

Design Notebook for Queuing System

CS 422 Software Engineering Principles

November 8, 2000

**Our Color Name
Group #X**

Student Name

Student Name

Student Name

Student Name

Student Name

Student Name

Version 2.0

Abstract

The Design Notebook for the Queuing System describes, in detail, the overall design of the Queuing System.

The purpose of the Queuing System is to simulate a parallel server system consisting of five servers fed by a single queue. Customers enter the system and are sent directly to the first available server. If a server is not available, the customer enters the queue according to one of two queuing options. If a customer does not receive satisfactory service from a server, the customer is sent to the queue according to the specified queuing option. A customer remains in the system until receiving satisfactory service.

The design of the Queuing System is based on the Structured System Analysis and Structured Design Method. The methods of this design are explained further in the introduction section of this document. The only design constraint of the system design is time.

There has been one modification to the original requirements of the Queuing System. The new requirement is the system shall be extensible. This requirement is an added feature intended to make the system, as implemented by Yellow NEARPS , modifiable for other applications of a parallel server system.

The members and assigned tasks of Yellow NEARPS are as follows:

1. Student Name: Project Lead
2. Student Name: Testing Engineer
3. Student Name: Requirements Engineer
4. Student Name: Software Engineer and Marketing
5. Student Name: Lead Programmer
6. Student Name: Lead Designer

Table of Contents

<u>1</u>	<u>Introduction</u>	4
1.1	<u>Project Purpose and Goals</u>	4
1.2	<u>Design Approach</u>	4
1.3	<u>Traceability Approach</u>	5
1.4	<u>Background</u>	6
1.5	<u>Organization of This Document</u>	6
<u>2</u>	<u>Requirements Analysis</u>	7
2.1	<u>Functional Requirements</u>	7
2.1.1	<u>User-Defined Queuing Option</u>	7
2.1.2	<u>User Simulation Control</u>	7
2.1.3	<u>User Controlled Queue Modification</u>	8
2.1.4	<u>Simulation Requirements</u>	8
2.2	<u>Non-functional Requirements</u>	9
2.2.1	<u>High Level Non-functional Requirements</u>	9
<u>3</u>	<u>Design Representation</u>	10
3.1	<u>Context Diagram</u>	10
3.2	<u>Level One Data Flow Diagrams and Process Specifications</u>	10
3.2.1	<u>Level One Data Flow Diagram</u>	11
3.2.2	<u>Process Specification for Level One Data Flow Diagram</u>	11
3.3	<u>Level Two and Three Data Flow Diagrams and Process Specifications</u>	12
3.3.1	<u>Level Two Data Flow Diagram 1.0</u>	13
3.3.2	<u>Process Specification for Level Two DFD 1.0</u>	13
3.3.3	<u>Level Two Data Flow Diagram 2.0</u>	15
3.3.4	<u>Process Specifications for Level Two DFD 2.0</u>	15
3.3.5	<u>Level Two Data Flow Diagram 3.0</u>	16
3.3.6	<u>Process Specifications for Level Two Data Flow Diagram 3.0</u>	17
3.3.7	<u>Level Three Data Flow Diagram 1.3</u>	18
3.3.8	<u>Process Specifications for Level Three Data Flow Diagram 1.3</u>	19
3.3.9	<u>Level Three Data Flow Diagram 2.2</u>	19
3.3.10	<u>Process Specifications for Level Three Data Flow Diagram 2.2</u>	20
3.4	<u>Transaction Analysis</u>	21
3.4.1	<u>Partitions of the System DFD</u>	21
3.4.2	<u>Identified Transactions</u>	23
3.5	<u>Transform Analysis</u>	24
3.5.1	<u>Structure Chart</u>	24
3.6	<u>Design Decision Log</u>	25
<u>4</u>	<u>References</u>	27
<u>5</u>	<u>Glossary</u>	28
	<u>APPENDIX A: Data Dictionary</u>	29
	<u>APPENDIX B: Project Schedule</u>	35
	<u>APPENDIX C: Requirements Traceability Matrix</u>	36
	<u>APPENDIX D: Identified Test Cases</u>	38

Table of Figures

<i>Figure 1: Context Diagram</i>	10
<i>Figure 2: Level One Data Flow Diagram</i>	11
<i>Figure 3: Level Two Data Flow Diagram (1.0)</i>	13
<i>Figure 4: Level Two Data Flow Diagram (2.0)</i>	15
<i>Figure 5: Level Two Data Flow Diagram (3.0)</i>	17
<i>Figure 6: Level Three Data Flow Diagram (1.3)</i>	18
<i>Figure 7: Level Three Data Flow Diagram (2.2)</i>	20
<i>Figure 8: GUI and Simulation Engine Interaction Chart</i>	22
<i>Figure 9: Simulation Engine and Animation Interaction</i>	22
<i>Figure 10: Structure Chart</i>	25
<i>Figure A-1: Queuing System Project Schedule</i>	35

Table of Tables

<i>Table 1: Design Decision Log</i>	26
<i>Table A-1: Schedule Tasks Key and Limitations</i>	35
<i>Table C-1. Requirements Traceability Matrix</i>	36

1 Introduction

This section of the Design Notebook covers the project purpose and goals, design approach, traceability approach and background of the Queuing System. The overall organization of this document is also covered.

1.1 Project Purpose and Goals

The Queuing System shall simulate a parallel server system, containing five servers, fed by a single queue. Customers entering the system go directly to the first available server, if a server is available. If a server is unavailable when the customer arrives in the system, the customer is sent to the queue according to one of two queuing options. Queuing option one sends each entering customer to the end of the queue. Queuing option two enters the arriving customer in the queue according to the total number of times, in descending order, the customer had been previously serviced by a server. A customer does not exit the system until the customer receives satisfactory service from a server.

1.2 Design Approach

The design of the Queuing System follows the Structured System Analysis and Structured Design Method. Fredrick T. Sheldon supplied the outline that is shown below.

Steps 1-2: Structured systems analysis (see Budgen Figure 10.5)

- **Level 0 = context diagram**
- **Level 1 = top level DFD**
- **Level 2 = explosion of level 1 DFD bubbles**
- **Level 3 = use this level where appropriate**

Step 3: Transaction analysis step and has five basic components:

- 1. The event in the systems environment that causes the transaction to occur**
- 2. The stimulus that is applied to the system to inform it about the event**
- 3. The activity that is performed by the system as a result of the stimulus**

4. The response that this generates in terms of output from the system

5. The effect that this has upon the environment

Therefore, you will need to identify a simple but comprehensive example of a transaction (see page 218 and Figure 10.8 of Budgen) that accounts for all of the five items described above. This will help you to define a good set of test cases. There should be four transactions identified. These will be your main starting points for the demonstration.

Step 4: Identify the central transform in the DFD:

You do not have to redraw the DFDs but if you add a “boss” bubble redraw showing where the boss fits. Which will allow you to develop a hierarchical structure chart. Develop structure charts for all of your level 1-2 DFDs.

Step 5: Merge the Structure Charts

The objective of this step is to produce a single structure chart (see Figures 10.23, 24 of Budgen).

1.3 Traceability Approach

The ability to verify the satisfaction of requirements in the design is essential to the success of the Queuing System’s design and implementation. To verify that all system requirements are satisfied Yellow NEARPS has decided on the following approach:

- Each system requirement is assigned a unique identification number.
- Each system requirement is inserted into the Requirements Traceability Matrix (RTM).
- In the RTM, each requirement is assigned a DFD identifier during the design phase.
- Each requirement is associated with a Module in the implementation phase.
- Each requirement is assigned a testing method for verifying that each requirement is met.
- The right-most column in the RTM is used to check-off each requirement test as it is completed.

The completion of the RTM (Appendix C) gives strong evidence that each requirement outlined in the Software Requirements Specification was satisfied.

1.4 Background

The Queuing System design has changed very little since the presentation of the Preliminary Design Review. There have been two major changes. The program design has been enhanced by making the Queuing System extensible. To accomplish this task all constant values associated with system elements (number of servers, simulation speed, etc.) have been changed to variables. This allows the user to modify the system to suit a specific task.

The other major change to the system regarded the user simulation control. Yellow NEARPS decided that it would be best to allow the user to not only start and stop/restart the simulation, but to also allow the user to terminate the simulation before the simulation time expired. This, in effect, allows the user to cut the simulation time short and still view the simulation statistics as if the simulation time had expired.

1.5 Organization of This Document

This document will be organized as follows:

1. Introduction
2. Requirements Analysis: analyses functional, non-functional and derived requirements
3. Design Representation: follows SSA/SD Method
4. References
5. Glossary
6. Data Dictionary
7. Project Schedule
8. Requirements Traceability Matrix
9. Identified Test Cases

2 Requirements Analysis

This section gives an overview of functional, non-functional, and derived requirements for the Queuing System. The requirements were originally defined in the Software Requirements Specification and later modified in the Critical Design Review.

2.1 Functional Requirements

The Functional Requirements for the Queuing System are defined below to ensure that all user-specified system functionality is incorporated into the system's design. Each Top-level functional requirement is followed by its derived lower-level functional requirements.

2.1.1 User-Defined Queuing Option

The system shall provide a way for the user to specify the preferred queuing option.

2.1.1.1 User-Defined Queuing Option Derived Requirements

The derived requirements for the queuing option requirement are the two choices the user has regarding placing customers to the queue. The first choice is to send each new and returning customer is added to the end of the queue. The second choice is to insert new and returning customers into the queue in descending order using the number of times each customer has received service as the sorting key. The reason the number of times serviced is used is because the probability that a customer will receive satisfactory service increases each time a customer is serviced.

2.1.2 User Simulation Control

The system shall allow the user to pause or stop the simulation process. This requirement was requested by the customer. The ability to pause or stop the process gives the user greater control over the simulation process.

2.1.2.1 User Simulation Control Derived Requirements

The requirements derived from giving the user greater control involve letting the user restart the process after pausing the execution and displaying a simulation statistics screen upon a user request to stop the simulation. Because the user can pause and restart the process the system is required to not lose or alter any of the information in the Simulation Parameters or Master List

data stores, unless modified by the customer.

2.1.3 User Controlled Queue Modification

The user shall be allowed to modify the queue and servers during simulation pauses by either inserting or deleting customers from the simulation. This requirement was requested by the customer, and it also supports system extensibility. Allowing the insertion or removal of customers from the system is analogous to disallowing a person's use of a lab computer because of violation of laboratory policy.

2.1.3.1 User Controlled Queue Modification Derived Requirements

The requirements derived from allowing the user to modify add and delete customers from the system are mostly error checking requirements. Any system modification made by the user must not cause the simulation to run improperly.

2.1.4 Simulation Requirements

The requirements for the simulation involve the behavior of the simulation and the statistics that the simulation calculates. The system shall initially be empty and idle, customers shall be sent from the front of the queue to the first available server, if all servers are busy, the customer is sent to the queue using the specified queuing option and a customer shall exit the system after receiving satisfactory service.

The statistics calculated during the simulation fall into two categories, customer statistics and simulation statistics. The customer statistics include the number of times serviced, the total time in the system and the amount of service time each customer receives. The simulation statistics involve the interarrival times for new customer, whether a customer receives satisfactory service and all other information required for the Simulation Statistics data store.

2.1.4.1 Simulation Derived Requirements

The requirements derived from the simulation requirements for system behavior and statistical calculation cover the details that were over looked by the high level requirements.

The system is required to be empty and idle at the beginning of a simulation. To achieve this, the queue shall be empty and each server shall be idle and available. After the simulation

begins, a new/returning customer shall be sent to the left-most available server, if all servers are unavailable, the customer shall be added to the queue based on the selected queuing option. When a customer is moved from the queue to a server, the customers remaining in the queue shall be advanced one space in the queue, and the queue shall be empty if the last customer in the queue is moved to a server. If a customer does not receive satisfactory service, the system shall increase the customer's number of times serviced variable. Every time an event takes place in the system, the system records all statistics generated by the system in the Simulation Statistics List data store.

The simulation system calculates statistics that enables simulation execution. It also calculates statistics for each customer that passes through the system. The simulation execution specific statistics include customer interarrival times, service times, customer satisfaction and number of times each customer is serviced. The statistics calculated for each customer are number of times serviced, average service time and the average/maximum time in the system. During the simulation, the system calculates statistics for the display screen. The statistics are the average/maximum time each customer is in a server and in the system, the average/maximum length of the queue, the number of serviced/satisfied customers, the time-average and the maximum number of busy servers.

2.2 Non-functional Requirements

The non-functional requirements for the Queuing System are those requirements not related to the system's functionality. These non-functional requirements are still essential to the Queuing System's design and implementation.

2.2.1 High Level Non-functional Requirements

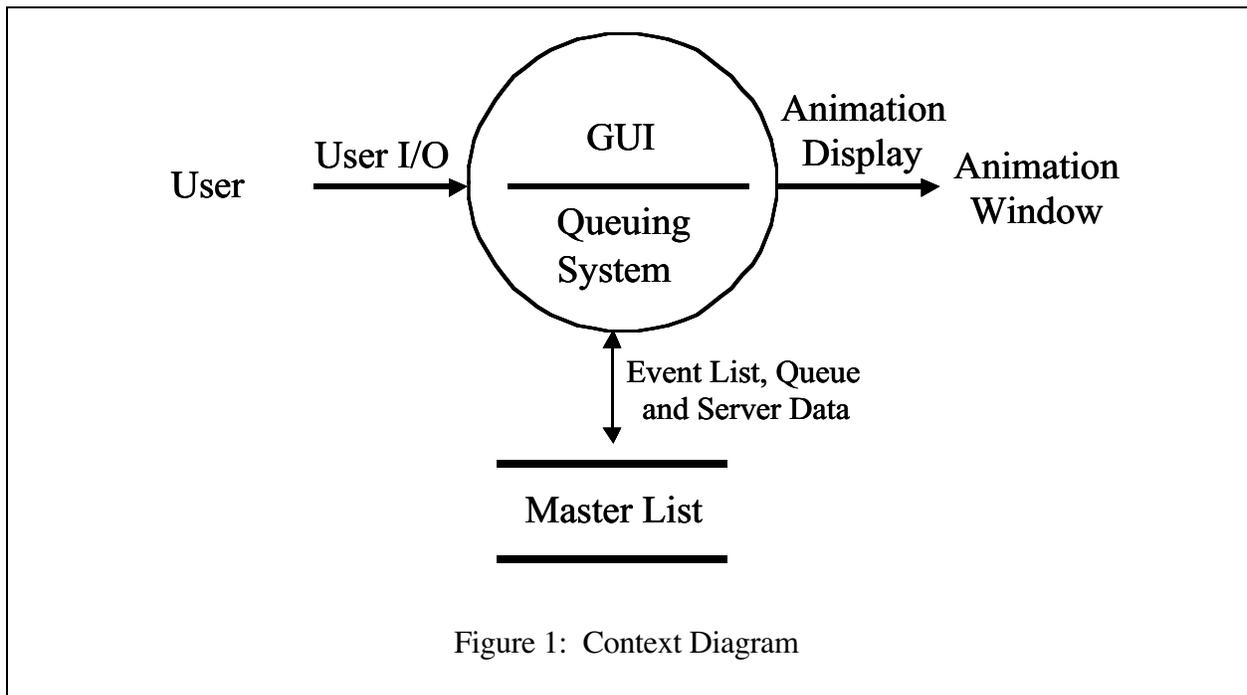
The Queuing System shall be implemented using the Java programming language. The Queuing System shall have a graphical user interface. The Queuing System shall provide an animation of the simulation in real-time.

3 Design Representation

The Queuing System was designed using the Structured System Analysis and Structured Design Method (SSA/SD). This section of the Design Notebook for the Queuing System follows the steps of the SSA/SD and describes each step in detail.

3.1 Context Diagram

The Context Diagram, shown below in *Figure 1*, shows the Queuing System at its highest level of abstraction. The system's external entities are the User and the Animation Screen. The main process bubble is a combination of the user interface, GUI, and the Queuing System, which controls all interaction with the Master List and the Animation Screen. The Master List is the only data store at this level.



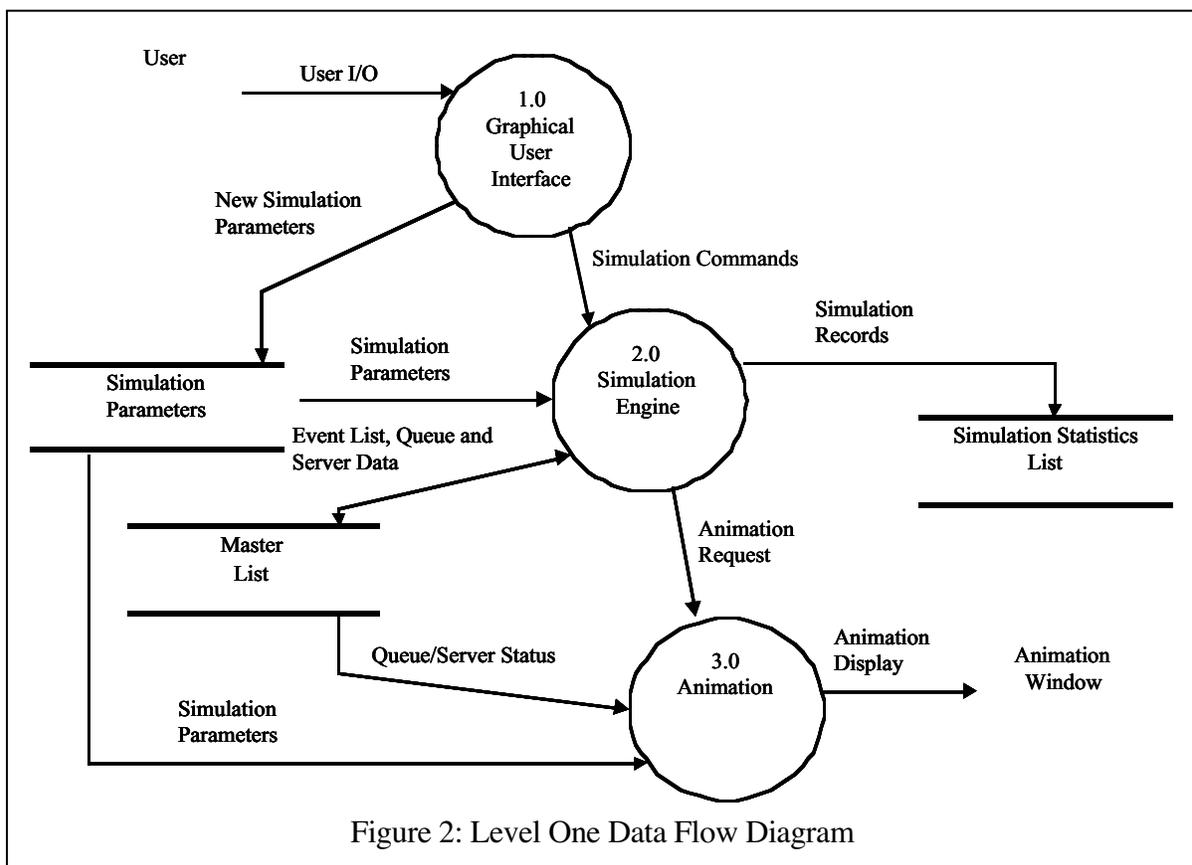
3.2 Level One Data Flow Diagrams and Process Specifications

At Level One, the Queuing System is split into three main processes. The GUI (1.0) acts as a middle man between the User and the Simulation Engine (2.0). The Simulation Engine (2.0) executes the simulation of the Queuing System using data from the Simulation Parameters, and Master List data stores. The Animation (3.0) uses the data in the Simulation Parameters and

Master List data stores to create an animated representation of the activities executed during the system simulation.

3.2.1 Level One Data Flow Diagram

The Level One Data Flow Diagram (DFD), *Figure 2*, for the Queuing System explodes the Context Diagram into three processes, bubble 1.0 is the GUI, process bubble 2.0 is the Simulation Engine and bubble 3.0 is the Animation. Two new data stores are introduced at this level, Simulation Parameters and Simulation Statistics List. The Master List is also included at this level. A definition of each data store can be found in the Data Dictionary (Appendix A).



3.2.2 Process Specification for Level One Data Flow Diagram

The process specifications for level one are an abstract narrative of how each process works in relation to the other processes conducted by the Queuing System. The process specifications for each process bubble are outlined below.

3.2.2.1 Graphical User Interface (1.0)

The Graphical User Interface handles all user input and output. User output is a collection of windows used either to gather information from, or to display information to the User. User input is considered as all commands and data submitted by the user through the various output windows displayed. The GUI also relays command messages, called simulation commands, to the Simulation Engine (2.0) so each command given by the user is reflected in the behavior of the simulation.

3.2.2.2 Simulation Engine (2.0)

The Simulation Engine process receives input from the Simulation Parameters and Master List data stores and the GUI in order to execute the simulation. While the simulation is executing, the Simulation Engine process continuously interfaces the Master List regarding the status of the Event List, the servers and the queue to determine if any action is necessary. For each customer that is processed by the Simulation Engine statistics are recorded in the Simulation Statistics data store. During execution of the Simulation Engine Process the Simulation Engine Process sends animation requests to the Animation Process (3.0) to create and update the animation display.

3.2.2.3 Animation (3.0)

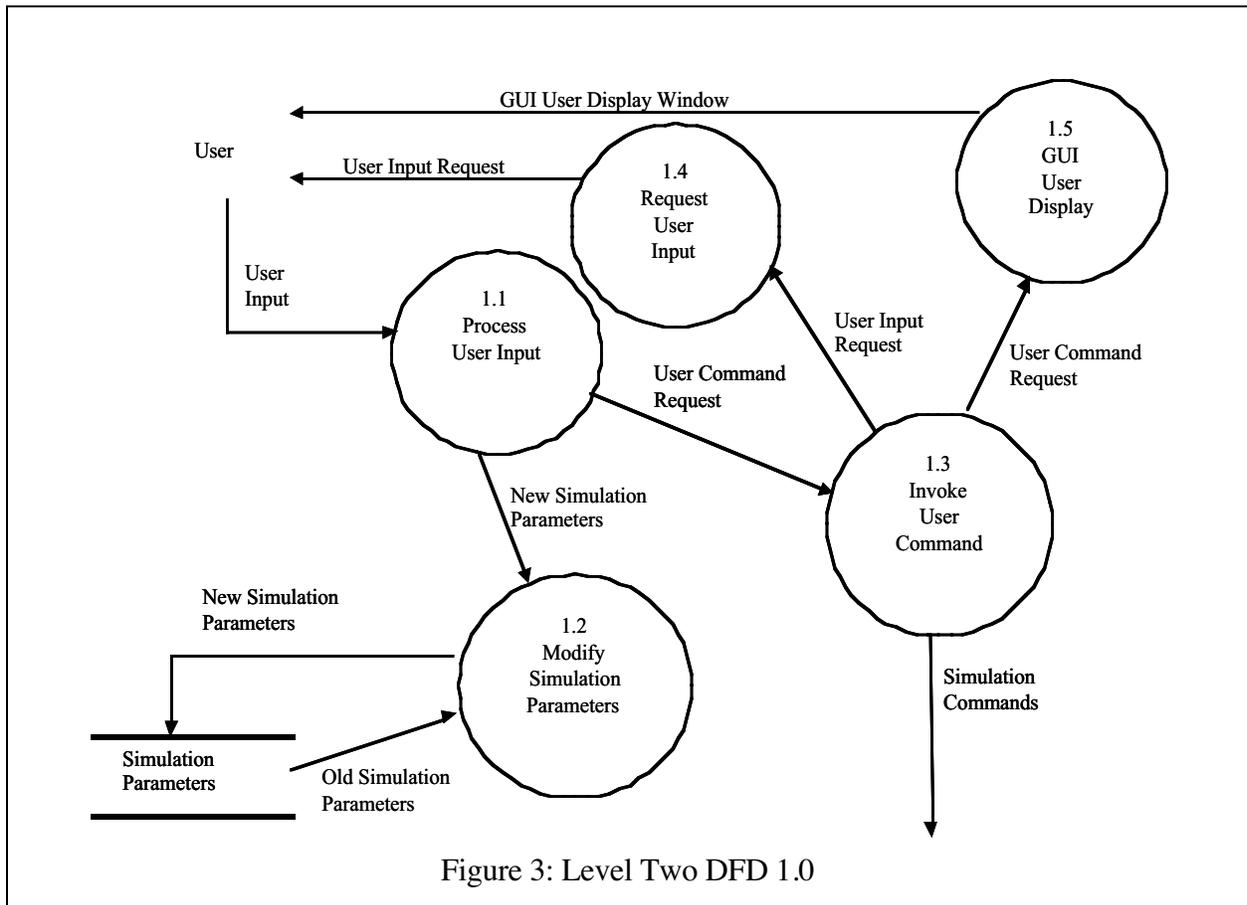
The Animation process inputs data from the Simulation Parameters and Master List data stores. This information is used to create a display drawing that represents each server, the queue and each customer in the system. The customers are drawn in one of two places, at a server or in the queue. The animation display is updated each time the Animation process receives an animation request from the Simulation Engine.

3.3 Level Two and Three Data Flow Diagrams and Process Specifications

The Level Two and Three DFDs further expand the process bubbles in the Level One DFD. The GUI, System Engine and Animation process bubbles are expanded in *Figure 3*, *Figure 4* and *Figure 5* respectively.

3.3.1 Level Two Data Flow Diagram 1.0

The Level Two Data Flow Diagram 1.0 (Figure 3) expands the GUI process bubble from Level One. At this level, the processes executed by the GUI are clearer. Refer to the process specification for more information on the operations of each process bubble shown below.



3.3.2 Process Specification for Level Two DFD 1.0

The Level Two DFD 1.0 process specifications elaborate on the process specifications from the Level One DFD GUI process bubble (1.0). The tasks of the GUI process bubble (1.0) have been expanded into five different processes and each is described below.

3.3.2.1 Process User Input (1.1)

Every time the user makes an entry in a dialog box or clicks on a button on the main GUI display, a command message is passed to the GUI Process User Input process bubble. The information is

examined to determine whether it is a command or data. If the input is a command, the Process User Input process bubble relays that command request to the Invoke User Command process bubble (1.3). If the user input is data, the Process User Input sends the new data to the Modify Simulation Parameters process bubble (1.2).

3.3.2.2 Modify Simulation Parameters (1.2)

The Modify Simulation Parameters process bubble receives new simulation parameters from the Process User Input process bubble (1.1) and saves the new parameters in the Simulation Parameters data store for future use by the rest of the system. The Modify Simulation Parameters process must determine if the new simulation parameters are within the determined range to avoid invalid input to the Simulation Engine (2.0). If any of the new parameters are invalid, the user is notified of the error and prompted to enter an input within the valid range. The previous simulation parameters are not overwritten until all user input is valid.

3.3.2.3 Invoke User Command (1.3)

The Invoke User Command process at this level is still abstract. The main function of the process is to input user command requests from the Interpret User Command process bubble (1.1) and act as an interface between the User and the Simulation Engine (2.0). The system's user output is either in the form of a user input request or a display window. The Invoke User Command process is responsible for determining which type of user output is required. If a user input request is necessary, the request is relayed to the Request User Input process bubble (1.4). If a user output screen is necessary, a request is sent to the GUI User Display process bubble (1.5).

3.3.2.4 Request User Input (1.4)

The Request User Input process bubble is responsible for displaying a window to receive necessary user input. The window type is determined by the user input request message passed from the Invoke User Command process bubble (1.3). When the user input request message is received by the Request User Input process, the process displays the corresponding window.

3.3.2.5 GUI User Display (1.5)

The GUI User Display process displays the contents of the Simulation Statistics data store to the user at one of two times during the simulation. By default, the GUI User Display is executed upon termination of the simulation. The other occurrence of the GUI User Display process is during a simulation pause upon user request.

3.3.3 Level Two Data Flow Diagram 2.0

The Level Two Data Flow Diagram 2.0 (Figure 4) expands the Simulation Engine process bubble (2.0) of the Level One Data Flow Diagram. The expansion is shown below followed by its process specification.

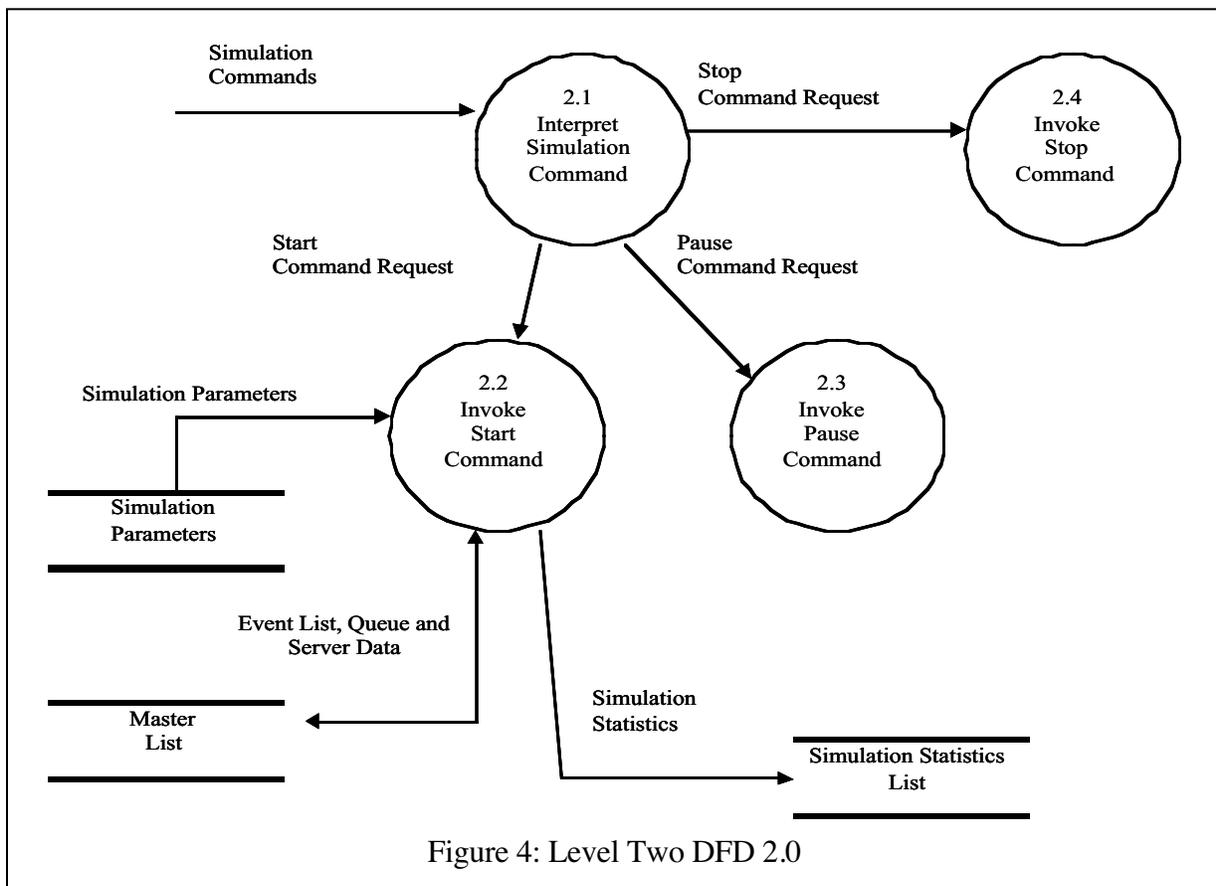


Figure 4: Level Two DFD 2.0

3.3.4 Process Specifications for Level Two DFD 2.0

The Level Two DFD 2.0 process specifications explain, in greater detail, the process executed by the Simulation Engine (2.0). Each process specification for the Level Two DFD 2.0 in figure 4 are illustrated below.

3.3.4.1 Interpret Simulation Command (2.1)

The Interpret Simulation Command process bubble receives a simulation command message from the GUI (1.0) and determines the command type. Once the command type is determined, the Interpret Simulation Command process sends a command request to the corresponding function.

3.3.4.2 Invoke Start Command (2.2)

The Invoke Start Command process is the backbone of the simulation. Because of this, the process is expanded and explained in greater detail in Level 3 DFD 2.2 (*Figure 7*). When the Invoke Start Command process is called the simulation executes using information from the Simulation Parameters and Master List data stores. As the simulation runs, animation requests are sent to the Animation process (3.0) and simulation records are stored into the Simulation Statistics data store. The simulation continues to execute until one of three events happen: (1) The simulation time expires or (2) a stop command is executed or (3) a pause command is executed.

3.3.4.3 Invoke Pause Command (2.3)

The Invoke Pause Command process temporarily stops the simulation execution. The time remaining in the simulation is saved and the Invoke Pause Command GUI process (1.3.4) is called. The simulation remains paused until the user chooses to resume the simulation, that process is carried out by the Invoke Start Command (2.2).

3.3.4.4 Invoke Stop Command (2.4)

The Invoke Stop Command process stops the simulation as if the simulation time had elapsed. Since the Invoke Stop Command process does not save information, like the Invoke Pause Command process does, a warning message is displayed prompting the user to verify their wishes to terminate the simulation. If the user chooses to stop the simulation, the Invoke Stop Command GUI process is called and the simulation is stopped.

3.3.5 Level Two Data Flow Diagram 3.0

The Animation Process is pictured below (*Figure 5*). The Animation process is intended to give a real time graphical representation of the simulation. The Level Two DFD 3.0 illustrates the

how information flows through the Animation process from the Simulation Parameters and Master List data stores, through the interpreter and finally to the Animation Window.

3.3.6 Process Specifications for Level Two Data Flow Diagram 3.0

The process specifications for the Level Two DFD 3.0 explain how each process converts the information in the Simulation Parameters and Master List data stores into the picture displayed in the Animation Window.

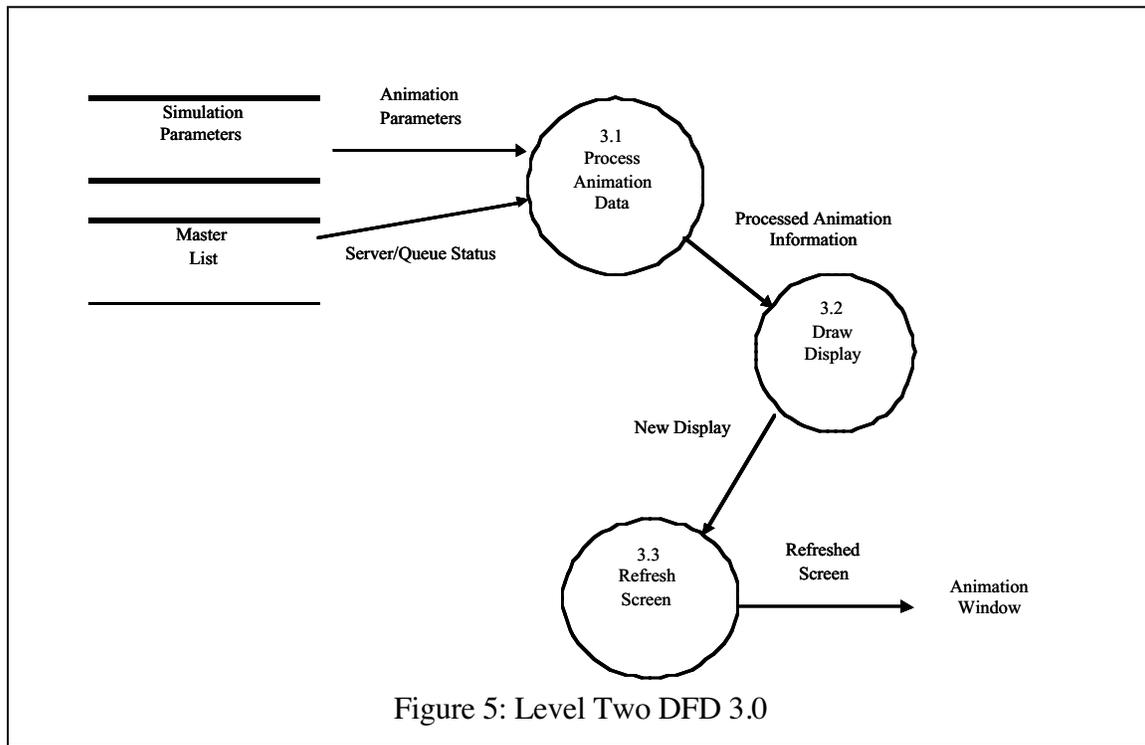


Figure 5: Level Two DFD 3.0

3.3.6.1 Process Animation Data (3.1)

The Process Animation Data bubble receives information from the Simulation Parameters and Master List data stores. The Simulation Parameters data store provides the simulation speed, which is necessary to calculate the animation's refresh rate. The refresh rate is calculated and the information necessary to generate the animation is requested from the Master List data store accordingly. The information necessary to generate the animation are the number of customers in the queue, which servers are occupied and how many servers are in the system. This information is pulled from the Master List data store. The above parameters are passed to the Draw Display process (3.2)

3.3.6.2 Draw Display (3.2)

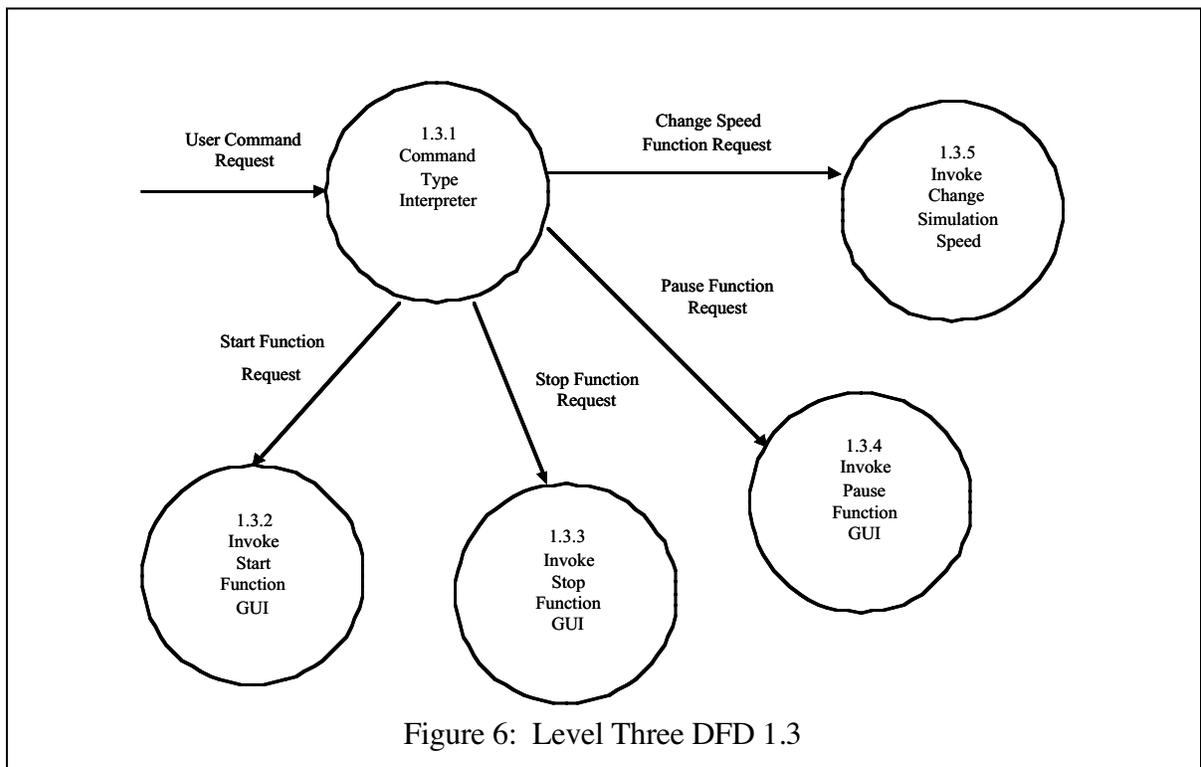
The Draw Display process uses the animation parameters from the Process Animation Data bubble (3.1) to determine where to draw each entity (servers, queue, and customers) in the system in the proper place. The newly generated display and is passed to the Refresh Screen process (3.3).

3.3.6.3 Refresh Screen (3.3)

The Refresh Screen process simply outputs the new drawing created by the Draw Display process (3.2) to the Animation screen. The Animation screen is a window inside of the main GUI display.

3.3.7 Level Three Data Flow Diagram 1.3

The Level Three Data Flow Diagram 1.3 (*Figure 6*) clarifies the functions of the Invoke User Function process (1.3) from Level Two Data Flow Diagram 1.0 (*Figure 3*). The process specifications for each new process bubble follow.



3.3.8 Process Specifications for Level Three Data Flow Diagram 1.3

The process specifications for Level Three Data Flow Diagram 1.3 explain the functionality of each process bubble in *Figure 6*.

3.3.8.1 Command Type Interpreter (1.3.1)

The Command Type Interpreter process receives a user input request and determines which command to invoke. If the command is either start, stop or pause, two different processes are invoked. The Command Type Interpreter process sends a function request to both the corresponding function's GUI process and the Simulation Engine. If the command is not one of the above functions, the Command Type Interpreter process sends a command request message to the corresponding function's GUI process.

3.3.8.2 Invoke Start Function GUI (1.3.2)

The Invoke Start Function GUI process sends a GUI request message to the GUI User Display process (1.5) that tells the GUI User Display process (1.5) to generate the Simulation Execution Graphical User Interface.

3.3.8.3 Invoke Stop Function GUI (1.3.3)

The Invoke Stop Function GUI process sends a GUI request to the GUI User Display (1.5) requesting the generation of the End of Simulation Statistics Window.

3.3.8.4 Invoke Pause Function GUI (1.3.4)

The Invoke Pause Function GUI process send a User Input request to the Request User Input process (1.4) that tells the Request User Input process (1.4) to display the Pause GUI Window on the screen.

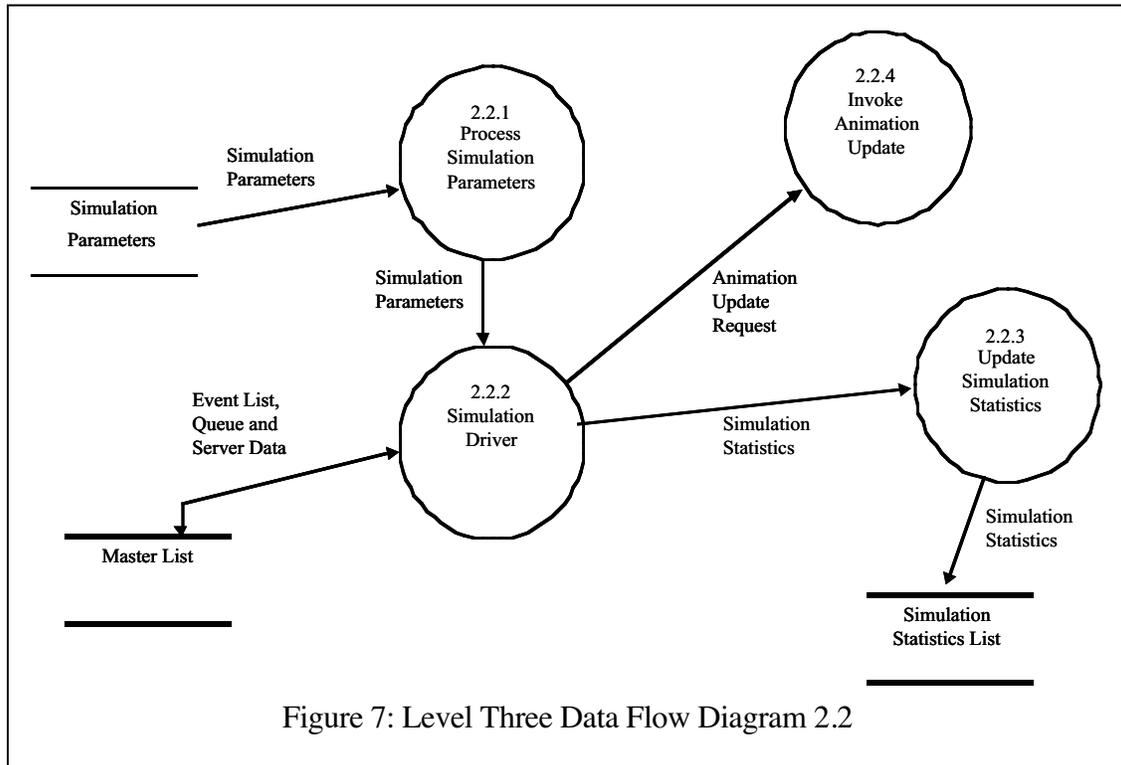
3.3.8.5 Invoke Change Simulation Speed (1.3.5)

The Invoke Change Simulation Speed process modifies the Simulation Speed parameter stored in the Simulation Parameters data store while the simulation is executing.

3.3.9 Level Three Data Flow Diagram 2.2

The Level Three Data Flow Diagram 2.2 (*Figure 7*) explicitly demonstrates the numerous processes involved when the Invoke Start Command process (2.2) is requested. The Invoke Start

Command process controls the execution of the Simulation Engine. The process specifications that follow the Level Three Data Flow Diagram explain how each process controls the flow of data between the Invoke Start Command process and the Simulation Parameters, Master List and Simulation Statistics List data stores.



3.3.10 Process Specifications for Level Three Data Flow Diagram 2.2

The following process specifications explain each process bubble in the Level Three Data Flow Diagram 2.2 (Figure 7).

3.3.10.1 Process Simulation Parameters (2.2.1)

The Process Simulation Parameters process inputs the current Simulation Parameters from the Simulation Parameters data store and sends them to the Simulation Driver (2.2.2) and Invoke Animation (2.2.4) processes.

3.3.10.2 Simulation Driver (2.2.2)

The Simulation Driver process uses the Master List data store to execute the simulation. Each time an event takes place in the simulation, the Simulation Driver process generates an Animation

Update Request and sends the request to the Invoke Animation Update process (2.2.4) and it sends the current Simulation Statistics to the Update Simulation Statistics process (2.2.3).

3.3.10.3 Update Simulation Statistics (2.2.3)

The Update Simulation Statistics process inputs Simulation Statistics from the Simulation Driver process (2.2.2) and writes the statistics into the Simulation Statistics data store.

3.3.10.4 Invoke Animation Update (2.2.4)

The Invoke Animation Update inputs an Animation Update Request from the Simulation Driver process (2.2.2) and relays the message to the Animation process (3.0).

3.4 Transaction Analysis

The purpose of a transaction analysis is to separate a large system into smaller components of related operations (Budgen, 216). In the following sections, the system DFD is partitioned into smaller sub-systems and the transactions involved in the partitions are explained.

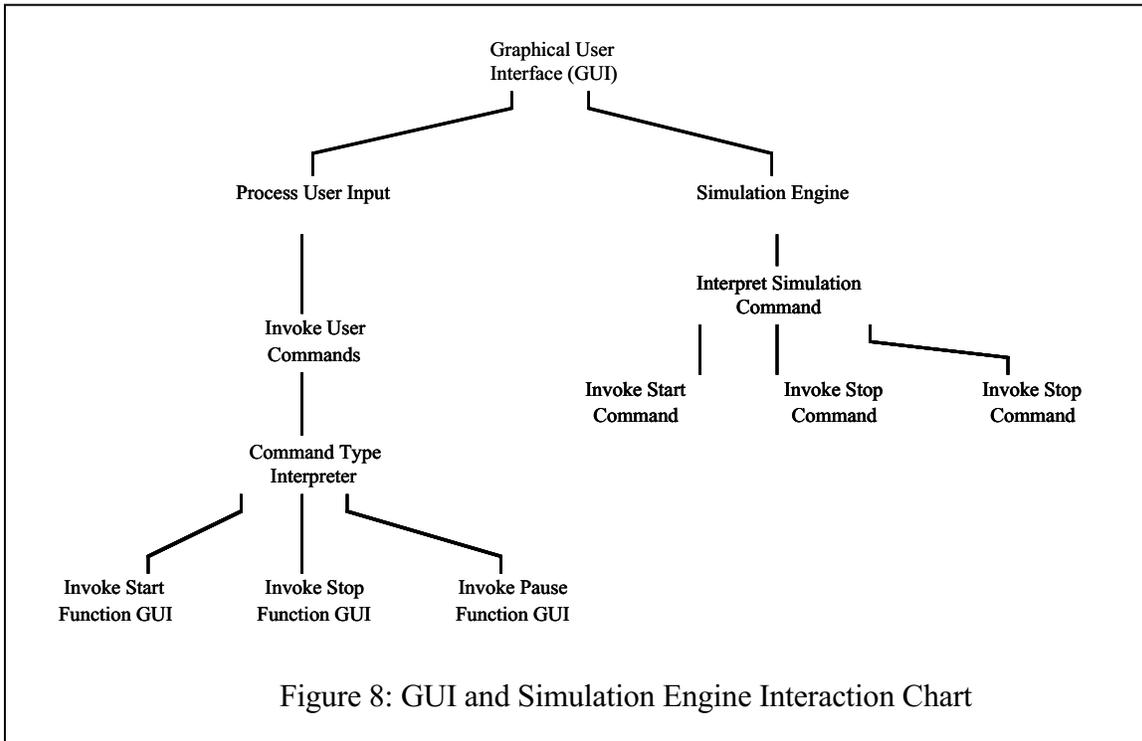
3.4.1 Partitions of the System DFD

The Queuing System design consists of three major sub-systems, the GUI, the Simulation Engine and the Animation. This section shows the interaction of these three modules. The interaction of the whole system is illustrated in section 3.5 by the system Structure Chart (*Figure 10*). The GUI is the sub-system that controls all interaction between the user and the Simulation Engine. The Simulation Engine and the Animation sub-system interact to produce a real-time animated representation of the Queuing System's simulation execution. The interaction of the three systems are illustrated in *Figure 8* and *Figure 9* displayed below.

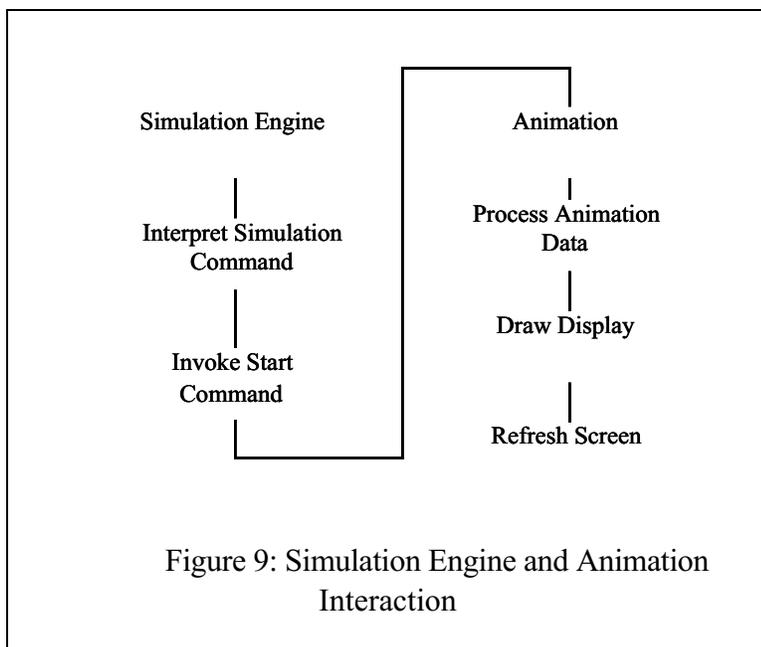
3.4.1.1 GUI and Simulation Engine Interaction

The GUI is used to translate information from the user into a form understood by the Simulation Engine. The GUI also translates information from the Simulation Engine into a form understood by the user. In *Figure 8*, the structure of these interactions is illustrated. The Interpret User Input module determines if the user input is a Simulation Command and sends ,through the GUI module, the request associated with that command to the Simulation Engine. The Simulation

Engine interprets that command request, invokes the corresponding function and tells the GUI module to invoke the proper GUI display for that command.



3.4.1.2 Simulation Engine and Animation Interaction



Each time the Simulation Engine's Invoke Start Command process executes an event it sends an animation request to the Animation process. Upon receiving an animation request, the Animation process produces a new display drawing and updates the display screen. This interaction is illustrated in *Figure 9* to the right.

3.4.2 Identified Transactions

Each identified transaction for the Queuing System is defined using the SSA/SD Method explained in the introduction of this document (Section 1.2). The transactions are explained using the following fields.

- The **EVENT** in the systems environment that causes the transaction to occur
- The **STIMULUS** that is applied to the system to inform it about the event
- The **ACTIVITY** that is performed by the system as a result of the stimulus
- The **RESPONSE** that this generates in terms of output from the system
- The **EFFECT** that this has upon the environment

Each of the identified transactions are listed below:

3.4.2.1 Simulation Execution

The first transaction covers running the simulation using user defined simulation parameters or the default parameters supplied by the system. The only effect that simulation parameter modification has on the system are the statistics produced during simulation execution.

EVENT: The system user selects the Play button on the GUI

STIMULUS: The Invoke Start Command process is called

ACTIVITY: The simulation of the Queuing System is executed

RESPONSE: Simulation Statistics are saved and Animation is displayed

EFFECT: User can view animation without entering simulation parameters

3.4.2.2 Pause Simulation Execution

This transaction covers pausing simulation execution.

EVENT: The system user selects the Pause button on the GUI

STIMULUS: The Invoke Pause Command process is called

ACTIVITY: The simulation is temporarily stopped

RESPONSE: Simulation Pause GUI is displayed

EFFECT: User can choose options from Simulation Pause GUI

3.4.2.3 Stop Simulation Execution

This transaction describes how Simulation Execution Stops are handled.

EVENT: The system user selects the Stop button on the GUI

STIMULUS: The Invoke Stop Command process is called

ACTIVITY: The simulation execution is terminated

RESPONSE: The Simulation Statistics GUI is displayed

EFFECT: The user can view the Simulation Statistics, or exit the program

3.4.2.4 Insert and Delete Functions

The insert and delete functions are available to the user during simulation pauses. The transaction associated with the insert and delete functions are described below.

The insert function allows the user to modify the queue and servers by inserting new customers into either.

EVENT: During a pause, the user chooses the insert option

STIMULUS: A dialog box is displayed giving the user insert options

ACTIVITY: The options selected by the user are executed

RESPONSE: A dialog box asking for user request confirmation

EFFECT: The user can insert customers into the queue or servers

The delete function allows the user to modify the queue and servers by deleting an existing customer from either.

EVENT: During a pause, the user chooses the delete option

STIMULUS: A dialog box is displayed giving the user delete options

ACTIVITY: The options selected by the user are executed

RESPONSE: A dialog box asking for user request confirmation

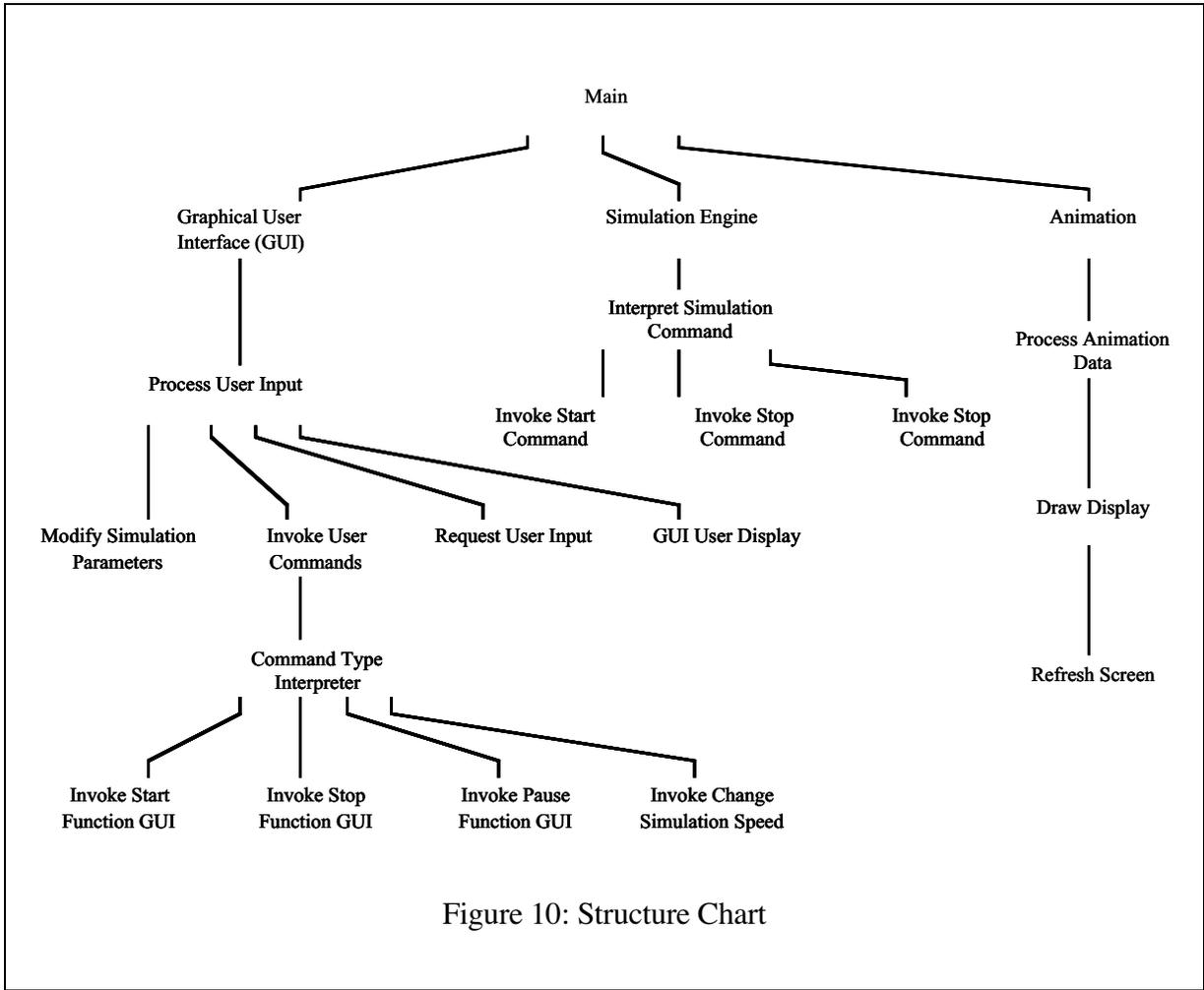
EFFECT: The user can delete customers from the queue or servers

3.5 Transform Analysis

This section illustrates the merging of the Data Flow Diagram partitions from the previous section. This section follows step 5 of the SSA/SD Method (Section 1.2).

3.5.1 Structure Chart

The structure chart of the Queuing System (*Figure 10*) merges the structure charts from section 3.4. The structure charts from section 3.4 were also rearranged to show an overall system hierarchy. The box labeled Main in the following structure chart handles all interaction between the GUI, the Simulation Engine and the Animation modules.



3.6 Design Decision Log

The Design Decision Log is a collection of all major decisions made throughout the design of the Queuing system. Each decision listed in table 1, below, is accompanied by the date the decision was made, an explanation of the decision and the rationale that led to the decision.

Table 1: Design Decision Log

Date of Decision	Design Decision	Rationale for Decision
September 10, 2000	Project Requirements decided and prototyping started.	Beginning of project, need to develop a starting point.
September 28, 2000	SRS version 1 completed and sent to Design team for PDR.	Requirements tentatively stated, so design process can begin.

Date of Decision	Design Decision	Rationale for Decision
October 10, 2000	PDR revision planned.	Initial design needs to be reworked to clarify DFD's and process specifications.
October 26, 2000	CDR revision planned.	DFD's and process specifications need further clarification.
November, 5 2000	Final GUI selected	Found the most user friendly GUI for the time allowed for the decision/
November 7, 2000	Design Notebook version 2 completed.	Design is complete and ready to pass to implementation and testing group.
November 7, 2000	Design Notebook revision planned.	Design Notebook may need revision before included with final submission of project artifacts, so time is being reserved.

4 References

1. IEEE Standard 1016.1-1993. IEEE Guide to Software Design Descriptions, 1993.
2. Pressman, Roger P. Software Engineering: A Practitioners Approach, Fourth Edition. McGraw-Hill, New York, 1997.
3. Sheldon, F.T. Data Element Description Handout, date received: October 25, 2000.
4. Sheldon, F.T. Design Notebook Guidelines and Standards, <http://www.eecs.wsu.edu/~sheldon/cs422.html>, date viewed: October 24, 2000.
5. Sheldon, F. T. General Documentation Style Guidelines and Standards, <http://www.eecs.wsu.edu/~sheldon/cs422.html>, date viewed: October 23, 2000.
6. Sheldon, F.T. Overview of SSA/SD Structured System Analysis and Structured Design Method, <http://www.eecs.wsu.edu/~sheldon/cs422.html>, date viewed: October 30,2000.
7. Sheldon, F.T. Project Requirements, <http://www.eecs.wsu.edu/~sheldon/cs422.html>, date viewed: September 30, 2000.
8. Sommerville, Ian. Software Engineering, Sixth edition. Addison-Wesley, 2001.

5 Glossary

CDR – Critical Design Review: Final design review presented to customer to demonstrate the design for the Queuing System.

DFD – Data Flow Diagram: a diagram used to describe a problem oriented view of the workings of a system. A DFD provides a description based on modeling the flow of information around a network of operational elements, with each element making use of or modifying the information flowing into that element (Budgen, 96).

GUI – Graphical User Interface: a user interface based on the WIMP interaction style.

PDR – Preliminary Design Review: Initial design review presented to customer. Action Items are produced to determine the direction of the design strategy.

P-Spec – Process Specification: a specification of the processes symbolized by bubbles in a Data Flow Diagrams.

RTM – Requirements Traceability Matrix: a table that is used to identify, track and verify each system requirement.

SRS – Software Requirements Specification: an artifact produced by David Doran of Yellow NEARPS that specifies all requirements of the Queuing System.

Structure Chart – a chart that illustrates the hierarchy of the system's processes.

WIMP – Windows, Icons, Menus and Pointers interaction style. An example of a WIMP interaction style is the Microsoft Windows operating system.

APPENDIX A: Data Dictionary

The Data Dictionary for the Queuing System contains explicit definitions of each major variable used and each major message passed through the Queuing System. The entries are organized into three sections 1) Simulation Parameters 2) Simulation Statistics and 3) List Variables. Each Data Dictionary entry is defined using the following criteria (Dr. Sheldon handout):

- Name: identifies the variable or message name
- Description: gives a brief description of the variable
- Used In: provides a reference to the functional units using this variable
- Units: indicates the unit of measure for the data contained in the variable being used
- Range: specifies the acceptable range of data values for the variable
- Data type: specifies the data type to be used when declaring the variable during coding
- Attribute: indicates whether or not the variable contains data, control information or a data condition
- Data Store Location: references the common region where the variable must be stored
- Accuracy: dictates the degree of accuracy required for output comparisons to be made between implementations.

If any of the above fields do not apply to a given variable, the field is marked with N/A or TBD where the value is to be determined later.

Simulation Parameters:

Name: Line cutting (line_cutting)

Description: The method to be used when dissatisfied customers rejoin the line.

Used In: Simulation queuing option

Units: none

Range: 0 or 1

Data Type: boolean

Attribute: data condition

Data Store Location: Simulation Parameters

Accuracy: N/A

Name: Number of servers (num_servers)

Description: The number of servers fed by the queue in the simulation.

Used In: Simulation execution

Units: none

Range: 1 to 10

Data Type: int

Attribute: contains data

Data Store Location: Simulation Parameters

Accuracy: N/A

Name: Interarrival stream (stream_interarrival)

Description: The random number stream used to generate arrival times.

Used In: Simulation execution

Units: minutes

Range: 0 to 10

Data Type: int

Attribute: control information

Data Store Location: Simulation Parameters

Accuracy: N/A

Name: Number of busy servers (num_busy)

Description: Keeps track of the number of busy servers for the time average statistics.

Used In: Simulation Execution

Units: none

Range: 0 to number of servers

Data Type: int

Attribute: contains data

Data Store Location: Simulation Parameters

Accuracy: N/A

Name: Service stream (stream_service)

Description: The random number stream used to generate service times.

Used In: Simulation Execution

Units: minutes

Range: TBD

Data Type: int

Attribute: control data

Data Store Location: Simulation Parameters

Accuracy: N/A

Name: Satisfaction stream (stream_satisfaction)

Description: The random number stream used to generate satisfaction probabilities.

Used In: Simulation Execution

Units: none

Range: TBD

Data Type: int

Attribute: control data

Data Store Location: Simulation Parameters

Accuracy: N/A

Name: Uniform distribution a (dist_a)
Description: The lower limit used to determine uniform service time.
Used In: Simulation Execution
Units: none
Range: TBD
Data Type: double
Attribute: contains data
Data Store Location: Simulation Parameters
Accuracy: N/A

Name: Uniform distribution b (dist_b)
Description: The upper limit used to determine uniform service time.
Used In: Simulation Execution
Units: none
Range: TBD
Data Type: double
Attribute: contains data
Data Store Location: Simulation Parameters
Accuracy: N/A

Name: Dissatisfaction probability (prob_dissatisfied)
Description: The probability that a client will not be satisfied after a service completion.
Used In: Simulation Execution
Units: none
Range: TBD
Data Type: double
Attribute: contains data
Data Store Location: Simulation Parameters
Accuracy: N/A

Name: Simulation runtime (runtime)
Description: The length of time in minutes the simulation should run.
Used In: Simulation Execution
Units: minutes
Range: 0 to 1000
Data Type: double
Attribute: control data
Data Store Location: Simulation Parameters
Accuracy: N/A

Name: Mean interarrival time (mean_interarrival)
Description: The mean time used to determine arrival times.
Used In: Simulation Execution
Units: minutes
Range: TBD

Data Type: double
Attribute: control data
Data Store Location: Simulation Parameters
Accuracy: N/A

Name: Simulator time (sim_time)
Description: The current time of the simulation.
Used In: Simulation Execution
Units: minutes
Range: 0 to simulation runtime
Data Type: double
Attribute: control data
Data Store Location: Simulation Parameters
Accuracy: N/A

Name: Random number generator (rand)
Description: An object with the random number generation utilities.
Used In: Simulation Execution
Units: none
Range: TBD
Data Type: class SimRandomNumber
Attribute: control data
Data Store Location: Simulation Parameters
Accuracy: N/A

Name: Current play speed (sim_speed)
Description: Keeps track of the current play speed of the GUI.
Used In: Simulation Execution
Units: none
Range: TBD
Data Type: class Jslider
Attribute: control data
Data Store Location: Simulation Parameters
Accuracy: N/A

Name: Current simulator state (sim_state)
Description: Keeps track of the current state of the simulator (playing, paused, etc.).
Used In: Simulation Execution, Simulation Pause
Units: none
Range: TBD
Data Type: int
Attribute: control data
Data Store Location: Simulation Parameters
Accuracy: N/A

Simulation Statistics:

Name: Service time statistics (servicest)

Description: Keeps track of the total service time statistics of clients.

Used In: Simulation Statistics Display

Units: minutes

Range: TBD

Data Type: class SimSampst

Attribute: contains data

Data Store Location: Simulation Statistics List

Accuracy: N/A

Name: Busy server statistics (busyst)

Description: Keeps track of the number of busy servers over time.

Used In: Simulation Statistics Display

Units: minutes

Range: TBD

Data Type: class SimTimest

Attribute: contains data

Data Store Location: Simulation Statistics List

Accuracy: N/A

Name: Number of clients (num_clients)

Description: The total number of clients served.

Used In: Simulation Statistics Display

Units: none

Range: not bounded

Data Type: int

Attribute: contains data

Data Store Location: Simulation Statistics List

Accuracy: N/A

Name: Number satisfied (num_satisfied)

Description: The number clients satisfied after the first service.

Used In: Simulation Statistics Display

Units: none

Range: 0 to number of clients

Data Type: int

Attribute: contains data

Data Store Location: Simulation Statistics List

Accuracy: N/A

List Variables:

Name: Servers (server[num_servers])
Description: Five servers are used to service a customer
Used In: Simulation Engine and Animation
Units: N/A
Range: N/A
Data Type: array of class SimList
Attribute: control information and data condition
Data Store Location: Master List
Accuracy: N/A

Name: Queue
Description: A line of customers waiting for an available server
Used In: Simulation Engine and Animation
Units: TBD
Range: TBD
Data Type: class SimList
Attribute: contains data
Data Store Location: Master List
Accuracy: N/A

Name: Event List (event_list)
Description: Used to drive the simulation.
Used In: Simulation Engine
Units: TBD
Range: TBD
Data Type: class SimEventList
Attribute: control information
Data Store Location: Master List
Accuracy: N/A

APPENDIX B: Project Schedule

Each member of Yellow NEARPS was assigned separate tasks involved in the construction of the Queuing System. This appendix covers the group members, their responsibilities and the time schedule for each of the Queuing System’s deliverable artifacts. The Schedule includes a table defining each of the tasks and any limitations each task may encounter. The Schedule and table are current as of November 8, 2000. (Student names removed in Fig. A-1)

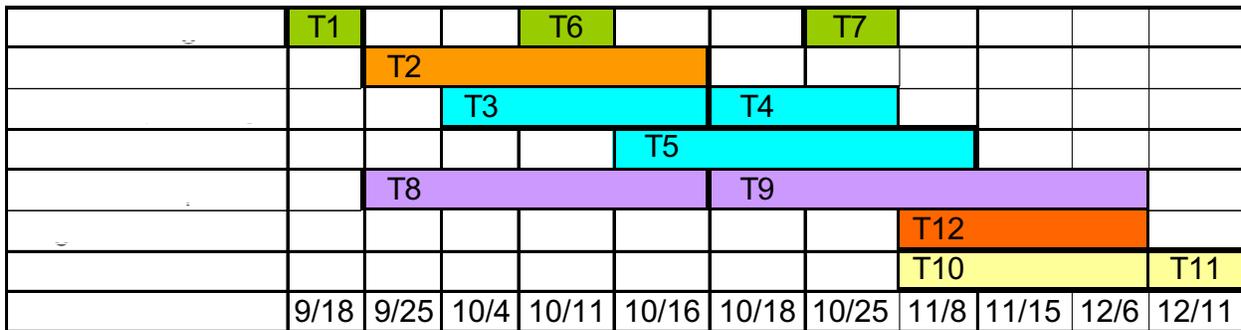


Figure A-1: Queuing System Project Schedule

Table A-1: Schedule Tasks Key and Limitations

Tasks	Limitations
T1: Project Plan	N/A (Completed)
T2: SRS	N/A (Completed)
T3: PDR	N/A (Completed)
T4: CDR	N/A (Completed)
T5: DNB	Time Limitation
T6: PDR Presentation	N/A (Completed)
T7: CDR Presentation	N/A (Completed)
T8: Simlab Java port	N/A (Completed)
T9: Core Coding	Time Limitation
T10: UM	Time Limitation
T11: Demo	Hardware Availability, Time Limitation
T12: Test Report	Time Limitation

APPENDIX C: Requirements Traceability Matrix

Appendix C contains the Requirements Traceability Matrix (RTM). The RTM is used to aid the verification of requirements for the Queuing System, as outlined in the Traceability Approach section of this document’s Introduction (Section 1.3). The blank columns in this table will be filled during the implementation and testing phases (Module Name and Tested, respectively).

Table C-1. Requirements Traceability Matrix.

Req. ID System Level.	Req. ID Sub-system Level.	DFD Identifier(s)	Module Name	Verification Method*	Tested
A001		1.0			
	A001.1	1.3		T	
	A001.2	1.3		T	
A002		2.0			
	A002.1	2.3		I or A	
	A002.2	2.2		T or I	
	A002.3	2.4			
A003		2.3			
	A003.1	2.3/1.4		A	
	A003.2	2.3/1.4		T or A	
	A003.3	2.3/1.4		T or A	
A004		2.3/1.4			
	A004.1	2.3/1.4		A	
	A004.2	2.3/1.4		A or T	
A005		2.0			
	A005.1	2.0		T or I	
	A005.2	2.0		A or I	
A006		2.1.2			
	A006.1	2.1.2		T or A	
	A006.2	2.1.2		A or I	
A007		2.1.2			
	A007.1	2.1.2		I or T	
	A007.2	2.1.2		A or T	
A008		2.1.2			
	A008.1	2.1.2		A or T	
	A008.2	2.1.2		I or T	

Req. ID System Level.	Req. ID Sub-system Level.	DFD Identifier(s)	Module Name	Verification Method*	Tested
A009		2.1.2			
	A009.1	2.1.2		A or I	
	A009.2	2.1.2		A or I	
A010		2.1.2			
	A010.1	2.1.2		A or I	
	A010.2	2.1.2		A or I	
A011		2.1.2			
	A011.1	2.1.2		T or A	
	A011.2	2.1.2		T or A or I	
A012		2.1.2			
	A012.1	2.1.2		A or T	
	A012.2	2.1.2		T or A	
A013		2.1.4			
	A013.1	2.1.4		T or I	
	A013.2	2.1.4		T or I	

* Verification Method Key: T = Testing, I = Inspection, A = Analysis

APPENDIX D: Identified Test Cases

The test cases in this appendix were derived from the Preliminary Design Review (PDR), the Critical Design Review (CDR) and the Data Notebook version 1.5. The test cases are divided into three groups. The first group of test cases are designed to test the basic functionality of the Queuing System. The second group of tests are designed to test the functions of the GUI. The third set of test cases are designed to test the customer requested insert and delete functions. All test cases are described using the following fields.

- Purpose(s): The data or functionality that is verified by the test case.
- Testing Method: The actions taken to execute the test case.
- Expected Output: The output that is expected from the test case.

D.1 Queuing System Functionality Test Cases

There are two test cases that are designed to test the Queuing system's basic functionality. The two test cases are defined below.

Case1: Run Simulation Without Modifying Simulation Parameters

Purpose: To verify correct execution of the simulation engine and animation.

Method: Select Play button on the GUI with default Simulation Parameters.

Expected Output: The system runs for the duration of the simulation time and displays the Simulation Statistics upon completion of the simulation.

Case 2: Modification of Simulation Parameters

Purpose: To verify that modification of Simulation Parameters does not adversely effect the Simulation Execution.

Method: Change Simulation Parameters and select the Play button on the GUI.

Expected Output: The system runs smoothly and at the end of the simulation time, the Simulation Statistics are displayed on the screen.

D.2 GUI Functionality Test Cases

The GUI Functionality Test Cases are designed to verify the correct execution of the GUI during simulation stops, pauses, and simulation speed changes.

Case 3: Stop Simulation

Purpose: To verify that the correct GUI is displayed after the user chooses to stop the simulation.

Method: During a simulation execution, select the Stop button on the GUI.

Expected Output: The simulation is terminated and the Simulation Statistics GUI is displayed on the screen.

Case 4: Pause and Resume a Simulation

Purpose: To verify that the correct GUI menu is displayed when the user chooses to pause the simulation. To verify that the simulation resumes correctly after the user chooses to resume the simulation.

Method: During a simulation execution, select the pause button on the GUI, then select the Resume button.

Expected Output: The simulation is paused and the Pause GUI menu is displayed. After the simulation is resumed, the simulation continues without trouble.

D.3 Delete and Insert Functions Test Cases

The two following test cases are designed to verify that the user defined insert and delete functions execute correctly.

Case 5: Inserting Customers into the System

Purpose: To verify that a customer can be added into the system without negatively effecting the simulation execution.

Method: During a simulation pause, choose the insert option from the menu and fill in requested information to insert a customer into the system.

Expected Output: The customer is inserted into the system at its assigned position and the simulation runs smoothly after execution is resumed.

Case 6: Deleting Customers from the System

Purpose: To verify that a customer can be removed from the system without adversely effecting the simulation execution.

Method: During a simulation pause, choose the delete option from the menu and fill in requested information to insert a customer from the system.

Expected Output: The selected customer is removed from the system and the simulation runs smoothly after the simulation execution is resumed.