ClinicMerge.py 21

8 Database: Database Structure **21**
8.1 Implementation Details 21
8.2 Data format 22
8.3 Sanitation 23
8.4 Requirements 23

9 Controls: Web Application Design **23**
9.1 Frameworks 23
9.2 models.py 24
9.3 views.py 24
9.4 test.py 25
9.5 urls.py 25

1 Introduction

1.1 Purpose

The Josef Brant Hospital pre-operative clinic (the client) schedules upwards of 50 patients per day. The appointment times are digitized, yet manually chosen by clinic staff. Once at the clinic, each patient undergoes a varying set of procedures with different durations. While staff have a good feeling of how to schedule patients, mistakes and inefficiencies often occur considering the numerous constraints and temporal variation of events. The client has approached us to explore the potential of simulating the flow of the clinic to help them understand the underlying problems at the clinic.

1.2 Description

This project aims to produce a tool which allows parties in the preoperative clinic sector to simulate the flow of patients amongst different procedures at the clinic. Demand for the solution stems from a lack of relevant products, and the reliance of clinic staff on intuitive and error prone manual scheduling. With this tool, prior patient temporal data will be fed in and used to build a model of all the variables involved. A simulation engine will then be created to produce data that could be used by the providers to optimize scheduling. This would allow clinic staff to reduce scheduling errors, as well as test the possible additions of new modules to the clinic.

1.3 Scope

A desktop application or web-interface application can allow users to insert necessary data, such historical patient procedure durations, doctor and nurse shift hours, and other constraints such as break allotments or soft constraints such as employee shift end times. Based on provided data and inputs, the system will generate simulations of the clinic. This tool will be designed with the intention to be implementable in multiple health care institutions for helping to manage patient scheduling.

2 Overview

2.1 Scheduled Deliverable and Development Time

The final deadline for the project is mid April 2017. The detailed deliverable and their respective deadlines are listed below:

- Requirements Document - revision 0: October 12, 2016
- Proof of Concept Plan: October 26, 2016
- Test Plan - Revision 0: November 2, 2016
- Proof of Concept Demonstration: November 21, 2016
- Design Document - Revision 0: January, 11, 2016
- Demonstration - Revision 0: February 13, 2017
- User's Guide - Revision 0: March 1, 2017
- Test Plan - Revision 0: March 22, 2017
- Final Demonstration: Mid-April, 2017
- Final Documentation: April 5, 2017

2.2 Revision history

Table 1: Revision history

Date	Comment
Jan 11, 2017	First draft
Apr 1, 2017	Second draft
Apr 9, 2017	Third draft

3 Overall Structure

The product will be delivered in a web application format. This maximizes its accessibility and allows it to be device agnostic. The core simulation component is written in Python 3, and makes use of the discrete event simulation libraries provided as part of the SimPy package. To maintain consistency in development language, we opted to use the Django framework for the web interface to our application. Data to be inputted and generated will be stored in a MongoDB database, and will use the provided Python MongoDB Drivers to interface between the view and data.

4 System Architecture

4.1 MVC Architecture

Our product is designed using a MVC (Model-View-Controller) system architecture.

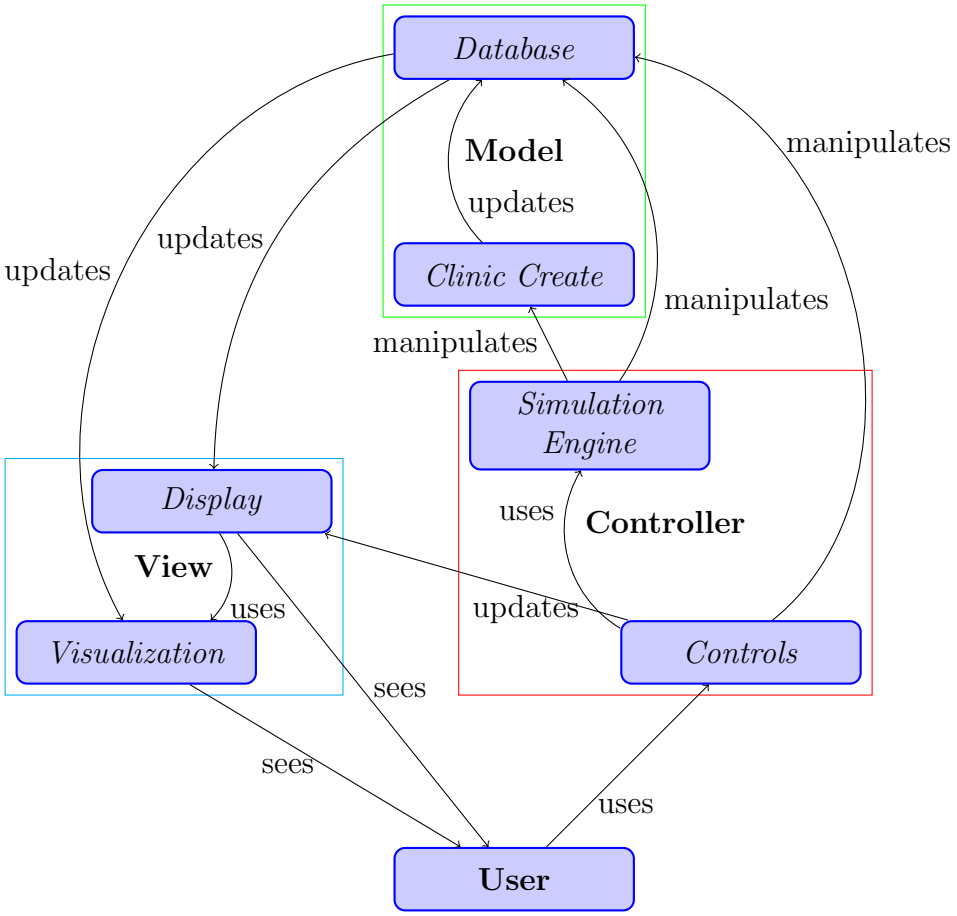

Figure 1: Breakdown of MVC Design

The **Controller** can be broken up into two primary components: *Controls*, which represents the how the user interfaces with the product and *Simulation Engine*, which manages and runs the clinic simulation. The **Model** also can be broken into two primary components: *ClinicCreate*, which manipulates the database information into a form usable to *Simulation Engine* and the *Database*, which manages all data base operations. The **View** can be broken down into two major components: *Display*, which handles the GUI for the website, and *Visualization* which displays the simulation data to the users.

4.2 Controls

Controls is a grouping of modules that are used by the user to interact with the system. They allow the user to enter input, issue commands to the simulation engine and change what is displayed by the view. It is primarily written in python 3 and uses the pymongo package to interact with MongoDB .
Controls interacts with the *Simulation Engine* and *Database* using the simengine() method. It also interacts with the display using the methods found in clinicflowx/views.py .
The component *Controls* allows the system to fufill both functional and non-functional requirements. It controls the interaction between elements in the program, meaning it determines what is going to be run. It also must do so in a way that is satisfactory to the end user.

4.3 Simulation Engine

Simulation Engine is a grouping of modules that are used by the program to create and run the simulations for a clinic. It manipulates the data from the *Clinic Create* to create the simulation. It then is able to generate multiple simulations, and commit the results the to *Database*. *Simulation Engine* is primarily written in Python 3, and it uses the SimPy discrete event simulation package.
Simulation Engine interacts with others modules through the method SimulationEngine.py . This method calls the other methods needed to run a simulation of the clinic. It takes as input:

1. a dataset that represents the clinic's information
2. a dataset that represents the provider's schedule
3. a dataset that represents the patient's schedule
4. a target destination for the output data

As output, it uses (or creates if it does not exist) a file containing all of the information generated by the simulation. It does this via AddSimResult.py .
The component *Simulation Engine* exists primarily to fulfill the functional requirements, namely actually being able to create and record a simulation.

It also addresses many of the nonfunctional requirements, such as simulation speed.

4.3.1 Data Analysis

Simulation Engine also has to solve the problem of how we implemented a discrete event simulation to provide realitic results. To allow it to be able to do this, we had to do background research into the clinic, and do analysis on the clinic's existing data sets.
Plots were created to understand the types of underlying variance in each of the sections of the clinic, as well as the variance in arrival times. For arrival times, we investigated the distribution of the differences between the scheduled arrival times and the actual arrival times.

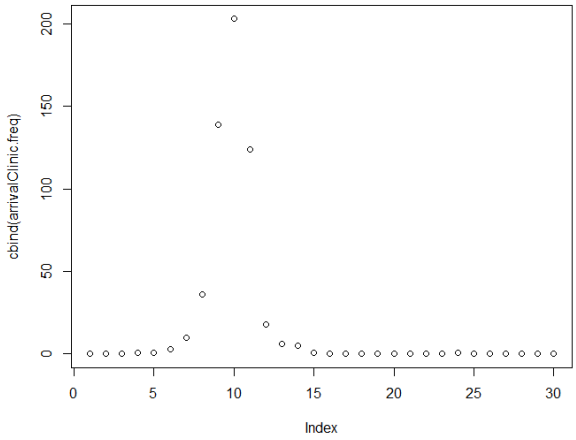

Figure 2: A plot of the distribution of scheduled arrivals versus actual arrivals

The X-axis in the above figure is the "bin" in which each difference was placed. 10 represents no difference between scheduled and actual arrivals, with 0 being -30 minutes the largest difference. The Y-axis in the above figure is the frequency of which each of these bins occur.
The above figure tells us that most people arrive around the time when they are scheduled, and that the variance appears to be approximately normal(Gaussian).

Next we looked at the first class of modules, that have fairly standard service times. To get an idea of the variance in these service times, we looked at the difference between the start time of the module and the finish time. The following plot is for"Registration", a typical example of this type of module:

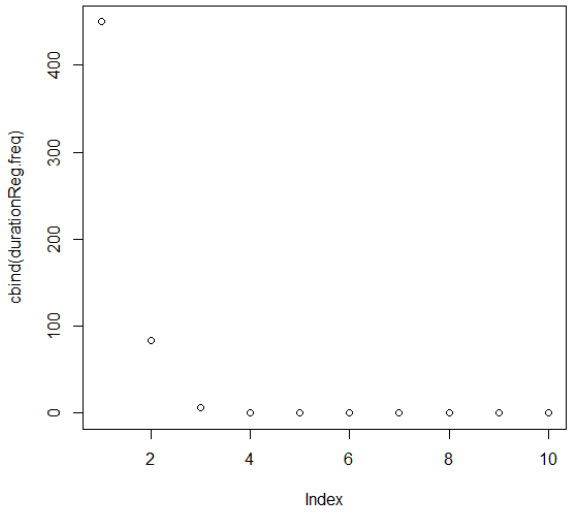

Figure 3: A plot of the distribution of time spent in "Registration"

The X-axis in the above figure is the "bin" in which each time length is placed. 0 represents 0 to 3 minutes spent in the module, with each remaining bin representing another 0-3 minutes. The Y-axis in the above figure is the frequency of which each of these bins occur.
From the above figure, we can see that most patients finish these modules quickly and with a consistent service time. There are a few patients who take longer. This suggests that we should use an exponential model for the variance in this module. Other modules of this class show a similar variance distribution.
Finally we investigated another class of modules, which have a more extreme spread of variance in the service times. This is because these modules contain more complicated proceedures that have no standard service time. To get an idea of the variance in these service times, we looked at the difference between the start time of the module and the finish time. The following plot

is for"XRayinRadiology", a typical example of this type of module:

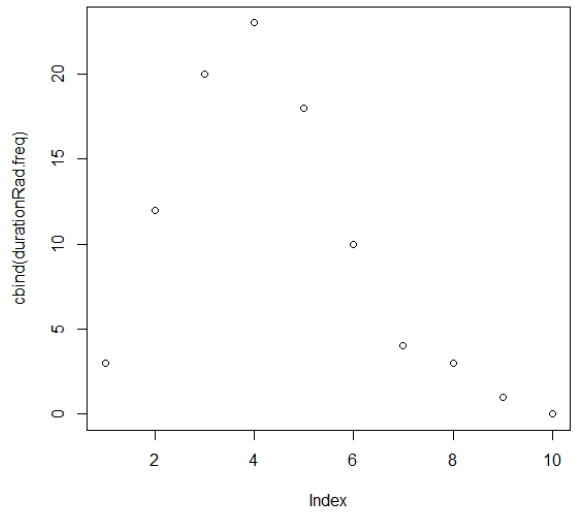

Figure 4: A plot of the distribution of time spent in "XRayinRadiology"

The X-axis in the above figure is the "bin" in which each time length is placed. 0 represents 0 to 3 minutes spent in the module, with each remaining bin representing another 0-3 minutes. The Y-axis in the above figure is the frequency of which each of these bins occur.
From the above figure, we can see that there is a large amount of variation between the amount of time any one patient would spend in this module. Some patients finish within 3 minutes, while most spend around 15 minutes. This suggests that we should try to model this variance with a different distribution, such as a normal (Gaussian) distribution.
In our implementation, we were able to take advantage of the results of this analysis to improve our models. We were able to use Python 3's libraries to generate random numbers as part of distributions built from the data sets we were provided on the clinic. We were also able to catergorize each module into one of two types (simple or complex). The variance of the arrival times was hard to model properly, so many method were tried before the final version used in the simulation engine was selected.

4.4 Clinic Create

Clinic Create is a grouping of modules that take the data from the *Database* and create useable files and tables for *Simulation Engine*. It maintains some of the clinic information, and general rules. *Clinic Create* is written in Python 3.
It takes input from other functions in the cliniccreate() method, and this method also provides files to *Simulation Engine* to allow that module to function.
ClinicCreate exists as a group of helpers for other methods. It does not solve any requirements itself, but it contributes to other modules being able to fulfill their requirements.

4.5 Database

Database is a grouping of modules that manage the database functions. It manages the tables, and it able to create and modify existing data tables. *Database* uses a MongoDB NoSQL framework.
This is cluster of database creation and management functions. They exist primarily in ClinicMongoDB.py . It does not contain input and output functions, but it manages contructs that get used by other modules.

4.6 Display

Display is a grouping of modules that manage the web display to the user. It displays the simulation results and information from *Database*. It controls what the user sees. *Display* is primarily written in Python 3, html and css. It shows the html pages to the user. It also takes requests from *Controls* to update and change the pages displayed.
Display exists to fulfill the function requirements involving showing the user results. It also must fulfill many non-functional requirements related to usabaility.

4.7 Visualization

Visualization is a grouping of modules that create the visualizations of the data that are displayed to the users. It gets the data from *Database* and give the visualizations to *Display*. *Visualization* will be created using Python 3 Pandas and Google Charts. It will be able to display graphs and charts related to the data processed by *Simulation Engine*.

5 Display: User Interface Design

5.1 Overview

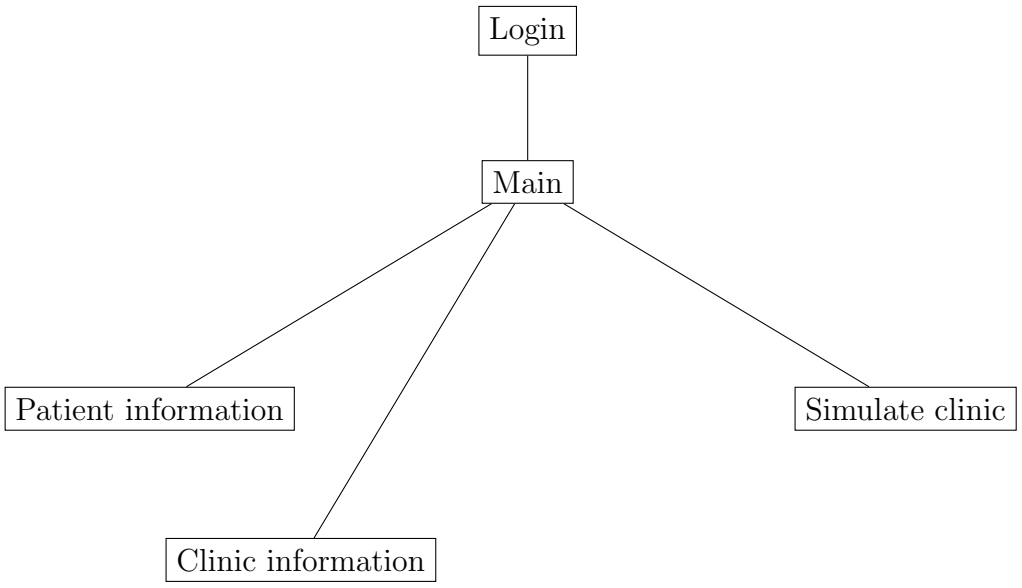

Figure 5: User Interface Design

5.2 Navigation flow

The first page presented to the user is the login page. The user must login as a manager or a authentic viewer before further operation. After logging in, the user will be redirected to the main menu page. The main menu page can direct user to the patient information page, clinic information page and simulate schedule page. The user who log in as a viewer can only view patient

information and generated schedules. Meanwhile the manager account would have viewer capabilities, in addition to being able to add/change patient information, add or change clinic attribute data, run simulations on specific patient data.

Each page has a navigation bar which let user transfer to different page. Each page has a logout button, thus the user can quit the system at any time.

5.3 Login

The user has to log in with the correct account and password before using the system. The login page has straight forward design which allows users to easily log in. This page contains internal validation to protect the system. This page can distinguish the user as a manager or a viewer, and gives the user corresponding authority.
It helps meet the following requirements:

- The program should verify the users
- The non-functional security requirements are enforced here

5.4 Main Menu

This page outlines general information such as today's reservation schedule, today's patients in list, and patients need schedule for next day which can give the user a quick overview. This page can guide the user to patient information,clinic information, clinic simulation, and schedule generation pages.
It helps meet the following requirements:

- The program should return to the main menu in case of errors
- The application should have a clean design

5.5 Patient Information

The manager account user can insert patient specific information such as name/id, appointment time, and required procedures into the system's database. There is a list of empty fields for the user to fill up. The fields for appointment time and procedures are vital and must be entered in order to add the

patient to the system, while other fields may be optional . The save button will validate the inputs before saving the patient data into database. The validation function will check the inputs to ensure the data consistency. If errors occur, the system will retain the inputs and inform the user of the source of error. Bulk patient data import is also supported through import of standardized csv files.

Both manager and view accounts are able to also see information on all patients current in the system. This will be presented both as list and timetable formats. Search and filter functionality will be included to allow users to target a particular subset of patients. Modification of existing records is also allowed, but only be the manager account.
It helps meet the following requirements:

- The user should be able to enter information
- The user should be able to manage the data
- The program should verify the users
- The program should be able to use real data to populate the simulation
- The program should display the database information to the user in a useful way
- The program should store the user's state
- The program should be easy to use
- The program should be scalable
- The program should be fault tolerant

5.6 Clinic Information

This page will give an overview of the attributes that model the clinic. This includes employee numbers, break times, starting times, etc. Both manager and view accounts will have access to this page, but only manager accounts will be able to edit the information therein and save it to the database. Multiple profiles for clinic information are also supported. This allows the user to simulate the clinic under models and compare the results.
It helps meet the following requirements:

- The program should verify the users
- The program should be able to use real data to populate the simulation
- The program should display the database information to the user in a useful way
- The program should store the user's state
- The program should be easy to use

5.7 Simulate Clinic

This page will be the interface to the simulation engine. The user will be able to run a simulation of the clinic given the data provided in the patient and clinic information sections of the application. The results of the current simulation and its summary statistics, as well as those of previous simulations, will be displayed here. The user is able to choose the day, and clinic data profile to use in the simulation. This page will also allow users to modify patient or clinic attributes (such as number of nurses available) to quickly see their effects on simulated clinic operation.

While managers and viewers will both be able to see past results, only managers have the authority to run simulations.
It helps meet the following requirements:

- The program should display the database information to the user in a useful way
- The program should be able to use any reasonable patient schedule
- The program should be able to use any reasonable
- The program should store the user's state
- The program should be easy to use

6 Simulation Engine

6.1 Design Description

The simulation engine is the core component set which is responsible for running the simulation of the clinic. The patients and their associated data are the entities that are tracked by the simulation, the clinic workers are the available resources that can service the patients, while the clinic environment acts as the constraints under which the patients flow through the system. This lends itself to a natural tree like organization structure. The modules thus have low coupling and high cohesion, with details pertinent to their operation being compartmentalized. *SimulationEngine* and Simulation run the simulation events, *PatientSchedule*, *HealthCareSchedule*, and *Clinic* maintain the objects used in the simulation, while *Patient*, *HealthCareWorker*, and *ClinicModule* keep track of the data if individual units in the simulation.

6.2 Simpy

Simpy is a Process-based discrete-event simulation framework which works with Python 3. It is well known and has a solid support base. The framework is relatively easy to work with and allows for a variety of event simulation types which are useful to this project.

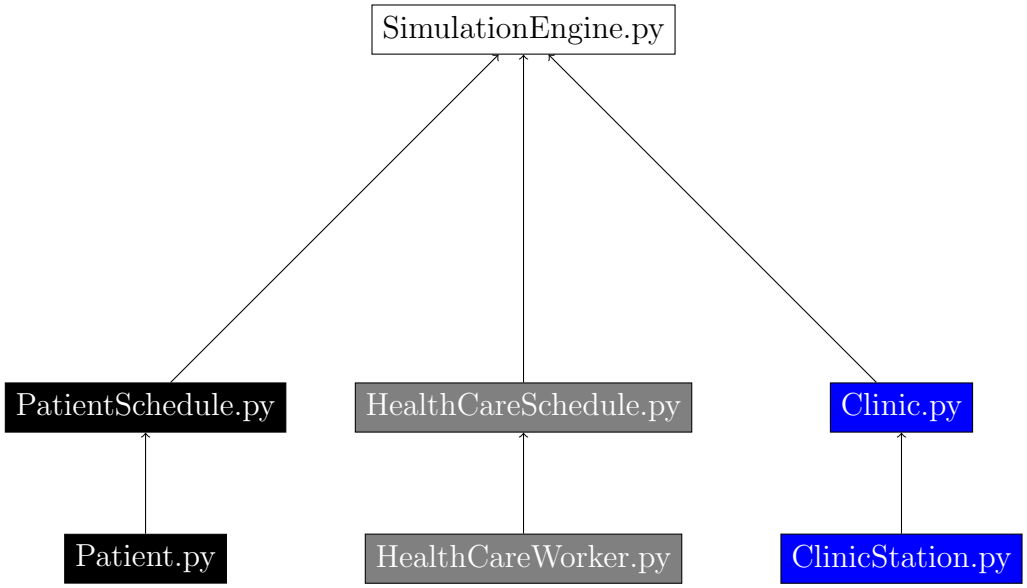

Figure 6: Simulation Engine Uses Diagram

6.3 SimulationEngine.py

6.3.1 SimulationEngine(clinicFile,employeeFile,patientFile,outFile)

- Acts as a top level for the program
- Handles File I/O for simulation
- Uses HealthCareSchedule, PatientSchedule and Clinic tables
- Calls the simulation function
- It helps meet the following requirements:
  - The program should use real data
  - The program should be able to run multiple simulations and generate different results.

6.3.2 Simulation(env,clinic,workerSchedule,patientSchedule,outfile)

- Manages the simulation of the clinic

- Uses SimPy to run a discrete event simulation
- Is called by SimulationEngine(), and uses workerRun() and patientRun() to create the simulation
- This function also handles the data analysis for the simulation results
- It helps meet the following functional requirements related to the simulation:
  - The program should use real data
  - The simulations should be realistic
  - The program should be able to handle reasonable schedules
  - The program should be able to generate resonable data

6.3.3 workerRun(env,worker,clinic,resources)

- Is the function that represents the "logic" that the worker threads follow
- Is used by Simulation to simulate the health care providers
- It helps meet the functional requirements related to the workers:
  - It helps the simulation be realisitic
  - It should be able to handle any provider break time schedule

6.3.4 patientRun(env, patient,clinic,resources)

- Is the function that represents the "logic" that the patient threads follow
- Is used by Simulation to simulate the patients
- It helps meet the functional requirements related to the patients:
  - It helps the simulation be realisitic
  - It should be able to handle any patient schedule

6.4 Clinic.py

6.4.1 Clinic

- Class file that contains the information about the clinic
- Constructor method reads in clinic data from file
- Uses ClinicStation
- It helps meet the following functional requirements :
  - It should be able to add and subtract clinic modules

6.5 ClinicStation.py

- Contains information about an individual station in the clinic
- Contains methods for activating (activate(self)) and deactivating (deactivate(self))a station
- Contains a method for getting a random number, generated from that particular station randomness (getRandomness(self))
- Contains a toString method to be able to see clinic information (str (self))

6.5.1 init (self,newName,prereqs,newMax,newMin,varType,avg,dev)

- The constructor method for the clinic module
- Stores all of information required for each clinic module
- It helps meet the following functional requirements :
  - It should be able to add and subtract clinic
  - The simulation should be realistic

6.5.2 getRandomness(self)

- A method to generate a random variation for each usage of the clinic
- Should be based on information passed to the class during construction
- It helps meet the following functional requirements :
  - The simulation should be realistic
  - The simulation should be able to run multiple simulations and get different results.

6.6 HealthCareSchedule.py

- Contains information about the provider's schedule in the clinic
- Uses HealthCareWorker.py
- It helps meet the following functional requirements :
  - The simulation should be realistic
  - The simulation should be able to use any resonable schedule for provider breaks

6.7 HealthCareWorker.py

- Contains information about a particular provider in the clinic
- Contains methods to change the worker's scheduled times (schedule(self, time)), and which station they are at (changeStation(self,newLoc))
- Has a toString (str (self))method for ease of use

6.7.1 init (self,newName,breakTimes,workStation)

- The constructor method for the HealthCareWorker
- Takes in information from another function to create the class
- It helps meet the following functional requirements :
  - The simulation should be realistic

  - The simulation should be able to use any resonable schedule for provider breaks

6.7.2 breakTime(self,currentTime)

- Method to handle the break times for each provider
- Returns the next break time for this provider
- It helps meet the following functional requirements :
  - The simulation should be realistic
  - The simulation should be able to use any resonable schedule for provider breaks

6.8 PatientSchedule.py

- Contains information about the patient's schedule in the clinic
- Contains methods for generating a schedule
- Uses Patient.py
- It helps meet the following functional requirements :
  - The simulation should be realistic
  - The simulation should be able to handle any realistic set of patients

6.8.1 schedule(self,fileName)

- Takes in a file containing patient schedule information, and creates the patient objects associated with all the information contained within the file
- Tries to assign patients every 15 minutes, like in the clinic
- Also reads in the variation information for arrivals
- It helps meet the following functional requirements :

  - The simulation should be realistic
  - The simulation should be able to handle any realistic set of patients
  - The simulation should be able to generate different results each simulation
  - The simulation should use real data

6.9 Patient.py

- Contains information about a patient in the clinic
- Has methods to adjust the patient's schedule(assignTime(self,time)), and help keep track of their actions throughout the clinic (completed(self, time) and addServiceTime(self,time))
- Has a toString (str (self)) method for ease of use
- It helps meet the following functional requirements :
  - The simulation should be realistic
  - The simulation should be able to handle any realistic set of patients
  - The simulation should generate useful data

7 Clinic Create

7.1 Design Description

Clinic Create is the component of the overall design responisble for taking the database files and converting them into a useful form for the Simulation Engine. It handles removal of incomplete data tuples and calculates the mean and variation of the various clinic modules.

7.1.1 ClinicCreate.py

- Uses pandas and numpy python packages for data analysis
- Reads in the data filesfrom the database, and uses ClinicMerge.py
- Calculates the mean and variance for each arrivals and for each clinic module
- Uses the file returned by ClinicMerge.py to create a complete clinic file, with proper variance so the simulation engine can generate realistic results
- It helps meet the following functional requirements :
  - The simulation should be realistic
  - The simulation should generate useful data
  - The simulation should use real data to create the variation
  - The simulation should be able to generate different results for each simulation run

7.1.2 ClinicMerge.py

- Takes a database file with qualitative clinic information and converts it into a form usable by ClinicCreate.py
- It helps meet the following functional requirements :
  - The simulation should be realistic
  - The simulation should be able to generate different results for each simulation run

8 Database: Database Structure

8.1 Implementation Details

MongoDB will be used as a database to store all input data related to modeling the clinic, as well as output data generated from simulation. The database will also be used to recall previously stored data. MongoDB does not have

a rigid structure, but its simplicity is well suited for this project. All data sets will be stored in a corresponding MongoDB collection. All data has a key attribute to improve searching efficiency and avoid duplication.

8.2 Data format

The **patient** collection contains all required patient information. This includes patient name/id, appointment date, time, as well as the list of services required by the patient. This information will be used to simulate clinic operations on a per day basis.

Table 2: Patient Table

Attribute Name	Sample Value
Patient Name	Jack Square
Reservation Date	2017-01-01
Reservation Time	8:15
Procedure	Interview, Bloodwork, x-ray

The **clinic** collection contains all attributes of the clinic we are modeling. This data will act as constraints on the clinic simulation, and can be modified by staff to accommodate operational changes.

Table 3: Clinic Table

Attribute Name	Sample Value
Nurse Number	3
Available Interview Room	3
Clinic Open Time	8:00
Clinic Close Time	18:00
Reception Close Time	15:00

The **result** collection contains all results from the simulations once they complete. This data will be used directly by the staff to help in deciding patient flow within the clinic during operation. This data will also help inform staff of attribute or scheduling effects on clinic operations.

Table 4: Results Table

Attribute Name	Sample Value
Day	2017-01-01
Simulation Run #	1
AVG duration bloodwork	20 minutes
AVG patient wait time	1 hr
Final patient end time	17:00

8.3 Sanitation

Simulation engine relies on the format and accuracy of data. The patterns of data must match the pattern inside simulation engine to ensure the simulation engine runs properly. Upon insertion or modification of data, a validation process verifies the inputs or changes before storing into database. Invalid inputs or changes would be rejected and the system would inform the reason for rejection. If no error in input or changes is detected, the system should allow the updates.

8.4 Requirements

The Database section helps meet the following requirements :

- The program should allow the user to enter data
- The user should be able to manage the data
- The system should store previous simulations in the data base
- The system should be scalable

9 Controls: Web Application Design

9.1 Frameworks

The Django framework is used in this project. It provides the users with a way to access the simulation engine, which includes import data, running the

simulation and utilizing other functions, and presents the simulation results in a concise and clear manner. At the same time, It helps to organize the backend structure of the project, even though it also supports the front end user interface. The following is a breakdown of the framework as it applies to our project.

9.2 models.py

These files integrate the *Simulation Engine*. They also deal with data flow from database to the simulation engine, and sends the corresponding outputs of the engine to the views.py. Other modules may be required to interact with the MongoDB as a database interface.

It helps meet the following requirements :

- The simulation should generate realisitic simulations, and present that information in a usable way.
- The system should be fault tolerant

9.3 views.py

It consists of static folder and templates folders. When the users send requests to the presetting URL address, it acts as a router and directs the users to pages that are generated from the template html in the templates folder. Since the static template HTML will be merged into the simulation results by a dynamic mechanism, the users can view the simulation results and operate the simulation engine instantly.

It helps meet the following requirements :

- The user should be able to manage the data on the system
- The system should be able to store the state of the program
- The system should be responsive and quick to use

9.4 test.py

To be used for testing cases, it ensures the backend works correctly and meets the design requirements.
It helps meet the following requirements :

- The system should be fault tolerant

9.5 urls.py

This stores the settings of the URL address for users to visit.