Bernard Kakengi

Modern Queueing Management System: QCracker

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Bachelor´s Thesis
5 November 2012

Abstract

Author(s) Title Number of Pages Date	Bernard Kakengi Modern Queueing Management System QCracker 47 5 November 2012
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Software Engineering
Instructor(s)	Godfred Mathias, QCracker-Project Manager Olli Hämäläinen , Senior Lecturer

The purpose of this project was to develop a modern and affordable queueing management system that would take advantage of modern available software development tools and networking infrastructure unlike traditional queueing management systems. The primary targeted markets are the emerging markets in Africa especially the eastern part of Africa. The system was intended to be affordable by any business entity and would be localized to meet people's expectations. In order to achieve this vision a company called Kifaru software solution Ltd was set up to carry out the development.

QCracker is based on client-server architecture. The QCracker client software communicates with QCracker server over the Local Area Network. The QCracker server and client software use QT (Nokia) framework. Currently both the client and the server are running in an MS Windows environment, but in the near future there is a plan of porting the server to Linux.

The result of the project shows that it is always fair to be treated equally with respect considering the first in, first served (FIFO) system. Modern Queueing Management System is applicable anywhere where people queue up for services. It saves the customer's and service provider's time, increases efficiency, productivity of the staff and customer confidence towards their service provider.

Keywords	Modern Queueing Management System

Contents

1 Introduction

The QCracker system is an attractive and modern computerized Queueing Management System. QCracker is made up of a digital LED display mounted on the counter station to display the queue number of a customer being served, client software application installed in a counter's computers, a touchscreen displaying available services to customers, a thermal Point of Sale (POS) ticket printer and a loudspeaker for announcing the queue number to be served next and the station location.

The goal of this project was to design from scratch and develop a modern queueing system that would take advantage of modern available software development tools and networking infrastructure unlike traditional queuing management systems.

The following situation could be considered as an example. A group of 100 people are waiting to be served in a post office for instance without any order. Undoubtedly this will lead to a chaos, unfairness and a noisy and unpleasant environment to be served in. The time of customers and service providers will be wasted. The service provider will end up with unhappy customers and that is negative thing to the service provider in today's competitive business.

Figure 1 below illustrates a general idea of QCracker in graphical notation, tells how the system start by arrival, waiting positions and server which serve the client and then departure.

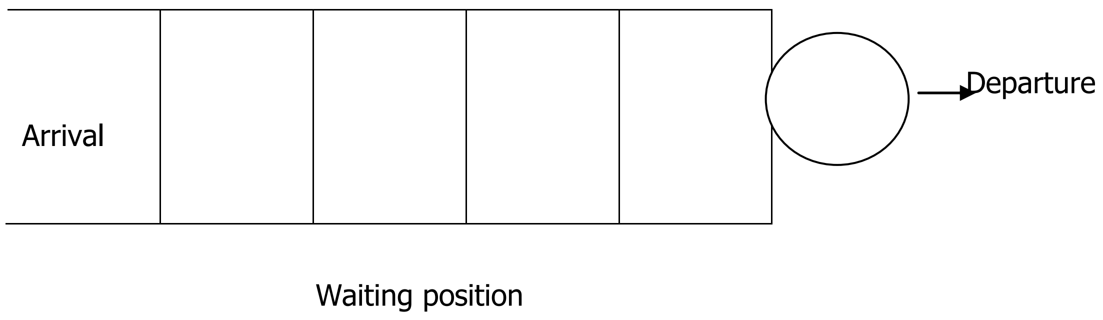


Figure 1. Standard graphical notation for QCracker

Figure 1 above, a standard graphical notation for QCracker is illustrated. The open rectangles with slots represent queues, and the circle represents the server. The path is expressed as lines with arrows. Although the graphical notation expressively depicts the process of a queueing system, does not distinguish between tokens.

The objective of QCracker system is to provide the following functionality to the service provider:

- Thermal printed tickets with queueing information
- Real-Time counter service analysis and queue trends
- Easy to add/delete/edit features of QCracker queues
- Improved counter visibility with bright counter LEDs
- The system to take advantage of the existing local area network (LAN) infrastructure and therefore no extra hardware or cabling required when installing and commissioning the system. The only required hardware is the LED and loudspeaker (this makes the system cheaper than other Queueing Management Systems on the market).
- QCracker to provide all User Interfaces in African local chosen official languages. This makes the system user- friendly and easily acceptable by local people.
- The system to increase efficiency and productivity of the staff working at counters and their managers. This with other benefits described above will increase trust amongst its customers.
- Queue jumping, unaccounted delays and contention amongst customers to make history by introducing QCracker.
- The image of the service provider to emerge as organized, professional and concerned with the well being of customers

QCracker follows a client-server model; the architecture of the server plays an important role in determining the performance and scalability of the entire system. A server application in this system is multi-threaded to enable concurrent processing of multiple client requests. The multi-process model involves frequent context switching between the various processes and the use of Inter- Process Communication primitives.

Figure 2 illustrates a designed of QCracker which consists of several clients and a server which give clients services needed, and all this connection is through a Local Area Network

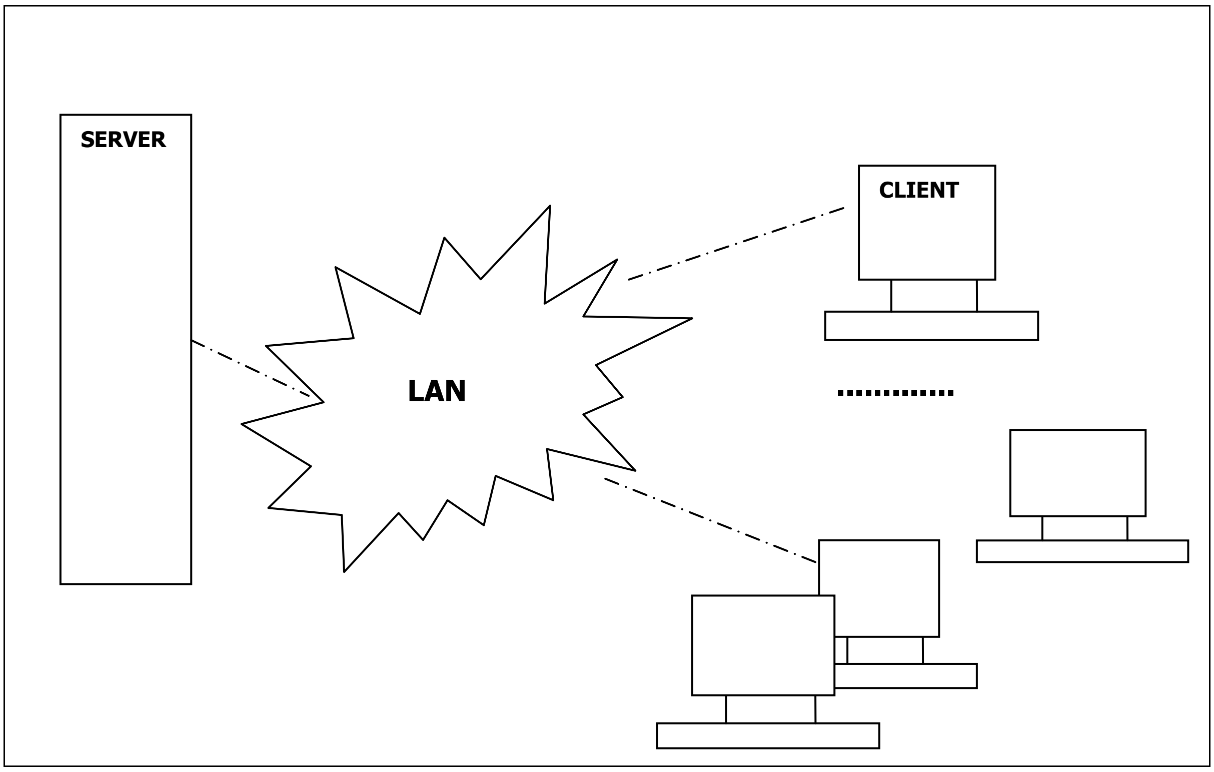


Figure 2. QCracker client-server general model

Figure 2 above shows a QCracker client-server general model, a client is the side that initiates the communication process, where as the server responds to incoming client requests and the server provides services to several clients across the local area network (LAN). The clients are the ones which are service requestors and send requests for services and data to the server. The server is the service provider. On receiving a client request, the server processes the request and replies back to the client from whom the request originated. All communication from the server application passes through the network subsystem of the operating system and is then directed to the outside world through the network adapter. Three major components that constitute an Internet server are the server application, the operating system and the interfaces from the server application to the operating system.

2 Qcracker System Context

Figure 3 below elaborates how the QCracker system interacts with External users as well as External output devices and External Input Output devices, which are all connected to the QCracker system

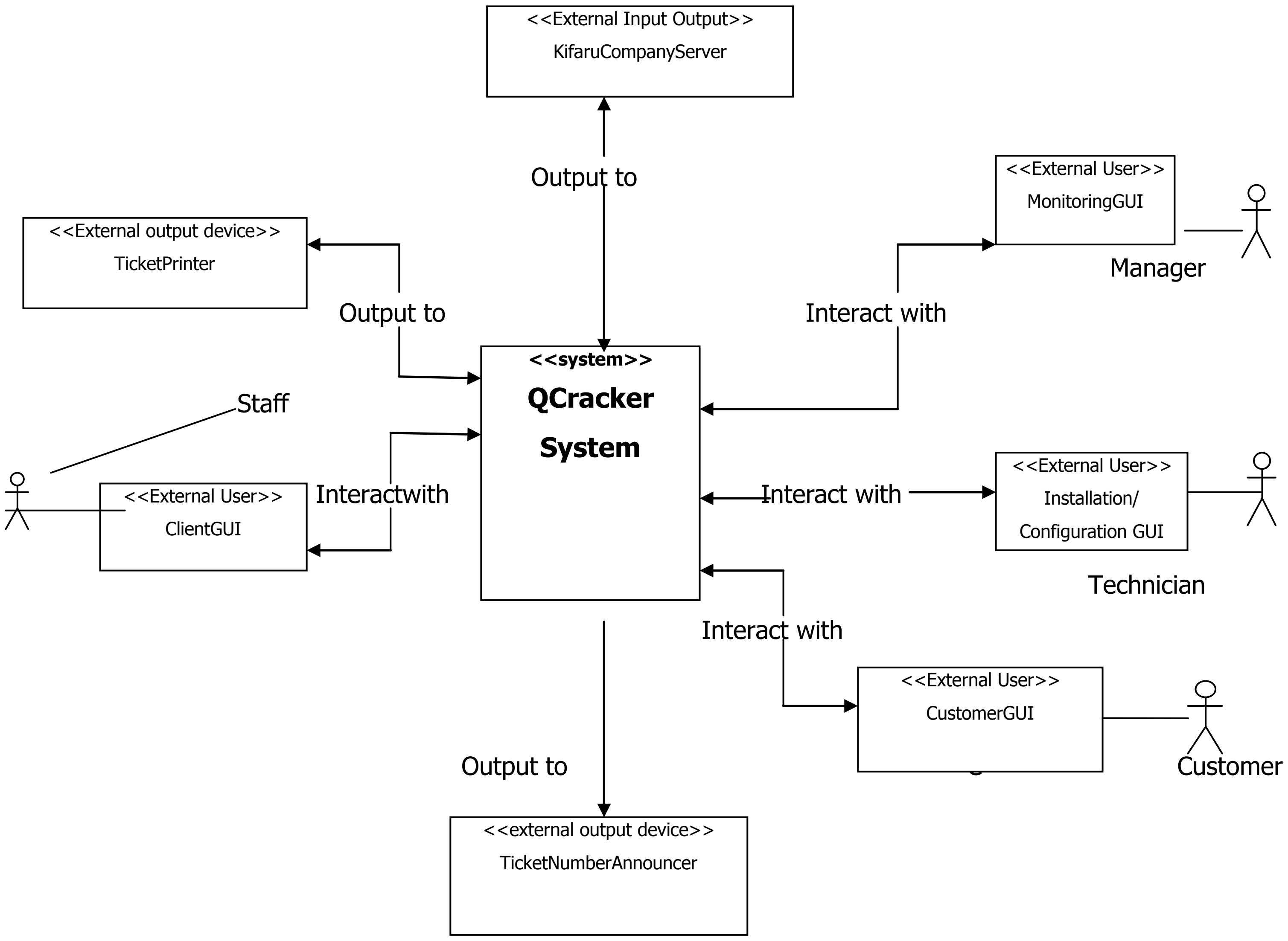


Figure 3. QCracker System Context Diagram

The QCracker system context diagram in Figure 3 above depicts the interface between the system and the external environment. The diagram shows the external users and how they interact with the external system classes. The staff or clerk interacts with the system via a Client GUI (Graphical User Interface) interface class. The customer interacts with the system using the class interface Customer GUI.

The technician who is responsible for installing and configuring the system interacts with the system using the interface class the Installation/Configuration GUI. The QCracker system will use the TicketPrinter interface class to output the ticket data to the Point of Sale (POS) printer and it will also output the audio messages to the loud-speakers via an interface TicketNumberAnnouncer. The manager monitors the queues using the external interface class MonitoringGUI.

3 Qcracker System Configuration

This section describes the general overview of the QCracker System configuration. It contains the QCracker basic components and how each component is connected to each other so that the whole system could function.

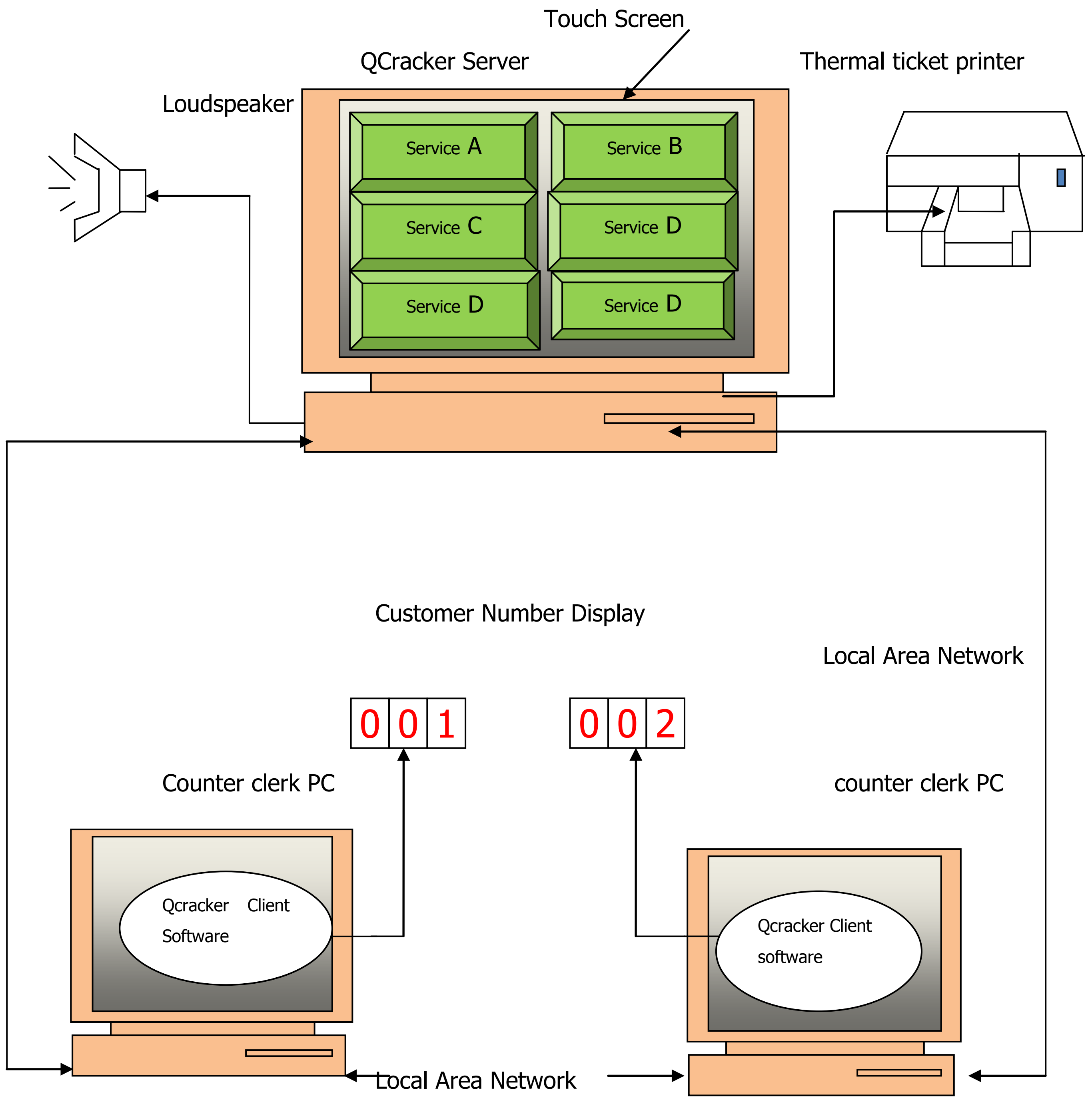


Figure 4. QCracker System Configuration

Figure 4 shows the basic overview of the QCracker system configuration. The client is connected to the LED digit display via USB. The client software controls the digit LED display. The server and clients are connected via Local Area Network. The server is connected to a thermal POS printer via USB. The loudspeaker is connected to the server via an audio output interface.

The customer will be presented on a modern touch screen connected and controlled by the QCracker Server with a list of available services/departments in a graphical form. The services are presented as soft buttons. The customer using a finger selects a service. The ticket with queuing information including a branch name, date and time of issue (A number indicating the customer position in the queue, waiting time and other customized messages) is issued by the thermal printer connected to the QCracker server as shown in figure 4. The customer takes a ticket, relaxes and waits anywhere in the lobby to be called. Light emitting diodes (LEDs) are connected through the QCracker client software and they are capable of bright 7 segment units to displays client number and the serving counter.

4 System Top Level Requirements

There were several software development steps taken into account when QCracker was built:

- Feasibility involved gathering information from potential customers in East Africa.
- High- level design based on our potential customer requirements. Some of the requirements have been generated internally.
- Implementation(actual coding) of software
- Testing the software which is still in progress
- Deployment which will take place soon
- Maintenance which will be done after deployment

The company's clients who showed interest in the system allowed Kifaru Software Solution Ltd (the company) to carry out the development of the system. The main goal was to create with a working prototype that could be demonstrated to the clients and then after the demo clients would specify more detailed requirements.

The QCracker customer (user of the system) upon arrival is presented with a list of available services in a graphical form on a modern touch screen controlled by the QCracker Server. The services are presented as soft buttons. The customer using a finger selects a service. The ticket with queueing information e.g. a number indicating the customer position in the queue, waiting time and other customized messages is issued by the thermal POS printer connected to the QCracker server. The customer takes a ticket and waits anywhere in the hall to be called.

The counter clerk presses the next customer button on the QCracker client software which sends a request to the QCracker server which retrieves the next number in front of the queue and sends it to the QCracker client software as a response. Upon receipt of the number the client software sends the number to an LED display which is connected to the counter clerk's PC. The LED will flash the displayed number several times to indicate the location of the clerk counter. As the number flashes the server via the loudspeaker will announce, in a local language, the customer number and the location of the serving clerk.

The QCracker system collects, analyzes and stores the counter clerk's service statistics which will be sent over to the manager when the clerk exits the QCracker client software.

After the customer is served, he or she can leave his or her feedback on the customer feedback units; managers/supervisors can monitor the performance of their staff through the real time reports generated by the QCracker server [3]. Section 4.1 specifies the functional requirements of the QCracker in terms of use cases, class and sequence diagrams.

4.1 Top level use case diagram

This section specifies the top level functional requirements by a top level use case diagram as shown in figure 5.

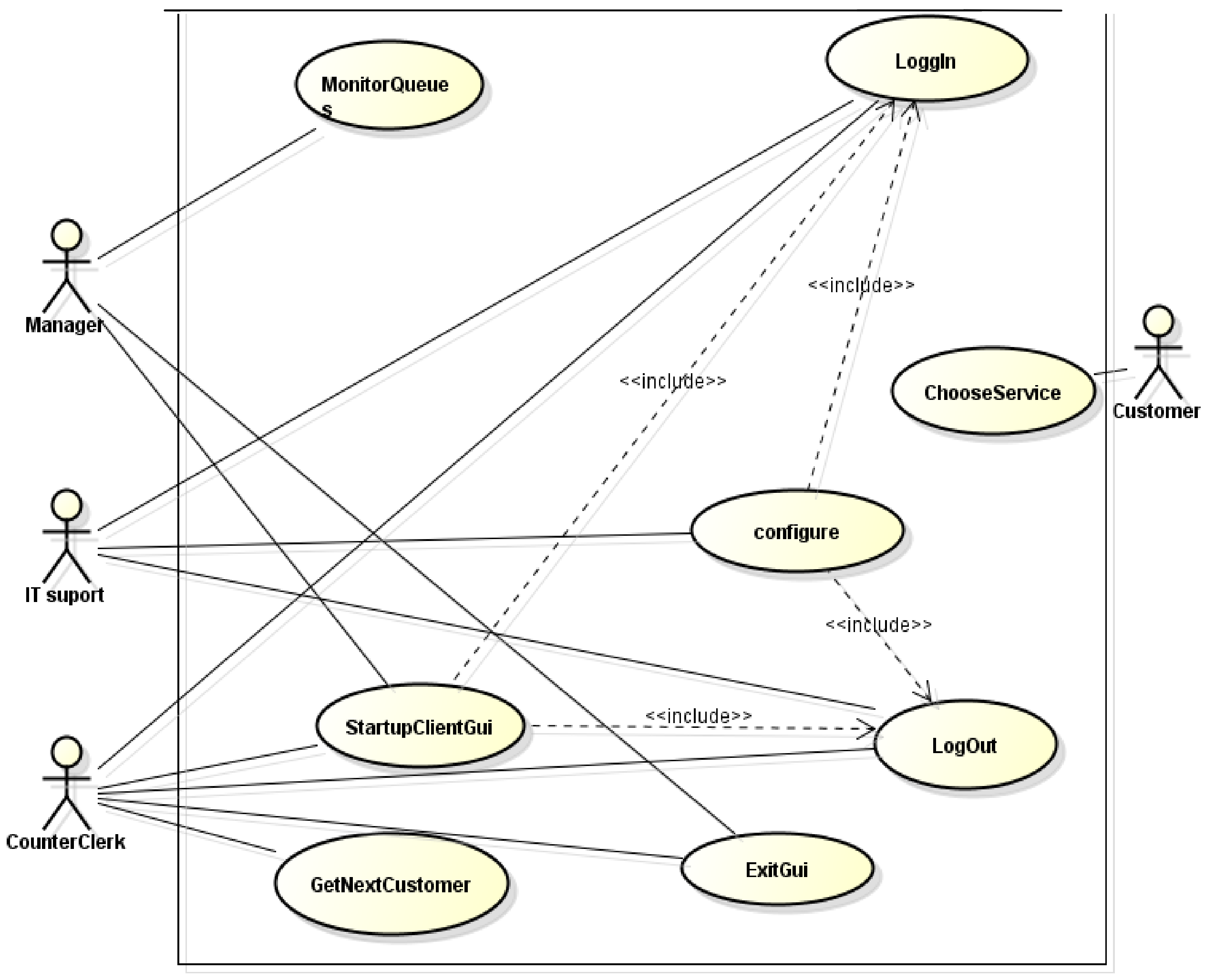


Figure 5. Top level use case diagram

Table 1 is detailed description of actors taken from figure 5 above in the QCracker system and what they do specifically. Table 1 provides a simple classification of actors and full descriptions of what specific actor interacts or uses the system.

Table 1. Description of use case diagram

Actor	Description
Counter Staff/Clerk	The Counter Staff is interested in getting the next customer in the queue. The system provides the next queue number as a digit.
IT Support	The IT staff installs the client and server client software and configures them
Manager	Logs on to the system and monitors the queues on real time basis.
Customer	Presses the button of the desired service and gets the ticket showing the position in the queue.

As table 1 illustrates, main actors in the system are divided into two, in which one concerns about staff and other is non-staff which is only a customer in this case.

4.2 Detailed Description of Actors and Use Cases

Table 2 specifies the functional requirements primarily associated with Counter Clerks.

Table 2. Counter Clerk Description

Actor	Definition	Required capabilities
Counter Clerk	Role played by service provider employee who provides front line service to customer	Basic computer (window) skills Use a QCracker Client Software to log onto and navigate the QCracker Client User Interface

Figure 6 shows an actor counter clerk and five different use cases of what he can perform in the QCracker system.

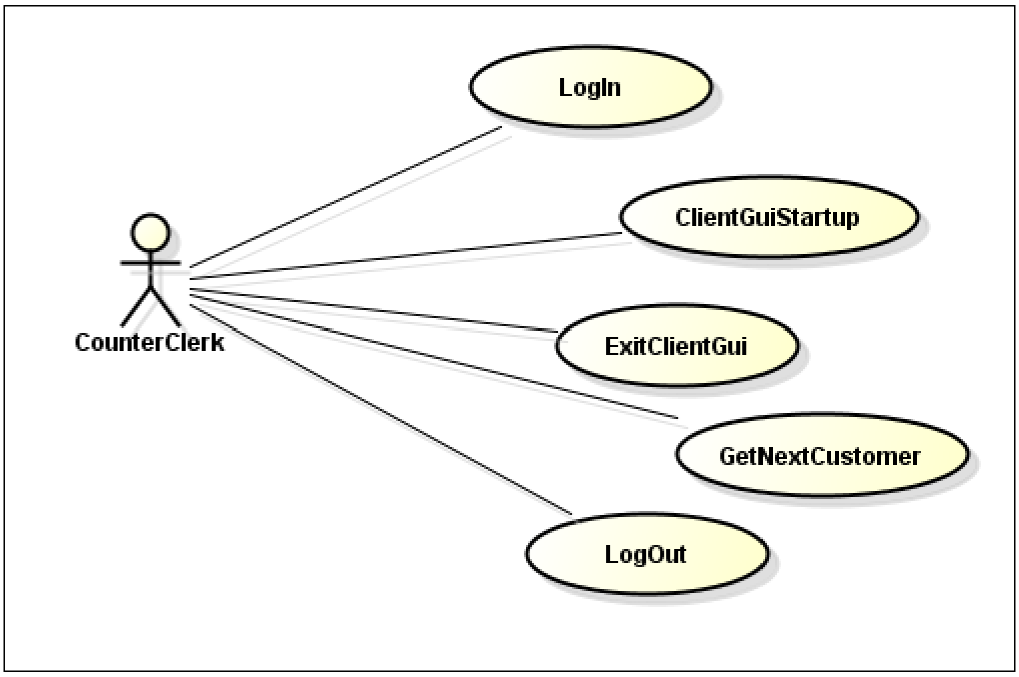


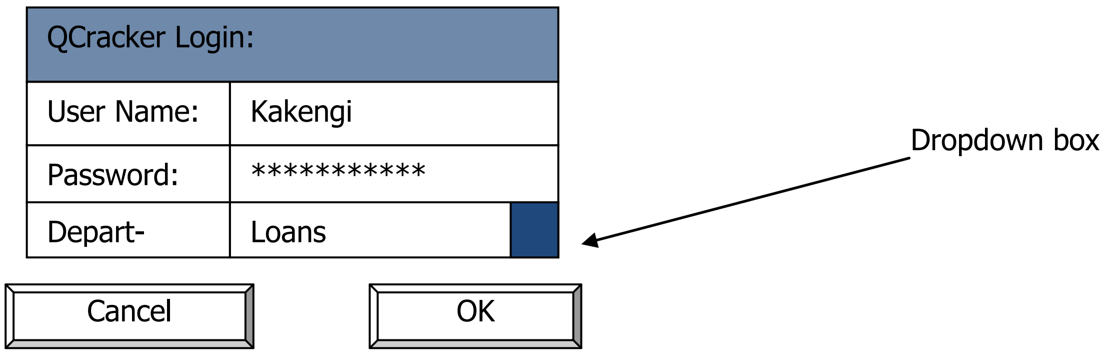
Figure 6. Counter Clerk Use Case Diagram

Figure 6 shows that counter clerk can first login the system and then he can start up

the client GUI and exit client GUI as well and after his day is over ,then he can logout from the system.

4.2.1 Description of Use case: Client GUI startup

The QCracker system by counter clerk starts up the client software GUI as illustrated in figure 7, and most of the preconditions taken into account before startup are that no QCracker client process is running on the counter clerk computer and the QCracker server must be up and running at that time. Counter clerk launches the client by either double clicking the QCracker client icon on the desktop or selects the program from Start->All Programs. Then the counter clerk is prompted with the following QCracker login dialog box.

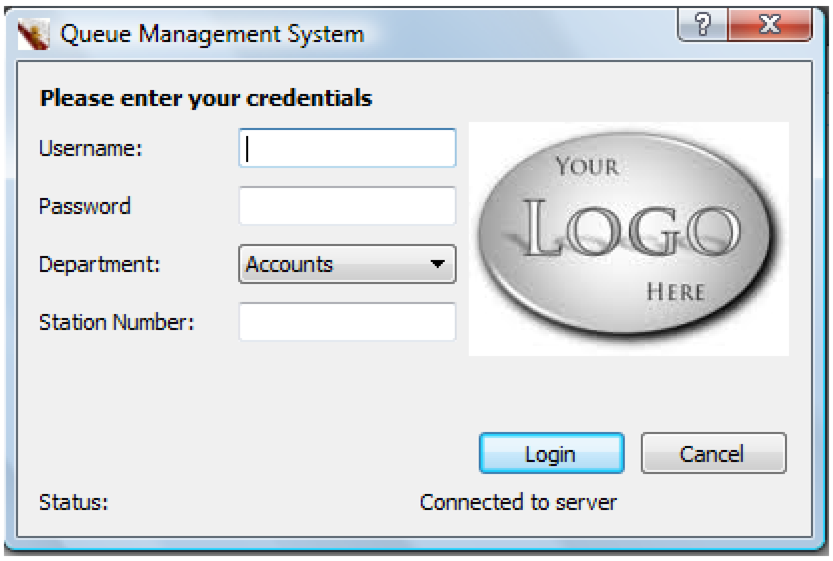


QCracker Login:	
User Name:	Kakengi
Password:	***********
Depart-	Loans

Figure 7. QCracker Login Dialogue Box

As seen in figure 7, the counter clerk enters the credentials and presses the OK button. The QCracker system successfully validates the Clerk, dismisses the login dialog box and displays the following Client. All buttons in the GUI are enabled.

Figure 8 illustrates a client GUI showing how a clerk can be authenticated before going to serve the customer, the system will validate and allow him and then the really client GUI will appear as shown in figure 8



Queue Management System

Please enter your credentials

Username:	
Password	
Department:	Accounts
Station Number:	

Status: Connected to server

Figure 8. Client GUI

Figure 8 shows how the clerk can log into the system and the system will validate if the clerk have access.

If the clerk has access to the QCracker system, then another client GUI will prompt. This GUI is the one which will allow the clerk to operate the NextCustomer button

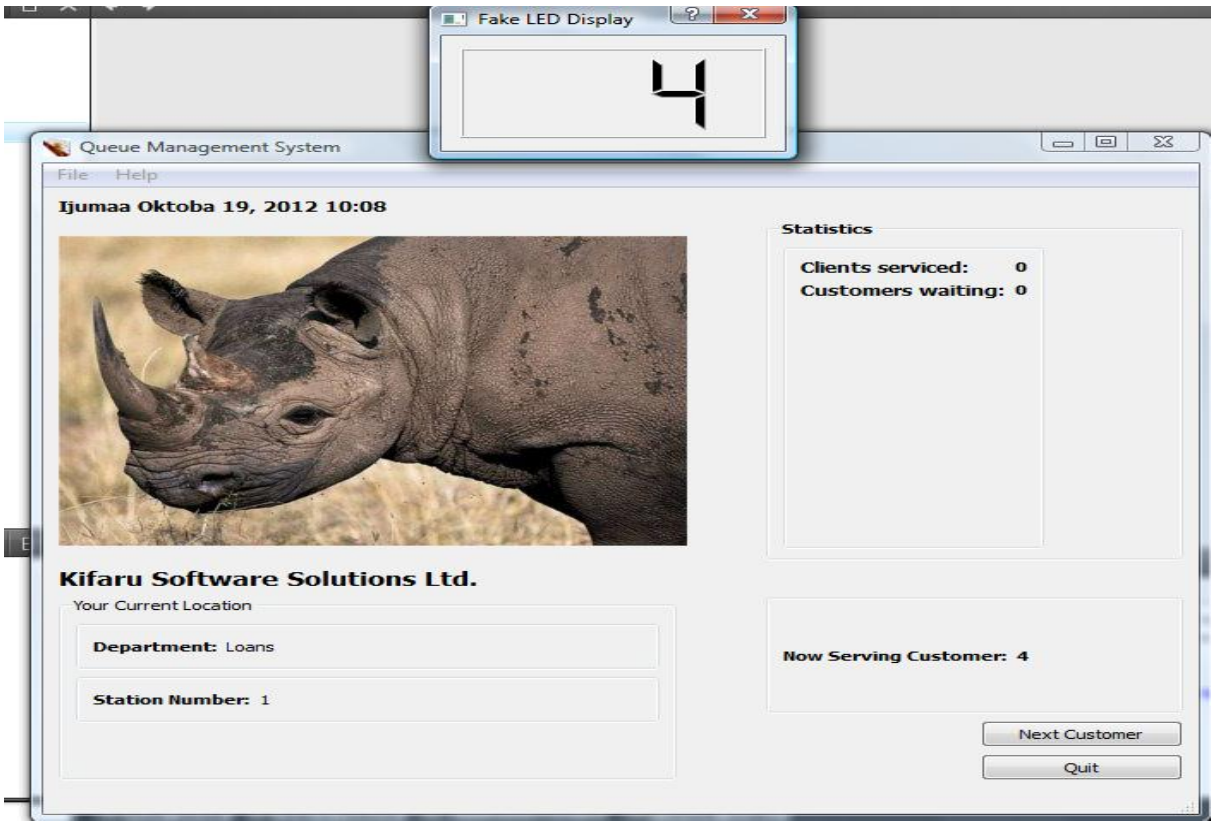


Figure 9. Client GUI after login

Figure 9 above illustrates that, if the system fails to connect the clerk, then these are the options taken into account by the system:

- If the credentials become rejected, the system will clear all the fields in the log-in dialog box and display the message 'The password or username is wrong, enter again…. The system will allow three attempts before dismissing the dialog box and displaying the message 'Login failed´, please see the administrator'
- If for some reason the system cannot establish the communication with the server when trying to log into the system, then the message 'Cannot establish connection with the server, see the administrator will prompt,' then the dialog box will be dismissed.

4.2.2 Description of Use Case: Get next customer

The system retrieves the next customer in the queue to be served. These are some of the pre-conditions taken into account before the next customer button is pressed:

1. The counter clerk presses the ‘Get Next Button,’ the system updates the ‘Now Serving Customer No’ field in the Client GUI with the next number on the queue. The buttons are disabled in the GUI. The system displays this number on the LED display and flashes the number 10 times.
2. As the number flashes in the LED, the loudspeaker will announce the flashing number and the counter number. The announcement would be like “Customer number X goes to counter number Y please!” This announcement will be re-peaed three times.
3. The buttons become enabled once the flashing stops.

Alternatively if the LED becomes faulty, the system will notify the counter clerk using a dismissable dialog box.

4.2.3 Description of Use Case: Exit Client

The system shuts down the client GUI by counter clerk, but the pre-condition must be that the QCracker client must be connected to the server. To logout of the sys-tem the counter clerk shall press the exit button; alternatively, if the LED becomes faulty, the system will notify the counter clerk using a dismissable dialog box.

4.3 Actor Customer

Table 3 specifies the functional requirements primarily associated with a customer and tells what a customer is capable of doing as an external user of QCracker system.

Table 3. Customer Description

Actor	Definition	Required capabilities
Customer	The role played by the service provider customer	No skills required apart from using a finger to press the service button and take a ticket.

Table 3 demonstrates that a customer needs no skills in the QCracker system except pressing the button by a finger to a select service he or she wishes.

Figure 10 shows an actor customer and how she or he interacts with the system. The customer does one task, which is to select a service.

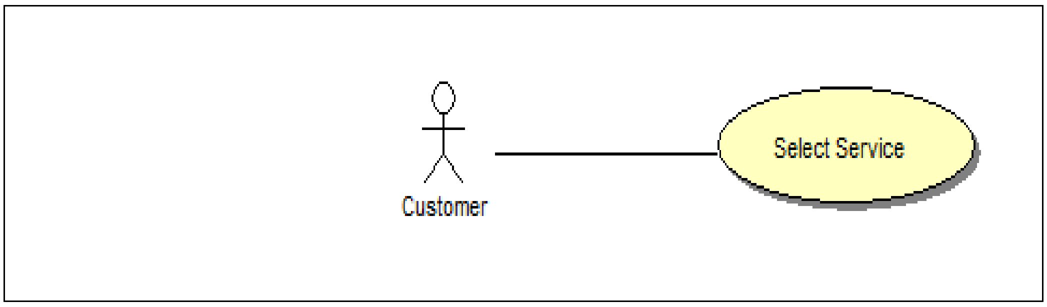


Figure 10. Customer Use case Diagram

Figure 10 above shows a description of how the customer is interacting with QCracker system.

Description of Use Case: Select Service

The system is up and running and the Customer GUI is displayed. The Customer presses the service button by a finger and the system issues a ticket via a POS printer. The ticket has a number on it representing the position in the queue.

The ticket will have service provider customizable messages printed above and below the number. The messages could be adverts customized by the service provider.

Figure 11 illustrates customer GUI, where several services are available for a customer to choose from. The customer can choose one service from available ones at a time and ready to be serviced.

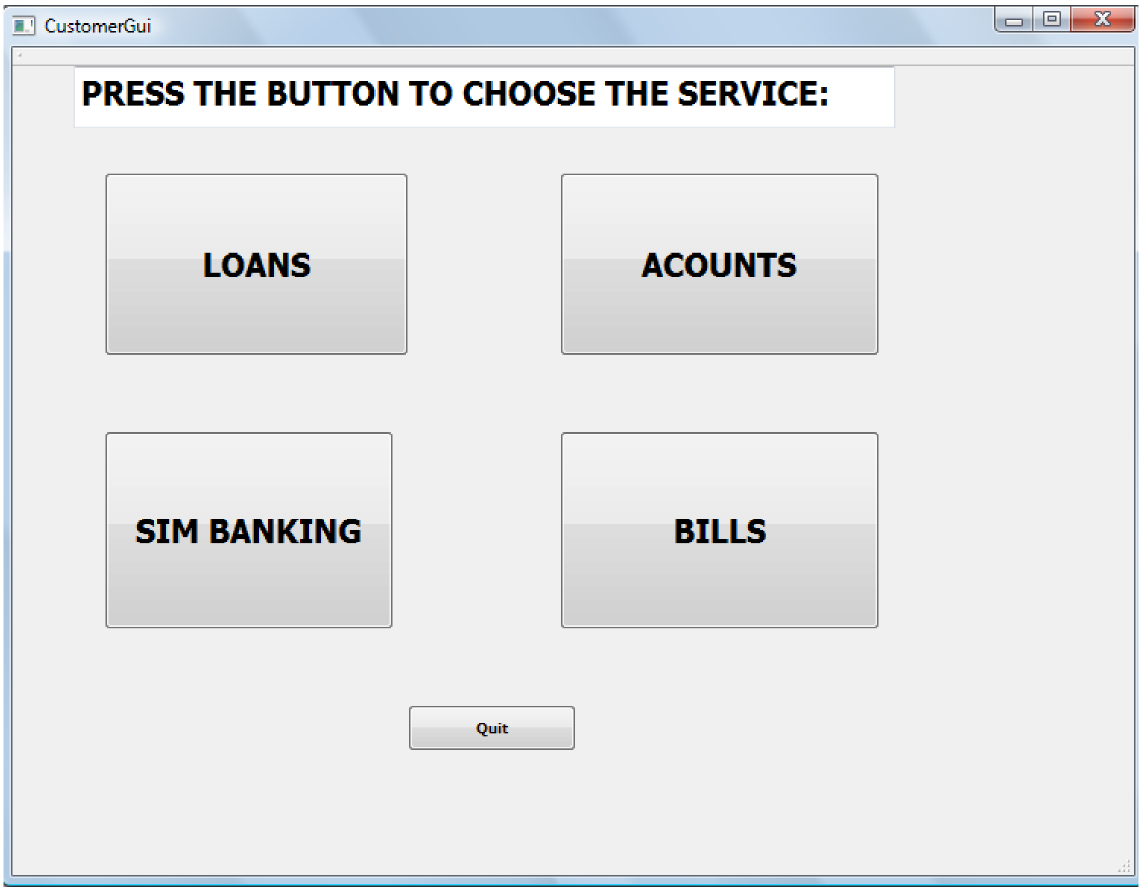


Figure 11. Customer GUI

Figure 11 illustrates a case where a customer chooses a service he or she desires and after choosing the service, the process goes to the printer for printing the ticket however if something goes faulty or if the ticket printer is faulty, the system will display critical error and display a message 'QCracker is out Of service'. The service buttons will become disabled. The QCracker clients will be notified the 'Get Next Button' on the client GUI will become disabled and the LED display will indicate that the counter is out of service.

However, it should be noted that Quit button is disabled all the time, and customer cannot do anything with it.

4.4 Actor IT Support

Table 4 specifies the functional requirements primarily associated with *IT Support* actor. This role is played by a technician who has known QCracker system in depth and has qualified for that role.

Table 4. IT support Description

Actor	Definition	Required capabilities
IT support	The role played by the service provider IT Support in the IT department or an employee of kifaru software solution Ltd.	Service Provider Local Area Network knowledge QCracker System knowledge

Table 4 shows how IT support interacts with a system and the special requirements or capabilities that the IT support should have.

Figure 12 shows the IT support as an actor and how it uses the QCracker system.

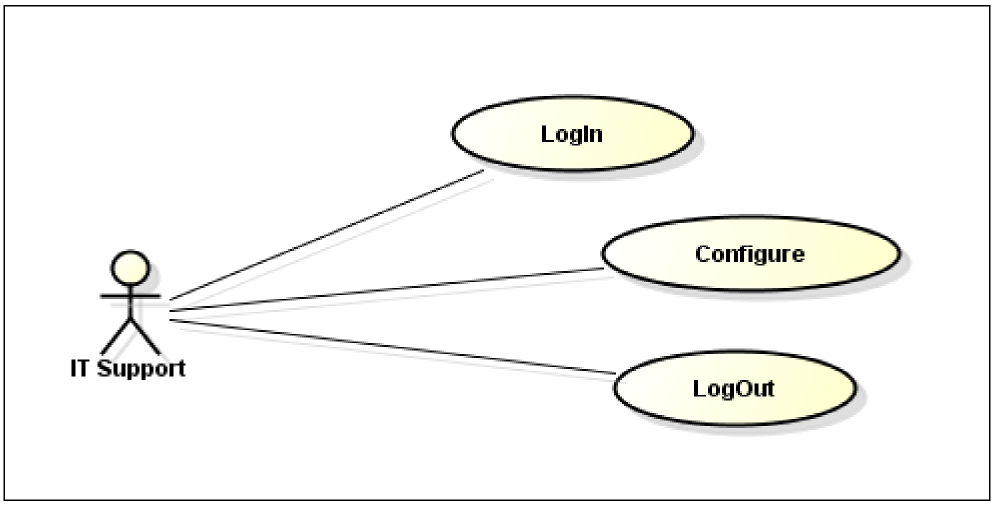


Figure 12. IT support use case diagram

Figure 12 shows the IT support and how he or she interacts with a QCracker system by log into the system, then configuring or installing system before logging out from the system.

4.4.1 Description of Use Case: Server Startup

The QCracker system starts up the server tests the POS printer and the loudspeaker. One major precondition is that no server is running. The printer is switched on and the loudspeaker is functional and the IT Support staff starts the server by double clicking its icon or selecting it from the Star->All programs. The system displays the Customer GUI.

Description of Use Case: Configuration

The system can be updated and being monitored by the IT support for configuration, and some installation as well at any time needed

4.4.2 Description of Use Case: LogOut

The system stops the server tests, the POS printer and the loudspeaker, if they are off. The authority of stopping the server is given by the Technician who is IT support trained and has knowledge of QCracker system. A major precondition with this is that no server is running. The Printer is switched off and the loudspeaker is in the wait functioning mode.

The IT Support stops the server by double- clicking its icon or selecting it from the Start->All programs and then shuts down the menu. The system displays nothing after stopping the server.

4.5 Actor Manager

Table 5 below specifies the functional requirements primarily associated with the *Manager* Actor.

Table 5. Manager Description

Actor	Definition	Required Capabilities
Manager	The role played by the service provider employee whose task is to supervise Counter Clerks.	Basic Computer (window) skills Use a QCracker Client software to log onto and navigate the QCracker client User Interface

Figure 13 shows how the manager is interacting with the QCracker system.

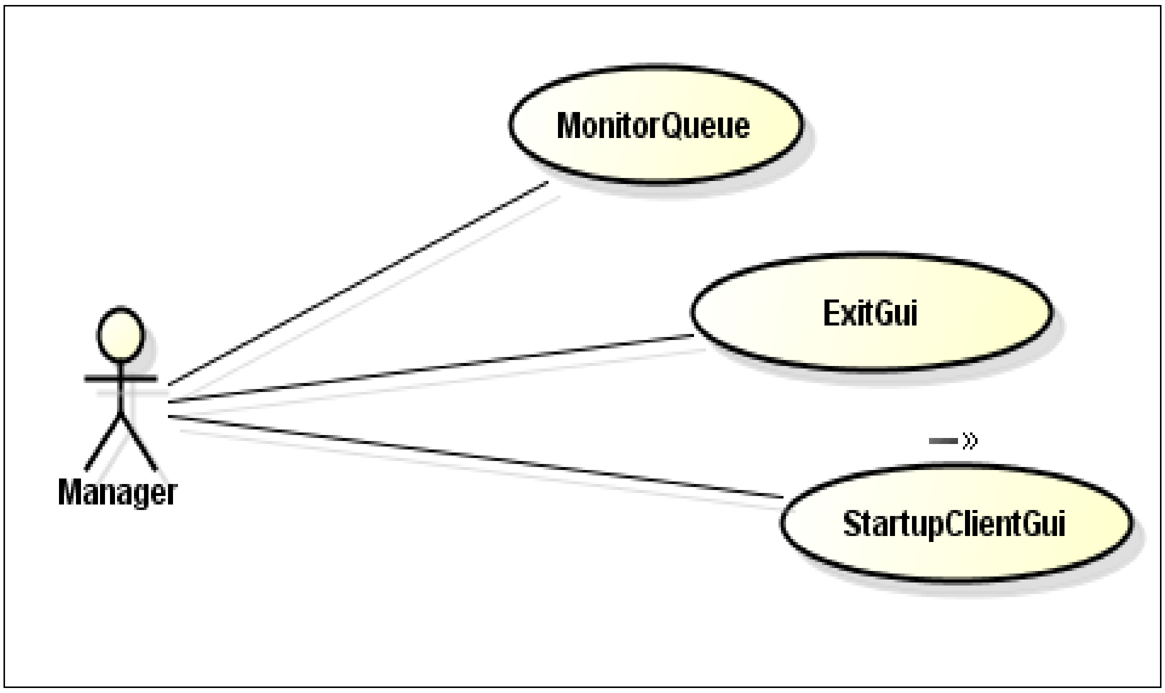


Figure 13. Manager Use case Diagram

Figure 13 illustrates that manager can operate several duties in the QCracker system including monitoring the Queue by the first starting up client GUI and at the end exiting client GUI. In addition of that the manager will have his own GUI which will enable him to do duties which no one else can do in the system.

4.5.1 Description of Use Case: StartupClientGui

The Manager double- clicks the monitoring tool icon on the desktop or selects the monitoring tool from Start->All Programs. The user interface starts up and displays all the queues created in the system. The system starts up the client GUI, while this is done the server must be up and running. The manager selects the queue to do the monitoring. The system displays the following data:

- The queue name
- The number of customers currently waiting in the queue
- The average waiting time in the queue
- The name of each counter clerk attending this queue, the counter number and the list of customers attended so far. The list will have the following details:
    - The time the customer was called to be served
    - The time taken to serve the customer.
    - The customer number currently being served.
    - The average time it takes to serve each customer

Description of Use Case: ExitGui

The system shuts down the monitoring tool GUI under the pre-condition that the monitoring tool is displayed.

5 Design

5.1 Qt Framework

Qt is a cross-platform application and UI framework. It includes a cross-platform class library, integrated development tools and a cross-platform IDE. Using Qt in QCracker was beneficial because developers were able to write applications once and deploy them across many desktops and embedded operating systems without rewriting the source code. [10]

QT was chosen for QCracker because of the following reasons:

- It is good for developing GUI application, so we used it to develop the client and customer GUI and other dialog boxes such as login dialog boxes.
- It provides good and easy support for multithreading and inter-process communication. When the server spawns threads to serve client it uses QT multithreading framework. Objects like QueueManager and CustomerGUI communicates using the signal or slot feature of the framework. The sender of the message emits the signal and the receiver just registers for that signal and receives with one line of code.
- We have plans to port our server to Linux. At the moment it is running under Windows. Because of the Qt nature of cross platform not so big effort needed when porting to Linux.
- We wanted to develop in C++ and QT is in C++.
- With the Qt Creator cross-platform IDE, Qt is fast to learn and easy to use, and its modular class library means that we can spend more time on innovation of QCracker, and less time on infrastructure coding getting the software to the market faster .
- Qt allows QCracker to integrate with the WebKit web rendering engine, which means that we can quickly incorporate content and services from the Web into our native application, and can use the web environment to deliver our services and functionality impressing our users in the process.[10]

Qt in QCracker provides a portable API for creating and synchronizing threads, and offers the option of building the Qt library with thread support in QCracker. With the emergence of multi-processor computers, multithreaded programming is rapidly gaining popularity. [11]

Qt provides thread support in QCracker in the form of platform-independent threading classes, a thread-safe way of posting events, and signal-slot connections across threads. This makes it easy to develop portable multithreaded Qt applications and take advantage of multiprocessor machines. Multithreaded programming was a useful paradigm in developing QCracker for performing time-consuming operations without freezing the user interface of an application. [12]

Thread support in Qt for QCracker includes the following:

- Thread-affinity in Qobject
- Per-thread event loops
- Post events to any thread
- Signals and slots across threads
- Thread-safe reference counting

5.2 Boost Standard Library

Boost speeds initial development, results in fewer bugs, reduces reinvention-of-the-wheel, and cuts long-term maintenance costs. Moreover Boost has now been accepted to be part of C++ compiler. Boost is also familiar sto developers. These were the main reasons why we chose Boost. In our code we are using Boost mainly in resource handling and control. Memory monitoring facilities, locking and unlocking of resources are all handled by the Boost library.

5.3 Server Design

The QCracker server is a multithreaded application designed to handle multiple client requests simultaneously. To achieve this functionality it utilises the services provided by the Qt framework. The Qt framework provides wrappers around the operating system calls. Figure 14 below depicts the high level architecture of the server.

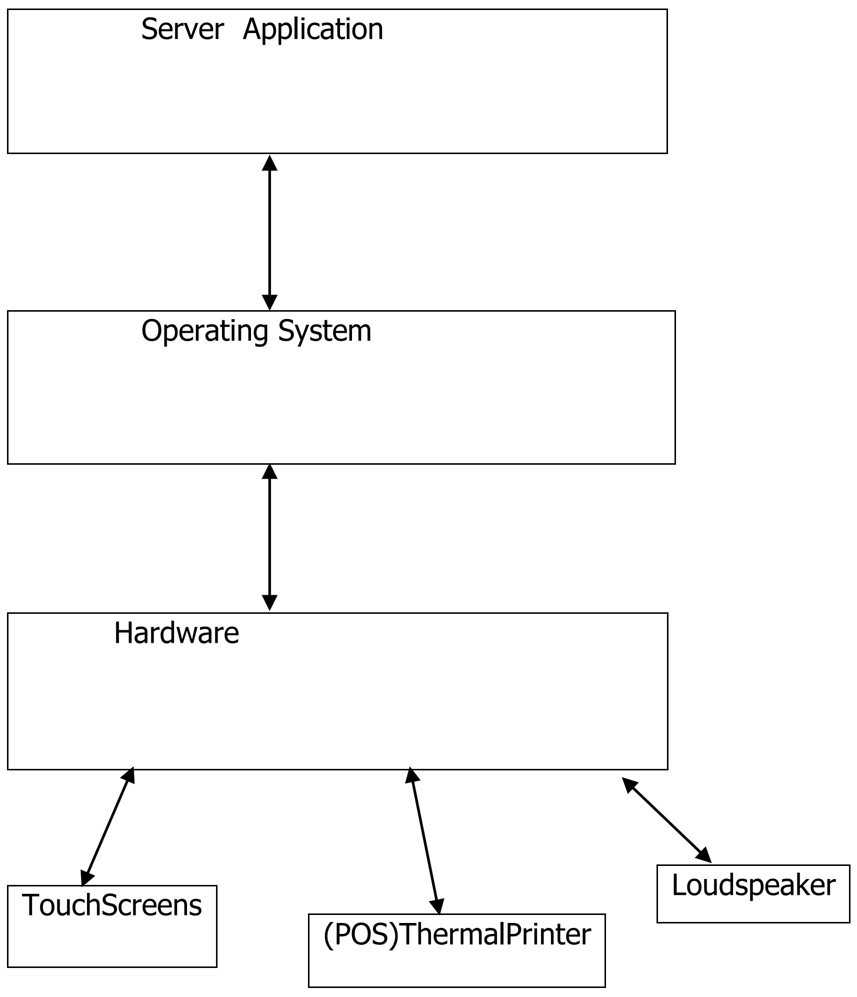


Figure 14. QCracker Server Architecture

As figure 14 shows, the server application utilizes the services provided by the Operating system. Currently the server runs on MS Windows OS via Qt framework to communicate with its hardware.

Figure 15 describes in detail how the objects of the server communicate with each other using a synchronous call.

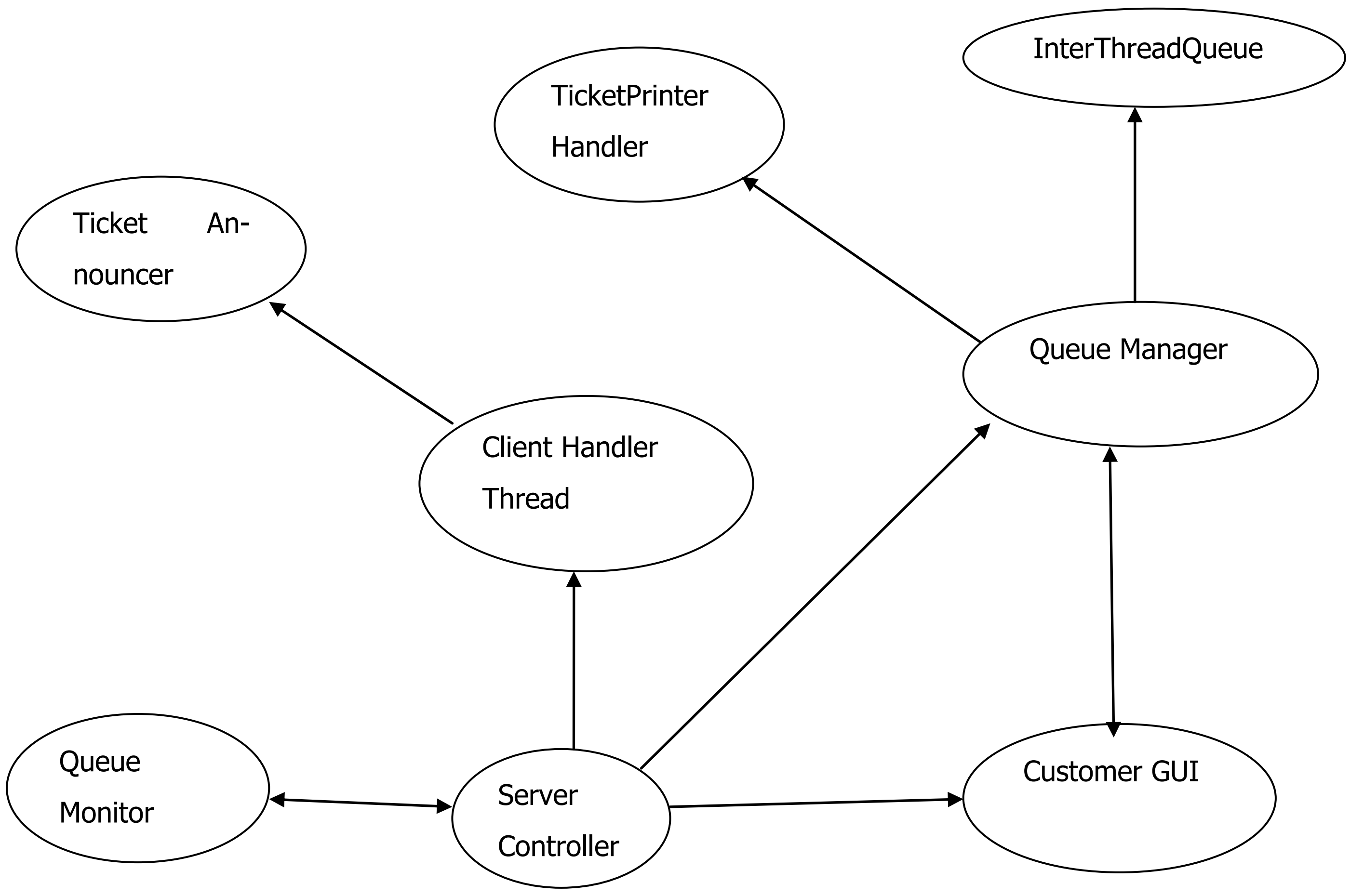


Figure 15. QCracker main objects structure

Figure 15 describes a detailed server design. At startup the following objects become created and start to sending and communicating with other objects through a synchronous call.

5.3.1 Server Controller

At startup the Server Controller registers its TCP port number. The port number will be used by clients to connect to the server. When the connection requests arrive from the clients, the Qt framework will deliver a message to the QCracker server with the socket descriptor of the client. The socket descriptor is the one which enables the server to communicate with the client.

Upon receiving the connection request from the client, the server using the QT framework creates a thread to serve the client requesting the connection. The server passes the socket descriptor received earlier from the framework as shown in figure 16. The thread makes a connection to the client and waits for requests. The same thread will serve the client as long as the client is connected

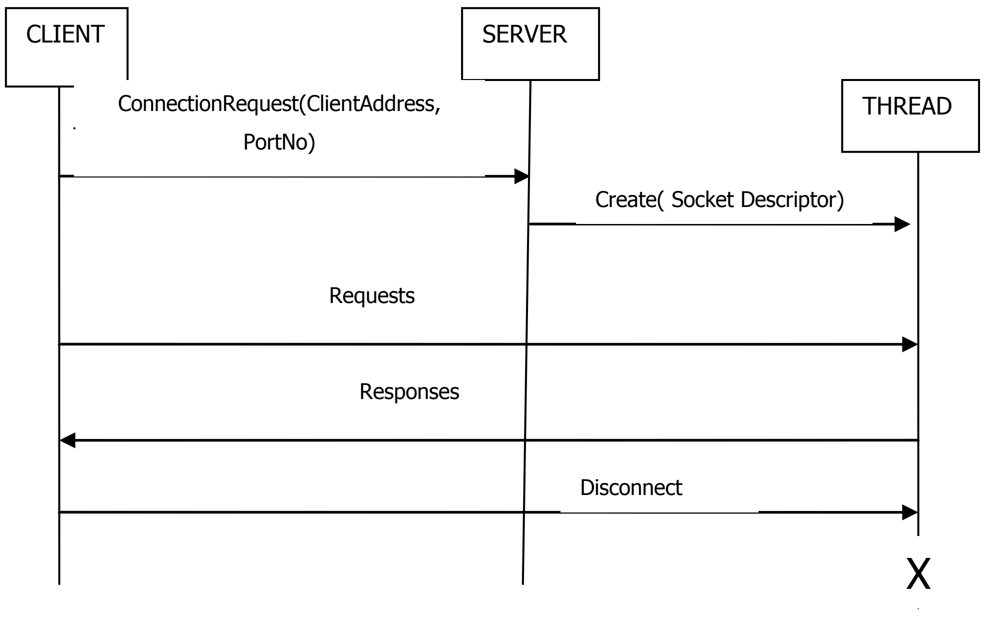

CLIENT
SERVER
THREAD
ConnectionRequest(ClientAddress, PortNo)
Create(Socket Descriptor)
Requests
Responses
Disconnect
X

Figure 16. Server Controller Communication

Figure 16 shows how the server controller communicates with the client and how the thread is made in order for client to be served for the entire period when the client is on.

5.3.2 CustomerGUI

The customer GUI at startup displays the customer Graphic User Interface and responds to the button pressed. It maps each button with its associated service. The buttons must be available at any time to the customer, so the queue manager just emits a message to any object that has registered for that message and returns very quickly to respond to more presses. Using this method the GUI becomes responsive at any time.

5.3.3 QueueManager

The Queue Manager is responsible for all the queues. When the object gets created it reads from the configuration file the details of the queues to create and creates them. It does listen for button press events from the Customer GUI. When it receives the events it determines the queue type and creates a ticket object and sends it to the relevant queue to be queued and the same ticket gets sent to the TicketPrinter object.

5.3.4 InterThreadQueue

InterThreadQueue objects get created by the Queue Manager. The underlying implementation of the actual Queue is the C++ STL (Standard Template Library) queue. The queue is based on First in First out (FIFO). The InterThread Queue has been designed to accommodate multi-threads hence the name. As many threads try to read and write from-to the queue the mechanism called mutexe (Mutually Exclusive) for controlling or serialising the operations has been introduced in the multiThreadQueue. The mutexes used are from the Boost Library. When a thread needs to read off the queue it must acquire a lock. Only one thread can acquire a lock at any given time. Other threads trying to acquire the lock while locked will be blocked until it is released or operation fails and they return and try again later (See the diagram below). This mechanism ensures the integrity of data being read or written. The implementation has been deliberately remove

Figure 17 illustrates in more detail the sequence diagram: how locking and unlocking of queues is performed in QCracker system.

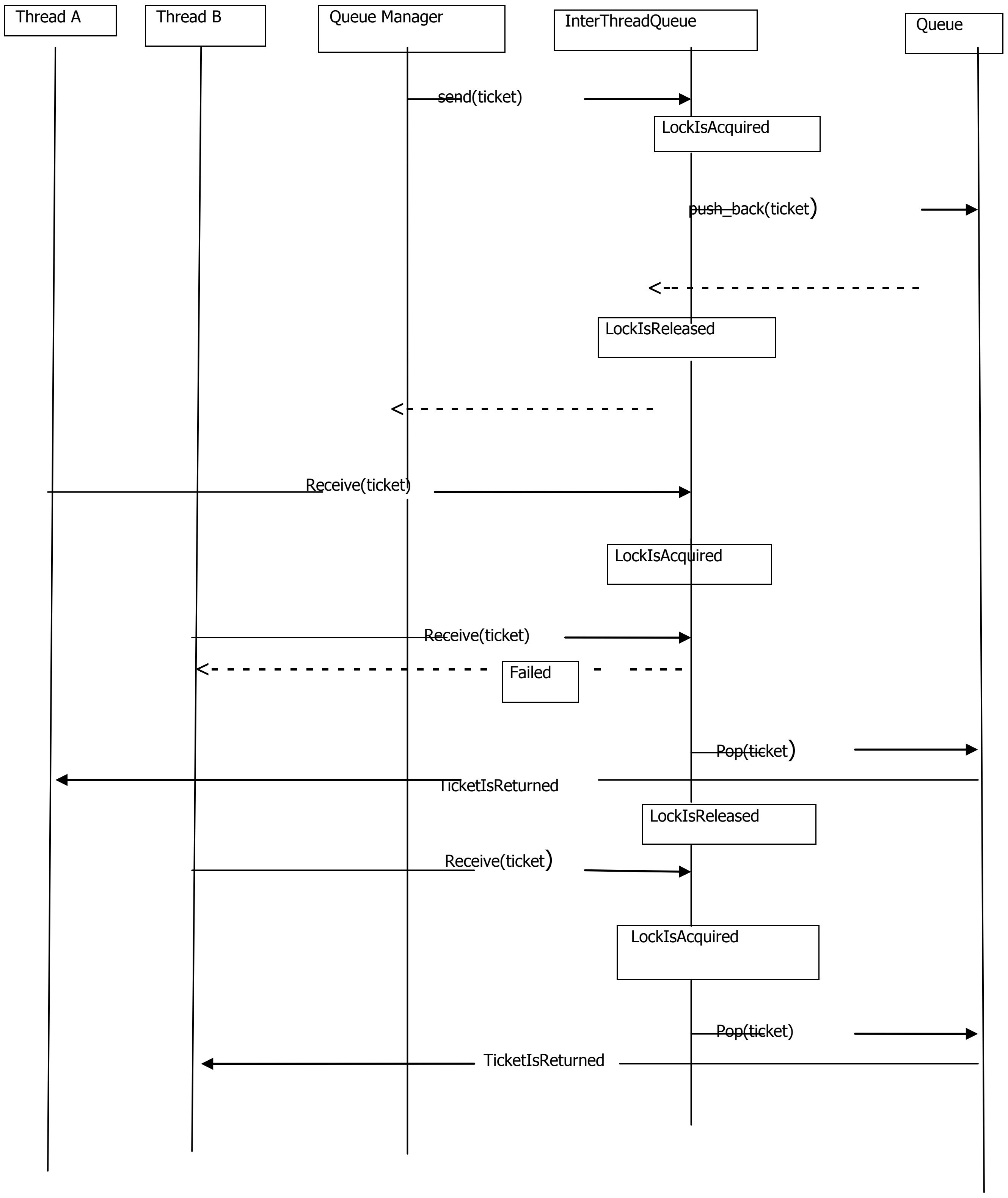


Figure 17. Locking and unlocking of queues

Figure 17 is a top level sequence diagram expressing the concept of how locking and unlocking of queues is taking place in the QCracker system.

5.3.5 Client Handler Thread

The client handler thread is created by the server upon receipt of the incoming connection from the server. They are the ones which do the actual work of serving the client. There is one thread per each connected client. The requests from the client become responded by the thread. The threads keep client transaction statistics in a file and they are responsible for sending emails to the manager at the end of the business day.

When a client presses the 'Get Next Customer', the thread will retrieve the queue ticket from the queue. It will extract the number from the ticket object and send it to the client. When the customer is issued with the ticket, the threads will get this information and notify the client about the customer waiting in the queue.

5.3.6 Ticket Printer

The ticket printer object is created by the Queue Manager. This object is responsible for interfacing with the QPrinter, the Object in the Qt framework which talks directly with the printer driver.
Ticket Number Announcer
The ticket Number announcer is responsible for handling the conversion from text to speech using the text to the speech library.

5.4 Server Class Diagram

Below in figure 18, a static model of the server classes is presented. The model defines the classes of objects in the server, some of the attributes of the classes and the relationships between classes.

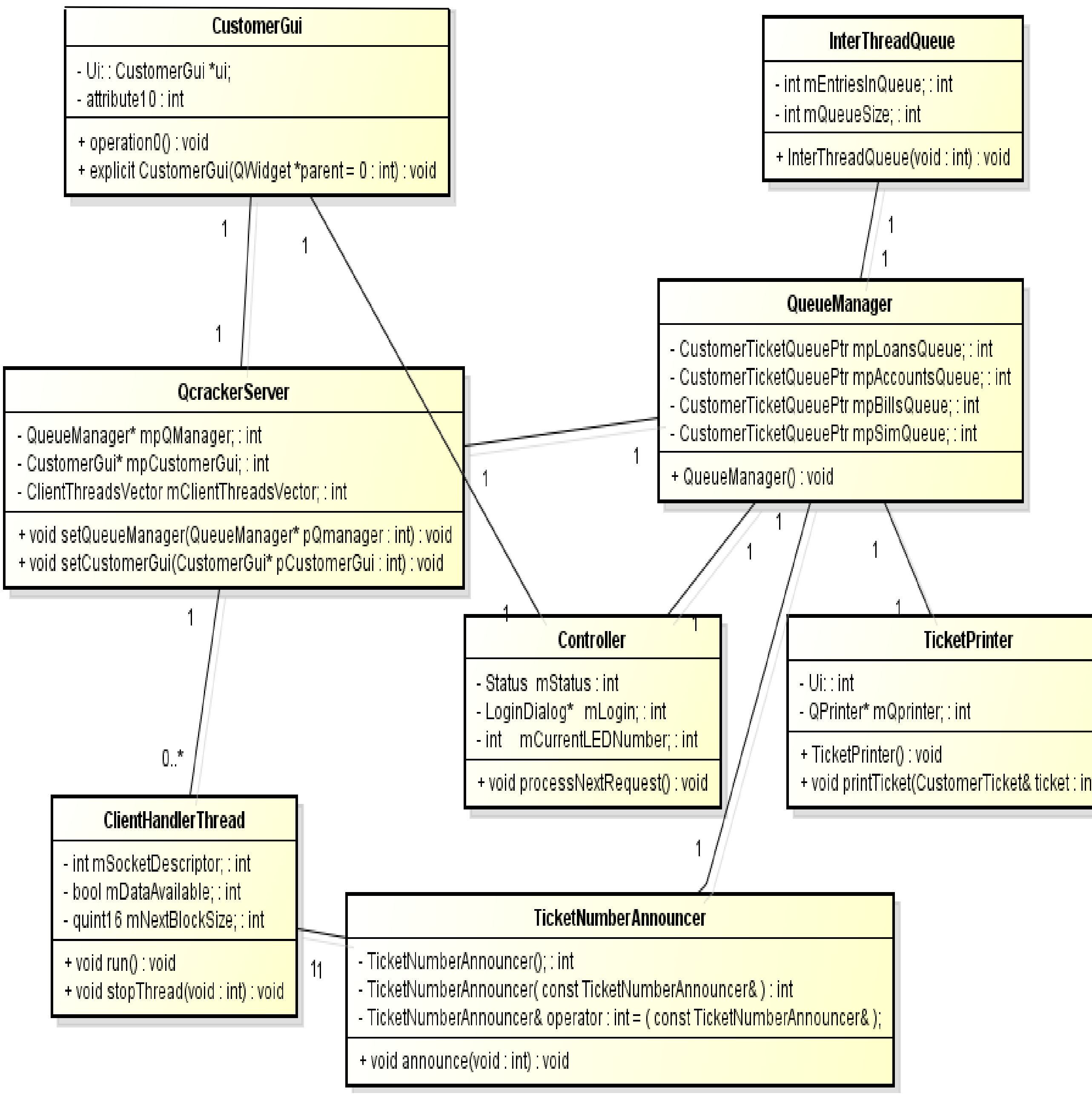

CustomerGui
- Ui: : CustomerGui *ui;
- attribute10 : int
+ operation0() : void
+ explicit CustomerGui(QWidget *parent = 0 : int) : void
InterThreadQueue
- int mEntriesInQueue; : int
- int mQueueSize; : int
+ InterThreadQueue(void : int) : void
QueueManager
- CustomerTicketQueuePtr mpLoansQueue; : int
- CustomerTicketQueuePtr mpAccountsQueue; : int
- CustomerTicketQueuePtr mpBillsQueue; : int
- CustomerTicketQueuePtr mpSimQueue; : int
+ QueueManager() : void
QcrackerServer
- QueueManager* mpQManager; : int
- CustomerGui* mpCustomerGui; : int
- ClientThreadsVector mClientThreadsVector; : int
+ void setQueueManager(QueueManager* pQmanager : int) : void
+ void setCustomerGui(CustomerGui* pCustomerGui : int) : void
Controller
- Status mStatus : int
- LoginDialog* mLogin; : int
- int mCurrentLEDNumber; : int
+ void processNextRequest() : void
TicketPrinter
- Ui: : int
- QPrinter* mQprinter; : int
+ TicketPrinter() : void
+ void printTicket(CustomerTicket& ticket : in
ClientHandlerThread
- int mSocketDescriptor; : int
- bool mDataAvailable; : int
- quint16 mNextBlockSize; : int
+ void run() : void
+ void stopThread(void : int) : void
TicketNumberAnnouncer
- TicketNumberAnnouncer(); : int
- TicketNumberAnnouncer(const TicketNumberAnnouncer&) : int
- TicketNumberAnnouncer& operator : int = (const TicketNumberAnnouncer&);
+ void announce(void : int) : void
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
0..*
1
11

Figure 18. QCracker server class diagram

From the diagram in figure 18 above it can be seen that the QCracker server communicates with Customer GUI, Queue Manager and Client Handler thread and these other classes are the ones responsible for sending data to other subclasses.

Server Sequence Diagrams

Below are some of the sequence diagrams representing the dynamic model of some of the interaction between objects when the server starts up and when a customer selects a service. It should be noted that these are just some of the chosen sequence diagrams on the QCracker system chosen for this specific project.

Server Startup Sequence

Figure 19 illustrates the Server start up sequence diagram and table 6 explains exactly what is going on in the figure 19.These are just explanation of the Server startup sequence diagram from figure 17.

Table 6 below elaborates on all the steps taken into account in the sequence diagram in figure 19.

Table 6. Server startup sequence explanation

Message	Description
1:	The main function creates a customer GUI object, and stores the address of the GUI object.
2:	The main function creates the Queue Manager. The Queue Manager is responsible for creating and managing all customer queues.
3:	The main function passes the address of CustomerGUI object to the QueueManager object. The two objects do communicate with each other directly.
4:	The main function creates QCrackerServer object. This object will be responsible for handling client connections and create threads to process client requests.
5:	The main function passes the address of the customerGUI object to the QCrackerServer.

The following figure 19 explains the server startup in a sequence diagram.

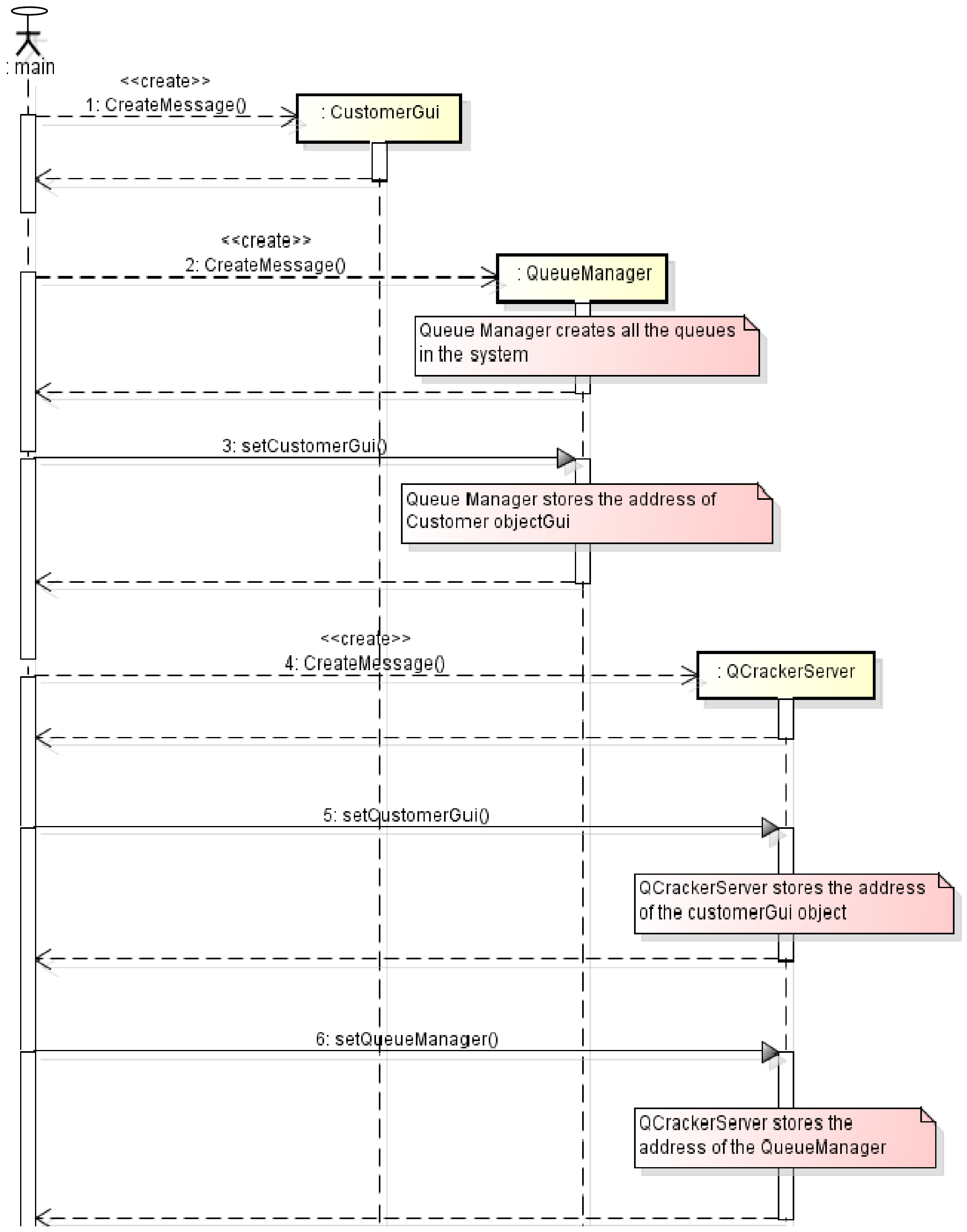

: main
<<create>>
1: CreateMessage()
: CustomerGui
<<create>>
2: CreateMessage()
: QueueManager
Queue Manager creates all the queues in the system
3: setCustomerGui()
Queue Manager stores the address of Customer objectGui
<<create>>
4: CreateMessage()
: QCrackerServer
5: setCustomerGui()
QCrackerServer stores the address of the customerGui object
6: setQueueManager()
QCrackerServer stores the address of the QueueManager

Figure 19. QCracker server startup sequence diagram

Figure 19 illustrates a sequence diagram for the server in the QCracker system from the beginning when the server starts to run and make communication by taking requests from the client.

The following figure 20 below illustrates the process where a customer starts to select a service.

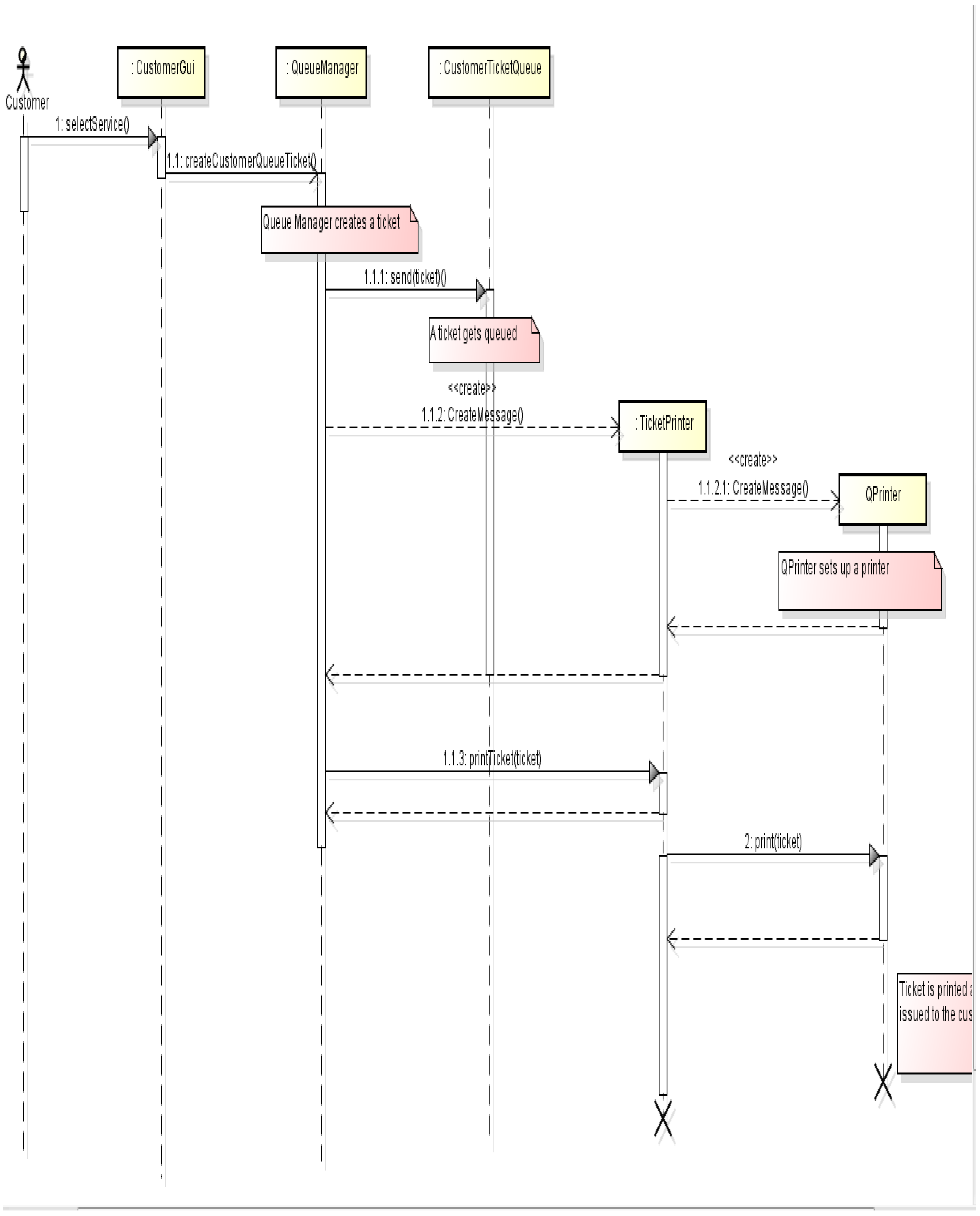


Figure 20. QCracker Customer Sequence Diagram

Figure 20 above elaborates on what is going on when a customer selects the desired service until he is served

Customer selects a service

Table 7 clarifies step by step what is going on when the customer selects service from the touch screen on wards.

Table 7 is explanation of the Customer select service sequence diagram from figure 20 above telling all necessary steps drawn from the figure above

Table 7. Customer select service explanation

Message	Description
1	Customer selects a desired service by pressing a button on the touch screen. The customerGUI object receives a command.
1.1	The customer GUI sends a message to a QueueManager to process the ticket service request. The QueueManager creates a ticket.
1.1.1	The QueueManager sends a ticket to the relevant queue object.
1.1.2	The QueueManager creates a TicketPrinter object.
1.1.2.1	The TicketPrinter object creates a Qt Framework object Qprinter which interface with the thermal printer. Upon creation it sets up the printer
1.1.3	Once the printer is ready, the QueueManager will commands a TicketPrinter to print a ticket.
2	TicketPrinter relays the message to the Qprinter object which commands the printer to print the ticket and the ticket is printed and issued to the customer.

Table 7 above clarifies the point when the customer is exposed with a customer GUI with different options for choosing the desired services. This is a table from the sequence diagram telling the important steps taken from figure 20 point of view.

5.5 Client Design

Below is a high- level design of QCracker Client and Server system diagram including a top high- level class diagram with its attributes, the classes showing the relationships of each other and how information is distributed through a client system

Figure 21 describes how the client is architectured from inside the system.

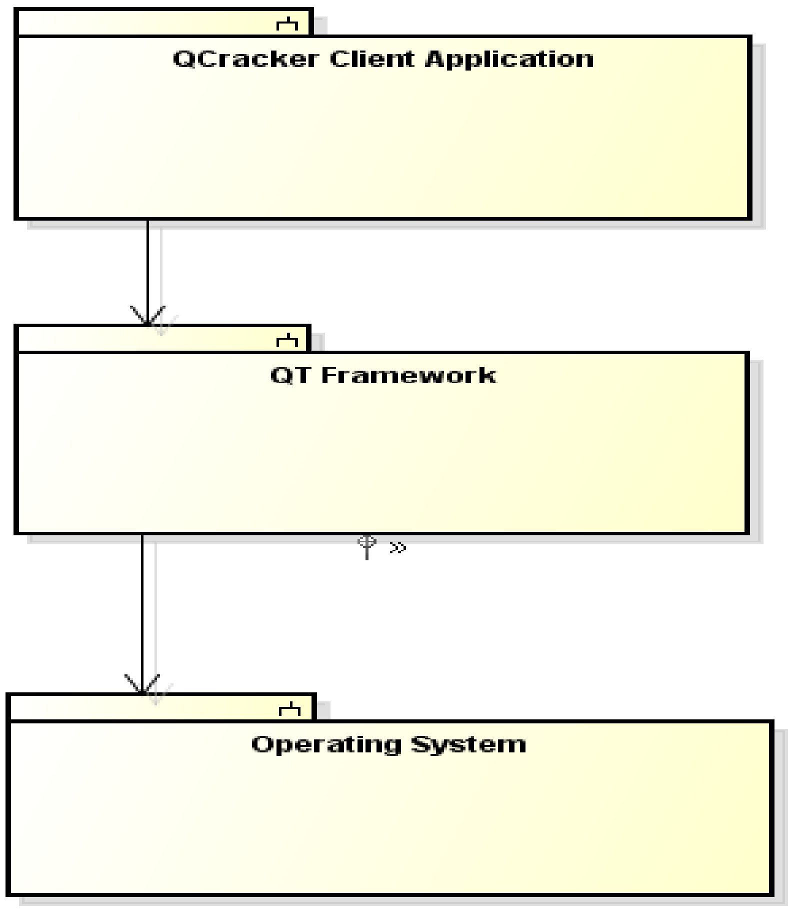


Figure 21. QCracker Client Architecture

Figure 21 shows high-level design of client. Basically this shows how data flows when the clerk presses the next button. The data first is flowing from QCracker client Application, then to Qt framework which is using Windows operating system which TCP/IP is within. Windows OS is using TCP/IP protocol stack to send messages to the server. This applies to the client as well, so it is the same concept used to architect the client

Figure 22 illustrates a simplified design of the client showing how the client is composed of and designed in the QCracker System.

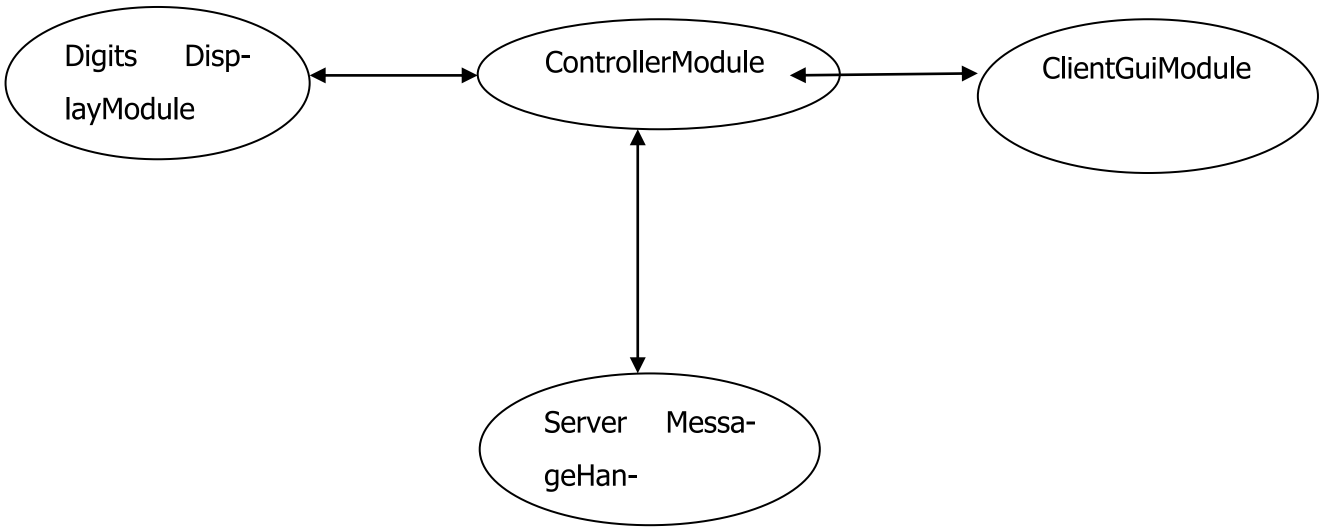


Figure 22. Client Design

Figure 22 shows the message handler handles the messages to and from the server. It receives the messages, decodes them and passes them onto the controller, which forwards them on to the relevant object. Accordingly ClientGui is the one which communicates with the clerk and Digit Display is the one responsible for displaying numbers on the screen. This object is responsible for interfacing with the LED Digit display, it will receive the number to display and will forwards it to the driver to be displayed. It will also process messages coming from the digit number display.

The controller module is the coordinator of all other objects in the client. It is the center of everything logical in design of the client. It communicates with other objects by knowing their references (pointers), it assigns them tasks and after the tasks are done they will be back to the client controller using the same address they have sent the task from. This feature is supported by the Qt framework. In other words it is known as a synchronous call or a call back function where the object communicates using references or addresses. Figure 23 illustrates a static model of the client classes. The model defines the classes of the objects in the client, some of the attributes of the classes and the relationships between the classes.

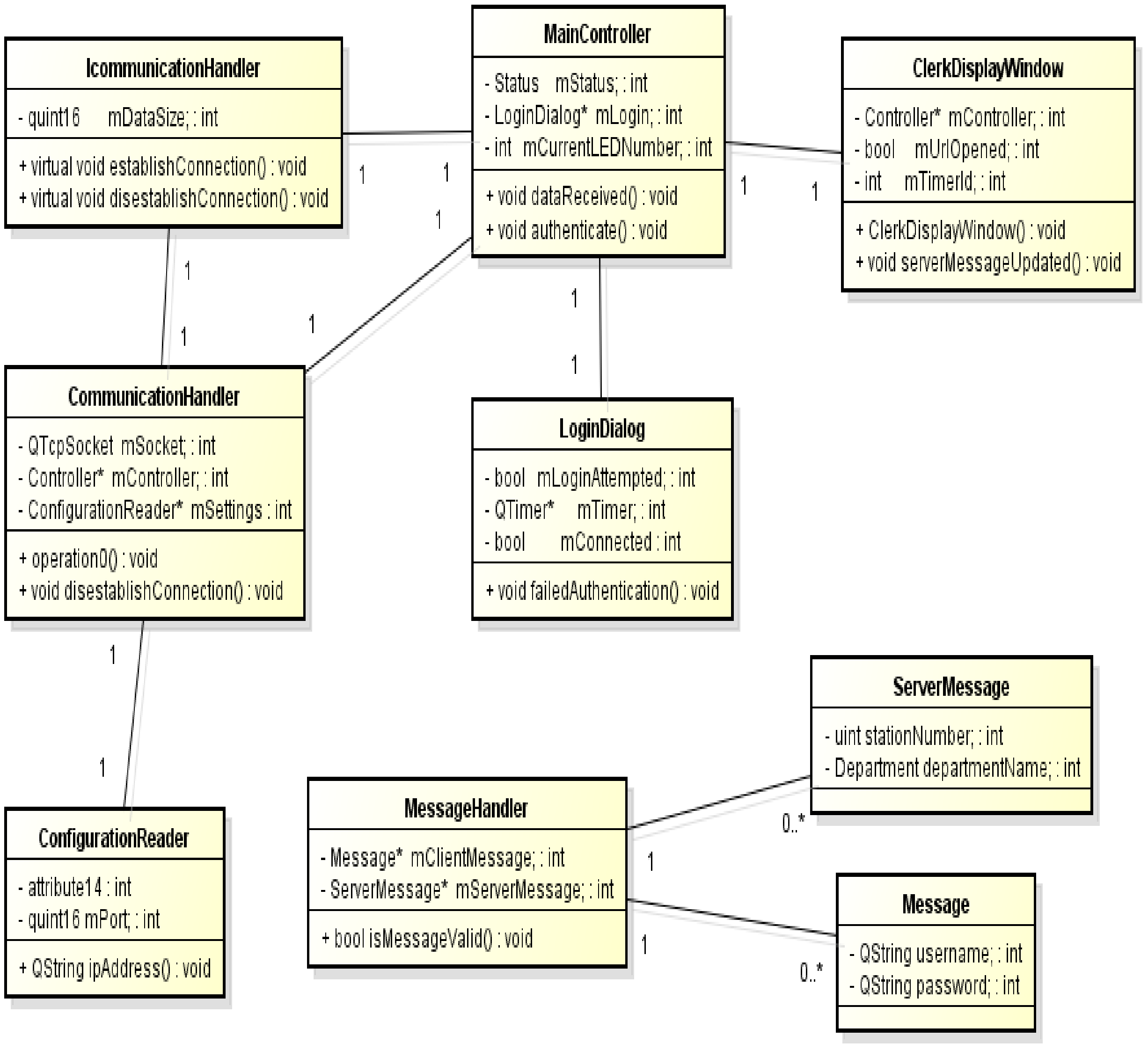

IcommunicationHandler
- quint16 mDataSize; : int
+ virtual void establishConnection() : void
+ virtual void disestablishConnection() : void
MainController
- Status mStatus; : int
- LoginDialog* mLogin; : int
- int mCurrentLEDNumber; : int
+ void dataReceived() : void
+ void authenticate() : void
ClerkDisplayWindow
- Controller* mController; : int
- bool mUrlOpened; : int
- int mTimerId; : int
+ ClerkDisplayWindow() : void
+ void serverMessageUpdated() : void
CommunicationHandler
- QTcpSocket mSocket; : int
- Controller* mController; : int
- ConfigurationReader* mSettings : int
+ operation0() : void
+ void disestablishConnection() : void
LoginDialog
- bool mLoginAttempted; : int
- QTimer* mTimer; : int
- bool mConnected : int
+ void failedAuthentication() : void
ServerMessage
- uint stationNumber; : int
- Department departmentName; : int
MessageHandler
- Message* mClientMessage; : int
- ServerMessage* mServerMessage; : int
+ bool isMessageValid() : void
ConfigurationReader
- attribute14 : int
- quint16 mPort; : int
+ QString ipAddress() : void
Message
- QString username; : int
- QString password; : int
1
0..*

Figure 23. Qcraker Client class diagram

Figure 23 shows the QCracker high- level client class diagram showing the relationships between classes, and each class showing its attributes and some of the methods in each class used to design a client.

6 Discussion

QCracker was developed using a mixture of water fall and iterative methodology. It was started off with the idea of building a queuing system. The idea came about when I visited East Africa (Tanzania in particular) and saw long queues in a noisy and chaotic environment and thought that there must be a better and modern way of handling such long queues. As software engineering student I quickly thought about developing a computerized queuing system that would be modern and still affordable to common service providers in East Africa and Africa as a whole.

I started off by speaking to stakeholders in businesses where customers normally queue up for services and they were interested by the idea of having a computerized system. They expressed their wishes of what the system should be like and what it should do. As I discussed with them I gathered the requirements. I even spoke to ordinary customers and they told us how it would be to have such a system as it would solve many of the problems caused by unordered, chaotic and unfair queues.

The main concern of the service providers was the pricing. They urged us to make the system as cheap as possible but still modern. We decided to take up the challenge and develop the system.

I captured the requirements and specified them using use case diagrams as shown in section 4.1. I did the analysis of tools and framework and started to design the system. I found that some of the requirements needed adjustment as a result of design. So I went back and adjusted them accordingly. I then started implementation followed by verification. The verification included unit testing, integration and then the system which is still ongoing.

Below are the steps followed and still following to build the system:

$$\begin{tikzcd}
\text{Requirement gathering} \arrow[d] \\
\text{Design} \arrow[d] \\
\text{Implementation} \arrow[d] \\
\text{Verification} \arrow[d] \\
\text{Maintenance}
\end{tikzcd}$$

The result of the project shows that there are several parts of the system which are running and functioning as planned. These include the QCracker client where the requests are sent from client to server over existing LAN through TCP/IP. The server is up and running and communicates well with the client. It communicates accordingly with client. The client software uses a queue number from the server to the fake LED, and a 3 digit number should be displayed on the digit display. This feature is temporary until we get the real LED, once I install the system to our customer. The client can log into the server accordingly, the client GUI represents all the necessary information to the user and the user (clerk) can call the next customer.

The results of a modern queuing management system apply to many seemingly unrelated situations, from serving customers at service counters to managing traffic congestion in a cosmopolitan city, and from designing switching equipment for telecommunications to understanding Internet behavior.

Recalling the purpose of the project, the aim was to create a system which would take advantage of the LAN and which would be cheaper and affordable to our main customer as mentioned. Another purpose was to design from scratch and develop a modern queuing system that would take advantage of modern available software development tools and networking infrastructure unlike traditional queuing management systems.

The main difference between the old queuing system and QCracker (modern queuing management system) is that a modern queuing management system attempts to do more through the use of a computerized system, and it helps the management by producing statistical reports on information such as arrival rates and patterns, waiting and service times, and default and reneging cases. Based on these statistics report from the server, the optimal use of resources can be achieved, helping the trade-off between service quality and service cost. The latest Internet-enabled systems allow remote system monitoring, report generation and system configuration across an Internet link. In that case QCracker is totally based on software and rely on a server to give all the services involved. All the client and other services will be tracked ad monitored, all information will be saved on the server, at the end of the day retrieved at any point unlike the old queuing system.

Generally the Qcracker system works accordingly and it is definitely an improvement over the old version of a queuing system. There are several points that might be debatable about the implementation of the improvements in QCracker:

1. The scope of this project was limited in time and, therefore, the improvements Over the Qcracker leave room for further development, such as improving the high tech devices into the system.

2. Due to the departure of several employees from the company who started the project the company had to suffer financially. Regarding this is small company and this is the first product for the company, it takes time to teach and introduce the concept to the new employers who did not start the project, which as a matter of fact delay the launching time of our product and incur more expenses in our company.
3. Delaying order of hardware devices from foreign countries to support software has resulted in employees not to fully concentrating on this project full time. As a result, it increases the expenses in the company.

Qcracker is still under development. As discussed earlier, it is supposed to be launched sometime at the end of this year in the eastern part of Africa. The product is working according to our customer requirements and the results show that it might help to improve the efficiency of the employees and make it easy for the employer to keep track of his/her employees on how many customer per given day can be served by a clerk and to calculate the average time taken per clerk to serve a single customer, according to which department he/she is assigned to.

Another result shows that QCracker would generate customer feedback on how clerks perform their duties. This will ensure that a customer is well served with fairness and with a positive attitude, and it would also give customers power to assess the customer services due to the fact that customers are the ones who makes the system work.

7 Conclusion

The goal of this project was to design from scratch and develop a modern queuing system that would take advantage of modern available software development tools and networking infrastructure unlike traditional queuing management systems.
The outcome of the project was the creation of a modern queuing management system.

The first objective of QCracker was to achieve better quality of service to customers. In its most basic form QCracker will issue a queue number ticket to an arriving customer and later call the number when the time slot for that number becomes available, eliminating the need to stand in line while waiting. In this way QCracker will provide comfort as well as fairness to customers. It will also allow customers to maintain their position in a queue while seated comfortably or engaged in constructive activities. The service provider can provide entertainment for waiting customers and can take advantage by displaying advertisements on the information screen. When the time slot becomes available, QCracker via loudspeaker will announce the queue number to be served and give instructions on the location of the counter and at the same time assist the customer to locate the counter. QCracker displays the customer number on the LED display fitted at counter's station.

As the staff at the counter serves the customer, QCracker collects statistics that show the performance of the staff. For example, the total number of customers served daily by the staff for example how long it took to serve one customer, how long the customer waiting time was. This statistics will be available to the management for analysis and definitely will assist them to make important business decisions. A statistics email is sent to the manager at the end of each business day if required. QCracker provides means for monitoring the status of the queues on the real time basis. The manager can remotely log into QCracker and view the status and progress of queues and based on that information he or she can take action.

The QCracker project has got developers with a high knowledge of programming in Object Oriented Design and C++ programming skills and an understanding of the Qt framework. The reason for choosing this topic was that I am currently working for Kifaru software solution limited and my main task is taking part of the development sector as well as the testing sector which is still in progress. The project itself motivated me to the degree that I wanted to be part of the team. Being part of a small and startup company would give me a tremendous opportunity to learn, grow professionally and meet challenges that would make me a complete engineer.

The results proved that it is possible to create a simplified model of a computerized intelligent queueing system where a server communicates with a client over an existing local area network, with some basic dynamic functionality. However, the results also demonstrated the challenge of establishing a proper working system, which may vary its response functionality while interacting with the system. The results proved that it is possible as well to create a well working product from scratch according to the customer requirements.

This project was limited both in time and resources, thus concentrating on a simple mechanism of developing features which are more beneficial to the customer.
Another limitation on QCracker was the employees, especially in the software testing department. This was due to the fact that it is start -up company growing, so financially the company is not stable enough, which delay to launch the product. Though testing is ongoing it could have been done earlier.

The constraints ranged from equipment lack to the availability of applying real-life tests to the system. QCracker is open to future development and it has established itself to maintain and update the system accordingly.

Generally QCracker achieved its goal due to the fact that the system is working according to the customer requirements even though it is a bit late to launch it. Overall the customer is satisfied with the prototype produced so far and I am open to another challenge of developing another software product.

References

1 Bose S. An Introduction to queuing system [online]. Kluwer/Plenum; December 2002. URL: http://home.iitk.ac.in/~skb/qbook/qbook.html. Accessed 5 March 2012.

2 Zukerman M. Introduction to queuing theory. University of California: Moshe publisher; 2000.

3 Blanc J P C. Queuing Models [online]. Tilburg: Tilburg University; 14 January 2011. URL: http://lyrawww.uvt.nl/~blanc/qm-blanc.pdf. Accessed 10 March 2012

4 Bell D. UML Bascis Class Diagram [online]. IBM Corporation; 15 September 2004. URL:http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell. Accessed 10 March 2012.

5 Ambler S. Agile Modeling [online]. DDJ State of the IT Union,USA: Ambysoft Inc; 7 July 2009.
URL:http://www.agilemodeling.com/artifacts/classDiagram.htm#Classeshttp://www.wavetec.com/QueueManagement/Components.aspx. Accessed 24 March 2012.

6 Cooper R. Queueing Theory. [online]. Florida Atlantic University: Encyclopedia of computer science; 12 March 2000.
URL: http://www.cse.fau.edu/~bob/publications/encyclopedia.pdf. Accessed 24 March 2012.

7 GMS. The Queue Management Concept. [online]. Selangor Darul Ehsan: General Microsystems Sdn Bhd; 8 June 2011.
URL: http://www.gms.com.my/GMS_Home/concept.htm. Accessed 26 April 2012

8 Kenneth W. Notes On Queueing Theory. [online]. Department of Computer Science North Carolina: Microssoft professional; 20 August 2012
URL: http://williams.comp.ncat.edu/comp755/qnotes.pdf. Accessed 30 March 2012.

9 BOUML. Reverse Engineering. [online]. Object Management Inc; 27 May 2012.
URL: http://www.bouml.fr/doc/index_cpproundtrip.html. Accessed 4 April 2012

10 Digia Qt. Cross Platform Application and UI Framework. [online]. Helsinki Finland: Qt Digia; 10 October 2012.
URL: http://qt.nokia.com/. Accessed 22 june 2012.

11 Blachette J. Qt 4's Multithreading Enhancements. [online]. Qt Digia; 16 October 2011.

URL: http://doc.trolltech.com/qq/qq14-threading.html. Accessed 22 June 2012.

12 Digia. Qt Developer Network. [online]. Thread support Department: Qt Digia; 27 0ctober 2012.
URL: http://qt-project.org/doc/qt-4.8/threads.html. Accessed 24 june 2012.

13 Lewis B. A Guide to Multithreaded programming. New Jersey, USA: Prentice Hall PTR; 1995.

14 Mayhew L, Smith D. Using Queuing Theory to Analyse Completion Times in Accident and Emergency. London: Cas Business School; 2006.

15 Beveridge J, Wiener R. Multithreading Application in win32. Boston, Massachusettes,;Addison-Wesley Professional; 1996

16 Kleinrock L. Queueing Systems. Volume 1: Theory. University of California: Wiley; 1975

17 Gross D, Shortle F, Thompson M, Harris M. Fundamentals of Queueing Theory, Set (Wiley Series in Probability and Statistics). University of California: Wiley-Interscience; 2009.

18 Narayan Bhat U. An Introduction to Queueing Theory: Modeling and Analysis in Applications (Statistics for Industry and Technology). Basel, Switzerland: Birkhäuser; 2008.
19 Xu W. Application of Proxels to queuing Simulation with Attributed Jobs. Magdeburg,German: Otto-von-Guericke University; 1 March 2008.
URL: http://wwwisg.cs.ovgu.de/sim/files/theses/xu.pdf. Accessed 27 September 2012.