April 27, 2007

Introduction to Software Engineering

Group 13

http://software-ece.rutgers.edu/~group713/

REPORT THREE

Restaurant Point-of-Sale System

Moustafa H.Abdelbaky

Albert Lalaj

Waleed N.Mina

Boris F.Petrovsky

Bryan A.Rabin

Ling J.Yang

Table of Contents

2. Contributions

Moustafa

- Functional Requirements Specification:

  1. Revised the Actors and their goals as some changes were made, where some use case will not be implemented for the moment.

  2. Worked on the full description for eight of our fully described Use Cases and modified a few of the previously described Use Cases.

  3. Introduced four new Use Cases to our Functional Requirement Specifications

- Interaction Diagrams:
  1. Redesigned six of our Interactions Diagrams and made all the changes needed to accommodate the updated Use Cases.
  2. Described Interaction Diagrams, and explained the Design Principles as well as the Design Patterns for all the Use Cases of the Interaction Diagrams.
- Class Diagram and Interface Specifications:
  Described all the Designed Patters used to achieve the hierarchy of the Class Layout and the Interface Specifications.

Albert

- Described Summary of Changes and listed all the changes under each activity.
- History of Work and Current Status of Implementation:
  1. Described the History of Work throughout the whole lifetime of the project.
  2. Compared side by side the initial goals with what was actually achieved.
- Conclusions and Future Work:
  1. Describe the learning experience through out the project and important conclusions that were drawn from all the Team Members at the end of this course.
  2. Expressed the Future Plans of Group 13 in possible new products as well as describing the additional features for POS project that are planned to be implemented in the future.

Waleed

- Client's Statement of Requirements:
- Domain Analysis:
  1. Redesigned the Domain Model to reflect the changes made to the Use Cases and the goals of each Actor.
  2. Updated the attributes, associations and concepts were deleted to accommodate the project functionality.
- System Architecture and System Design:
  Made the necessary changes to System Requirements and Network Protocol to support the replacement of the PDA with BalackBerry phones and revised the diagram for Mapping Subsystems to Hardware.

Boris

- Functional Requirements Specification:
  1. Fully described four of the Use Cases that we decided to elaborate on this report and revised some of the Use Case described on the previous reports.
  2. Revised the description of Actors and Goals.
- Reviewed the Report I and Report II and summarized the uncompleted parts and the errors that were targeted to be revised on Report III.

Bryan

- Functional Requirements Specification:
  1. Described how the Refresh Class interacted with Use Cases and helped implementing it into the Class Diagram.
  2. Revised the names of the GUI classes.
- Design Analysis:
  1. Helped revising the Domain Model, implementing the different changes.
  2. Updated Operation Contracts for the Design Analysis

Bryan

- Interaction Diagrams:
  1. Redesigned six of our Interactions Diagrams and made all the changes needed to accommodate the updated Use Cases.
- System Architecture and System Design:
  1. Redesigned the Subsystem Analysis Diagram to help Identifying Subsystems.
- User Interface Design and Implementation:
  1. Introduced the Chef GUI interface and updated the Examples of the User Interfaces showing the progress on implementation of our User Interfaces.
  2. Revised the description of the User Interfaces and the way they interact with each other.

Ling

- Functional Requirements Specification:
  1. Revised the Use Case Diagram
  2. Designed the additional System Sequence Diagrams for new descriptions of the Use Cases.
- Nonfunctional Requirements: Revised the Nonfunctional Requirements to clarify some misunderstanding, and described the Implementation, Interface, Operation, Packaging and Legal requirement that weren't mention in the previous reports.
- Interaction Diagrams:
  Redraw all the Interaction Diagrams after all the updates were made to the Use Cases.
- Class Diagram and Interface Specifications:
  Revised the Class Diagram and specified the Data Types used as well as the Object Constraint Language.

All the team members contributed equally towards:
Glossary and **References.**

3. Summary of Changes

Client's Statement of Requirements

The statement of requirements was revised to remove all the unclear statements and words to make it more readable and easy to understand to the client. Host Access gained a new administrative feature by assigning severs to specific table. System network diagram was updated. Some of the System Actors roles were removed to approach a precise description of the project.

Functional Requirements Specification

Actors and their Goals were also changed based on the changes that were done to some of the used cases. Use Cases -#1 and -#2 (clock in, clock out) were completely removed and will not be implemented in the final demo. Four new Use Cases were introduced to satisfy more goals for our Actors, UC-#29 (Host assigns a default server to a table), UC-#30 (Chef or bartender marks order ready), UC-#31 (Cancel Item), UC-#32 (Send messages for any reason). Also more use cases were fully described. A total of 12 cases were fully described compared with only four on the previous reports. UC-#3, UC-#5, UC-#7, UC-#14, UC-#15, UC-#18, UC-#20, UC-#21, UC-#22, UC-#24, UC-#25, UC-#26.

Use Cases that will not be implemented but would be possible to implement in the future UC-#1, UC-#2, UC-#8, UC-#12, UC-#17, UC-#27, UC-#28 were clearly labeled so are easy to be noticed. Some use cases UC-#22, UC-#23, UC-#24 were modified from their previous description. System Sequence Diagrams were updated to incorporate the new and modified use cases that will be completed for the final demo. They were ordered in the correct way and more System Sequence Diagrams were drawn to match with the new Use Cases that were added.

Nonfunctional Requirements

The nonfunctional requirements were revised and to clarify some misunderstanding and five more requirements were added.

Domain Analysis

Domain model diagram was fully revised and updated to maintain a better understanding of the project. Changes including attributes of some old concepts were made and some new associations were added. Also some concepts were deleted to accommodate the project functionality. System Operational Contracts were updated to reflect the changes made in the Domain Model Diagram. The new

Interaction Diagrams

New interaction diagrams were designed and drawn as well as all the previous ones were completely changed not to allow any errors and perfectly match the rest of the report. A great effort was invested into the Interaction Diagrams as they are one of the most important parts of this report.

Class Diagram and Interface Specification

Class Diagram was revised to show all the recent changes that were made to our programming and the Refresh Class was added to the Diagram. Data Types Operations Diagram was also updated as well as the Object Constrain Language Contracts. Design Principles was a new part added to this section.

System Architecture and System Design

System Architecture was partially revised. The main areas of change included Identifying Subsystems, where the Diagram of Subsystems Analysis was changed to match all Use Cases. Mapping Subsystems to hardware also had its Diagram changed to reflect the use of BlackBerrry phones instead of PDA's. Our Database Schema was also changed to show the addition of the table Messages and also instead of having one table for Orders we actually use Inactive Orders and Active Orders where at the moment an order is active it temporarily stays on the Active Orders table and then as soon as it get closed it gets moved to the Inactive Orders table.

User Interface Design and Implementation

Under User Interface Design the part of Examples of GUI Interfaces was updated and the GUI for the Chef was created. Usages Scenarios were also revised and more discussion was about the User Interfaces was presented.

History of Work, Conclusions and Future Work

History of Work throughout the whole lifetime of the project was described in details and the changes made along the way were compared side by side with the initial plans. Conclusions that Group 13 arrived after working on this project were described in details and also the learning experience through out the project including important conclusions that were drawn after using the principles that we learned in this Software Engineering Course. At the end future plans of Group 13 were listed describing possible new products as well as describing the additional features for POS project that are planned to be implemented in the future.

4. Client's Statement of Requirements:

Since all restaurants operate on a very low scope, every dollar saved will count toward the bottom line. A POS system can give you a new level of control over your restaurant operations, helping you increase efficiency, boost profits, and fine-tune inventory management. Switching from a traditional cash register and paper-based orders to a restaurant computer system can be hard, but the return on investment can really make it worth your time and effort. A POS is a must if you wish to operate your restaurants at the maximum effectiveness possible.

Basic POS Features

- Operate more efficiently
- Reduce employee errors
- Increase table rotation speeds

Touch Screen Monitor for Order Entry

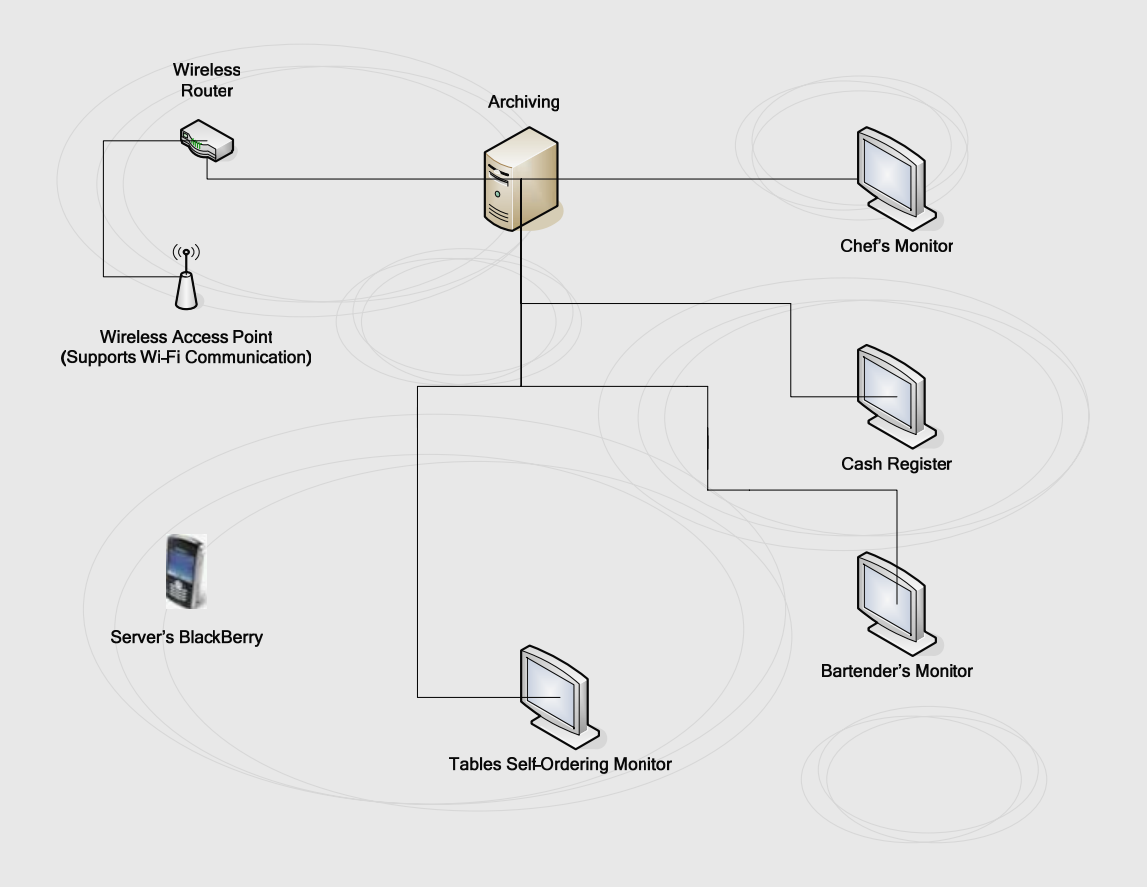
- Designed especially for touch screen & light pen operation

BlackBerry For Order Entry

- Server uses a BlackBerry with displayed interface menu to take an order

BlackBerry Pearl 8100

System Network



System Actors' Roles

Administrator Tools (Manager): Statistical business analysis: see profit break-down by hour or by menu-item, either graphically or textually.

Ability to check for example:

- Menu item popularity
- Average turnaround time (How long patrons spend in the restaurant.)
- Average preparation time (Time from when the order is placed to when it is ready)

Administrative power over employee profiles: the ability to create and modify profiles, track employee activities, and authorize restricted server activities:

- Profiles are data structures that contain an employee's personal information, such as his name, login ID, and time sheet
- If the employee is a server, his profile also contains information about the tables for which he is responsible. From his profile, the individual tabs for those tables can be accessed
- The concise and effective ability to manage all aspects of restaurant operations
- Maintain employee schedule based on previous sales volume
- Update floor plan (Modify how tables/section, how many sections)
- Access records (Statistics)

Host's Access: Assigns table to customer. Fairly distributes customers among tables that are assigned to specific server. Also assigns server to table, in the case when the customer desires to sit in specific place where all servers are busy.

Server's Access: Track multiple tables under one profile:
A manager assigns a group of tables to a specific server, and the server accesses a tab for each table in his or her profile. Now Server can electronically update kitchen orders: i.e., items entered onto a tab are sent to the kitchen staff through the network.

Bartender Access: Maintain tabs at the bar with ability to start, add to, and close orders. Maintain records of cash sales for end of shift report. Also provide alerts based on tab information on excessive alcohol intake to avoid over serving. In addition, Bartender report when an order is ready.

Chef's Access: The availability of computer monitors in the kitchen that are networked through the archiving to the ones on the restaurant floor to provide immediate access to orders. In addition to that orders placed by server using a BlackBerry are displayed to the kitchen staff through a queue, i.e., on a first-in, first-out basis. The priority scheduling will be included to allow more efficient expediting of orders.

Customer Access: Table top computers can be placed at certain booths to allow direct ordering and payment by the customer. Frequent customers could have their information stored by the system to further increase ease and efficiency.

Flexible Menu Item Pricing Support: Allow you to specify different menu item prices for each of the order types.

Employee Staff Cash Register: Allow each employee be their own cashier. It is easy to read shift end report. All money collected by the employee are accounted for.

Cash Register Functionalities: Full control over your money handling operations. Track your money trail with accountability.

The Old Fashion System

Solving the problem for this “old fashion” system works but yields a large amount of tab receipts, wastes a lot of time and is simply out-of-date. In old fashion systems, waiters have to carry pads around to take orders, always have a working pen and be sure to keep each bill organized and “synchronized” with the proper table. Another issue is record maintenance. In the old system, when everything is done by paper, the management is responsible to keep all information saved and organized, which is no easy task. Everyday tabs are collected, data needs to be organized and employees need to get paid. This requires a great deal of time and attention from the managers.

5. Glossary of Terms:

Clients: business owner or other parties buying our software

Customers: guests going to the restaurant and using the system

Chef: cook or chef preparing the food in the kitchen

Server: waiter or waitress serving customers of the restaurant

Administrator: Restaurant Manager(s)

Host: person in charge of seating customers as they come to the restaurant

POS: An Abbreviation for Point of Sale

Concepts (Domain Model): A Concept is something that has a distinct, separate existence, though it need not be a material existence. In particular, abstractions and legal fictions are usually regarded as concepts. In general, there is also no presumption that an Concept is animate. Entities are used in system developments as models that display communications and internal processing of say documents compared to order processing.

Attributes (Domain Model): An Attribute is a named property of a class defining a range of the values an object can contain.

Association (Domain Model): Is a relationship between two or more classes denoting the possible links between instances of the classes. An association has a name and can have multiplicity and role information attached to each of its ends.

Archiving:

Archiving is the process of getting rid of old data (or data that is not valid anymore) from the main production databases.

Text field:
An area in which it is necessary for the user to enter text instead of simply clicking or touching the screen.

Wireless Access Point: Access points used in home or small business networks are generally small, dedicated hardware devices featuring a built-in network adapter, antenna, and radio transmitter. Access points support Wi-Fi wireless communication standards.

A domain model: can be thought as a conceptual model of a system. A domain model will tell about the various entities involved and their relationships. The domain model is created to understand the key concept of the system and to familiarize with the vocabulary of the system.

User Layer Displays the content sent from the server, storing and retrieving any cookie state in the process

Presentation Layer
Processes and delivers display content to browser (e.g. HTML), along with any cookie state in the process

Logic Layer Specifies the business objects and rules of the application, and handles interfacing between the presented information and the stored data.

Database Layer Stores the nonvolatile state of the application, and exposes ways to access this state

JDBC is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.

Threads are a way for a program to fork (or split) itself into two or more simultaneously (or pseudo-simultaneously) running tasks. Threads and processes differ from one operating system to another, but in general, the way that a thread is created and shares its resources is different from the way a process does.

API – Application Programming Interface - a source code interface that a computer system or program library provides in order to support requests for services to be made of it by a computer program

Blackberry – Specific brand of Smartphone developed by Research in Motion (RIM). Includes its own API for application development.

JDE – Java Development Environment – a software development kit aimed specifically at Java development

Midlet – A program for embedded devices, more specifically the Java ME virtual machine. Generally, these are games and applications that run on a cell phone.

Servlet – An object that receives a request (ServletRequest) and generates a response (ServletResponse) based on the request.

Swing UI– a very popular Java graphics library used for GUI development

WYSIWYG – acronym for What You See Is What You Get

Response time - The amount of time elapsed between the dispatch (time when task is ready to execute) to the time when it finishes its job (one dispatch)

Structured Query Language (SQL**)** - The most popular computer language used to create, retrieve, update and delete data from relational database management systems. The language has evolved beyond its original purpose, and now supports object-relational database management systems. SQL has been standardized by both ANSI and ISO.

6. Functional Requirements Specification:

Stakeholders

Generally, the system under development (SuD) has five key stakeholders: Clients, systems analysts, systems architects and developers, software testing and quality assurance engineers, project Managers. In our system we, Group 13, will serve as systems analysts, systems architects and developers. Also, we will be handling the software testing and project management. Hence, our main concern will be the clients that will interact directly with our system.

Clients in our system are divided into four groups:

- Clients who will be buying our system i.e. restaurant owners
- Users who will be interacting directly with the system and that includes all the actors that will be using our system i.e. Host, Bartender, Chef, Server and Customers (we are using the term Customers to define the guests who come to the restaurant to eat)
- Managers who have the ability to modify any part of the system or how other users interact with the system, they also use the system to generate reports and analyze stats i.e. restaurant Manager
- Sponsors who will benefit directly or indirectly from our system i.e. Advertisers (by posting ads on Customer terminals located at tables)

Actors and Goals

Initiating Actors:

Customer:

- They place a new order or add to an existing order using the computer on the table; the order goes straight to the kitchen and/or the bar (UC-#25: AddItemAtTable)

- They also can request a Server; in case they don't want to use the computer on the table or if they need help with the system or want to leave (UC-#26: CallServer)

Chef:

- They log in and out to retrieve their specific interface (UC-#3:LogIn, UC-#4:LogOut)
- They also clock in and out in order to keep track of their hours **(Will not be implemented in Demo 2)** (UC-#1: ClockIn, UC-#2:ClockOut)
- They notify Server when an item or an order is ready and they remove it from the queue (UC-#22:MarkItemReady, UC-#30:MarkOrderReady)
- They can cancel an item from an order (UC-#31:CancelItem)
- They can send a message to Server (UC-#32:MessageServer)

Server:

- They clock in and out **(Will not be implemented in Demo 2)** (UC #1:ClockIn, UC-#2:ClockOut)
- They log in and out to retrieve their specific interface (UC-#3: LogIn, UC-#4: LogOut)
- They add/remove items to/from tab and place orders on behalf of Customer if Customers don't want to use the computer at the table (UC-#15: AddItem, UC-#16: RemoveItem, UC-#18:PlaceOrder)
- They also close orders, collect payment and notify Busboy to clean the table and finally mark table as ready (UC-#20: CloseTab, UC-#21: MarkTableReady)
- They adjust the price of an item on the tab to reflect a discount coupon **(Will not be implemented)** (UC-#17:AdjustPrice)
- They view tab for a particular table (UC-#19:ViewTab)

Host:

- They clock in and out **(Will not be implemented in Demo 2)** (UC-#1: ClockIn, UC-#2: ClockOut)
- They log in and out to retrieve their specific interface (UC-#3: LogIn, UC-#4: LogOut)
- They assign default Servers to tables (UC-#29:AssignDefaultServer)
- They assign Customers to tables and have the option of assigning a specific Server to that table for the duration of the Customer stay (UC-#14: SeatCustomer)

Bartender:

- They log in and out to retrieve their specific interface (UC-#3:LogIn, UC-#4:LogOut)
- They clock in and out **(Will not be implemented in Demo 2)** (UC-#1: ClockIn, UC-#2: ClockOut)
- They notify Server when drinks for restaurant Customers are ready for pick up (UC-#30: MarkOrderReady)
- They place orders received from bar Customers (UC-#23:AddItemAtBar)
- They also close orders or update open tabs and collect payments from bar Customers (UC-#24: CloseTabAtBar)
- They can cancel an item from an order (UC-#31:CancelItem)
- They can send a message to Server (UC-#32:MessageServer)

Manager:

- They log in and out to retrieve their specific interface (UC-#3: LogIn, UC-#4:LogOut)
- They manage users and that include adding new users removing current users or updating current user info (UC-#5: AddEmployee, UC-#6: RemoveEmployee, UC #7: UpdateEmployee)
- They update menu items by adding new items to the menu or removing menu items and update prices, recipe UC-#9: AddToMenu, UC-#10:DeleteFromMenu, UC-#11:UpdateItemInMenu)
- They also access various records and statistics for the restaurant operations (UC-#13: ViewStats)
- They can send a message to Server (UC-#32:MessageServer)
- They update floor plan **(Will not be implemented in Demo 2)** (UC-#8: UpdateFloorPlan)
- They schedule specials **(Will not be implemented in Demo 2)** (UC-#12:ScheduleSpecial)

Participating Actors:

Chef: Make orders based on order queue

Server: Bring food/drinks to the table when food/drinks are ready they also respond to Customers calls when Customers call them, they also receive messages from Chef/Bartender/Manager, and finally they respond to Host assignment for the tables to serve.

Bartender: Make drinks based on bar Customers orders or queue

Use Cases

Some of the uses cases have been elaborated in more details. These use cases are highlighted in light blue color differently formatted to be easily detected as they are the most important use cases of this project.

UC-#1: ClockIn **(Will not be implemented)**	Employee clocks in by entering the desired clock-in time. System first requests Employee to enter authentication code, then it requests the desired clock-in time, and when received, records both desired clock-in time and the actual clock-in time. **Should be considered for future implementation to increase flexibility of clocking in by Employee regardless of having to be logged into/out of one of the Employee interface.**
UC-#2: ClockOut **(Will not be implemented)**	Employee clocks out by entering the desired clock-out time **Should be considered for future implementation to increase flexibility of clocking out by Employee regardless of having to be logged into/out of one of the Employee interface.**

UC-#3:	LogIn
Related Requirements	
Goal In Context	Employee attempts to log into the system and open Employee specific interface
Preconditions	Connection to the database exists
Successful End Condition	Employee successfully logged into the system and is provided with Employee specific interface
Failed End Condition	Employee is denied access to the system and is provided with log-in interface
Primary Actors	Employee (Host, Bartender, Server, Chef, Manager)
Secondary Actors	
Trigger	Employee enters Employee password into the password field of the log-in screen at the terminal
Main Flow	Step Action 1 Employee enters password and attempts to log-in. 2 System receives password from the log-in screen and retrieves Employee information with given password. Then it opens appropriate Employee interface in a new window and closes log-in screen.
Extensions	Step Action 2.1 System does not find an Employee with password matching the provided password. System notifies Employee of failure to log in and resets log-in screen.

UC-#4: LogOut	Employee logs out of the system.

UC-#5:	AddEmployee
Related Requirements	
Goal In Context	Manager attempts to add new Employee to the Employee database
Preconditions	Manager is logged in and Manager interface is opened on the terminal Connection to the database exists
Successful End Condition	Manager successfully adds new Employee into the Employee database
Failed End Condition	Manager fails to add new Employee to the Employee database
Primary Actors	Manager
Secondary Actors	
Trigger	Manager requests to add new Employee from the Manager interface

Main Flow

Step	Action
1	Manager requests to add new-Employee from the Manager interface.
2	System receives request from the Manager and provides Manager with a new-Employee form.
3	Manager fills out the new-Employee form with new Employee's name, desired password, SSN, Wage and new Employee job title. Manager then submits the form.
4	System checks the validity of the information entered and that no other existing Employee has matching password. Then system adds new Employee to the Employee database, and closes the new-Employee form.

Extensions

Step	Action
4.1	System finds invalid information or another Employee in the database with the same password. System then provides Manager with failure by highlighting those fields of the new-Employee form which need to be changed along with description of why information needs to be changed.
4.2	Employee changes highlighted fields and submits the new-Employee form.

UC-#6:RemoveEmployee	Manager deletes former Employee profile. System deletes it from the database.

UC-#7:	UpdateEmployee
Related Requirements	
Goal In Context	Manager attempts to update existing Employee information in the Employee database
Preconditions	Manager is logged in and Manager interface is opened on the terminal Connection to the database exists
Successful End Condition	Manager successfully updates existing Employee information in the Employee database
Failed End Condition	Manager does not update existing Employee information in the Employee database
Primary Actors	Manager
Secondary Actors	
Trigger	Manager requests to update existing Employee information from the Manager interface

Main Flow

Step	Action
1	Manager requests to update existing Employee information from the Manager interface.
2	System receives request from the Manager and provides Manager with an existing Employee information form.
3	Manager selects particular Employee from the Employee list and fills out the form with the new information. Manager then submits the form.
4	System checks the validity of the information entered and that no other existing Employee has matching password if password field was modified. Then system updates Employee database, and closes the existing Employee information form.

Extensions

Step	Action
4.1	System finds another Employee in the database with the same password. System then requests a valid Employee password.
4.1	Manager changes password and submits the new Employee information form.

UC-#8: UpdateFloorPlan **(Will not be implemented)**	Manager updates the floor plan through a graphical interface by adding or removing tables, as well as changing table position on the floor plan. **Will not be implemented due to complexity of implementation. Should be considered for future implementation to decrease client's dependence on product supportability and increase product flexibility by allowing client to change floor plan on demand.**
UC-#9: AddToMenu	Manager adds an item to the menu by entering name of the item, recipe, preparation time, price etc. into the form.
UC-#10: DeleteFromMenu	Manager deletes an item from the menu.

UC-#11: UpdateItemInMenu	Manager selects an item from the menu and updates item's name, recipe, preparation time, price etc.
UC-#12: ScheduleSpecial **(Will not be implemented)**	Manager schedules special price for an item, by specifying start time, end time, day of the week, recurrence and special price. **Should be implemented in the future to give Manager an ability to schedule price-change events on demand.**
UC-#13: ViewStats	Manager views statistics for the restaurant. time, end time, day of the week, recurrence and special price.

UC-#14:	**SeatCustomer**
Related Requirements	
Goal In Context	Host attempts to place Customer at a table
Preconditions	Host is logged in and Host interface is opened on the terminal
Successful End Condition	Host successfully seats/assigns Customer to a table
Failed End Condition	Host fails to seat/assign Customer to a table
Primary Actors	Host
Secondary Actors	Server
Trigger	Host selects an empty table from the floor plan on the Host interface

Main Flow

Step	Action
1	Host selects an empty table from the floor plan on the Host interface.
2	System responds by presenting seat-Customer form requesting Customer party size and providing Host with an option to change Server from the default Server to any other logged-in Server.
3	Host enters the size of Customer party, and submits the seat-Customer form.
4	System changes the status of the table to occupied, creates a new blank order and records it in the Active Orders database, System then signals the assigned Server. Then it closes seat-Customer form.

Extensions

Step	Branching Action
1.1	All tables are either occupied or being cleaned. Server requests Customer to wait.
2.1	Host chooses to change Server.
2.2	Host enter Server ID to change. Proceed to step 3

UC-#15: AddItem

Related Requirements

Goal In Context: Server attempts to add an item to the tab from Server's terminal

Preconditions: Customer has been placed at the table and a tab for the table exists

Successful End Condition: An item has been added to the tab, Chef and/or Bartender have been notified

Failed End Condition: No items are added to the tab

Primary Actors: Server

Secondary Actors: Chef, Bartender

Trigger: Server selects to add item to an open tab on Server's terminal

Main Flow

Step	Action
1	Server selects to add item to an open tab on Server's terminal.
2	System responds by presenting Customer with the menu.
3	Server selects the desired menu item.
4	System responds by requesting the quantity of item desired.
5	Server responds by entering the quantity desired.
6	System responds by providing Server with options to attach a special comment, to proceed or cancel.
7	Server selects to proceed with the adding of an item to the tab.
8	System adds the item(s) to the tab.
9	System shows Server current item(s) in tab, and provides Server with options to select adding another item, or updating an existing item(s), or to place an order.
10	Server places the order.
11	System separates drinks from food and sends food order to the Chef's queue and drink order to the Bartender's queue. Then system sends an update to the active order database with the newly added items.

Extensions

Step	Branching Action
6.1.1	Server selects to attach a special comment to the selected item. Then Server chooses to proceed and continues with Step 8.
6.2.1	Server selects to cancel addition of the selected item.
6.2.2	System returns Server to the opened tab for the selected table. Proceed with Step 1.
10.1.1	Server selects to add another item to the tab. System goes back to step 2.
10.2.1	Server selects an item and chooses to update it.
10.2.2	System responds by providing Server with options to attach a special comment, to change quantity of the selected item or to remove selected item.
10.2.3	Server selects to remove item.
10.2.4	System removes item from tab and returns Server to the opened tab for the selected table. Proceed with Step 9.
10.2.3.1	Server selects to change quantity of the selected item.
10.2.3.2	System changes the quantity of the selected item. Proceed with step 9.

UC-#16:RemoveItem	Server removes an undesired item from the table's tab before placing the order. If Server removes an undesired item after order has been placed, Manager enters authorization code and completes the removal of item from the tab.
UC-#17:AdjustPrice **(Will not be implemented)**	Server adjusts the price of an item on the tab to reflect a discount coupon. **Should be implemented in the future to give Server more flexibility when interacting with the Customer, and to give Manager the ability to attract more Customers through the use of coupons.**

UC-#18	**PlaceOrder**
Related Requirements	
Goal In Context	Server places order for a specific table
Preconditions	Server has added items to tab of a certain table
Successful End Condition	Order is placed, and Chef/Bartender has been notified
Failed End Condition	Order is not placed and/or Chef/Bartender has been notified
Primary Actors	Server
Secondary Actors	Chef, Bartender
Trigger	Server requests to place order through the Server interface

Main Flow	Step	Action
	1	Server requests to place order through the Server interface.
	2	System adds order to the table tab in active order database.
	3	System notifies Server order is placed.

Extensions	
UC-#19:ViewTab	Server views tab for a particular table.

UC-#20	**CloseTab**	
Related Requirements		
Goal In Context		Server closes a tab of a certain table
Preconditions		Customer called Server and is ready to leave
Successful End Condition		Tab is collected, stored in inactive_order database, table tab is cleared
Failed End Condition		Tab is not stored in the database and/or table tab is not cleared
Primary Actors		Customer, Server
Secondary Actors		
Trigger		Customer asks Server for check
Main Flow	Step	Action
	1	Customer asks Server for check.
	2	Server selects table through the Server interface.
	3	System responds by presenting active tables.
	4	Server selects close tab for the desired table.
	5	System responds by presenting total amount due
	6	Server collects payment from Customer.
	7	Server selects confirm.
	8	System prints bill, and moves active tab from active_order database into the inactive_order database.
Extensions		

UC-#21	**MarkTableReady**	
Related Requirements		
Goal In Context		Server marks a certain table ready
Preconditions		Customer left and Busboy cleaned the table
Successful End Condition		Table is marked ready
Failed End Condition		Table is not marked ready
Primary Actors		Server
Secondary Actors		
Trigger		Server
Main Flow	Step	Action
	1	Server selects table through the Server interface.
	2	System responds by presenting active tables.
	3	Server selects mark table ready for the desired table.
	4	System marks table ready, and updates tabletops database.
Extensions		

UC-#22: MarkItemReady **(Modified)**	Chef announces item as ready to be served. This use case will be used at Chef's discretion to inform Server when items such as appetizers are ready.

UC-#22:	**MarkItemReady**	
Related Requirements		
Goal In Context		Chef attempts to inform Server of completion of preparation of selected item
Preconditions		Item has been prepared; connection to the database exists
Successful End Condition		Server is informed and item is marked as prepared to be delivered to Customer by Server
Failed End Condition		Item is not marked as prepared in the Chef's order queue and Server is not informed
Primary Actors		Chef
Secondary Actors		Server, Timer
Trigger		Chef selects an Item from the current order tab and marks that Item ready
Main Flow	Step	Action
	1	Chef selects an Item from the current order tab and marks that Item ready.
	2	System notifies the Server and highlights Item in Chef's interface.
	3	System sets Timer.
	4	Server delivers item(s), and mark item(s) delivered.
	5	System resets Timer.
Extensions	Step	Branching Action
	4.1	Server doesn't deliver item, or forgets to mark item(s) ready.
	4.2	Timer reaches limited time.
	4.3	Timer triggers System.
	4.4	System goes back to step 2.

UC-#23: AddItemAtBar **(Modified)**	Bartender places an item on a new or existing tab. Multiple tabs are showing on the Bartender's interface, one per Customer.

UC-#24	**CloseTabAtBar**
Related Requirements	
Goal In Context	Bartender closes a certain tab for a bar customer
Preconditions	Customer is ready to leave
Successful End Condition	Tab is collected, stored in inactive_order database, tab is cleared
Failed End Condition	Tab is not stored in the database and/or tab is not cleared
Primary Actors	Customer, Bartender
Secondary Actors	
Trigger	Customer asks Bartender for check

Main Flow	Step	Action
	1	Customer asks Bartender for check.
	2	Bartender selects tab through the Bartender interface.
	3	System responds by presenting list of active Customers.
	4	Bartender selects desired Customer
	5	System responds by presenting total amount due
	6	Bartender collects payment from Customer.
	7	Bartender selects confirm through the Bartender interface.
	8	System prints bill, and moves active tab from active_order database into the inactive_order database.

Extension

UC-#25		**AddItemAtTable**
Related Requirements		
Goal In Context		Customer attempts to add an item to the tab from Customer terminal located at table
Preconditions		Customer has been placed at the table and a new tab for the table is open
Successful End Condition		An item has been added to the tab, Chef and/or Bartender and Server have been notified
Failed End Condition		No items are added to the tab
Primary Actors		Customer
Secondary Actors		Chef, Bartender, Server
Trigger		Customer requests to add an item to the tab from the Customer terminal
Main Flow	Step	Action
	1	Customer requests to add an item to the tab from the Customer terminal.
	2	System responds by presenting Customer with the menu.
	3	Customer selects the desired menu item.
	4	System responds by requesting the quantity of item desired.
	5	Customer responds by entering the quantity desired.
	6	System responds by providing Customer with options to attach a special comment, to proceed or cancel.
	7	Customer selects to proceed with the adding of an item to the tab.
	8	System adds the item to the tab.
	9	System shows Customer current item(s) in order, and asks Customer to select adding another item, or updating existing item(s), or to place an order.
	10	Customer places an order.
	11	System adds order to the table tab in active order database.
Extensions	Step	Branching Action
	6.1.1	Customer selects to attach a special comment to the selected item. Proceed with Step 6.
	6.2.1	Customer selects to cancel addition of the selected item.
	6.2.2	System returns Server to the opened tab for the selected table. Proceed with Step 1.
	10.1.1	Customer selects to add another item to the tab. System goes back to step 2.
	10.2.1	Customer selects an item and chooses to update it.
	10.2.2	System responds by providing Customer with options to attach a special comment, to change quantity of the selected item or to remove selected item.
	10.2.3	Customer selects to update item.
	10.2.4	System updates item in the tab and returns Customer to the opened tab for the selected table. Proceed with Step 9.

UC-#26	**CallServer**	
Related Requirements		
Goal In Context		Customer attempts to call a Server because he/she doesn't want to use the system, or needs assistance
Preconditions		Customer has been placed at the table
Successful End Condition		Server is notified
Failed End Condition		Server is not notified
Primary Actors		Customer
Secondary Actors		Server
Trigger		Customer requests to call Server from the Customer terminal
Main Flow	Step	Action
	1	Customer requests to call Server from the Customer terminal.
	2	System notifies the assigned Server.
	3	System informs Customer that Server has been notified.
Extensions		

UC-#27: ManageTabAccounts **(Will not be implemented)**	Manager adds, removes or updates individual Customer tab accounts. Upon Customer's request a personal tab account can be maintained by the restaurant. Tab account will accumulate item orders, over a period of time specified by the Manager, and upon Customer's preferred payment type, cash or credit card, he/she will be periodically charged. As payment is received, tab is cleared. **This use case will not be implemented due to additional complexity and time restrictions. It should be implemented in the future to eliminate paperwork required to keep track of tab accounts.**
UC-#28: TransferTab **(Will not be implemented)**	Server/Bartender transfers an open tab onto an existing Customer tab account given proper identification of the Customer. **Should be implemented with UC-#27 to transfer active orders onto Customer's Tab at a touch of a button.**

UC-#29: AssignDefaultServer Host assigns a default Server to a selected table.
(New Use Case)

UC-#30: MarkOrderReady **(New Use Case)**	Chef/Bartender marks order ready, Server is notified to pick up the food/drink(s), timer is set to make sure the food is picked up.
UC-#31: CancelItem **(New Use Case)**	Chef/Bartender cancels item, Server is notified with the canceled item(s) and order.
UC-#32: MessageServer **(New Use Case)**	Manager/Chef/Bartender sends a message to Server for any other reason.
Refresh **(New)**	Refresh is an internal class that continually updates the a given PC based GUI (Chef/Bartender/Host) from the database to show any changes that occurred to the database since the last refresh. (Manager/Customer) GUI's are not going to have this function since they are event driven cases. For the Server, this function will not be implemented for the GUI, but will be implemented for a Communicator class that will continually checks the database for any changes and sends the changes to the BlackBerry through HTTP.

Use Case Diagram:

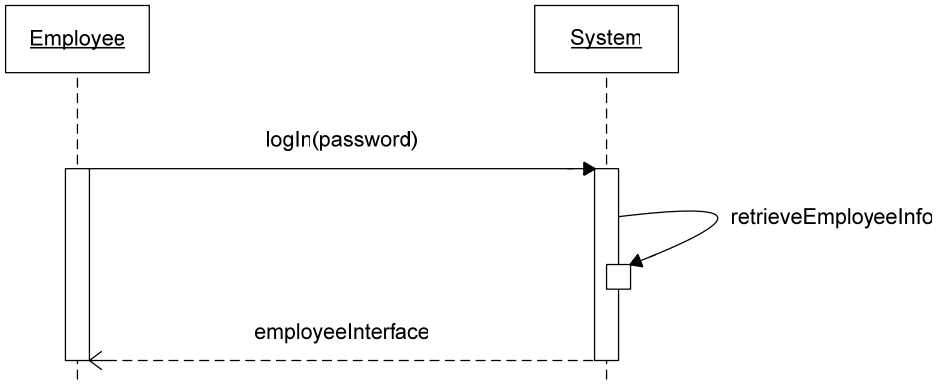
In the above use case diagram, since every employee who interacts with the system will be provided with individual terminal, UC3 and UC4 are only invoked once when employee logs into/out of his/her terminal. Once employee logs in he/she remains logged in until he/she logs out. Use of database actor has been removed from the case diagram to prevent clattering. Most of the use cases will be accessing database as that is where all restaurant operation related data is recorded and stored.

UC27 can be further broken down into use cases such as create new customer tab account, remove customer tab account, and update customer tab account. This is avoided to prevent cluttering.
To create a customer tab account, customer feels out a form on paper with his name, address, credit-card information and submits it to the manager through other employees. Manager examines the data, approves the creation of new tab account and then enters it into the system.

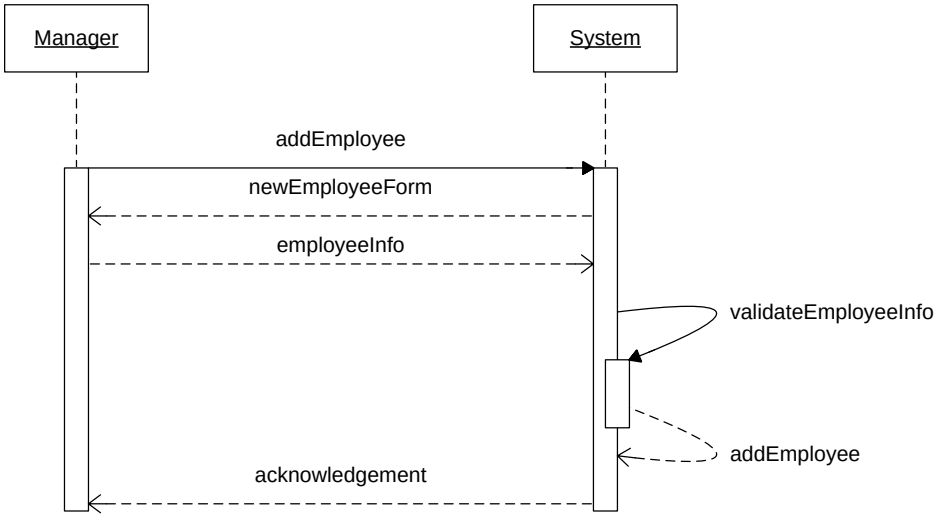
When manager creates a server-employee account, he/she specifies the section of the floor to which that server will be automatically assigned when server logs into the system on his/her terminal. Host has the liberty of reassigning servers to tables as he/she sees fit.

System Sequence Diagrams:

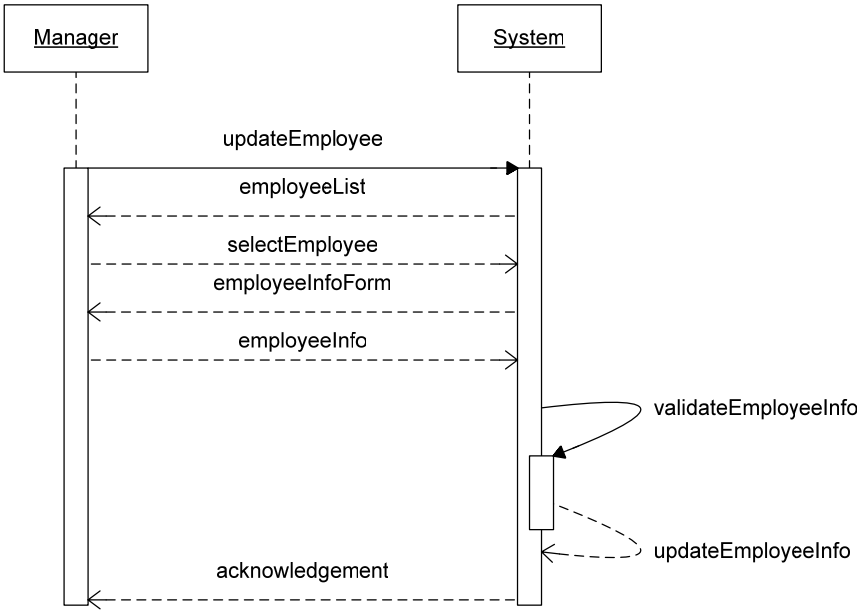
LogIn Sequence Diagram



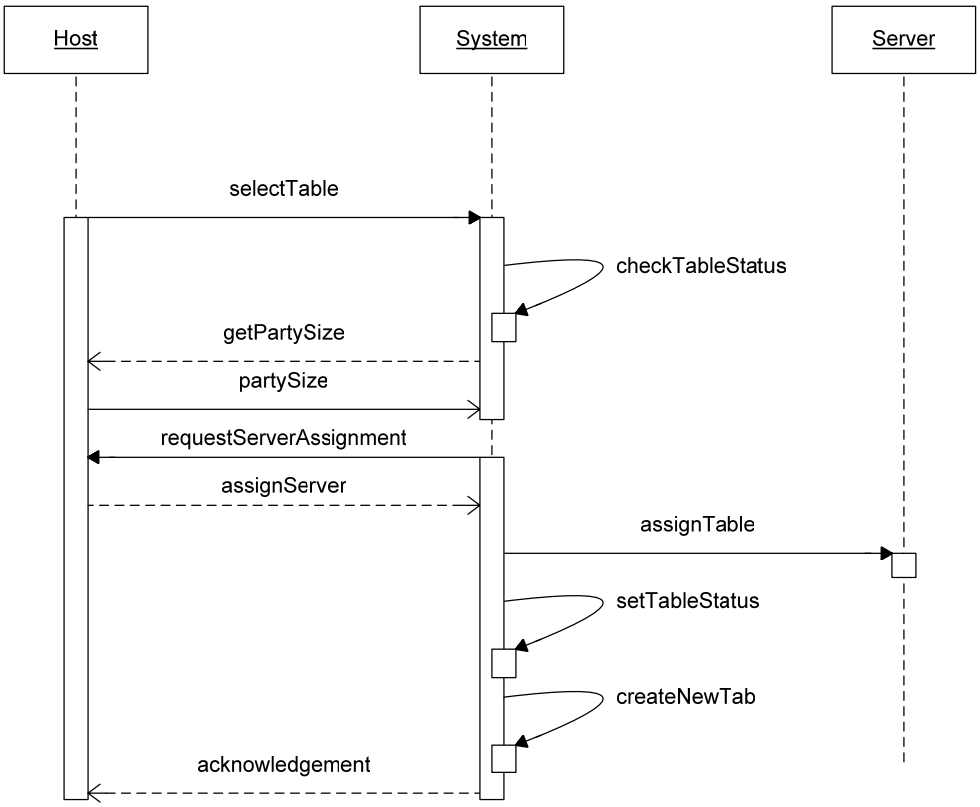
AddEmployee Sequence Diagram



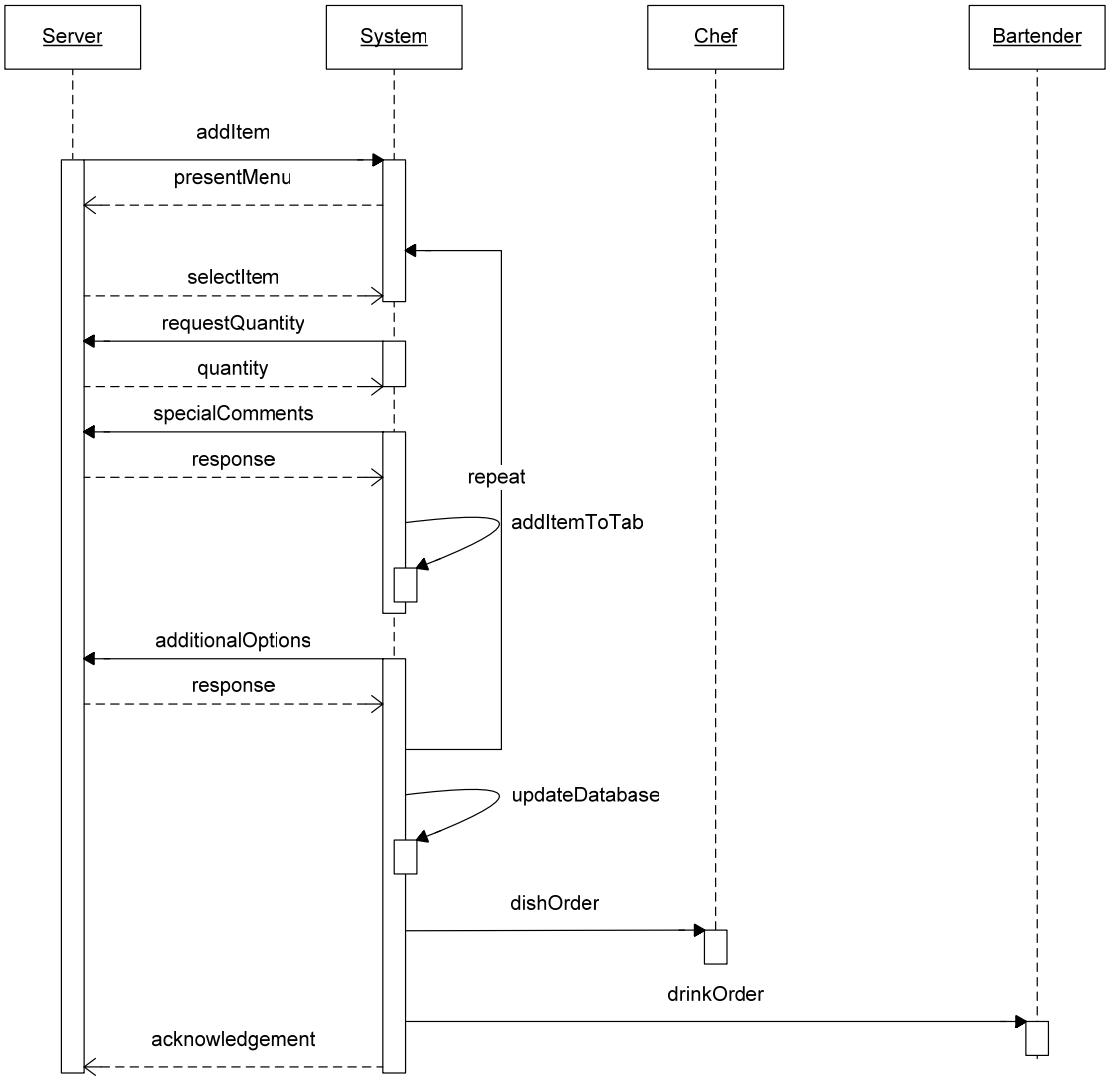
UpdateEmployee Sequence Diagram



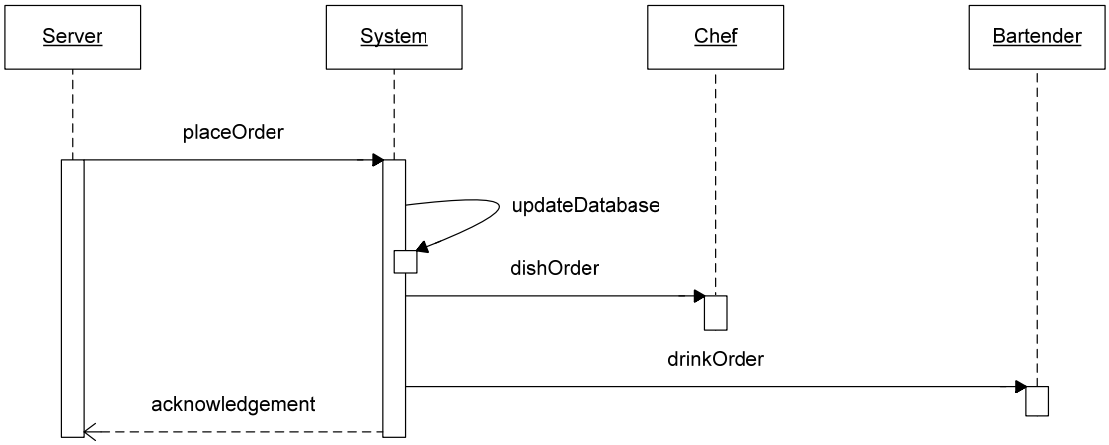
SeatCustomer Sequence Diagram



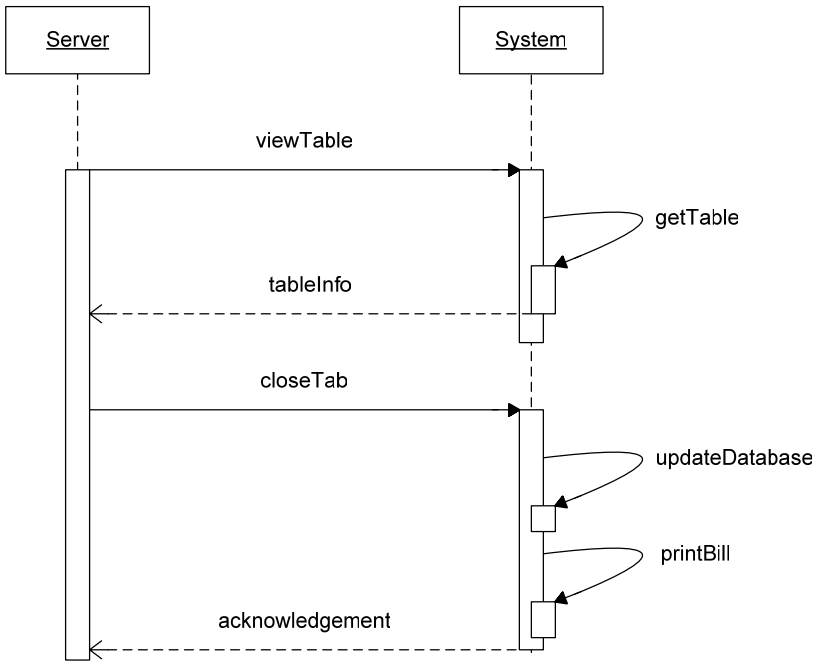
AddItem Sequence Diagram



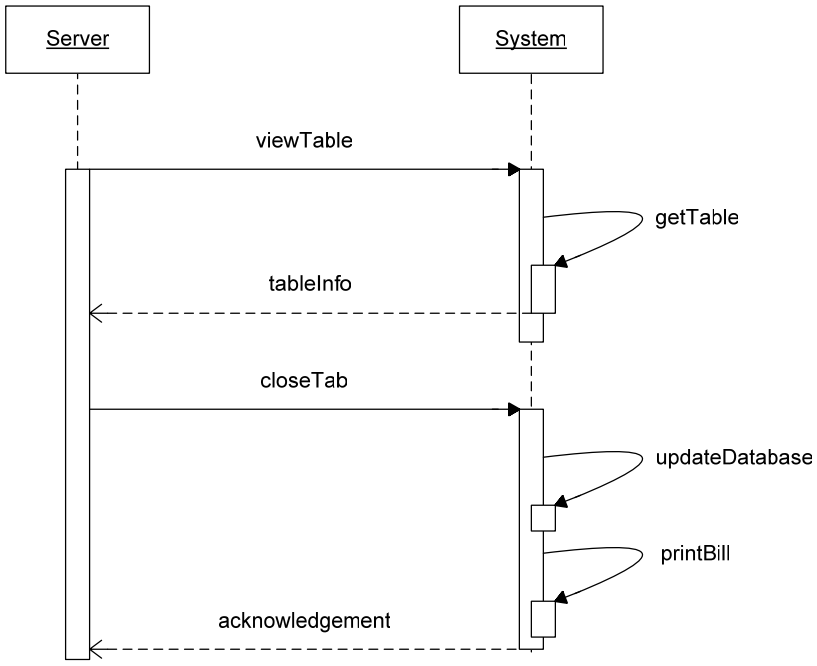
PlaceOrder Sequence Diagram



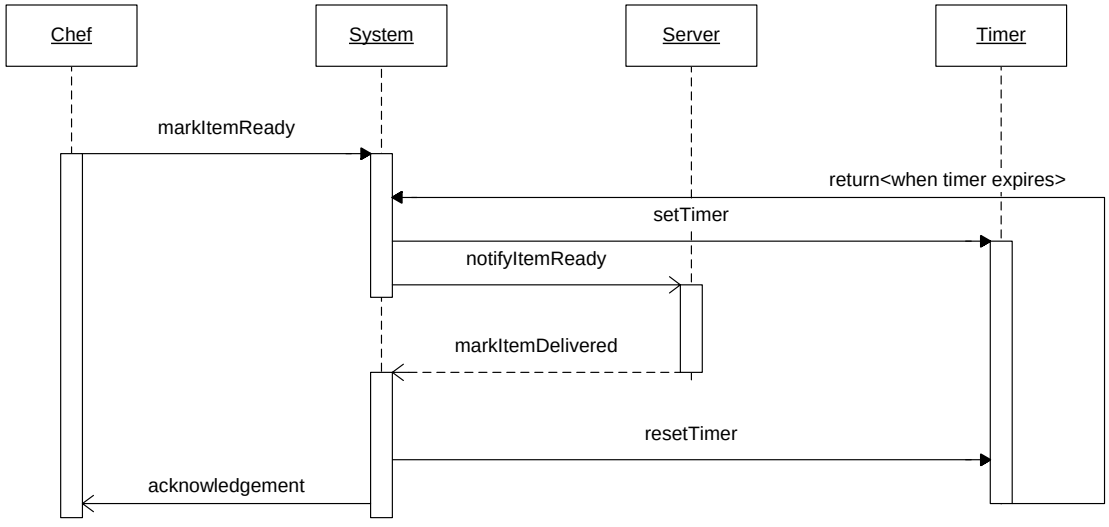
CloseTab Sequence Diagram



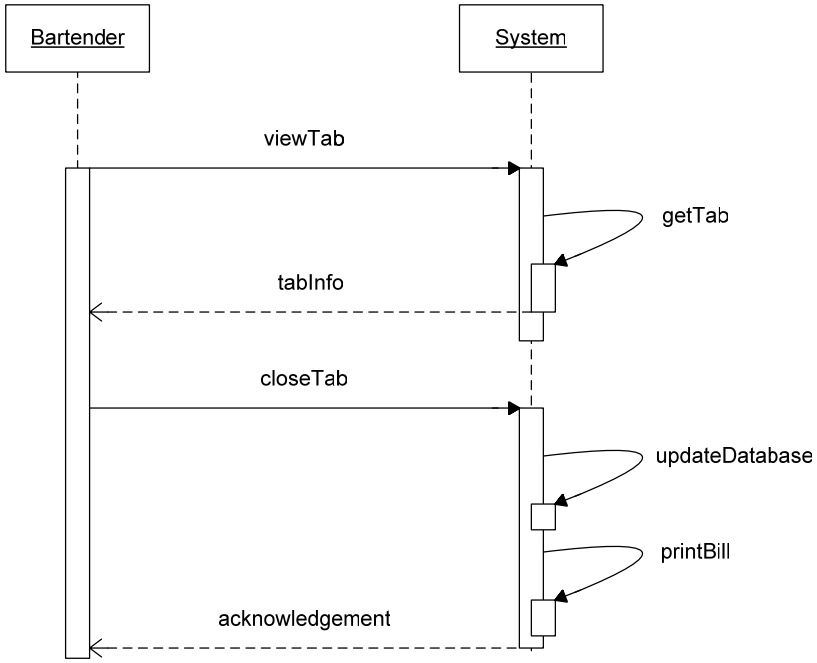
MarkTableReady Sequence Diagram



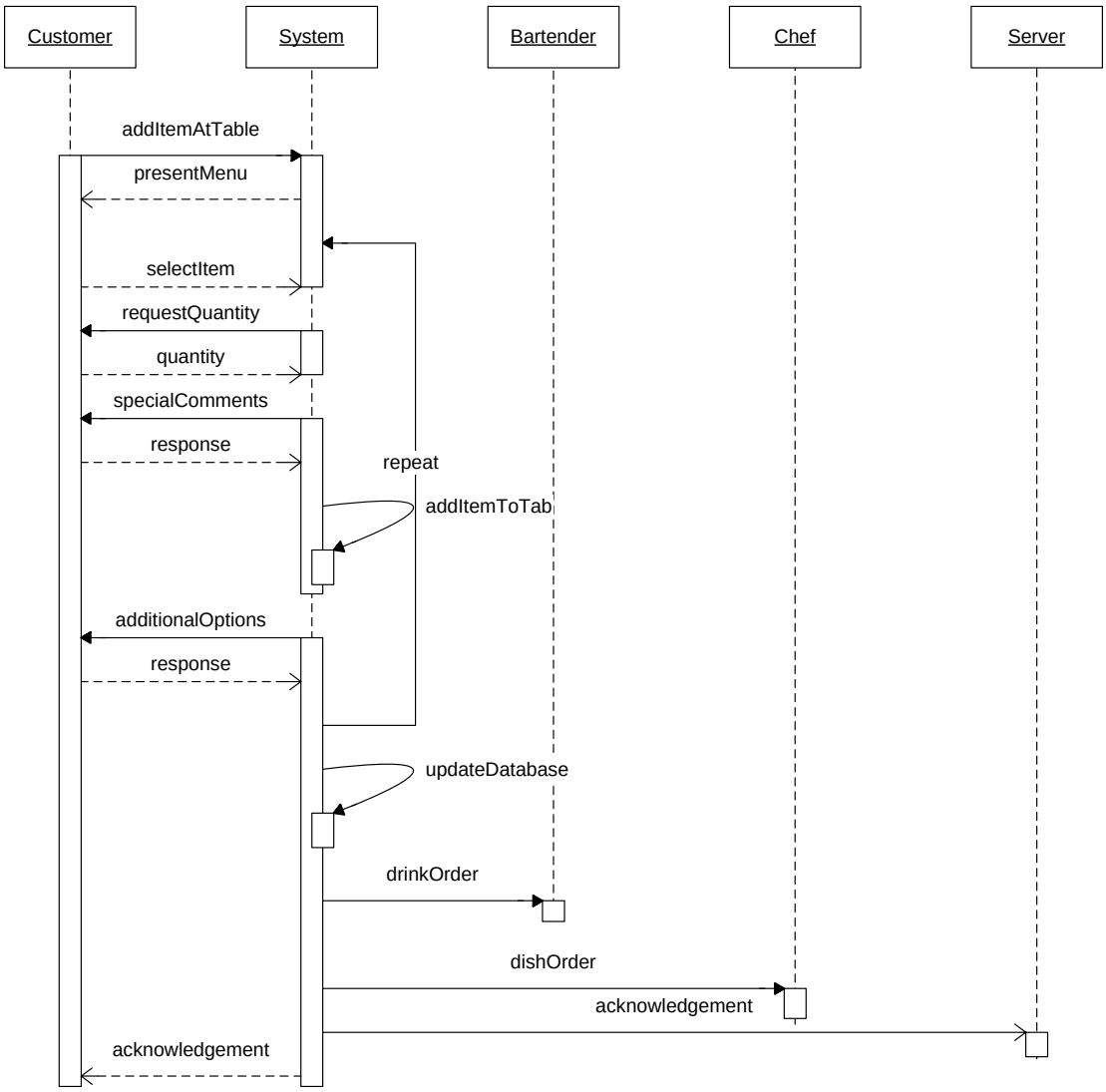
MarkItemReady Sequence Diagram



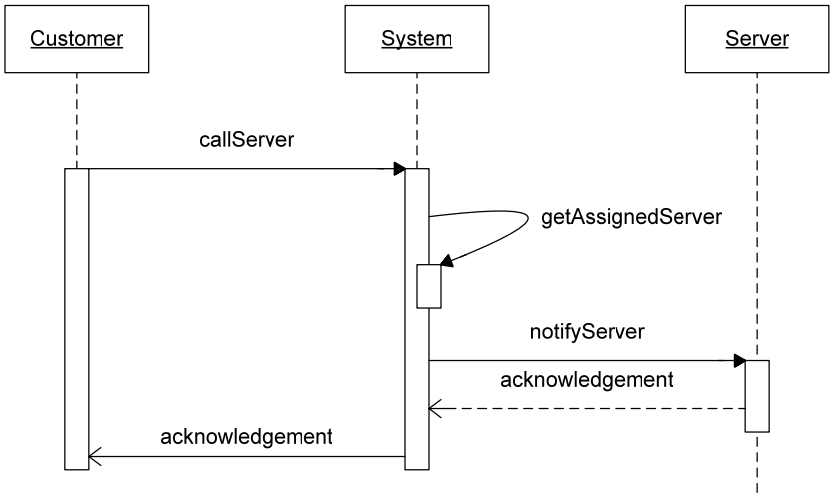
CloseTabAtBar Sequence Diagram



AddItemAtTable Sequence Diagram



CallServer Sequence Diagram



7. Nonfunctional Requirements:

Nonfunctional requirements describe aspects of the system that are not directly related to the functional behavior of the system, and they are part of the FURPS+ model. Nonfunctional requirements consist of four categories: **Usability**, **Reliability**, **Performance**, and **Supportability**.

1. **Usability**

   OpenBar restaurant POS system uses touch-screen computers stationed at different places of a restaurant, replacing traditional pen-paper method of order processing and communication. The interface, which changes according to different users logged in, is fairly easy to understand, thus making the software easy to use. A maximum of one hour training should guarantee clients operating the system proficiently. In case of questions regarding the software, online manual is also available on our website.

2. **Reliability**

   OpenBar system is based on a real-time server, and has minimal requirement on CPU and memory usages (1.2GHz Pentium III and 256M RAM or above), therefore the system will not go down easily. All terminals are connected to the central server, Intranet-based, which means no Internet is required, also preventing possible virus. In addition, the timer in the system forces automatic log-out after a period of inactivity, avoid unauthorized operations.

3. **Performance**

   Performance includes *response time*, *throughput*, *availability*, and *accuracy*.

   Response time: as stated above, all terminals are connected to the central server using Ethernet cables or bluetooth, with 10/100M connection, the system should response to user inputs almost instantly.

   Throughput: a medium performance computer as the server will allow 20 users accessing the database simultaneously, and be able to handle a busy night.

   Availability: this software requires no Internet, which means the system is on whenever the server is on.

   Accuracy: almost every operation involves a confirmation check, preventing inaccuracy.

4. **Supportability**

   Supportability includes *adaptability/portability* and *maintainability*.

   Adaptability/portability: OpenBar POS system is written in Java and SQL, therefore it works in any hardware or software environments that are Java and SQL-friendly.

   Maintainability: for basic maintainability, restaurant owners can log in as manager/administrator and modify items, prices, and floor layout etc. Most problems can be easily fixed by rebooting the system, if they do occur. Software functionality-wise, technical support contact information is available on our website.

5. **Implementation**

   OpenBar POS system is written in Java language and uses SQL server. While implementation of the system can be achieved by basic notepad coding, we suggest using a Java editing program—Netbeans IDE 5.5. Netbeans is updated periodically with the JDK (Java Development Kit) itself to fit into many different hardware platforms, including Windows, Mac OS, Linux, and Solaris.

6. **Interface**

   Considering most restaurants using OpenBar POS system will most likely run the system on relatively new computers, legacy systems and interchange formats should not cause any problem to clients. In case such problem does occur, JDK updates should solve it.

7. **Operation**

   OpenBar POS system provides all everyday restaurant operations you will ever need, ranging from basic order taking, to special event setup, to floor plan change. Administrator/manager can set accessing privilege of each user, therefore preventing unauthorized operation.

8. **Packaging**

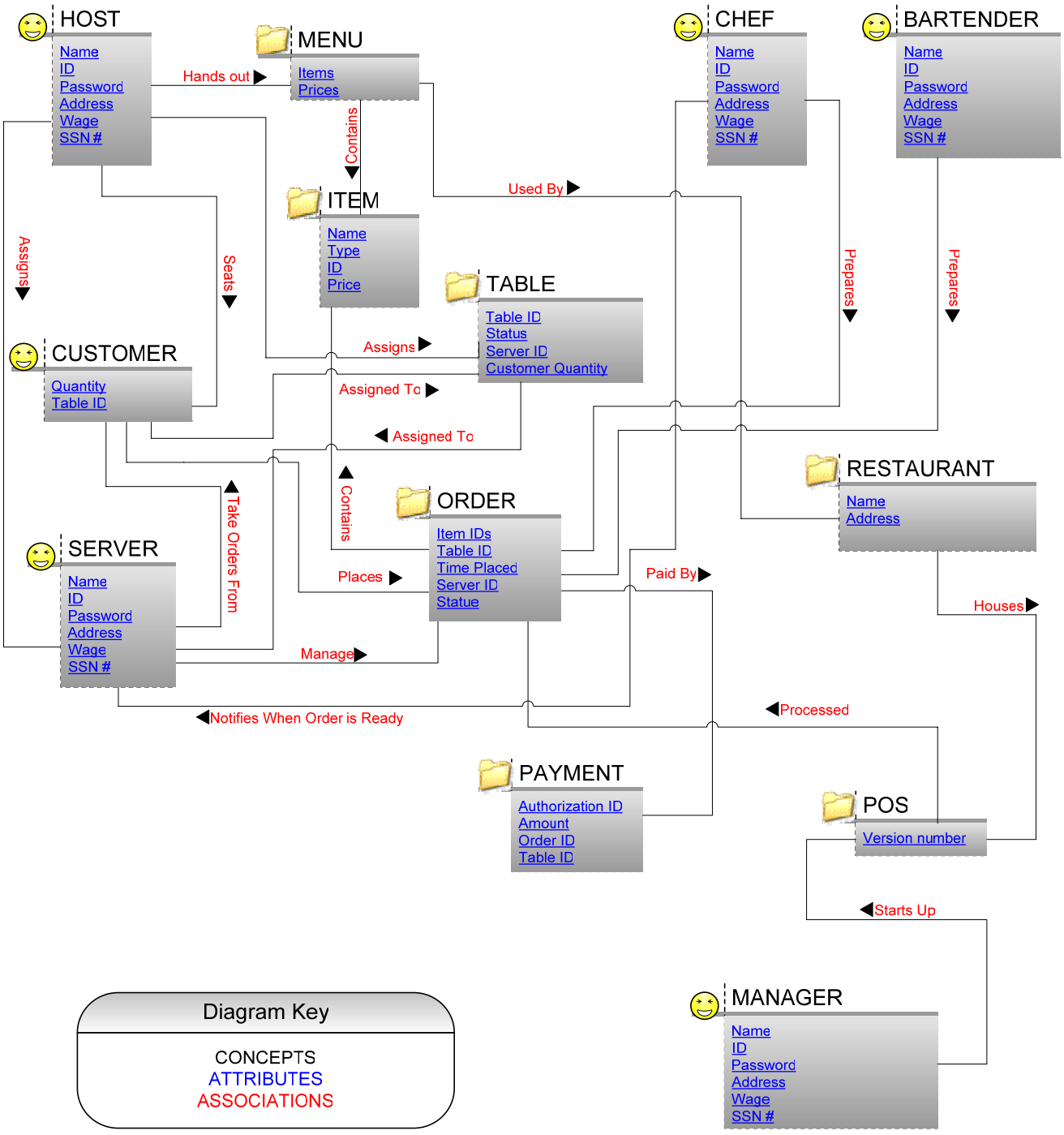
   Installation of OpenBar POS system requires JRE (Java Runtime Environment), SQL server (mySQL), and of course OpenBar itself. After purchasing, an easy installation CD including all the required softwares will be mailed to customers, in addition, on-site technician help will be provided upon request.

9. **Legal**

   While there are many similar POS systems available on the market, OpenBar is designed and written solely by our team. A license will be obtained when the system is finalized.

8. Domain Analysis:

Domain Model:



System Operation Contracts:

Operation	Add Item to Order
Preconditions	Customer has been placed at the table and a new tab for the table is open Table ID: known Server ID: known Item ID: known
Post conditions	An item has been added to the order, chef and/or bartender and server have been notified Item IDs: updated with Item ID Server/Chef/Bartender notified

Operation	Assign Table
Preconditions	Host is logged into the system.
Post conditions	Customer has been assigned to selected table, a server has been assigned to the table, and a new tab is opened for the table Table ID: assigned Server ID: assigned Customer Quantity: assigned Status = occupied

Operation	Place Order
Preconditions	Server is logged into the system Table ID: corresponds to table in which order is being taken Server ID: corresponds to server that is placing the order
Post conditions	Order has been placed by a server and submitted. Item IDs: contain items being ordered Time Placed: contains current timestamp

Operation	Chef Notifies Server
Preconditions	Server is logged into the system Table ID: corresponds to table in which order is being

	taken to Server ID: corresponds to server that is being notified Order ID: corresponds to order that is ready
Post conditions	Server has been notified that order is ready and what items are ready. Time Placed: contains current timestamp
Operation	Server Closes Order
Preconditions	Server is logged into the system. Customer is ready to close their tab. Table ID: corresponds to table in which order is being closed Server ID: corresponds to server that is placing the order Order ID: corresponding order number Status = Active
Post conditions	Order has been closed by the server. Status = Inactive Server has received payment from Customer.

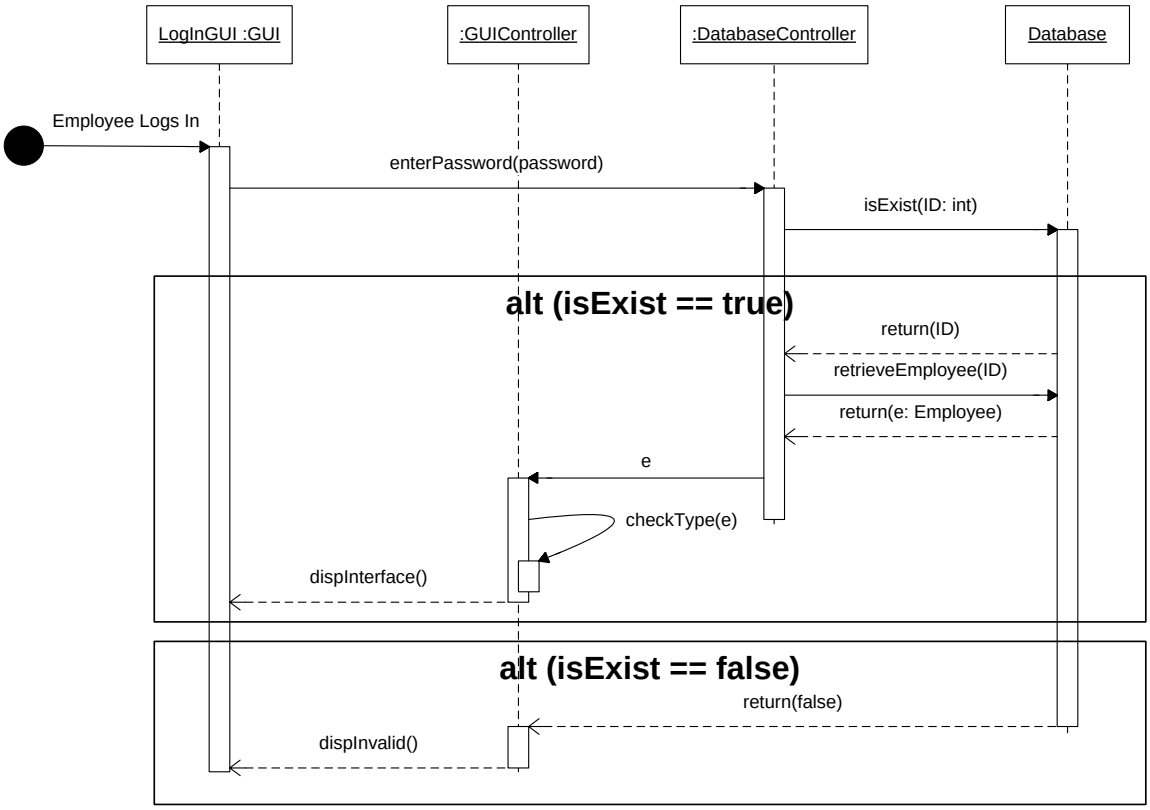
Operation	Server Clears Table
Preconditions	Customers have closed tab with Server and have left the table. Server has cleaned the table. Table ID: corresponds to table being cleared Server ID: corresponds to server that is clearing the table Status = Active
Post conditions	Table has been marked as inactive and is ready for more Customers to be seated by the Host. Status = Inactive

Operation	Bartender Closes Order
Preconditions	Bartender is logged into the system. Customer is ready to close their tab. Table ID: corresponds to table in which order is being taken Server ID: corresponds to bartender that is placing the order Order ID: corresponding order number Status = Active
Post conditions	Order has been closed by the bartender. Status = Inactive Server has received payment from Customer.

9. Interaction Diagrams

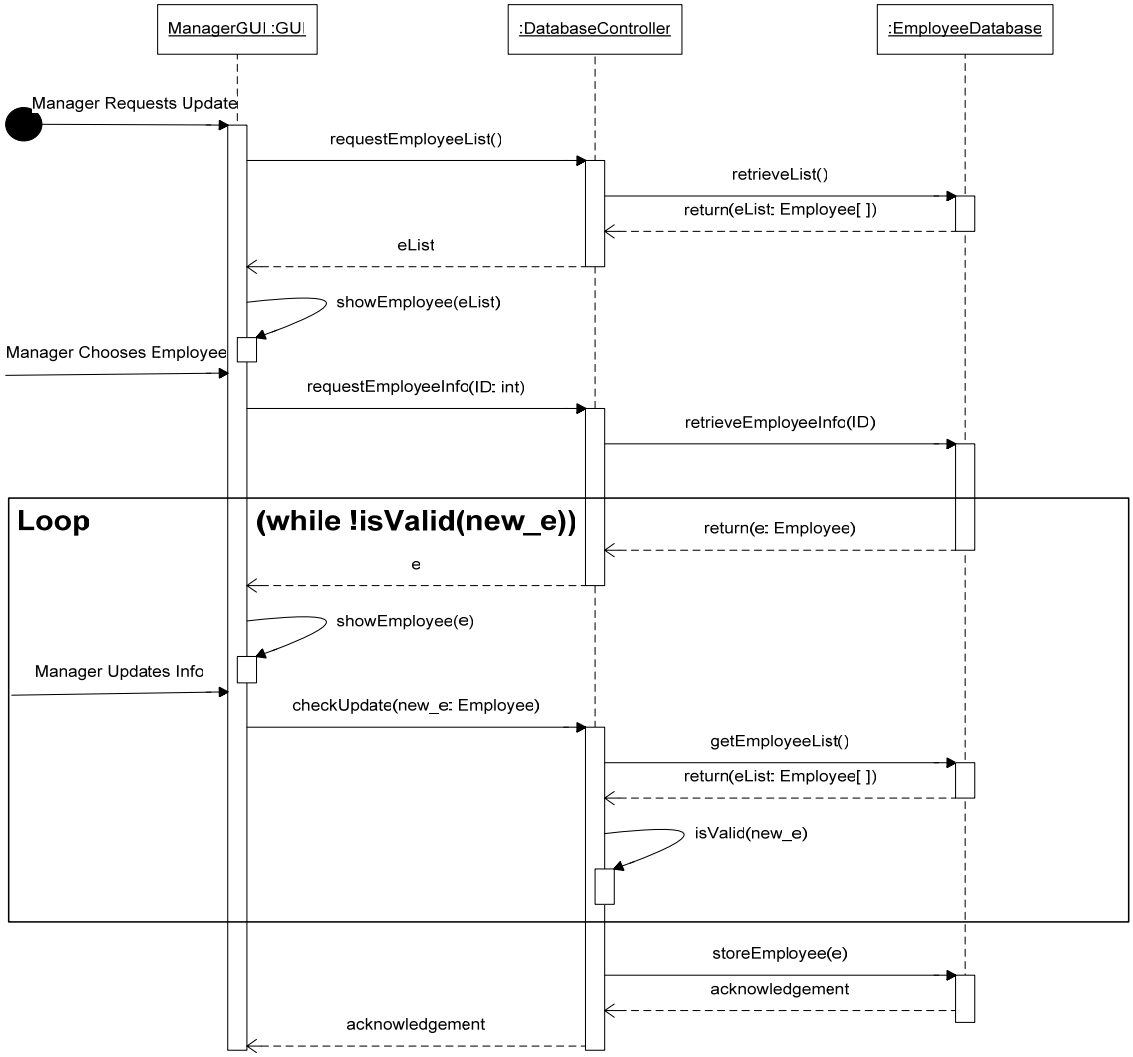
LogIn

This case is straight forward; only the employee logging in interacts with DatabaseController and it follows the ***Expert Doer Principle :*** only the employee and the DatabaseController needs to know and the controller signs the employee in to the database. This is a direct implementation of a **Publisher-Subscriber** design pattern where the GUIController is a publisher and the DatabaseController is the subscriber.



UpdateEmployee

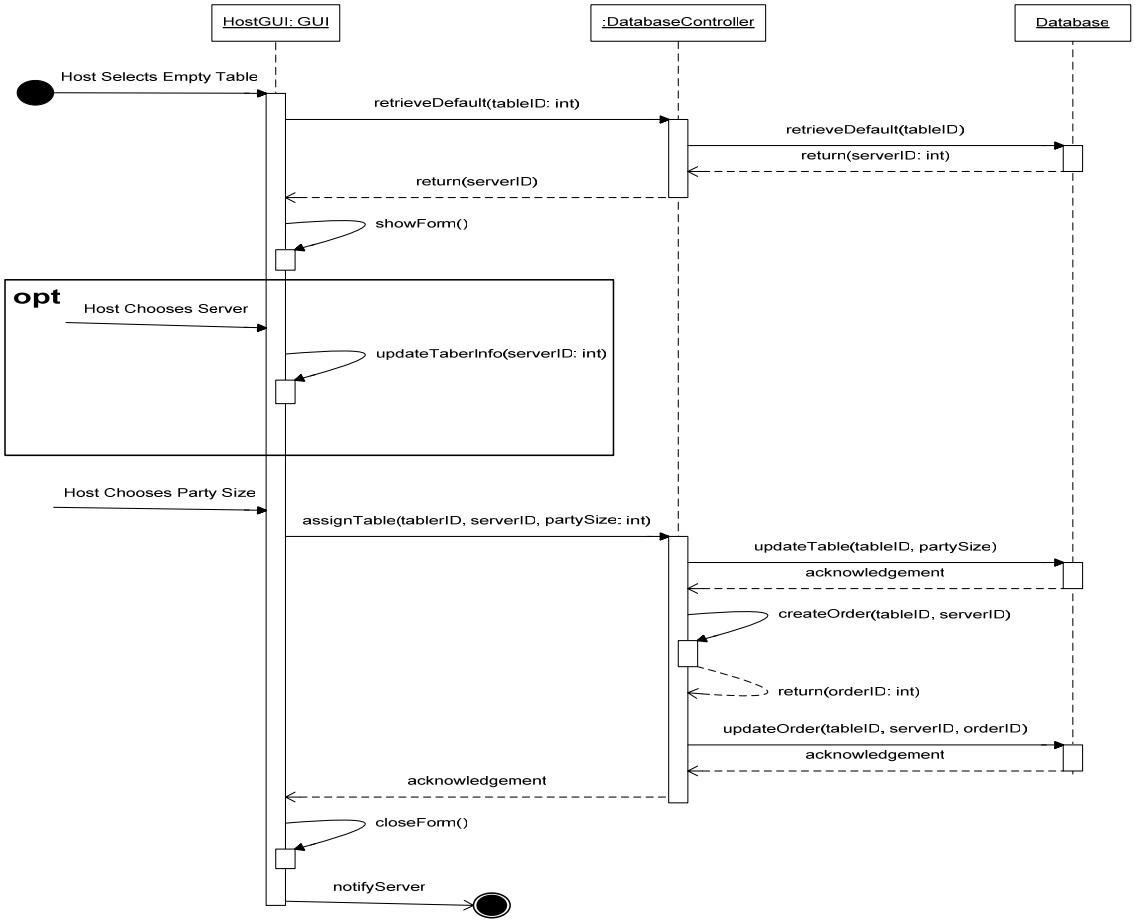
This case is straight forward where the manager using his interface access the EmployeeDatabase and updates it; it follows the ***Expert Doer Principle:*** Since only the manager needs to know so he does the job. This is a direct implementation of a **Publisher-Subscriber** design pattern where the GUIController is a publisher and the DatabaseController is the subscriber.



SeatCustomer

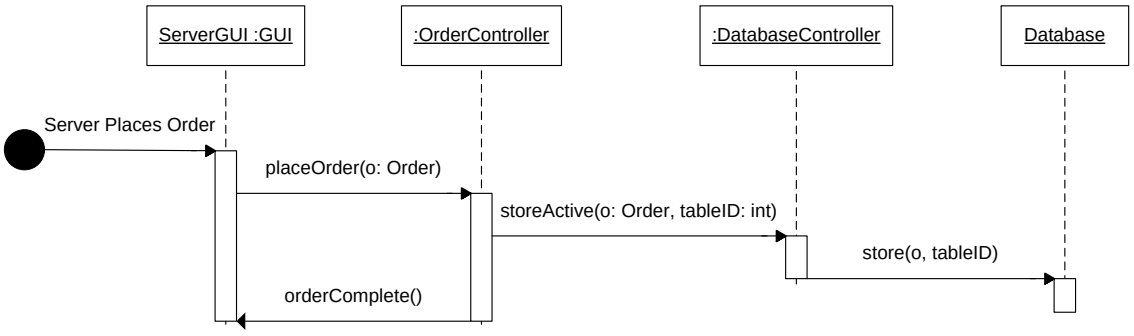
This case main role is seating the customer and updating the table and also notifying the assigned server based on a floor plan that keeps getting updated by the GUI class Refresh. The first role is assigned to the Host what follows the ***Expert Doer Principle:*** since the host need to know who is coming in and out. Once the table gets updated, the Refresh class that we implemented would update the Communicator with the new

modification, then the Communicator notifies the assigned Server. The workload is balanced between the all the interacting interfaces, and that follows the ***High Cohesion Principle.*** This is a direct implementation of a **Publisher-Subscriber** design pattern where the GUIController is a publisher and the DatabaseController is the subscriber.



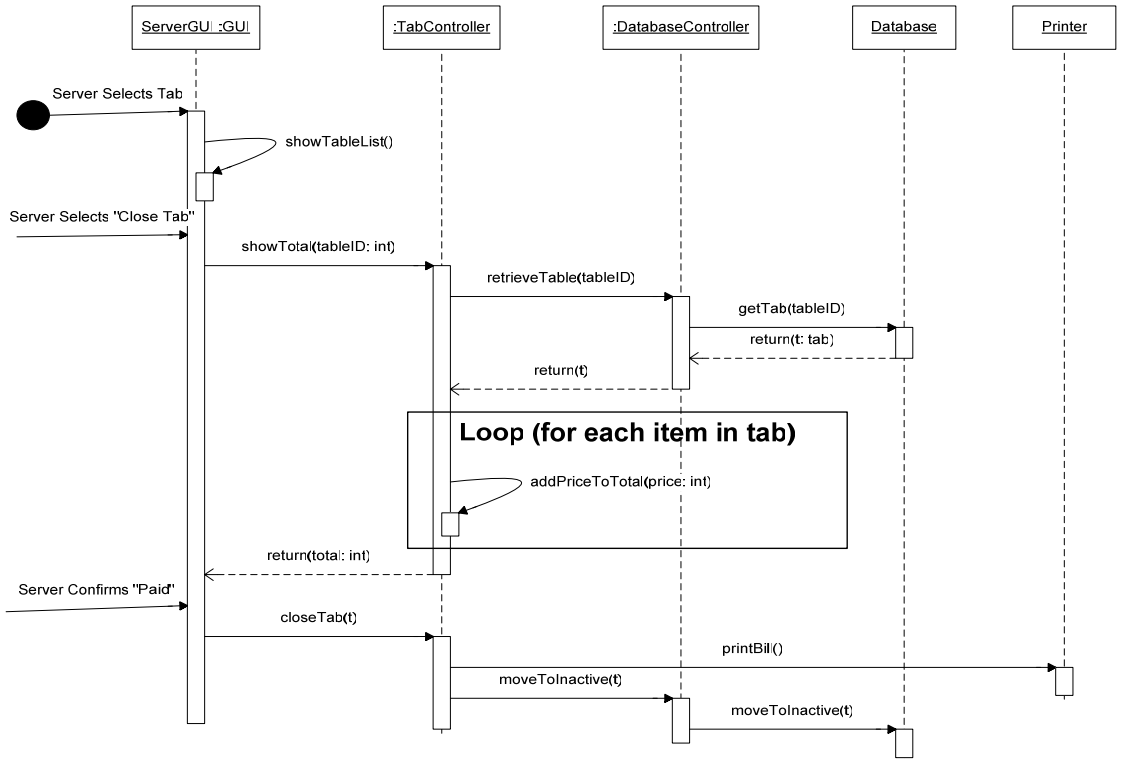
PlaceOrder

This case main role is to place an order that is already been selected and again we find low coupling: where ServerGUI reaches the DatabaseController Through the Communicator. Note that Communicator is not shown in the diagram because the Communicator is a fixed controller that serves as a Servlet for the Server interface, so there is no need to mention in it everytime. We also note that the design pattern implemented for all Server interface is the **Command** design pattern: for sending the data to the database. The communicator connection to the database is considered a **Publisher-Subscriber** pattern where the Communicator is the publisher and the DatabaseController is the subscriber.



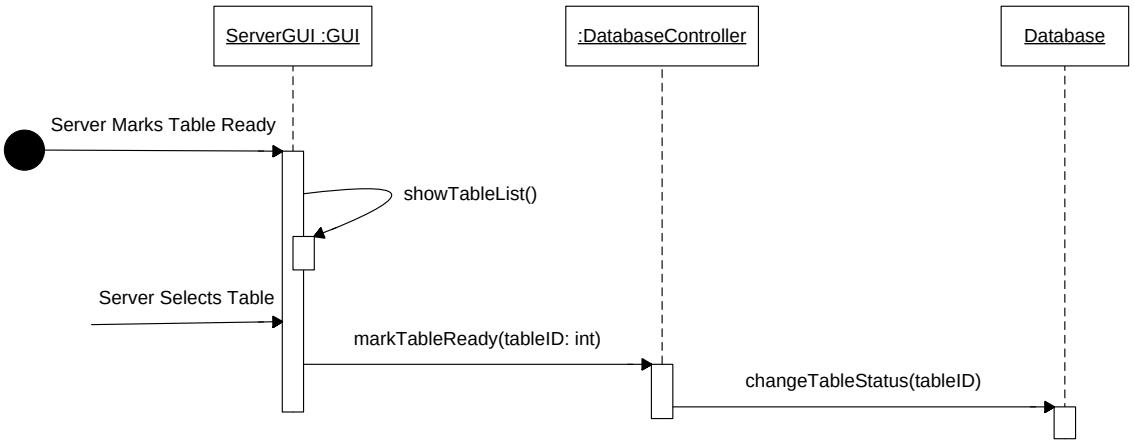
CloseTab

In this case we have two main roles: collecting payment, saving the order to the inactive database. After being called by the customer the server closes the tab, collects the payment, notifies the Busboy to clean the table and returns the printed receipt for the Customer. Second role is activated when the Server confirms closing the order the System saves a copy of the tab to the inactive orderd databases, then clears the tab for the next customer to come. This design follows the ***Expert Doer Principle*** since only the server needs to know about the tab when collecting the payment. This design also follows the ***High Cohesion Principle*** where the workload is balanced among the objects. For the design Pattern again we find the **Command** pattern is implemented for the Server interface: for sending the data to the database. Also, The communicator connection to the database is considered a **Publisher-Subscriber** pattern where the Communicator is the publisher and the DatabaseController is the subscriber.



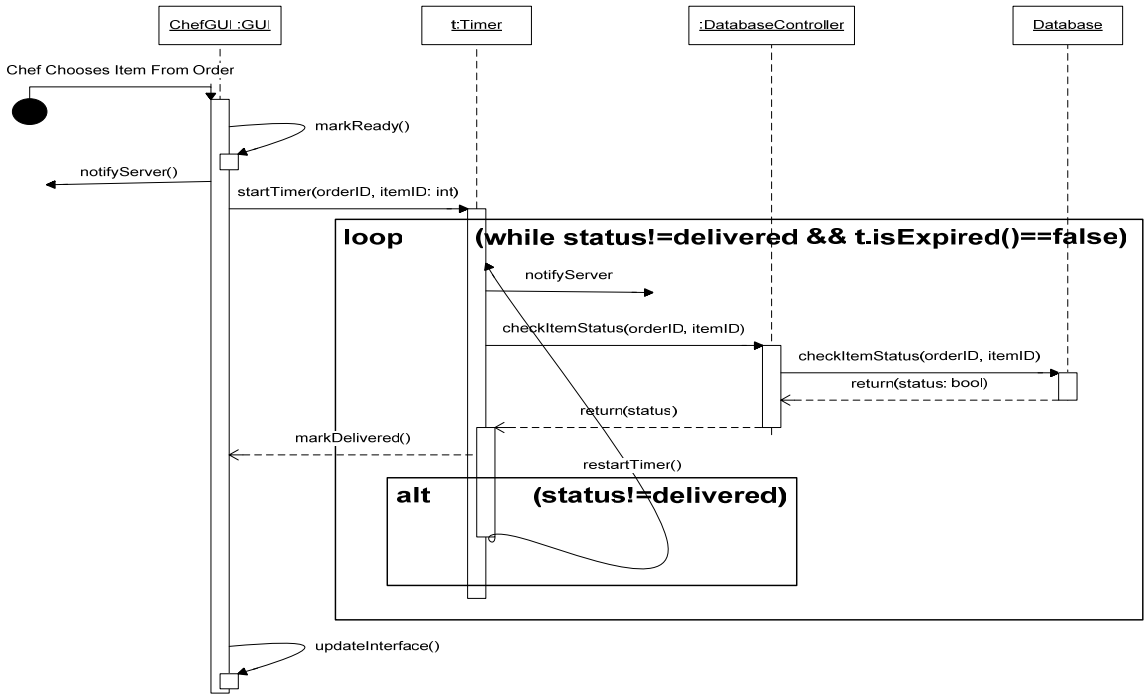
MarkTableReady

Another Straight forward case that completes the closing order process, and it was split because we wanted to take care of the case, where the Customer pays but still sits there. So when Customer leaves and Busboy cleans the table, the Server marks the table ready, and the ServerGUI sends a request to the DatabaseController to change the status of the table. For the design Pattern again we find the **Command** pattern is implemented for the Server interface: for sending the data to the database. Also, The communicator connection to the database is considered a **Publisher-Subscriber** pattern where the Communicator is the publisher and the DatabaseController is the subscriber



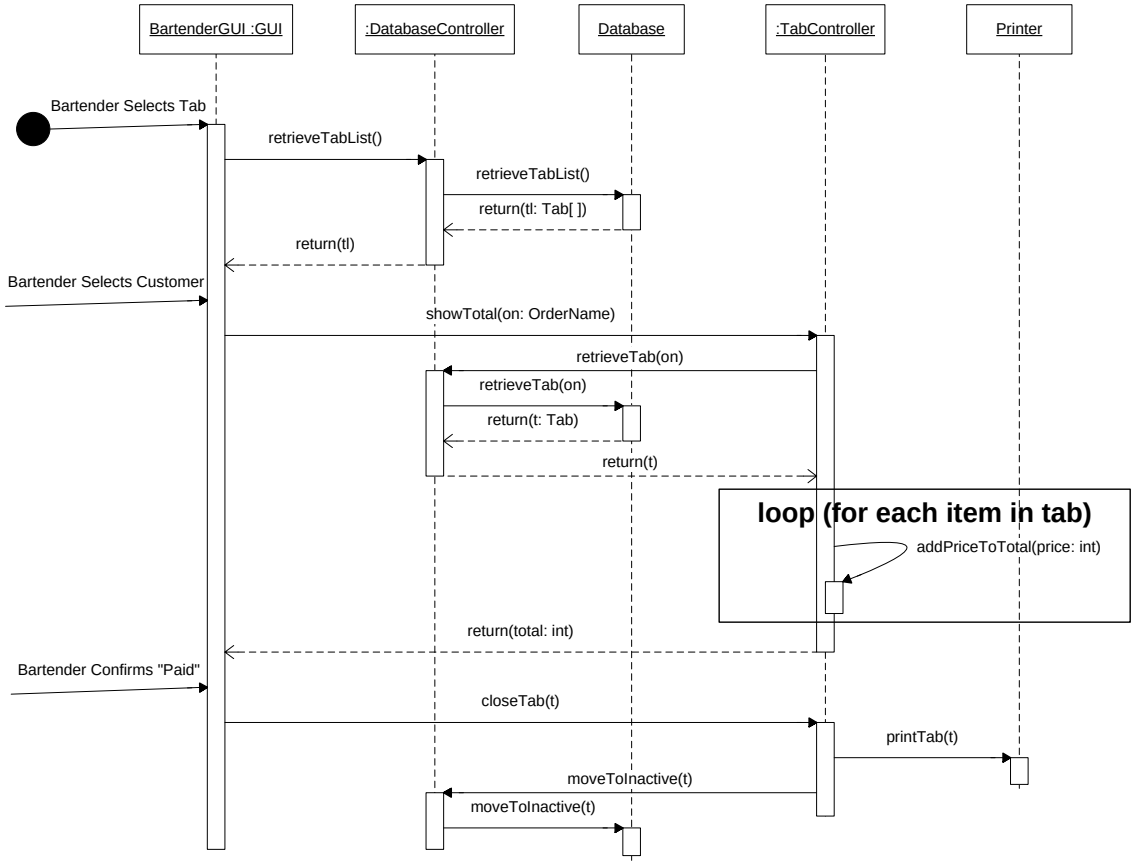
MarkItemReady

This case is very simple where the chef marks an item of the tab (Appetizer for example) ready as soon as they finish it to minimize turn around time. The item is marked ready in the database and upon the next refresh the Communicator sends a notification to the Server to come pick up the item. This design follows the ***Expert Doer Principle*** where the chef knows about the prepared item then stores it in the database and our database handles the communiation. This is Indirect communication and that follows the **Publisher-Subscriber** design pattern. And based on this case along with UC-25: AddItemAtTable we eliminated the role of the server to only serving the food when its ready and this way the communication is between customer straight to the kitchen and that follows the ***Low Coupling Principle***. The workload is balanced among all the actors interacting with our system (Customer, Server, Chef and Bartender) where each of them only gets to interact with the system only when they have to. So the total design follows the three principles (***Expert Doer Principle, Low Coupling Principle and High Cohesion Principle***) very
closely.


ChefGUI :GUI
t:Timer
:DatabaseController
Database
Chef Chooses Item From Order
markReady()
notifyServer()
startTimer(orderID, itemID: int)
loop
(while status!=delivered && t.isExpired()==false)
notifyServer
checkItemStatus(orderID, itemID)
checkItemStatus(orderID, itemID)
return(status: bool)
return(status)
markDelivered()
restartTimer()
alt
(status!=delivered)
updateInterface()

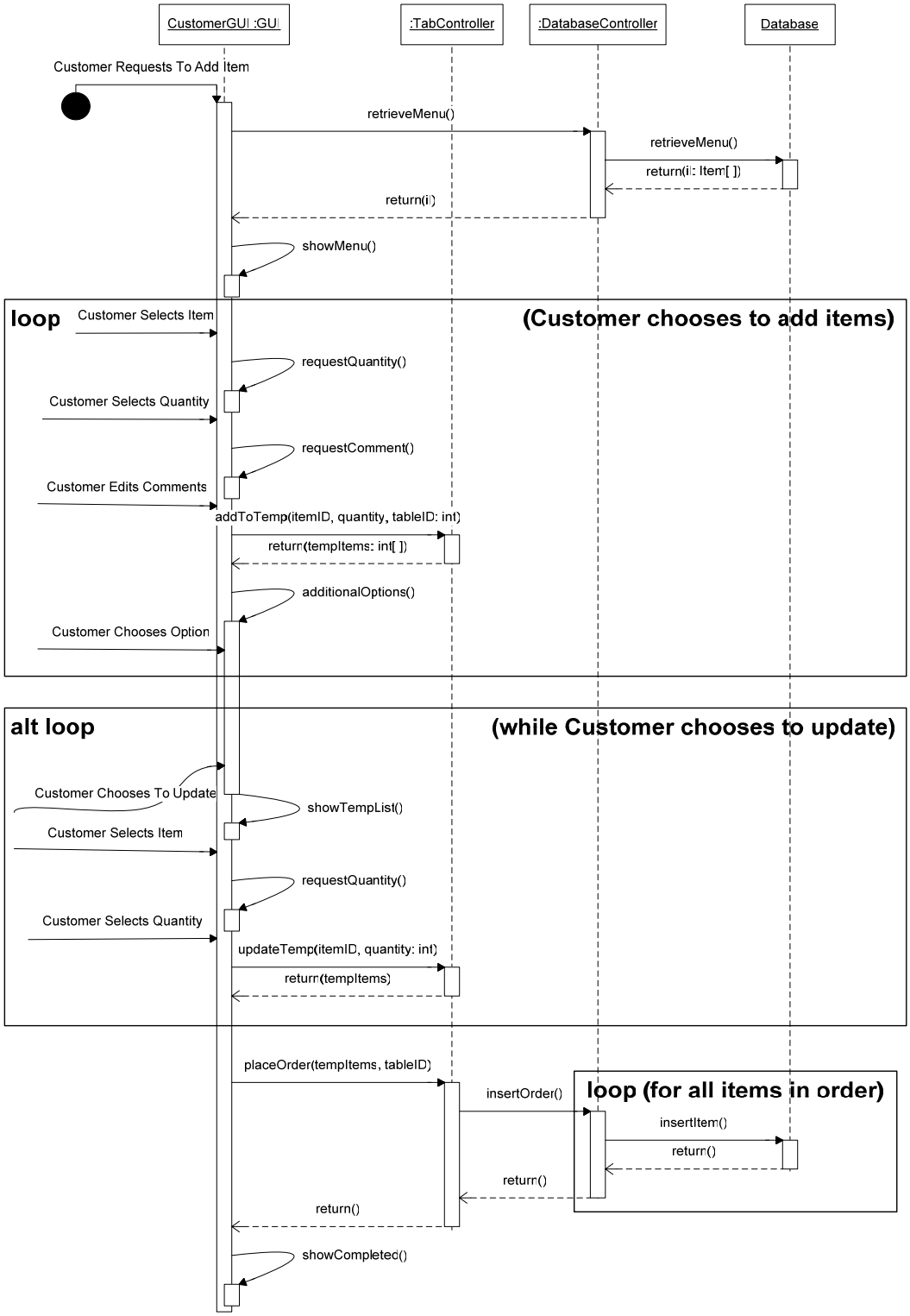
CloseTabAtBar

CloseTabAtBar is another Use Case that goes along with our **Publisher-Subscriber** design pattern: where the BartenderGUI is the Publisher and the DatabaseController, we need to note that here we have a TabController and this controller handles getting the total amount due for the active tab, and when the tab is cleared the controller takes care of storing a copy at the inactive order database before destroying the item. This design satisfies the (***Expert Doer Principle, Low Coupling Principle and High Cohesion Principle***) very closely.



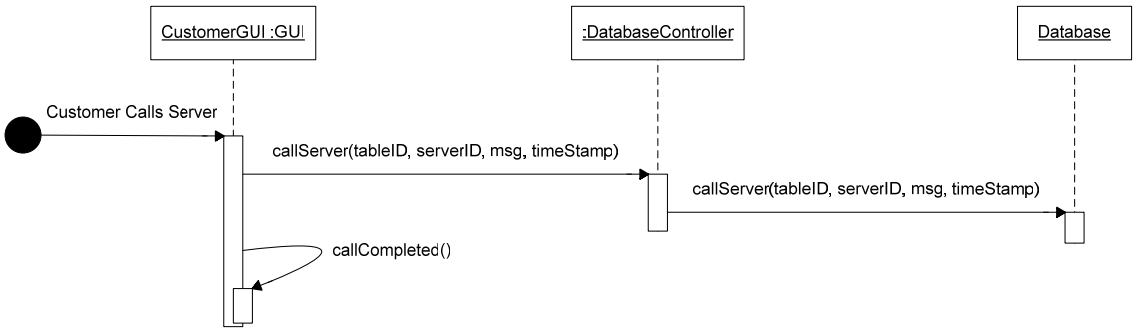
AddItemAtTable

In this case we have two main roles which are placing/modifying order and then storing them in the active database order for the selected table in order to get delivered to the Chef and/or Bartender next time their GUI gets refreshed. The customer performs the first task through the interface provided and the interface interacts directly with the menu stored at the database the Customer has the option of adding multiple item updating number of items(zero for removing the item from the order. Then, once the order has been placed the CustomerGUI stores the order in the active database for this table using the Database Controller. This design follows the ***Expert Doer Principle*** and ***Low Coupling Principle*** since only the users need to perform the task are the ones doing it (Customer and Menu), (Order and Database) or (Database and Chef/Bartender). This design also follows the ***High Cohesion Principle*** where the workload is balanced among all the objects (Customer, Database). Also we have the system taking care of dividing the order into food and/or drinks automatically. This is an Indirect communication and that follows the **Publisher-Subscriber** design pattern.


CustomerGUI :GUI
:TabController
:DatabaseController
Database
Customer Requests To Add Item
retrieveMenu()
retrieveMenu()
return(ii: Item[ ])
return(ii)
showMenu()
loop
Customer Selects Item
(Customer chooses to add items)
requestQuantity()
Customer Selects Quantity
requestComment()
Customer Edits Comments
addToTemp(itemID, quantity, tableID: int)
return(tempItems: int[ ])
additionalOptions()
Customer Chooses Option
alt loop
(while Customer chooses to update)
Customer Chooses To Update
showTempList()
Customer Selects Item
requestQuantity()
Customer Selects Quantity
updateTemp(itemID, quantity: int)
return(tempItems)
placeOrder(tempItems, tableID)
insertOrder()
loop (for all items in order)
insertItem()
return()
return()
return()
showCompleted()

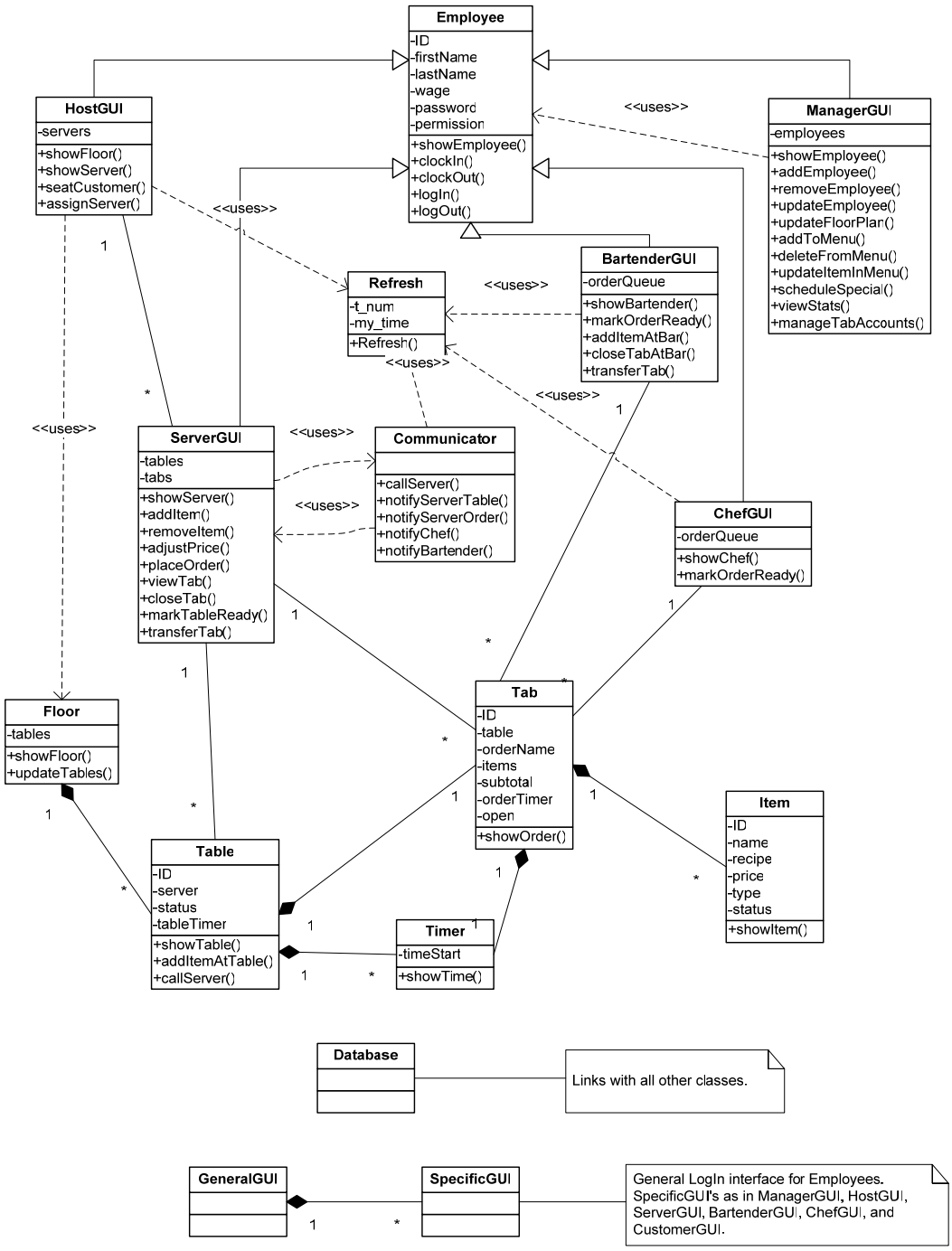
CallServer

This case is fairly easy and it is extremely important as it facilitates for the Customer to Call a Server whenever needed. The CustomerGUI directly sends a message to the DatabaseController. The message is stored with the tableID and the assigned server for this table. The rest of the call gets handled by the Refresh function for the Communicator, which retrieves the message and sends it over HTTP to the assigned Server; it follows the ***Expert Doer Principle.*** Only the communicator needs to know so it does the job. This is design is devided into 2 parts the CustomerGUI to the DatabaseController is an Indirect communication that follows the **Publisher-Subscriber** design pattern, while the rest of the scenario(Not listed in the case Diagram but will be implemented in the design) follows the **Command** design pattern.

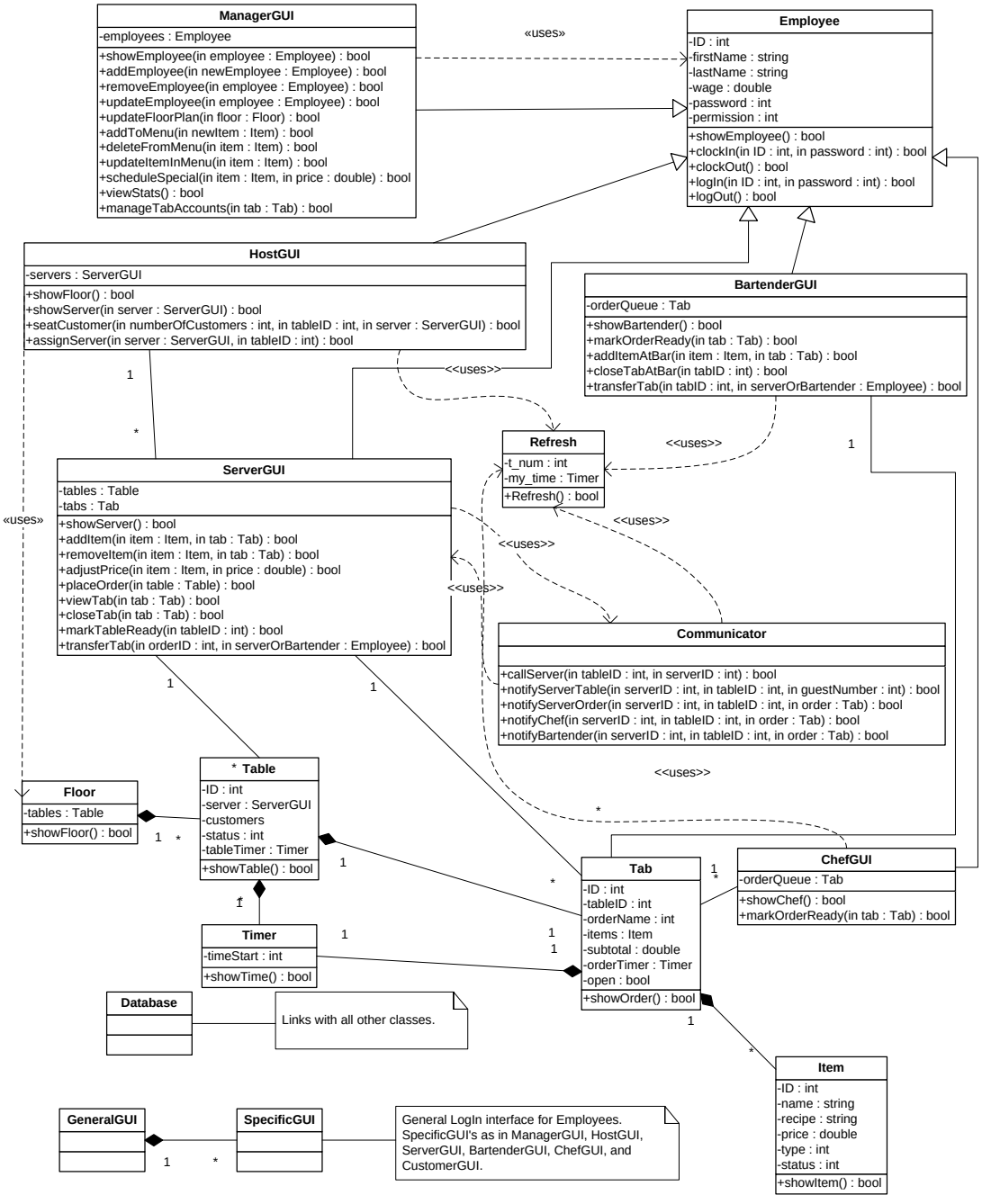


10. Class Diagram and Interface Specification:

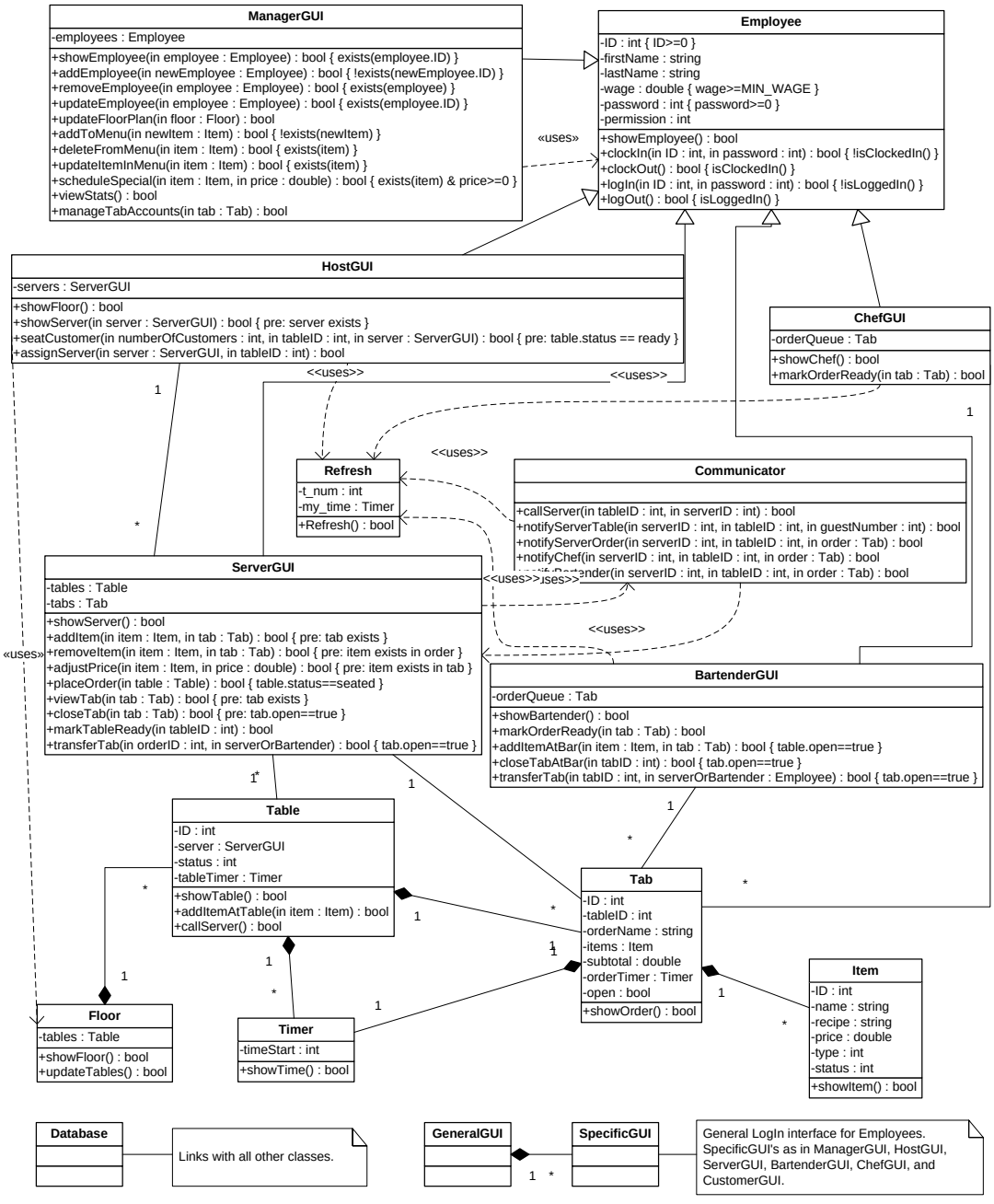
Class Diagram:



Data Types and Operation



Object Constraint Language Contracts



Design Principles

Since our system is a **Four-tier** architectural style we tried in our design to enforce the ***low coupling principle,*** and we do that by having our data stored in a database on the server, but the user interfaces have no direct communication with the database, instead they communicate with a databaseController, this provides low coupling because it sets the database implementation free (when changing the way we implement that database, we only need to change the databaseController, and all our interfaces remain unchanged)
A special case of our interface is the Server interface since this interface needs to communicate with the database and also with the other interfaces we implemented a Communicator, this Communicator functions to provide an interface to the system that will communicate with the databaseCommunicator, and when any of the other interfaces need to communicate with the Server they store a message on the database, then the Communicator accessing the databaseCommunicator extracts these messages and then sends them to the specified Server over HTTP. This design provides ***low coupling*** and ***high cohesion*** at the same time because each subsystem or interface has many objects that depend only on each other (***high cohesion***) and at the same time our system provides low coupling, even though the Server needs to interact with other interfaces low coupling is maintained by using the Communicator. Also our design follows the ***Expert Doer Principle*** by making sure only the user interface that needs to know about certain information is the only one that accesses the database to get this information.
For the **Design Patterns** we implemented couple of design patterns for different scenarios. For example in our Server interface we designed our classes to be completely reusable and also they are very independent, and all you need is to instantiate new instances of classes for your objects for them to inherit any of the main classes or methods, this follows the **Publisher-Subscriber** pattern. The **Publisher-Subscriber** pattern is also found in our design where it provides indirect communication between different users. Another pattern that is used is the **Command** pattern in operating across the Internet (the way the BlackBerry(Server interface) communicates with the database. Since it is difficult to employ remote method invocation (RMI) here, instead, the call is made from the BlackBerry by pointing the browser to the file containing the servlet. The

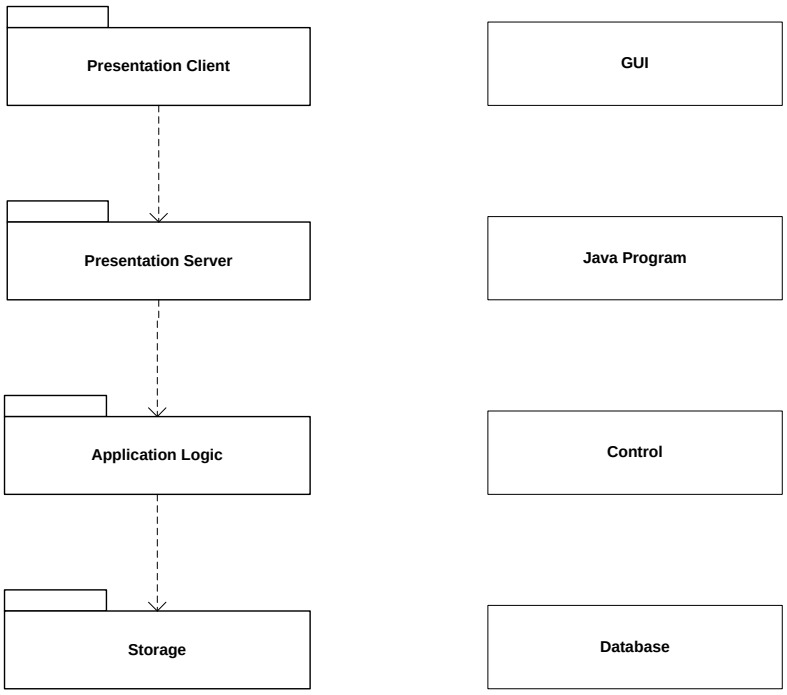
servlet then calls its service(HttpServletRequest, HttpServletResponse) method. The object HttpServletRequest includes all the information that a method invocation requires, such as argument values, obtained from the “environment” variables at standardized global locations. The object HttpServletResponse carries the result of invoking service(). This technique embodies the basic idea of the **Command** design pattern.

11. System Architecture and System Design:

Architectural Styles

The Four-tier Architecture will be used in the design of the system. In general, N-tier architecture divides computer hardware and software resources into multiple logical layers, where N is any natural number. Because each layer can be dealt with independently of other layers, the result is increased flexibility in developing, managing, and deploying a system. Developers only have to modify or add specific layers rather than disturb the entire system if technologies change, use scales up, or problems occur. This modularized architecture will also assist in dividing the workload amongst team members.

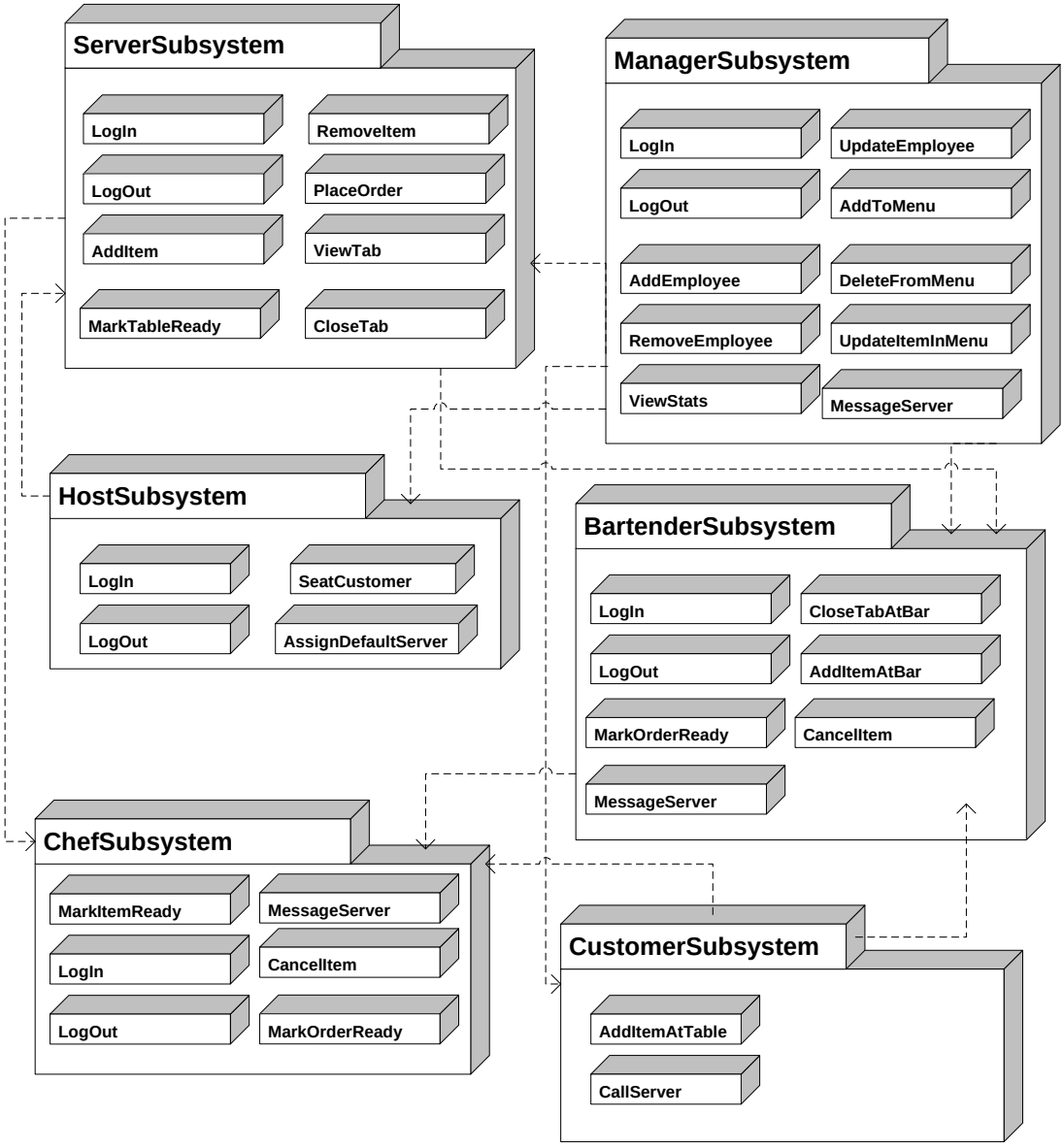
Our system will utilize this architecture as follows:



Four-tier Architecture Style (UML Class Diagram)

Identifying Subsystems:

Subsystem Identification is to keep functionally related objects together. First assign the participating objects that have been identified in each use case to the subsystems. Then create a new subsystem to accommodate them to the subsystem that creates these objects. (See next page for UML diagram)



Subsystems Analysis to match all use cases

Mapping Subsystems to Hardware

The system will be running on multiple computers to segment the hardware into its separate subsystems and to provide access to all users effectively.

The list of all running subsystems:

- Host Terminal: This terminal will provide system access specific to the Host. A floor plan will be provided for efficient and organized seating of Customers. This terminal will be implemented with a desktop computer and a touch screen monitor.

- Server Terminal: A wireless handheld BlackBerry with a strategically designed user interface specific to this device will be used to provide access to the Server at any time. This interface will also be used for notification messages. Each Server will have his/her own handheld device for interfacing with the system.

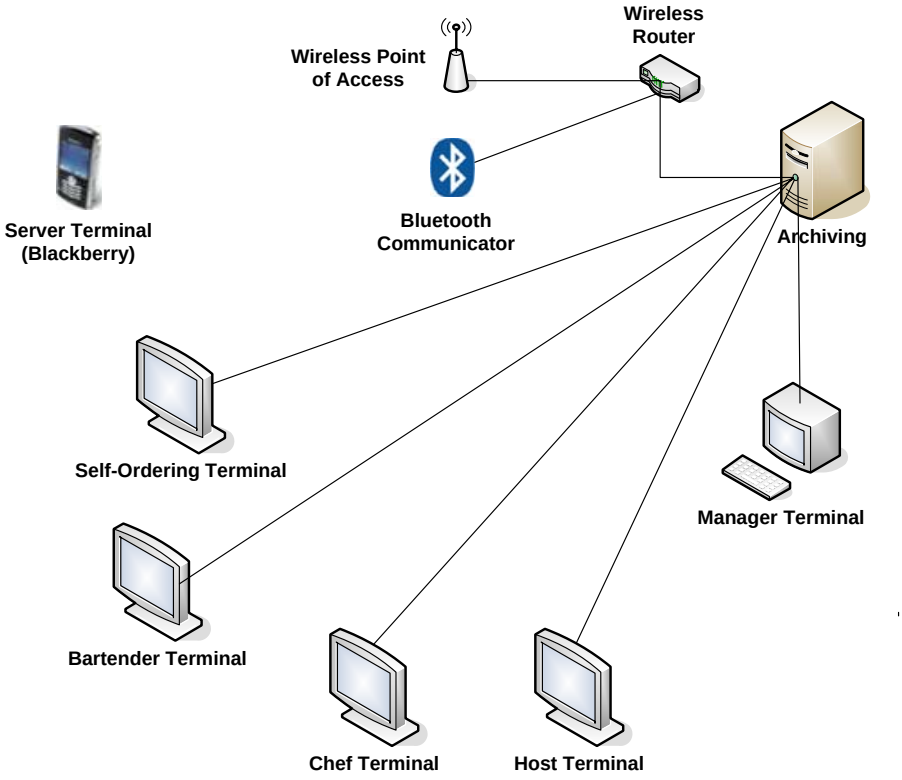
- Bartender Terminal: This terminal will provide system access specific to the Bartender. It will provide a menu system to facilitate ordering and notification of drinks ordered by Servers. This will be implemented on a desktop computer and a touch screen monitor.

- Customer Terminal: This terminal will provide system access directly to the Customer. It will provide a way for the Customer to order directly from his/her table or notify the Server when needed. This will be implemented on a desktop computer and a touch screen able to be stood up and presented at the table. This terminal will have a distinct look-and-feel aspect to appeal to the Customer, maximizing both form and function, whereas the other terminals will only need to maximize function.

- **Manager Terminal:** This terminal will provide system access to the Manager. It will provide a means for the Managers to interact with the system. Due to the fact that the Manager will do more typing when entering users and menu items into the system, a keyboard and mouse will be provided at this terminal. A flat-panel monitor and desktop computer will be used. This computer can also serve as a general purpose computer for any use the Manager sees fit.

- **Archiving Terminal:** This subsystem will be the centralized server for the entire system. It will contain the database and a backup of all necessary system files. Database backup will occur daily and a backup copy of all necessary system files will be saved to disk. This subsystem will also provide a means of communication between all the other subsystems. It will be implemented with just a desktop computer and has no need for a monitor, keyboard, or mouse. All changes will be done remotely from the manager terminal over the network.

All Terminals will be networked wirelessly over Wi-Fi, except for the BlackBerry which can communicate through either Bluetooth or wireless internet connection. Hard-wired Ethernet can also be used instead on Wi-Fi if distance or weak signal becomes an issue.



A simple diagram showing the hardware distribution of the system

Persistent Data Storage:

Persistent data storage is needed to store data necessary to outlive a single execution of the system. That is why some objects in the models need to be persistent. Where persistent objects provide clean separation points between subsystems with well-defined interfaces.

The system's persistent data storage will be realized with a database for multiple reasons:

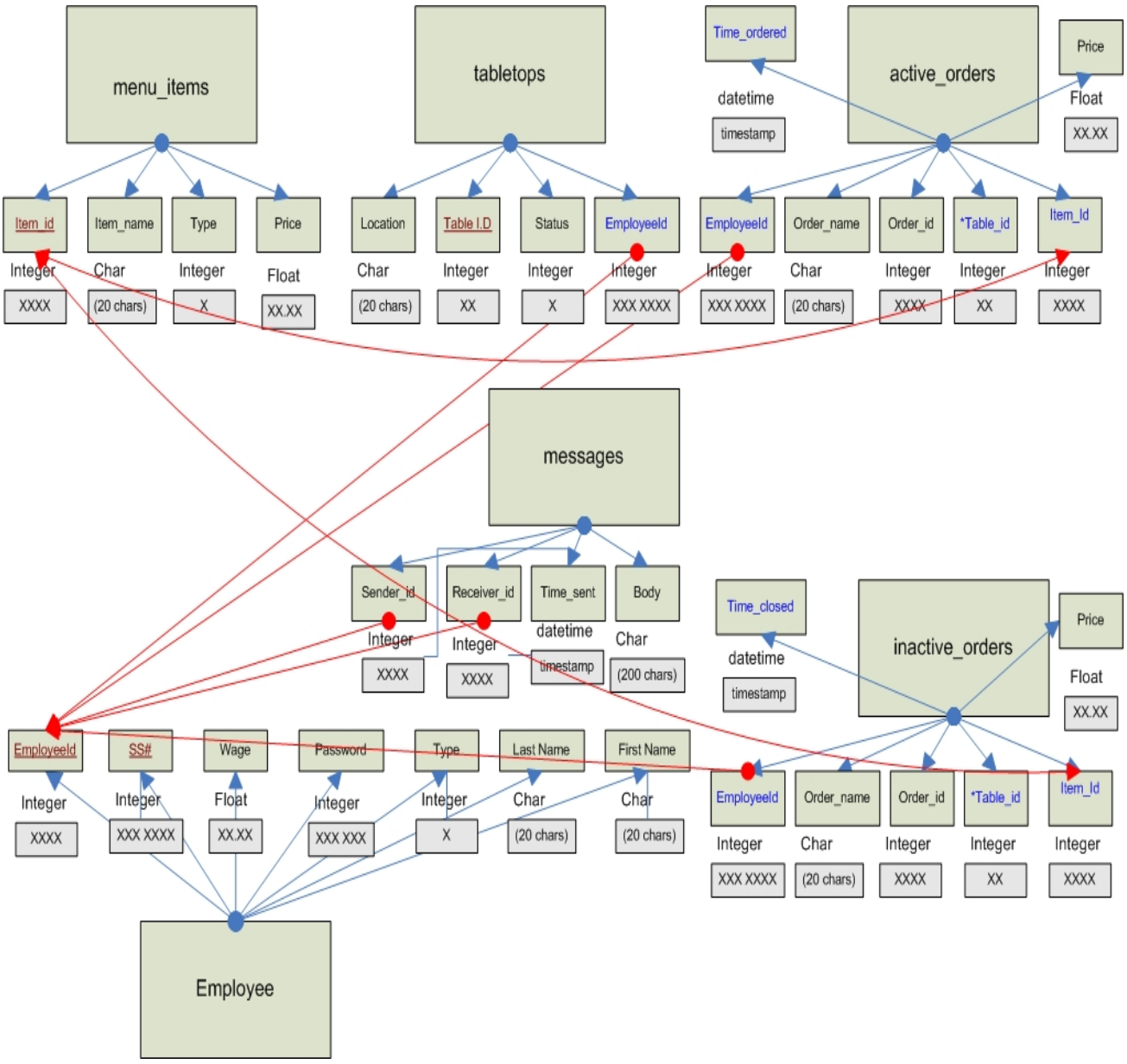
- Ability to hold large amounts of data
- Portability – data can be backed up or moved to another system
- Supports multiple writers and readers
- Provides interaction point between subsystems

Persistent data necessary for storage in the database:

- Items
- Employees
- Tables
- Order History

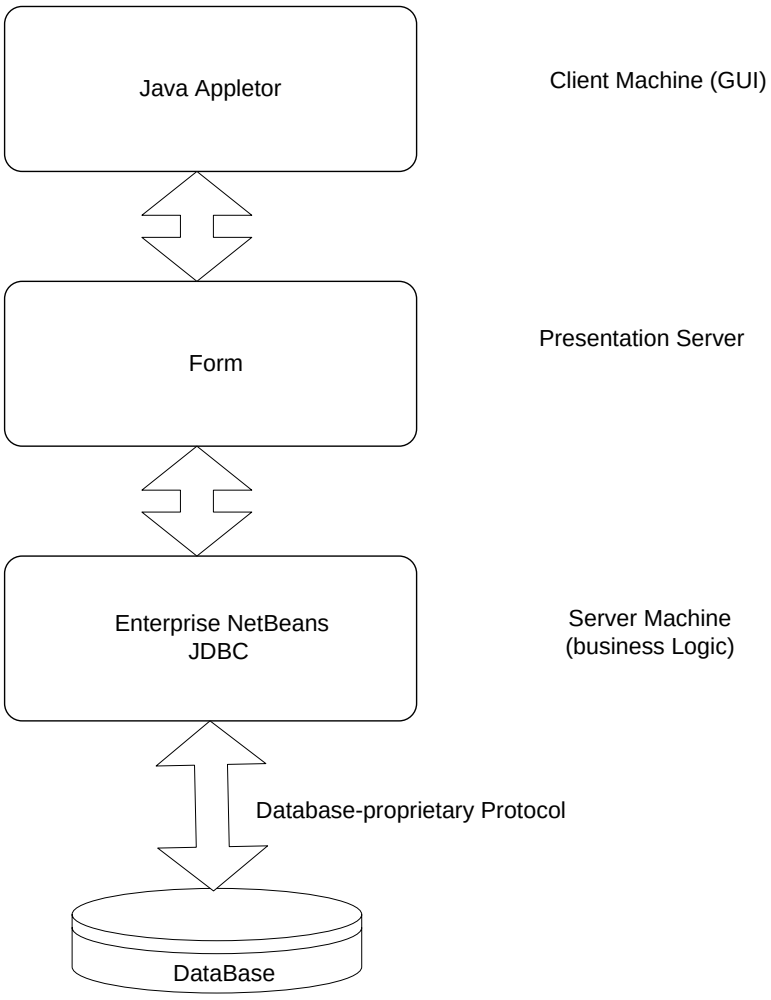
Executable files will be saved on the local hard disk at each terminal.
Database backup will also be saved to file on the archiving terminal's hard disk daily.

Database Schema:



Network Protocol:

The JDBC communication protocol will be used to access the database from within our Java program. JDBC is a Java API that enables Java programs to execute SQL statements. This allows Java programs to interact with any SQL-compliant database. Since nearly all relational database management systems (DBMSs) support SQL, and because Java itself runs on most platforms, JDBC makes it possible to write a single database application that can run on different platforms and interact with different DBMSs. (Illustrated diagram, see next page)



Four-Tier Architecture for Data communication

Global Control Flow:

- **Execution Orderness**: Our system will be an event-driven system. Actions can be generated by any user at any point in time. Once an action is generated the system will respond accordingly, managed by the control structure. Specific actions will follow a given procedure once they are generated. When no actions are being taken, the system will remain idle until user-interaction occurs. Also, multiple users can generate actions simultaneously and each action will be handled accordingly by the control structure.

- **Time Dependency**: Our system will contain multiple timers which will be implemented as a class of their own. The timers will be used to keep track of how long a table has gone without placing an order and also how long it has been since a specific order has been placed. After a certain amount of time without placing an order, the assigned Server will be notified to go check on the table. Keeping track of how long it has been since an order has been placed will assist the Chef with cooking times and will help in maximizing kitchen output by helping the Chef keep track of time more efficiently. Timers will also be used to refresh user interfaces based on current database information.

  The system is of event-response type due to the fact that the system will only need to react to input response from multiple interfaces. Event-Response Systems utilize a technique for specifying the syntax of multi-threaded dialogues. They can compactly represent the concurrency needed to implement multi-threaded dialogues. This concurrency support also allows interfaces to be structured differently than is possible with existing dialogue specification systems based on state transition specifications or grammars. This flexibility allows many interfaces, especially direct manipulation interfaces, to be specified with a more modular structure than most existing systems allow.

  This system has no concern for a real-time response. Simply, the response does not need to occur immediately. An order getting to the chef is not limited to the constraints of a real-time response. A delay of even a few seconds will not make a difference in functionality. Conceivably, an order could take a minute or 2 to get to the kitchen and would still not make a difference in functionality.

- **Concurrency:** The system uses multiple threads and also involves multiple subsystems running independently of each other. Interaction between the subsystems will be done either through the persistent data storage, the database, or through the control structure. A simple example of communication through the database would be as follows:
    - 1) Server places an order for 1 bottle of Budweiser.
    - 2) Database entry for Budweiser is updated.
    - 3) Bartender places an order for 2 bottles of Budweiser.
    - 4) Database entry for Budweiser is updated.
    - 5) Manager check sales analysis to see how much Budweiser has been sold.
    - 6) Database is queried and responds with a total of 3 bottles.

  As you can see from this example, no direct communication is done directly between subsystems. The communication is done through the database instead of communicating with the individual subsystems. This provides a centralized location for the data to be stored and provides a means for synchronizing data between subsystems. A simple example of communication through the control:

    - 1) Customer places an order for 1 Budweiser.
    - 2) Database entry for Budweiser is updated.
    - 3) Bartender is notified to fill the order.
    - 4) Bartender completes the order.
    - 5) Correct Server is notified to pick up the order.
    - 6) Server picks up the order and delivers to table.

  This example shows how separate subsystems will communicate through the Control Layer which will be realized by the Communicator class. The notification of both the Bartender and the Server is done by this layer and does not involve interaction with the database.

System Requirements:

- PC with Pentium® III 1.2 GHz or faster processor, Archiving Terminal will need 2.0 GHz or faster processor
- Microsoft® Windows XP Professional with Service Pack 2 or later, Windows Server 2003
- VGA (1024x768) or higher-resolution touch screen monitor
- Minimum 256 MB of RAM, Archiving terminal will need minimum 1 GB due to higher workload
- Approximately 3 GB of available hard disk space for the Point of Sale program files and the operating system. (Hard disk usage will vary, depending on Point of Sale's configuration and the size of the database), Archiving Terminal will need minimum 10GB free space
- CD-ROM or DVD drive
- Keyboard and Mouse or compatible pointing device
- Universal Serial Bus (USB) ports will be necessary
- Bluetooth Dongle
- Wireless Router, 2.4 GHz or above
- A Broadband Internet Connection
- Handheld BlackBerry device, BlackBerry 8100 recommended
- Receipt Printer: 1 USB port
- Credit Card Reader: 1 USB port

12. Algorithms and Data Structures:

Algorithms

The point of sale system most of the time provides help for the simple use cases between the actors and the system. These simple use cases only require relatively easy algorithms to provide adding, removing, updating items, or calculating totals of money generated in different time frames.

However when the administrator sees the need to monitor some statistical data to draw important business conclusions, then the algorithms involved to provide these functionalities become somewhat complex. The main types of complex algorithms used in our system are the search and the sort type.

Data Structures

Complex data structures are a necessity for an efficient software system. Our system makes full use of different complex data structures like arrays, linked lists, and hash tables based on the particular applications. Some applications require performance rather flexibility. For these types of applications we use arrays as our main data structure. Arrays are chosen because of their short processing time. Taking under consideration the use of mobile blackberry devices, which are limited in hardware performance, we tend to concentrate on the performance other than flexibility for almost all the functions involving these devices.

Hash tables on the other hand are the data structures that allow us to achieve both. The main reason is the ability to search on average with a constant-time O(1) just like arrays regardless of the number of items in the table. They also allow us to perform different searches over the same database tables giving us the flexibility to provide changes to our statistical data analysis at any time as might be needed by the client’s requests.

Linked lists are the other type of data structures that comes to a great help towards flexibility. We all are familiar with the advantage that linked lists provide in the dynamic use of memory. They also are the best way to implement the other data structures like queues and stacks. Queues are necessary to represent the food items waiting in the kitchen to be cooked and become ready for the server. Stacks on the other hand are very helpful in the user screen menu where the recursive process is required to come back to the initial screen. We acknowledge the possibility of using other data structures in this project but for right now our main functionalities involve arrays, linked lists and hash tables.

13. User Interface Design and Implementation:

Since the user will log-in before they are shown the main graphical user interface, the correct GUI can be displayed depending on the type of user that they have been set to in the system. For instance, a chef and a server would expect to see very different screens after logging in to the system. This will help to increase the efficiency of the system by allowing a separate interface to be built for each type of user to assist in performing specific tasks commonly used by each of these different user types. By doing this, the system can be navigated much quicker for the common tasks native to each user. The main screen can also be updated with commonly sold items to allow for even less time spent on menu navigation. Less time spent on user-interface navigation and more time spent on data entry will allow for a more efficient use of this Point-of-Sale interface.

This system also allows the Customer to interface directly with the system, without the need for a Server to take the order. The Customer will expect to see a GUI much like that of the Bartender, but possibly with more descriptive items and a flashier, eye-catching display. The Servers will also interface directly with the system through the use of a Blackberry smartphone. The GUIs for these devices will be designed to be more functional than eye-catching when compared to the tabletop displays for the customers. Both allowing the Customer to interface directly with the system and letting the Server carry their interface right on their belt or in their pocket will allow for very efficient ordering and delivery. The Chef's GUI will provide them with a current view of all unfinished orders, containing all the items included in these orders, as well as a means to call a Server at any time. And, the Host's GUI will provide the user with a graphical floor plan to enable them to seat customers easily and efficiently.

Some typical usage scenarios:

Bartender Scenario: Simple Cash-and-Carry Order

1) Tabbed menu groups will allow for navigation to the desired item to be ordered.
2) After choosing the item/items to be ordered, the "Send" button will be pressed to initiate the closing of the sale.
3) Total will be shown and cash given will be entered.
4) Cash drawer will open and the money will be entered and change will be given, thus completing the cash transaction.

Minimum # of clicks: 3

Host Scenario: Seating a Table

1) Table will be chosen from the on-screen 2D table view.
2) Number of Customers being seated will be entered.
3) Server will be notified that the table has been seated.

# of clicks: 2

Server Scenario: Placing Simple Table Order

1) Table will be chosen from the Tabs midlet.
2) After choosing the item/items to be ordered, the “Send” button will be pressed to update the table’s running tab.

Minimum # of clicks: 3

Server Scenario: Closing Table Order

1) The table will be chosen from list in the Tabs midlet.
2) The “Close” button will then be pressed to initiate the closing of the order.
3) Total will be shown and cash given will be entered.
4) The Server will then give the table the correct change and the transaction is complete.

# of clicks: 3

Customer Scenario: Ordering a Beer

1) Tabbed menu groups will allow for navigation to the desired item to be ordered.
2) After choosing the item to be ordered, the “Send” button will be pressed and the beer will be added to the table’s running tab.
3) The Bartender will be notified to make the drink and the Server will be notified to bring the drink to the table.

Minimum # of clicks: 2

Chef Scenario: Finishing an Order

1) Upon completion of the order, the chef will simply press the “Order Finished” button to notify the correct Server to come pick up their food.

# of clicks: 1

Administrator Scenario: Updating Item Cost

1) Select the Menu tab by clicking on it.
2) Navigate to the correct menu item using the tabbed menu groups.
3) Enter the new price and select whether tax is included in the price.
4) Click the “Update” button.

Minimum # of clicks: 3 + 1 text field

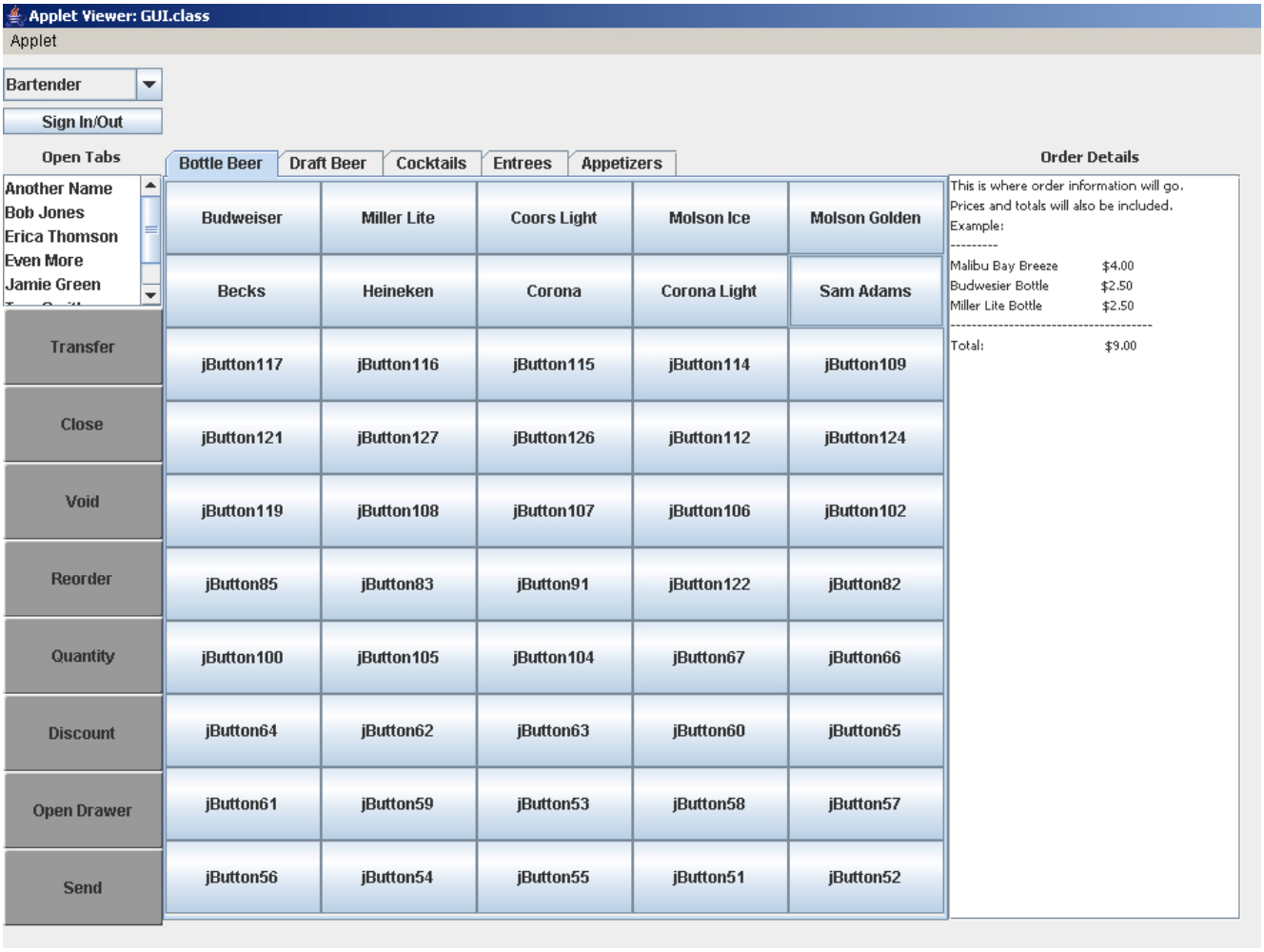
Administrator Scenario: Adding a New User

1) Enter new users First Name, Last Name, and Login ID.
2) Select the user type from the list of radio buttons.
3) Click the “Active User” checkbox to set the user as active.
4) The checkboxes to allow the user to Void and to Log Hours can also be set.
5) Then, click the “Add” button.

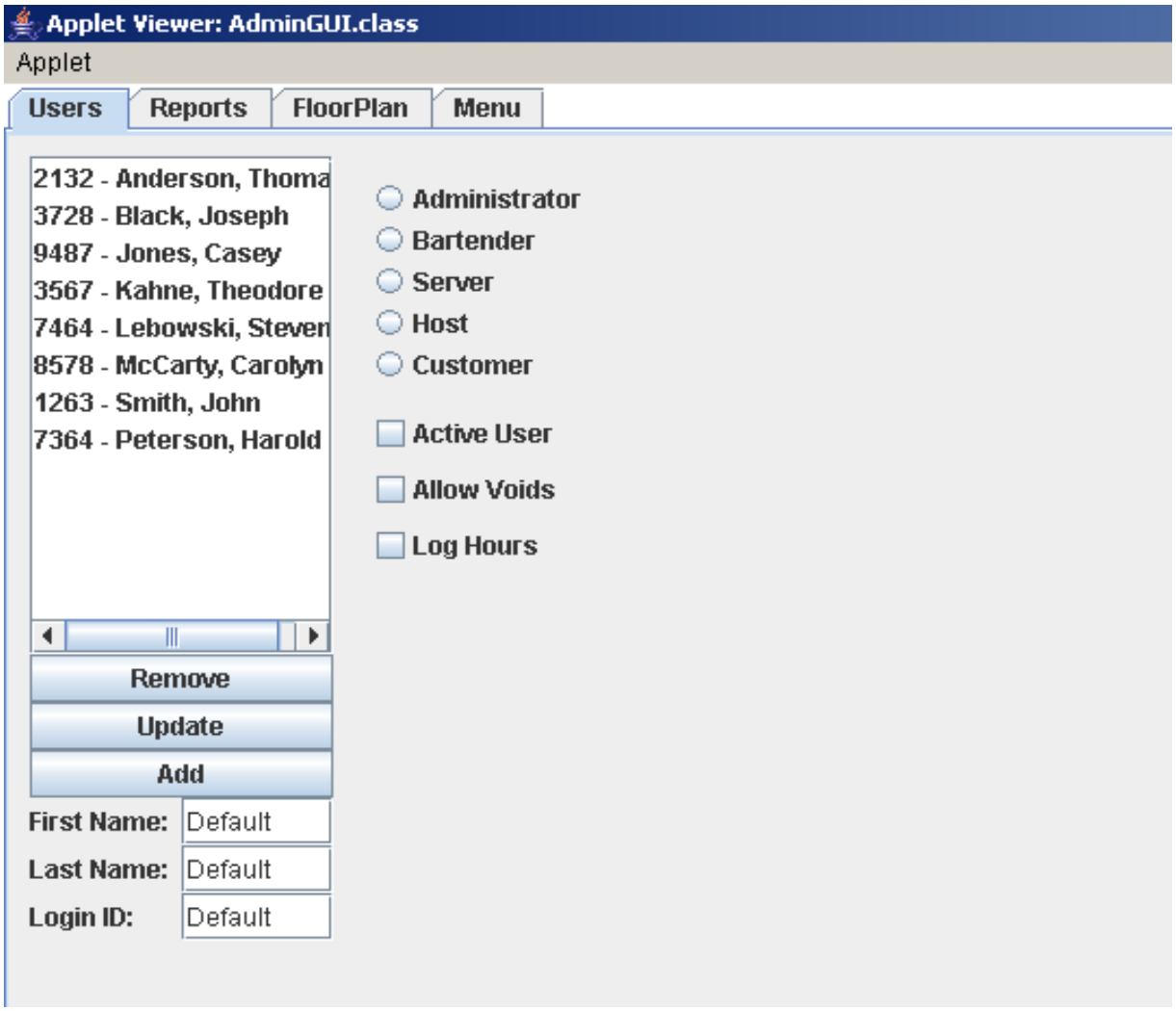
Minimum # of clicks: 3 + 3 text fields

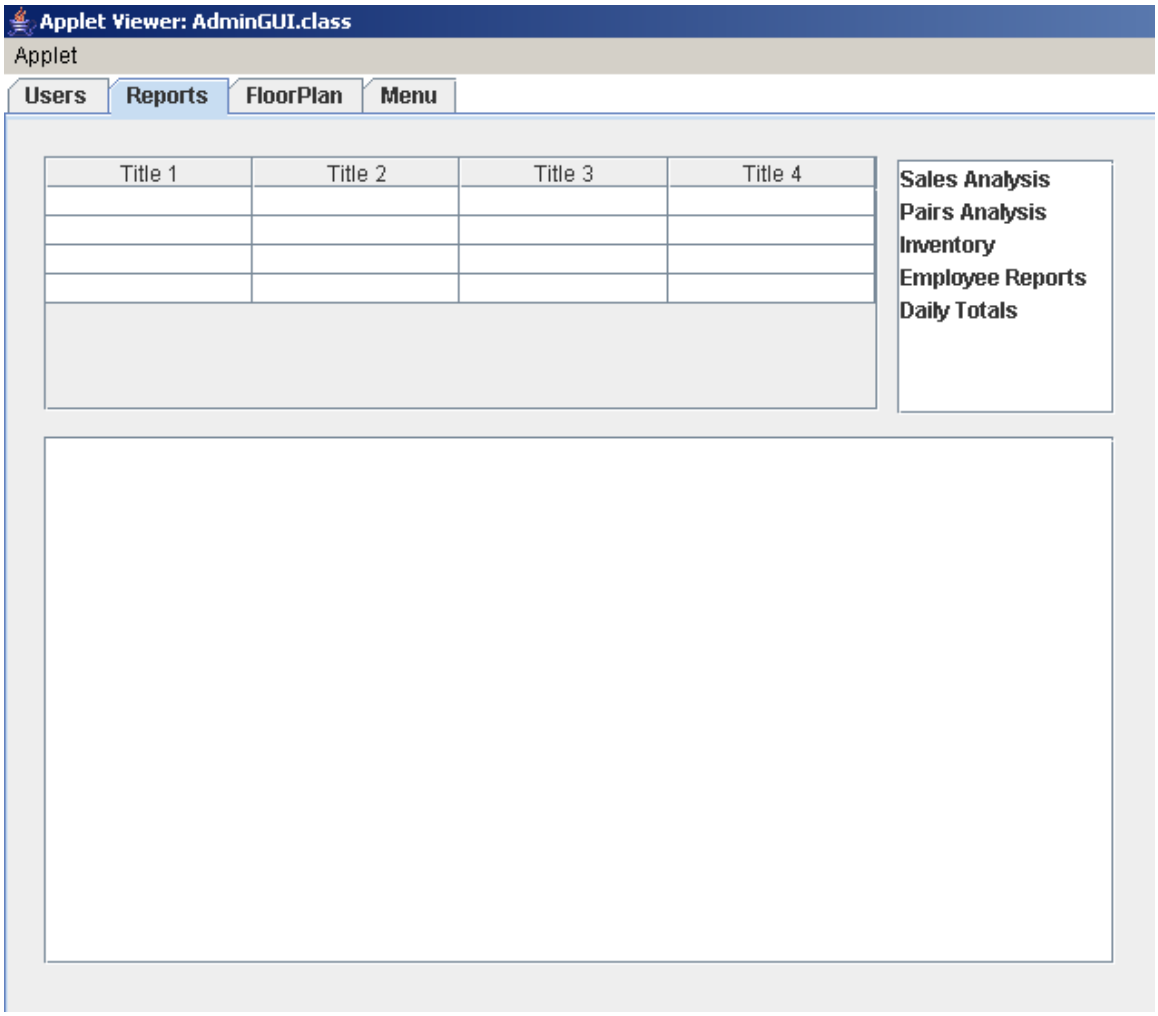
Example User Interfaces:

GUI for Bartenders and Customers:



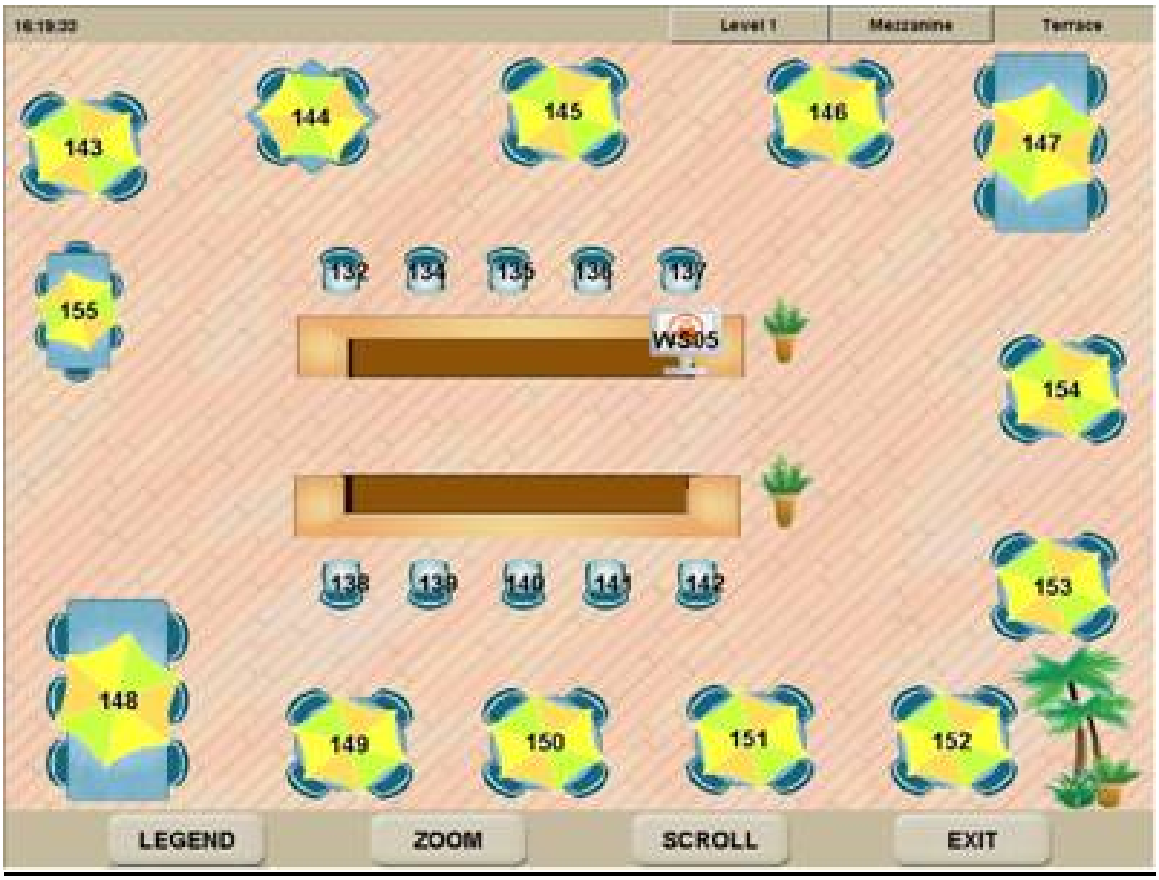
GUIs for Managers:



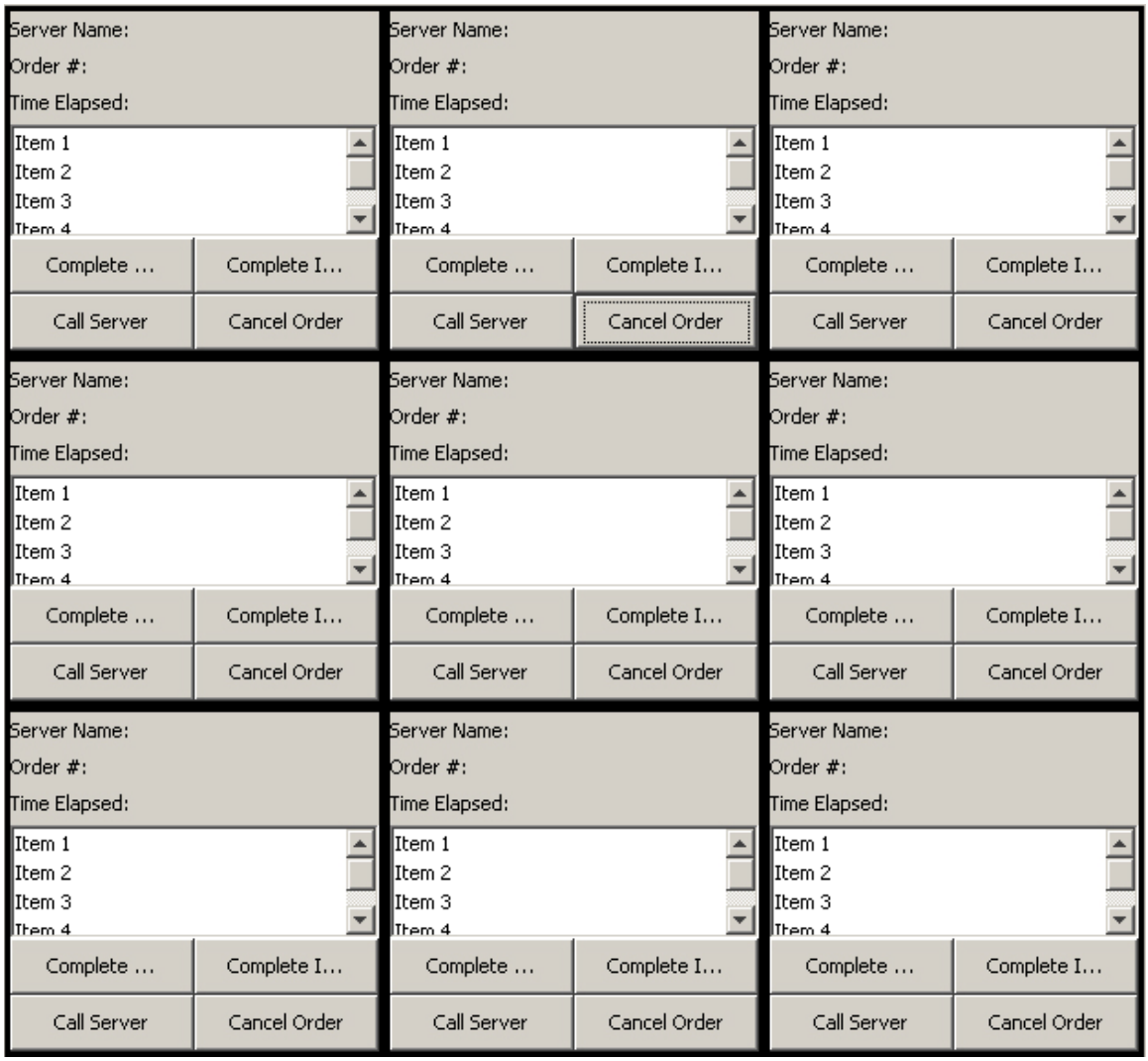

Applet Viewer: AdminGUI.class
Applet
Users
Reports
FloorPlan
Menu
Title 1
Title 2
Title 3
Title 4
Sales Analysis
Pairs Analysis
Inventory
Employee Reports
Daily Totals


Applet Viewer: AdminGUI.class
Applet
Users
Reports
FloorPlan
Menu
Bottle Beer
Draft Beer
Cocktails
Entrees
Appetizers
Budweiser
Miller Lite
Coors Light
Molson Ice
Molson Golden
Becks
Heineken
Corona
Corona Light
Sam Adams
jButton117
jButton116
jButton115
jButton114
jButton109
jButton121
jButton127
jButton126
jButton112
jButton124
jButton119
jButton108
jButton107
jButton106
jButton102
jButton85
jButton83
jButton91
jButton122
jButton82
jButton100
jButton105
jButton104
jButton67
jButton66
jButton64
jButton62
jButton63
jButton60
jButton65
jButton61
jButton59
jButton53
jButton58
jButton57
jButton56
jButton54
jButton55
jButton51
jButton52
Price:
Tax Included
Update

GUI for Host:


Level 1
Mezzanine
Terrace
143
144
145
146
147
155
132
134
135
136
137
WS05
154
138
139
140
141
142
153
148
149
150
151
152
LEGEND
ZOOM
SCROLL
EXIT

GUI for Chef:


Server Name:
Order #:
Time Elapsed:
Item 1
Item 2
Item 3
Item 4
Complete ...
Complete I...
Call Server
Cancel Order

The original screen mock ups were implemented using the Matisse GUI Builder for NetBeans 5.5. This is done using a WYSIWYG interface that automatically generates the corresponding Java source code. The Server GUI has been updated to show how it will look on the BlackBerry device. This UI will be implemented using the BlackBerry JDE and its contained custom UI package. Because of the small size of the BlackBerry device, more screens will need to be traversed to select a menu item. This will mean a customized menu system will be developed and popup menus will be utilized to switch between windows quickly. Notifications will be sent to the phone and cause a ringer or vibration, which each Server can customize to their own liking.

Example Main Screen with Icons for different tasks:

Focus will be shown by a change in the icon when rolled over.

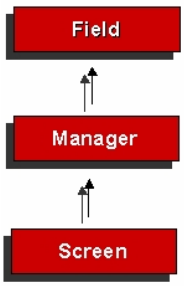
Example Blackberry GUIs for selecting menu items:

The Customer, Host, Chef, Bartender, and Manager graphical interfaces will be implemented using the Swing graphics package. The Swing package is contained in the Java Foundation Classes (JFC) and provides a set of buttons, panes, tables, and other common user interface components. This will provide the set of objects from which the non-BlackBerry GUI's will be built.

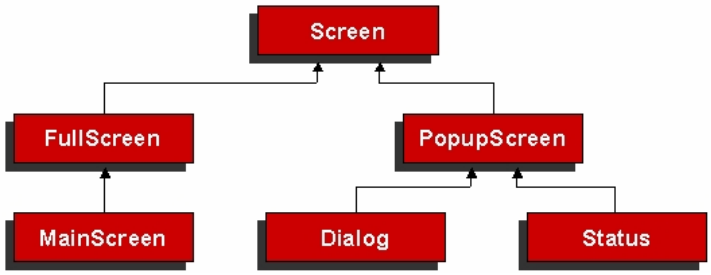
The BlackBerry JDE has a custom user interface (UI) application programming interface (API) built on a hierarchical structure similar to Java Swing. The most basic component of a UI is the field. Anything that can be output to the screen must be of type **Field**. The two other major components of a UI are the layout manager and the screen, which inherit from **Field** and **Manager** respectively.

Example BlackBerry GUI Layout

The BlackBerry GUI is a three level hierarchical structure in its most basic form:



Extensions of the Screen class available in the BlackBerry API set:



Ease-of-Use Analysis:

Customer:-

The **Customer**'s UI must be the simplest to use. No a priori knowledge of the system must be required for interaction with the system through this interface. This interface will concentrate on an eye-catching menu system with easy navigation and no need for learning. The reason for a flashy looking interface is to allure the **Customer** into using the system. Keep in mind that a reduced version of the full menu will be available on this interface to further facilitate ease-of-use. To place an order outside the domain of this reduced menu the **Customer** must use the "Call Server" button to request a server to come and take the order.

Chef:-

The **Chef**'s UI will be very simple to use and will require little to no training. The current orders will be listed in a queue. The orders will be listed in the sequence they are expected to be completed (some simple appetizer orders may take priority over entrée orders to increase turnover rate). Upon completion, the **Chef** will simply press the "Order Complete" button and the correct **Server** will be notified to pick up the order. The **Chef** will also be able to access recipes, retrieve past orders, and call a **Server**'s attention. These few simple tasks can be done very easily and efficiently.

Host:-

The **Host**'s UI will also be very simple to use and requires little to no training. A graphical floor plan will be shown will numbered tables. When seating a table, the **Host** will simply choose the table and enter the number of guests being seated. If necessary, the **Host** will be able to change the **Server** assigned to a given table.

Manager:-

The **Manager**'s UI will require only a small amount of training to use. Many of the **Manager** abilities are simple to use and the layout will help facilitate this ease-of-use. Although easy to use and to navigate, the **Manager** has many tasks available which is the reason for necessary training to fully utilize all the available functions.

Bartender:-

The **Bartender**'s UI will require a fair amount of training to increase speed and efficiency. To further increase ease-of-use, a tabbed menu system will be used. Each menu section can have its own tabs to break down the menu into specific sections. (Example: The entrée section could contain separate chicken, beef, fish, etc tabs, each with their own menu items.) This will make switching between item types much easier and much quicker. All necessary functions can be accessed from the main screen. Currently open tabs will also be listed and can be accessed directly from this screen.

Order details will also be listed on the screen and will contain a list of what the currently opened order contains, along with a subtotal.

Server:-

The **Server**'s UI will require the most training to use due to the fact that it will be accessed from a handheld device. Because of the smaller screen, a very different UI will be used. Navigation on this device will be slightly more involved due to the smaller screen, but after the initial learning curve it will greatly increase order efficiency. Multiple menu screens will be used to avoid cluttering on the BlackBerry interface. This will further increase efficiency and ease-of-use by reducing the number of items to choose from on each screen. Communication between the **Server** and other users in the system, including **Customers**, will also be greatly increased due to the fact that the **Server** can be notified or called at any time since they will be carrying this device with them at all times. The **Server** can modify the notification settings to choose a profile that fits them best. They can choose between vibration, ringers and tones, or a combination of ringers and vibration.

14. History of Work & Current Status of Implementation:

Grooup13 has started coding on February 12 of 2007 developing our first product “Open Bar” which it has a deployment day of May 3, 2007. It all started with the special interest of our team in using engineering strategies to develop a smart and mature Point of Sale software application. This application was designed to help any small or large size bar/restaurant owner and personnel to achieve their perfection in managing their limited resources.

High Velocity Bar & Grill has been our first client and has helped us through out the whole time with the many business meetings where we have discussed and analyzed their requirements. We have based most of our decisions on the main principles of Software Engineering and have followed the procedures of creating many important documents. The design principles and professional guidance that the software engineering course have offered to our team have made possible the delivery of this complex product in such a short time.

We list here the steps that we have been taking with the time frames corresponding to these steps, and the transitions between them:

1. We started with the establishment of a clear Client’s Statement of Requirements.

   (Feb 1, 2007)
2. From there we were able to build a better understanding about the main actors of this system and their functionalities. This was the key that open the door to our initial stages of coding. We started designing the User Interfaces and displayed mock up screens to our Clients.

   (Feb 12, 2007)
3. Functional Requirements and Nonfunctional Requirements were described and included in our first document of agreement with the client, where we had listed all the use cases and a use case diagram.

   (Feb 16, 2007)

4. Sequence diagrams were built to help with the Domain analysis and a new feature of using BlackBarry phones as mobile terminals for the waiters was added to our product. Group 13 was divided into two new development teams to manage the two different types of environments.

   (Feb 25, 2007)

5. System Architecture and System design were needed at this stage to be able to start our estimation of cost for this product. Hardware requirements were presented to the client in a second document of agreement. An SQL database was created for our system so functionalities could start being tested with both the BlackBarry phones and the regular PCs.

   (March 9, 2007)

6. Phase II of coding scheduled for March 15 was moved to a later date due to complications with the BlackBerry software which required a different connection to the database. Phase II involved combining and testing the different classes and also provide interaction between the different users. Focus remained on the first demo of our project.

   (March 20, 2007)

7. A few functionalities, like surfing through the menus of the BlackBerry and being able to generate dynamically the menu of items from the database were successfully displayed in our first demo. Interesting features were planned to be included in the future like notifying the waiter with a vibrating buzzer when action is needed.

   (March 23, 2007)

8. Group 13 continued to test on the different parts of the software, as well as progressing towards the goal of integration between the many modules of code. Great effort was dedicated to debug problems that were detected after the first demo.

   (April 15, 2007)

9. We created a website were we archived all the documents and all the files used trough out this project. We started preparations for our final demo and already created a brochure for advertising our product to potential new clients. Estimates for the final cost of our project were recalculated and revised.

   (April 20, 2007)

Summary of technical stages

- Used Net Beans 5.5 to build user interface for Bartender, Host and Chef.
- Installed My-SQL 5.5 and created a local database with a few tables to enable interaction between the user interfaces and the database.
- The database driver for SQL was installed in Net Beans which allows viewing the database tables in a GUI interface after establishing a connection.
- Different Net Beans projects were created developed independently.
- The most important classes were created and used over the different projects like Bartender, Host and Chief. These classes include the “connectDB”, “insertDB”, “refreshGUI”, “signIn”, “switchUser”, etc.
- Completed the GUI interfaces for the Server in BlackBerry environment.
- Established a connection to the database from the BlackBerry device.
- Completed all the User Interfaces and their functionalities.
- Designed the integration of the system and performed the integration.
- Tested the integrated system and did final adjustments. Prepared for final demo.
- Simulated a regular day of our product in action, for testing purposes before release.
- Electronically archived all project files and documentation in Group 13 website.

Current status of Implementation

Bartender User Interface and its functionalities:

We are currently working on the notifying messages between this user and other users as well as the closing an order or changing between orders. We are working on naming the orders by a name chosen by the customer rather than leaving them only with an order id.

Chef User Interface and its functionalities:

This user interface is in its initial stages but the work estimated to complete this user interface and its functionalities is corporately very small. We are also working on the notifying messages on this user as well.

Host User Interface and its functionalities:

Most of the Host Interface and the functionalities have been completed, however we are currently investing on having a dynamic floor plan where the administrator can update the floor plan due to some extension of tables or remodeling the store.

Server User Interface and its functionalities:

This User Interface is being implemented in BlackBerry phones and so far we have achieved the navigation between different screens but are currently working on the connection of this device to the database. Also the notification for messages which is desired to be done with a vibration ringer is still being worked on.

Manager User Interface and its functionalities:

We have designed all the functionalities but haven't starting implementing this User Interface yet as we think when most of our User Interfaces and their functionalities will be fully completed then enough information will be available to perform the Managers' functionalities.

Database Management:

Database management has been locally implement on each computer while we have been working independently, but now we are working on moving the database to the school's computers and accessing it via internet.

15. Conclusions and Future Work:

When Group 13 decided to create a point of sale system for restaurants we first envisioned it to be done with different computer programming languages such as C++ and Java. The program that we would build would then be installed separately on each computer. These individual computers would be directly connected to one another, allowing for data transfer through open sockets and ports. We also thought the entire system would be small, approximately 3 computers large, but quickly learned that this was not the case and many computers would be needed as we decided to extend our product for Bars & Restaurants. Knowing this we realized that a direct connection between all the computers would be extremely difficult. After much thought and research we decided to implement a "terminal-database" model with a SQL database and network connections to it by the different terminals. Our decision was also based in the different techniques that we learned through out this software engineering course. This "terminal-database" model allows all computers, or "terminals" to be connected to a single server, in essence lowering coupling. Communication would be much easier because all the most updated data would be stored on the database.

We also encountered other challenges when we made this decision. The innovated idea of providing servers with mobile terminals brought up the issue of having different types of connection to our database. The BlackBerry phones which used a totally different approach to connect to the database were suspected to be slower than the regular network connection chosen for the rest of the users. Testing this issue was extremely time consuming and virtually impossible.

The product that Group 13 had planned to develop initially had in mind the small Restaurants and the help that they needed to follow the easiest transition from the old fashion way of pen and paper to the new era of technology were the electronic data is used to offer more secure, more accurate and more flexible transactions. To achieve this transition we were aware that it would require a complex Software Product and maybe we didn't have the right level of experience. So we started analyzing our past experiences.

When given a coding assignment in the past, most of us would usually begin by just coding. With short preparation and no outlining this was often difficult and the desired result would get harder to implement as the size grew. We would often begin by coding what we believed was the correct software and alter and change the code as we needed. For simple programs this would work. Once the programs became more difficult, many problems would arise. For example, with no "blueprints," or preparation, an unexpected problem may be too difficult to resolve on and the only solution is to begin coding all over again. For this project we took a different approach, using methods we learned in the Intro to Software Engineering class.

In Intro to Software Engineering we were taught various techniques to develop software. One of these methods was UML or Unified Modeling Language. This is a developing standard that helped us to create all the small pieces of our software system. Once we constructed all the small parts, we were able to connect them and see how they would interact with each other before the actually coding. This allowed us to analyze every small piece of the software design, isolate problems and fix them without having to revisit the entire program.

We used the UML knowledge we learned in class to create use cases, system sequence diagrams, class diagrams as well as many other things. By doing this we were able to "blue print" the entire software project. This made it easy to code the final program. All we had to do is put all the little pieces together to make one working piece of software. This technique of software engineering also allows for simple extensions or updates in the program. Since the layout is devised a certain way, addictions would not need a restructure of the software.

When we began this course most of us had a deep understanding of C++. We have all taken courses in this language and understand it thoroughly. When faced with this project, we learned that C++ would not be enough. We began learning Java and became familiar with Net Beans. We also had to become familiar with BlackBerry software and learn how to query in SQL. Our final project would not meet the standards we were looking for if we weren't able to use of these other programming languages. Much of these programming languages are closely related and easy to understand, especially with the C++ programming methodologies learned in previous classes.

The conclusions that we are able to draw from our experience in this project can be summarized in saying that it would have been impossible to complete our product if we didn't use the principles of Software Engineering learned in this course. Not only we developed a better understanding in software development but we also realized that breaking down the project in subsystems and then reattaching them allows for future features to be added.

Keeping this in mind we plan to implement a few other features to our Point Of Sale product. One of the features involves the option for customers to order from a computer at home and pick up their order or have it delivered to them. Looking at the same feature but thinking of the use that we made of the BlackBerry phones we can easily supply a customer interface where a customer that owns a BlackBerry phone can order from anywhere. Perhaps this would offer the luxury of ordering the food on the way home from work and make a quick stop to pick it up, and everything is going to be ready exactly when the customer gets there.

Another feature that we plan to implement in the future is also the customers memberships where a customer will have an account and they are going to be provided with a pin that they can enter anytime they order, being online or at the restaurant or bar. Discounts will be offered to the customer for joining this membership for the fact being that this will reduce the employees work on taking down a payment and asking for signature.

Group 13 has been proud to work on this project and looks forward to our next project.

16. References:

Software Engineering, Ivan Marsic, Department of Electrical and Computer Engineering, Rutgers University:
http://www.caip.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf

Bernd Bruegge and Allen H. Dutoit:
Object-Oriented Software Engineering: Using UML, Patterns and Java, Second Edition, Prentice Hall, Upper Saddle River, NJ, 2004.
ISBN 0-13-0471100

Russ Miles and Kim Hamilton: *Learning UML 2.0,* Second Edition,
O'Reilly Media, Inc., Sebastopol, CA, 2006.
ISBN 0-596-00982-8

Java: The Complete Reference Seventh Edition by Herbert Schlidt

JSR 209: Advanced Graphics and User Interface Optional Package for the J2ME™ Platform:
http://jcp.org/en/jsr/detail?id=209

Trail: Creating a GUI with JFC/Swing:
http://java.sun.com/docs/books/tutorial/uiswing/index.html

Java: The Complete Reference Seventh Edition by Herbert Schlidt
Matisse GUI Builder for Net-Beans:
http://www.netbeans.org/kb/55/quickstart-gui.html

Blackberry Java Development Environment Fundamentals Guide:
http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/8067/645045

/8655/8656/1271077/BlackBerry_Java_Development_Environment_Fundamentals_Guide.pdf?nodeid=1271322&vernum=0

Desktop API Reference Guide:
http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/8067/645045/8655/336229/1195268/_Desktop_API_Reference_Guide.pdf?nodeid=1195442&vernum=0

Blackberry Graphical User Interface Part 1:
http://na.blackberry.com/eng/developers/resources/journals/jul_2005/BlackBerryDeveloperJournal-0202.pdf (p27-32)

Blackberry Developer - Getting Started:
http://blackberryforums.pinstack.com/540-blackberry_developer_geting_started.html

Blackberry Developer Journal:
http://www.blackberry.com/developers/journal/

What Is - Blackberry UI hierarchy:
http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/348583/796557/800332/800505/800608/What_Is_-_BlackBerry_UI_hierarchy.html?nodeid=800513&vernum=0

A Four-Tier Model of A Web-Based Book-Buying System:
http://web.mit.edu/6.826/www/notes/6826-project.pdf

JDBC Introduction:
http://java.sun.com/docs/books/tutorial/jdbc/overview/index.html

What is JDBC?:
http://www.webopedia.com/TERM/J/JDBC.html

Java Database Connectivity:
http://en.wikipedia.org/wiki/Java_Database_Connectivity

CS 4773 Object Oriented Systems
Threads and
Synchronization:http://vip.cs.utsa.edu/classes/cs4773s2000/notes/cs4773.topic08.html
Silverware POS Software and User Manuals, Forsys Corporation:
http://www.forsyscorp.com