

Concepts in Software Engineering

14:332:452

Restaurant Automation Project

Report 3 – System Design

04/27/07

Project Advisor

Prof. Marsic

TA: Jian Zhang

Group # 8 Project Members

Chris Eng

Sagar Mansukhani

Prakriti Gautam

Hitesh Ved

Nikethan Yerabaka

Srihitha Yerabaka

Individual Contributions

All team members contributed equally.

Table of Contents

3. Summary of Changes.....	1
4. Customer Statement of Requirements.....	2
5. Glossary of Terms.....	4
6. Functional Requirements Specification.....	5
7. Nonfunctional Requirements.....	45
8. Domain Analysis.....	47
9. Interaction Diagrams.....	53
10. Class Diagram and Interface Specifiaction.....	55
11. System Architecture and System Design.....	55
12. Algorithms and Data Structures.....	66
13. User Interface Design and Implementation.....	44
14. History of Work & Current Status of Implementation.....	99
15. Conclusion and Future Work.....	101
16. References.....	101

3. Summary of Changes

- Addition of new Use Cases in the Functional Requirements Section
 - o Elaborated on existing Use Cases
- Corrections of Nonfunctional Requirements
- Domain Analysis has been corrected to include the new Use Cases
- Interaction Diagrams have been updated and fixed. The new Use Cases have been diagramed.
- Addition of Design Patterns to existing Interaction Diagrams
- Class Diagrams updated to reflect all the new Classes
- Database Schema has been changed for the System Architecture and System Design section
- Classes have been mapped to their appropriate Subsystems
- User Interface pictures have been updated with current version
- Explanation of User Interface has been expanded upon and reasoning for certain choices has been made clearer

4. Customer Statement of Requirements

Aim:

The aim of this project is to develop a software system that would eliminate the need of traditional pen/paper approach for privately- owned restaurants. The project is focused on making the restaurant fully automated such that it is easier to co-ordinate various work activities that go on inside a typical restaurant. The main features of the project include:

- Organizing a database for a medium sized restaurant
- Coordinating work activities of the various actors – Host, Waiter, Cook, Busboy and Manager
- Increase efficiency by minimizing time between an order is placed and the billing
- Increase profits by reducing operating costs and increasing revenues by increasing efficiency
- Archiving information of the workers and hours worked

Problems with the primitive system:

The traditional pen paper approach has the following drawbacks:

- Keeping track of empty tables requires either keeping a “dry erase” diagram of tables or the host constantly keeps track of the status of the tables if it is a small restaurant
- The waiter jots the order on paper and has to transfer redundant information to the terminal system. This takes some time and reduces efficiency in peak hours of patron service, also there may not be enough terminals available in the restaurant as there are number of tables which would require particular waiters to wait until the others are done entering their orders into the systems
- The cooks could not notify the waiter that the food was ready
- Keeping billing and other statistical information was an issue of concern

Software Solution:

We propose a software solution to the above problems which would allow the restaurant management to be easier and offer more coordination for the everyday work. A touch screen will be used by the staff to log in and complete the desired task.

The supported employee roles are: Host, Waiter, Cook, Busboy and Manager. The various employees have user accounts and login using their passwords they need to remember except the cook. Logging in and out will be exploited as triggering events to update and organize the data.

When a person enters the restaurant the host will greet the customer and log in to see the tables that are free. The host can also show the floor status to the customer for their preference (e.g. if the customer prefers a free table near the window etc.). After being seated the assigned waiter for that particular table takes over from the host and takes the order from the customer on a PDA. The order is seen by the cooks in the kitchen who can right away start preparing the order. After the customer is done eating they are billed and the order is archived in the database for calculation of the restaurant revenues for that day/month/year. This also allows preparing easy statistics regarding high patron service hours etc. The Busboy who checks the table status can then take care of the dirty table and after he is done cleaning can mark them as ready to use in the system.

The manager has administrative power over employee profiles. They can do the following:

1. The ability to create and modify profiles
2. Track employee activities
3. Authorize restricted waiter activities.

We will take into account the number of clicks that are necessary to accomplish the individual tasks and try to minimize the number of clicks for efficient deployment of our system.

5. Glossary Of Terms Used

- **Manager** - Manages inventory, payroll, employee list and charts and statistics for the restaurant
- **Host** - Assigns and seats people who come to the restaurant
- **Waiter** – Takes the order from the customer onto a PDA and delivers the order to the customer
- **Cook** - Reads the order placed from a terminal in the kitchen and cooks food accordingly. Also the cook informs the waiter when the order is ready.
- **Busboy** – Keeps track of the dirty tables and updates status when he/she is done cleaning
- **Add/Edit Employee** – Button on the Management page to add or edit the information of an employee at the restaurant such as employee identification number, their password, employee type and their wages.
- **Manage Inventory** – Button on the Management page of all the items required for food preparation in the restaurant
- **Manage Payroll** – Button on the Management page to manage the payrolls of the various employees in the restaurant
- **Reports Screen** – statistical data analysis of the traffic flow in the restaurant
- **Grid** – GUI layout of the tables in the restaurant
- **Efficiency analysis** – Performance measure of all waiters in the restaurant

6. Functional Requirements Specification

Stakeholders

The following are the stakeholders in the application. They have a vested interest of some sort in the way that the application works. It is important to them that the application is easy to use.

- Owner of the restaurant
- Employees (Manager, Host, Waiters, Busboy, and Cook)
- Programmers

Actors and Goals

The table lists all the actors, their various goals and the related use cases.

Actor	Goal	Type	Use Case
Manager	Create employee account	Initiating	createAccount (UC-1)
Manager	Edit employee account	Initiating	editAccount (UC-2)
Manager	Delete employee account	Initiating	deleteAccount (UC-3)
Manager	Add new items to menu	Initiating	AddToMenu (UC-4)
Manager	Delete item from menu	Initiating	DeletefromMenu (UC-5)
Manager	Manage Payroll	Initiating	Paycheck (UC-6)
Manager	Graphical Analysis	Initiating	Graphs (UC-7)
Host	Seat customers and assign waiters to tables	Initiating	SeatCustomer (UC-8)
Cook	Prepare and cook the order	Initiating	PreparingOrder (UC-9)
Busboy	Clean and prep tables	Participating	clean(UC-10)
Waiter	Take orders	Initiating	Order (UC-11)
Waiter	Deliver order to table	Participating	Deliver (UC-12)
Waiter	Collect payment from customer	Initiating	PayBill (UC-13)
All employees	Login	Participating	Login(UC-14)

Manager	Managing Inventory	Initiating	addInventory(UC-15) UpdateInventory(UC-16)
---------	--------------------	------------	---

Use Cases

Casual Description

These are the various use cases that are contained within the program. A quick description of all the use cases follows below.

1. createAccount (UC – 1): Allows the manager to create new accounts for employees.
2. editAccount (UC -2): Allows the manager to edit existing accounts in the database.
3. deleteAccount(UC-3) : Allows the manager to delete existing accounts from the database for employees who are leaving the restaurant.
4. AddtoMenu(UC – 4): Allows the manager to add new items to the menu list for the restaurant.
5. DeletefromMenu(UC- 5): Allows the manager to delete items from the menu. When the manager wishes to discontinue an item in the restaurant he can delete that item from the list.
6. Paycheck(UC – 6): Allows the manager to prepare paychecks for the individual employees working in his restaurant based on the number of hours they worked during the week.
7. GraphicalAnalysis(UC- 7): Allows the manager accesses the database to create performance reports such as employee efficiency.
8. SeatCustomer (UC-8) - The Host selects a table for the customers and assigns a waiter to them.
9. PreparingOrder (UC-9) – The cook prepares the order after the waiter gives it to him and after the food is prepared, the cook changes the status of the order to ready, indicating to the waiter that it can be delivered. Also, each time a food item is prepared the inventory is updated directly. The system will know the names and quantity of the items used and will update the inventory.
10. clean (UC-10) – The Busboy cleans dirty tables and after they are clean, he changes the table to status to ready so that the host can assign it to new customers.
11. Order (UC-11) – The waiter takes the order from customers and gives it to the cook.
12. Deliver(UC-12) – After the waiter gets a ready signal from the cook indicating that the food is ready, he gets it from the kitchen and delivers it to the table.
13. PayBill(UC-13) – After the customers are finished with the food, the waiter gives them the bill and collects payment.
14. Login (UC – 14) – Shows how each employee logs in to the system and is taken to their respective homepage.
15. addInventory(UC-15) – The manager adds new items to the inventory.
16. UpdateInventory(UC-16) – The manager updates or deletes items from the inventory. If the quantity is specified as 0, the item is deleted. If the quantity is specified as a number other than 0, it is updated.

Fully-Dressed Description

Use Case UC-1	createAccount
Primary Actor	Manager
Actor's Goal	Add a new employee to the database
Stakeholders	Employee
Preconditions	There is a new employee to be added and is not already in the database.
Postconditions	The employee has been added to the database.
Main Success Scenario	
<p>> 1. Manager selects the interface to enter in a new employee and enters in the employee SSN, Wage, password, employee type, first name, last name.</p> <p>< 2. Validates the fields. System checks if the data entered is correct, complete, and not already in the system.</p> <p>< 3. Employee is entered into the system and informed of his ID and password.</p>	

Use Case UC-2	editAccount
Primary Actor	Manager
Actor's Goal	Edit existing employee account in the database
Stakeholders	Employee
Preconditions	An employee entry exists in the database for the particular employee whose information needs to be updated
Post conditions	The relevant employee information is updated
Main Success Scenario	
<p>> 1. Manager selects the interface to edit the existing employee account in the database which might require changing any of the following fields: SSN, Wage, password, employee type, first name, and last name.</p> <p>2. Validate the fields.</p> <p>3. Employee information gets updated.</p>	

Use Case UC-3	deleteAccount
Primary Actor	Manager
Actor's Goal	Delete employee from the database
Stakeholders	Employee
Preconditions	The employee account already exists
Post conditions	The employee has been deleted from the database.
Main Success Scenario	
<p>> 1. Manager selects the interface to delete the employee information of an existing worker.</p> <p>> 2. Manager selects employee from drop down menu.</p> <p>3. Delete employee from the system.</p>	

Use Case UC-4	AddtoMenu
Primary Actor	Manager
Actor's Goal	Add new items to the menu database
Stakeholders	
Preconditions	Menu database already exists.
Post conditions	New items are added to the existing menu.
Main Success Scenario	
<p>> 1. Manager selects the interface to enter in new item information into the menu database.</p> <p>2. Validates data.</p> <p>3. Item added to database.</p>	

Use Case UC-5	DeletefromMenu
Primary Actor	Manager
Actor's Goal	Delete items from the menu
Stakeholders	
Preconditions	The food item already exists in the menu.
Post conditions	The food item is deleted from the menu.
Main Success Scenario	
<p>> 1. Manager selects the interface to delete the food item.</p> <p>> 2. Manager selects food item from drop down menu.</p> <p>3. Food item is then deleted from the system.</p>	

Use Case UC-6	Paycheck
Primary Actor	Manager
Actor's Goal	Prepare weekly paychecks for the employees
Stakeholders	Employee
Preconditions	The employee information already exists in the system
Postconditions	The pay is calculated
Main Success Scenario	
<p>> 1. Manager selects interface.</p> <p>< 2. System returns employee hours, hourly wage, and total salary.</p> <p>3. Manager creates paycheck.</p>	

Use Case UC-7	GraphicalAnalysis
Primary Actor	Manager
Actor's Goal	Graph the performance charts
Stakeholders	Employee
Preconditions	Employee and Item information is collected
Post conditions	Graph generated
Main Success Scenario	
<p>> 1. Manager selects interface to view performance.</p> <p>> 2. Manger requests the daily/monthly/yearly performance of employees, sales reports and the popularity of food items.</p> <p>3. System generates graphs.</p> <p>< 4. Returns performance graphs to manager.</p>	

Use Case UC-8	SeatCustomer
Primary Actor	Host
Actor's Goal	Seat a customer at a clean empty table and assign a waiter to that table
Stakeholders	Employee
Preconditions	There is a new customer who hasn't been seated yet. A waiter hasn't been assigned to that table yet.
Postconditions	Customer is seated and a waiter is assigned to the table.
Main Success Scenario	
<ul style="list-style-type: none"> > 1. Host selects the view table interface, and selects a table. > 2. Host assigns waiter to table. 3. System changes table status to occupied. 	

Use Case UC-9	PreparingOrder
Primary Actor	Cook
Actor's Goal	Prepare food for the order and notify the waiter when it is ready
Stakeholders	Waiter
Preconditions	An order has been made by a customer and the waiter has brought it to the cook.
Postconditions	The waiter is notified when the food is ready to be delivered to the table. Also each time a fooditem is prepared, the corresponding inventory items are decreased by the pre-specified quantity.
Main Success Scenario	
<p>> 1. The cook selects one of the pending orders from the database and clicks on item to cook.</p> <p>2. System changes the status of the item to 'ready'</p> <p>< 3. The waiter is then notified by the system that the order is ready.</p>	

Use Case UC-10	clean
Primary Actor	Busboy
Actor's Goal	Clean dirty table.
Stakeholders	Host
Preconditions	There is a dirty table
Postconditions	The table status has to be changed from dirty to clean.
Main Success Scenario	
<p>> 1. The Busboy looks at the table layout screen and clicks on table to clean.</p> <p>2. System changes table status to ready.</p>	

Use Case UC-11	Order				
Primary Actor	Waiter				
Actor's Goal	Place the order in the queue for the cook to prepare.				
Stakeholders	Cook				
Preconditions	There is an order to be placed				
Postconditions	The order is placed in the queue to be prepared				
Main Success Scenario					
<p>> 1. Waiter chooses table and clicks the “Place Order” button</p> <p>< 2. System changes to the order page and displays only items that are in stock</p> <p>> 3. Waiter inputs the various items and quantity for the order and clicks the “Place Order” button</p> <p>4. System stores items into the database.</p> <p>Alternate Scenario:</p> <table border="1" style="margin-left: 40px;"> <tr> <td style="width: 50%;"></td> <td style="width: 50%;">If the customers choose to order more food.</td> </tr> <tr> <td></td> <td>Exceptions: The order should not be closed(Bill hasn't been paid yet)</td> </tr> </table> <p>> 1. If the customer wants to add more food and the bill hasn't been paid yet, the waiter will still have access to the order page for that table and he will be able to make the necessary additions.</p> <p>However, if the bill has been paid already, the new items will be associated with a different order ID.</p>			If the customers choose to order more food.		Exceptions: The order should not be closed(Bill hasn't been paid yet)
	If the customers choose to order more food.				
	Exceptions: The order should not be closed(Bill hasn't been paid yet)				

Use Case UC-12	Deliver
Primary Actor	Waiter
Actor's Goal	Deliver the food prepared by the cook to the customers.
Stakeholders	Manager
Preconditions	The cook has already prepared the food and it is ready for delivery.
Postconditions	The order is deleted from the cooks list of orders to do.
Main Success Scenario	
<p>< 1. The waiter gets an indication from the system that an item is ready.</p> <p>> 2. Waiter clicks on item to deliver.</p> <p>3. System updates status of item.</p>	

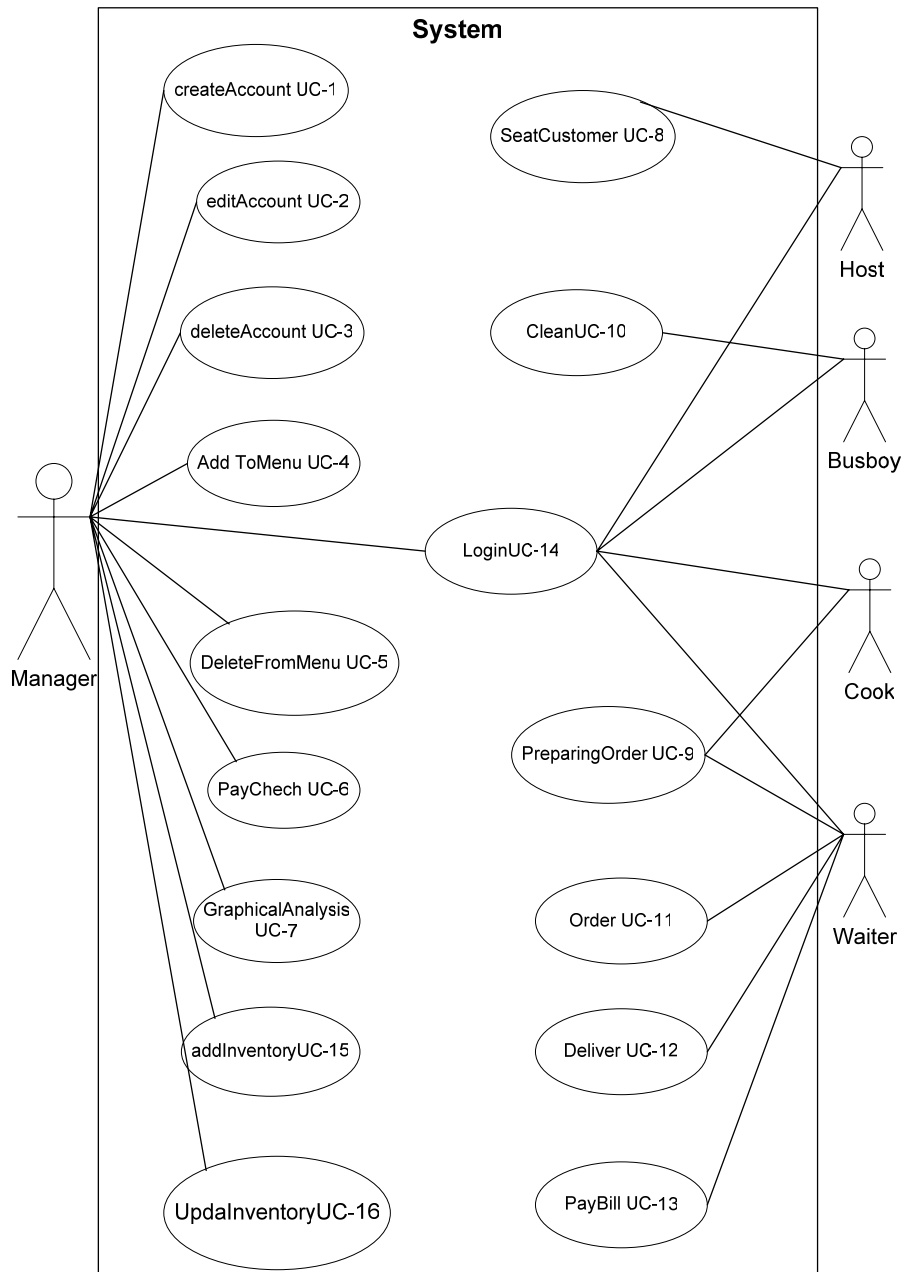
Use Case UC-13	PayBill
Primary Actor	Waiter
Actor's Goal	Place the order in the queue for the cook to prepare.
Stakeholders	Manager
Preconditions	The customers have ordered food and have finished eating it
Postconditions	The money has to be paid before the customers leave the restaurant and the table status has to be changed to dirty by the waiter.
Main Success Scenario	
<p>< 1. System generates check.</p> <p>> 2. Waiter processes payment and notifies system.</p> <p>< 3. System changes table status to dirty.</p>	

Use Case UC-14	login
Primary Actor	Employees
Actor's Goal	Login to the system to access pages required to their job.
Stakeholders	Employees
Preconditions	The person is an employee of the restaurant and already has a login account.
Postconditions	The employee is taken to his homepage depending on his job.
Main Success Scenario	
<p>> 1. Employee enters login name, password and employee type.</p> <p>< 2. System verifies the login information provided and if it is correct, the person is redirected to his homepage from where he will be able to access all pages required for his job. If the information is wrong, the system outputs a “Wrong Login” message and takes the user back to the main login page.</p>	

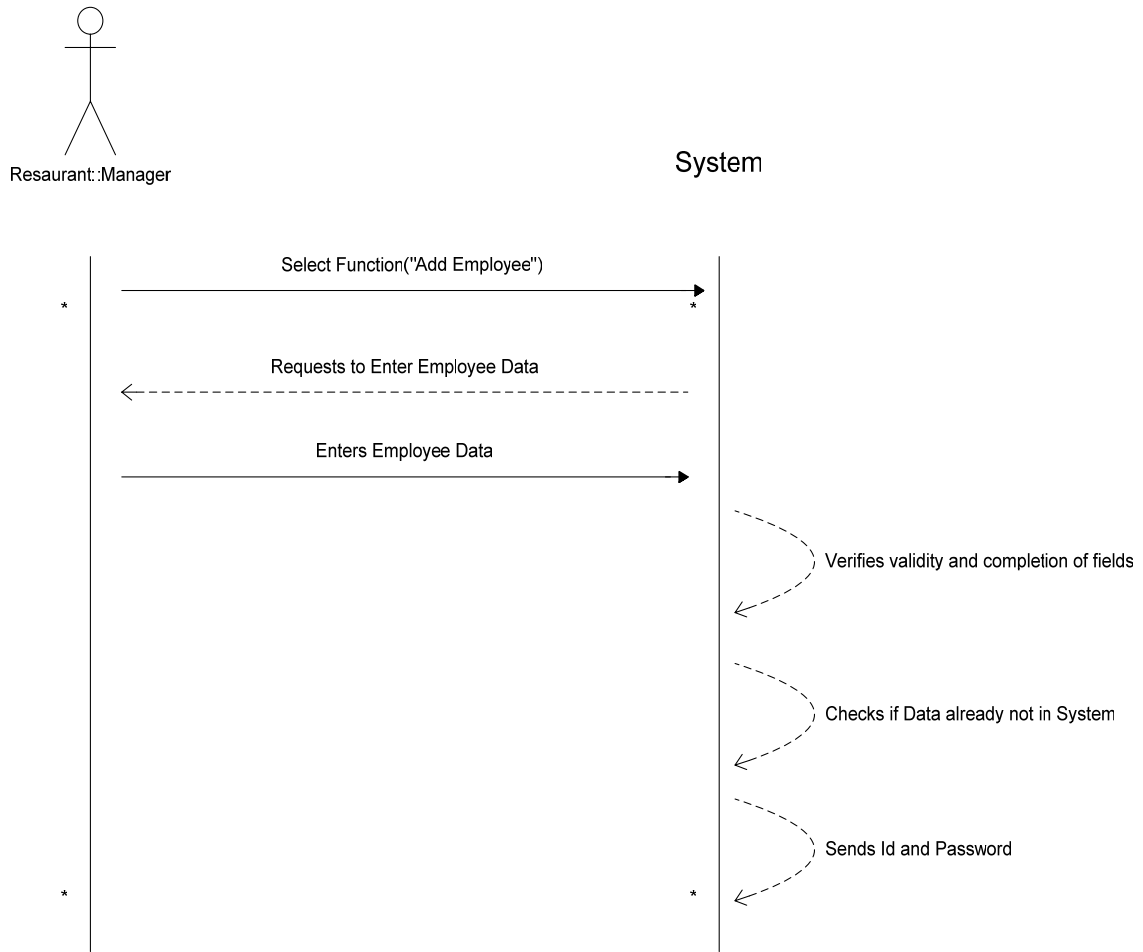
Use Case UC-15	addInventory
Primary Actor	Manager
Actor's Goal	Add items from inventory
Stakeholders	cook
Preconditions	Only the manager of the restaurant can make changes to the inventory. The itemID has to be unique.
Postconditions	The item is added and the system is updated.
Main Success Scenario	
<p>> 1. Manager selects add new item to inventory interface. Adds information of new product.</p> <p>< 2. System validates fields and checks for duplicate entry. Adds information to database and notifies manager.</p>	

Use Case UC-16	Update/DeleteInventory
Primary Actor	Manager
Actor's Goal	Delete and update items from inventory
Stakeholders	cook
Preconditions	Only the manager of the restaurant can make changes to the inventory.
Postconditions	If an item from the inventory is deleted, all items from the menu which cannot be prepared without that item should also be deleted.
Main Success Scenario	
<p>Main Success Scenario</p> <ul style="list-style-type: none"> > 1. Manager selects inventory management interface. > 2. Selects item to be updated/deleted from a drop-down menu. < 3. System validates fields and updates database. 	

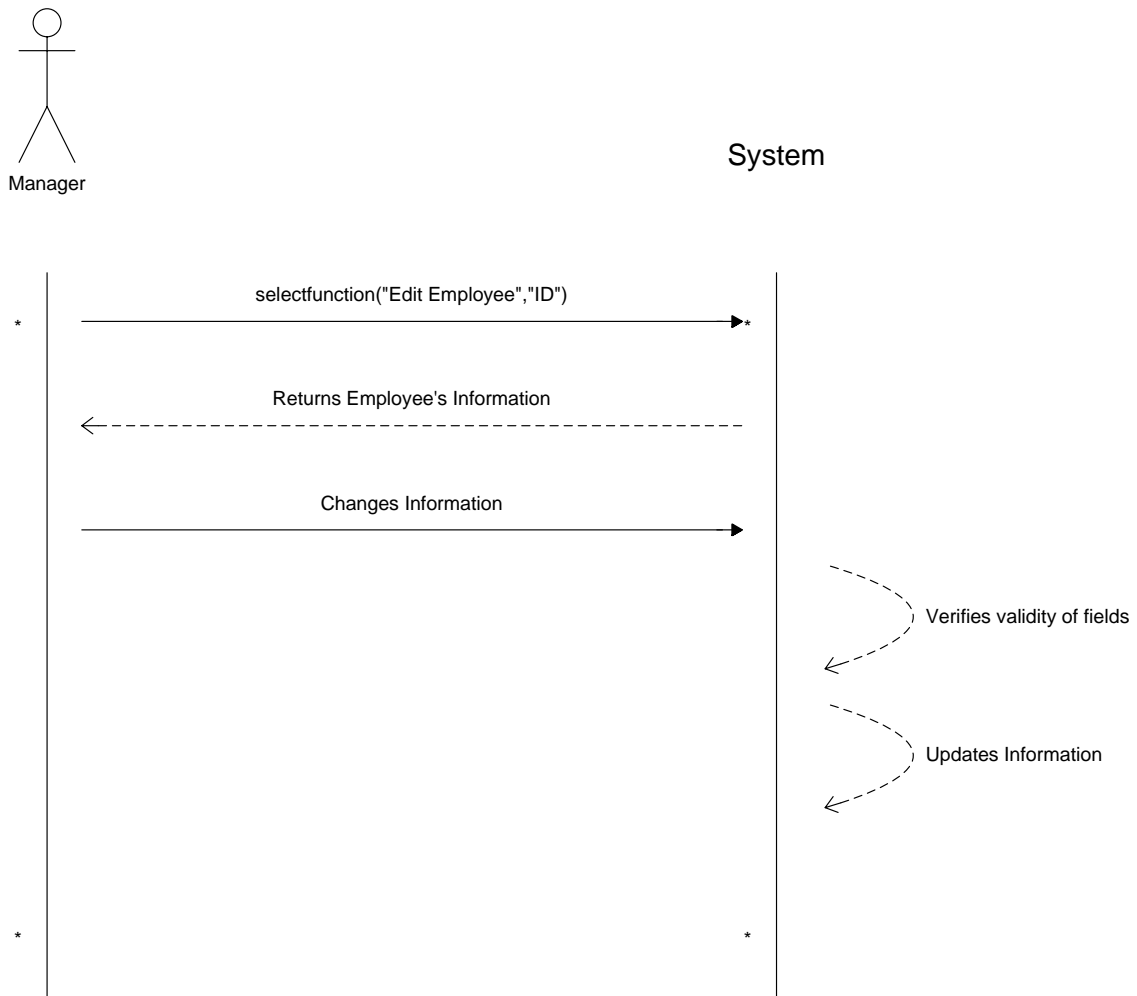
Use Case Diagram



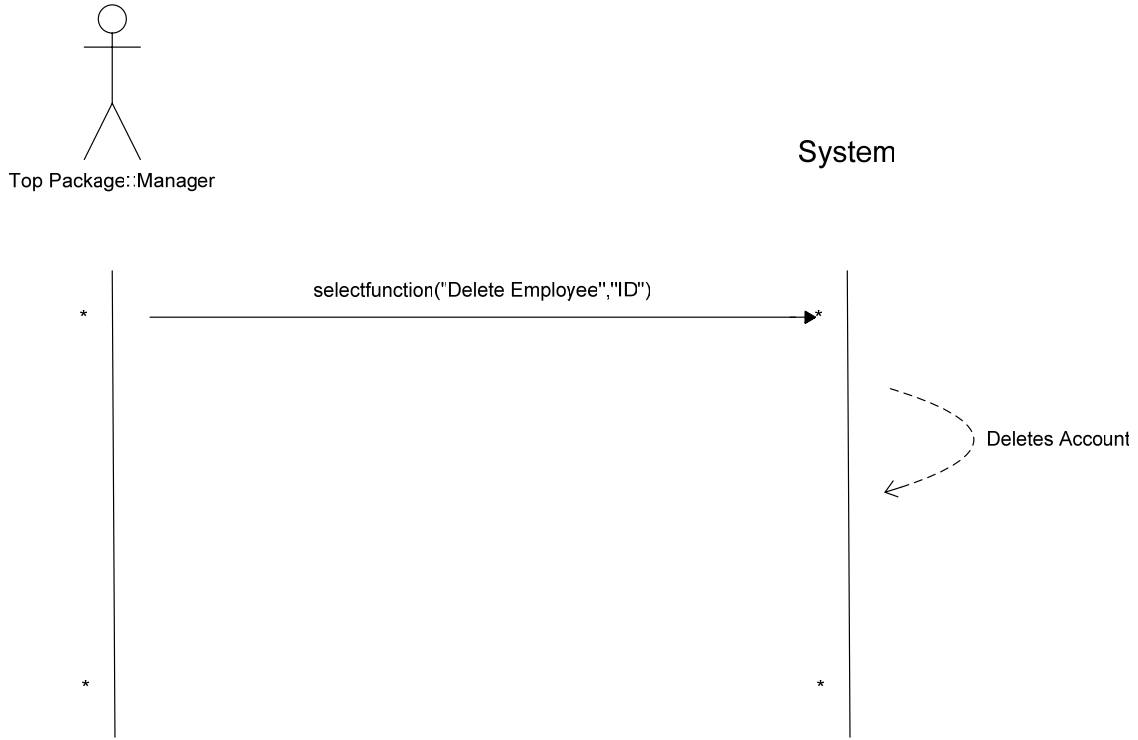
UC-1 createAccount



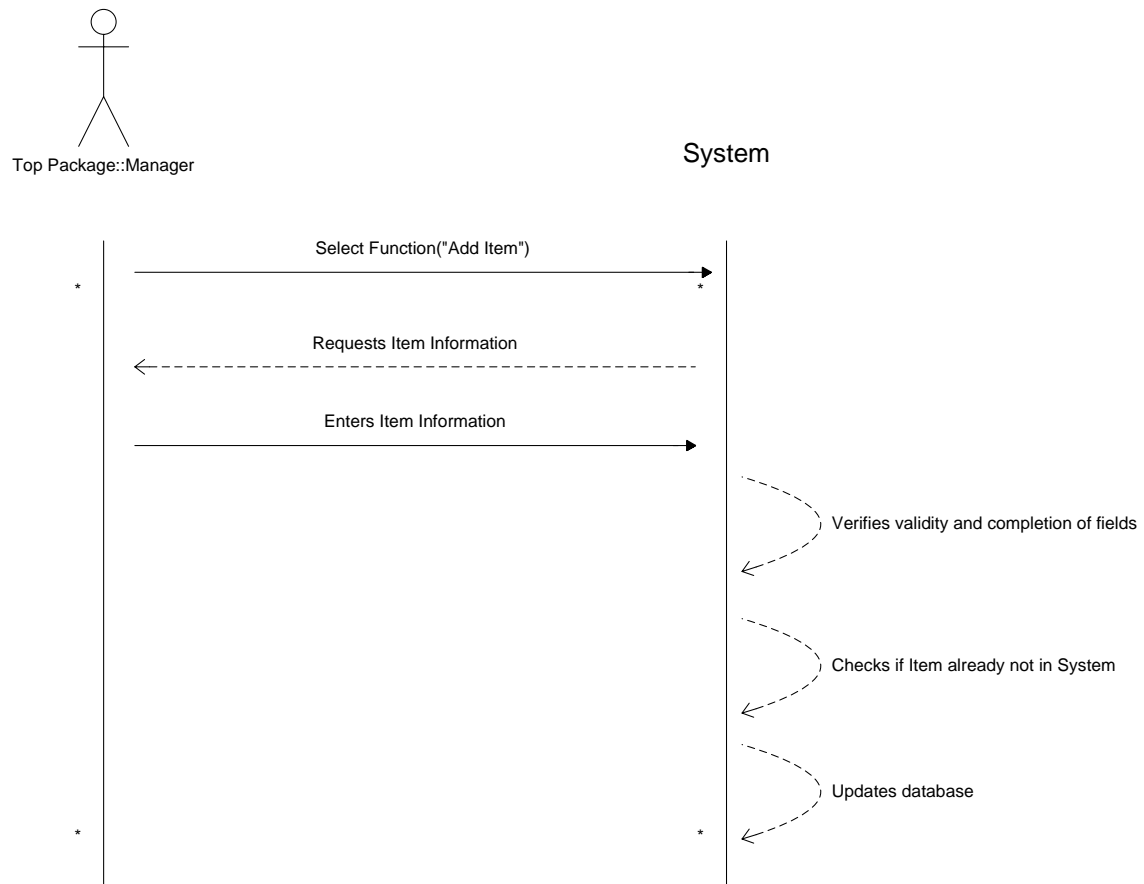
UC-2 editAccount



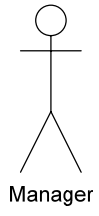
UC-3 deleteAccount



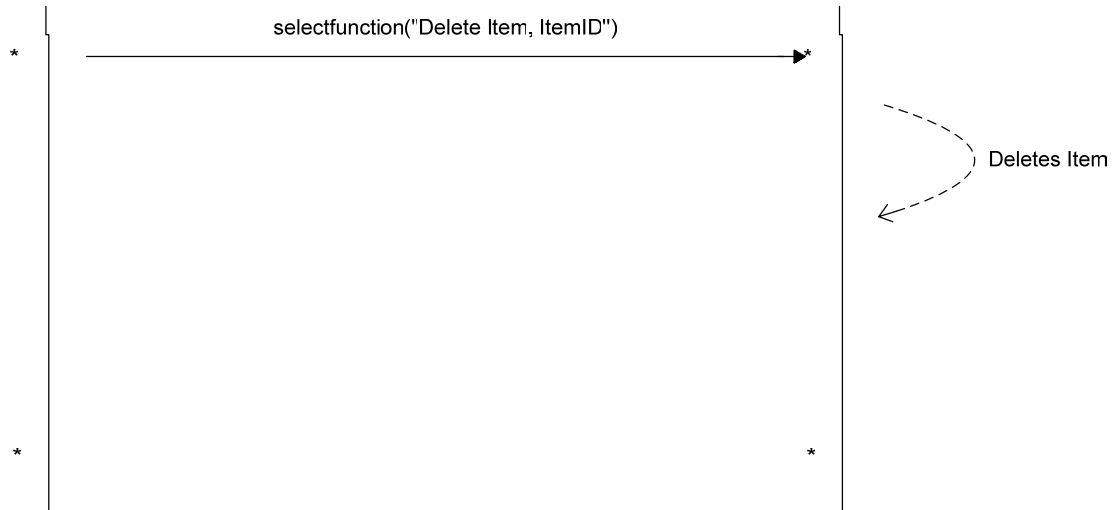
UC-4 AddtoMenu



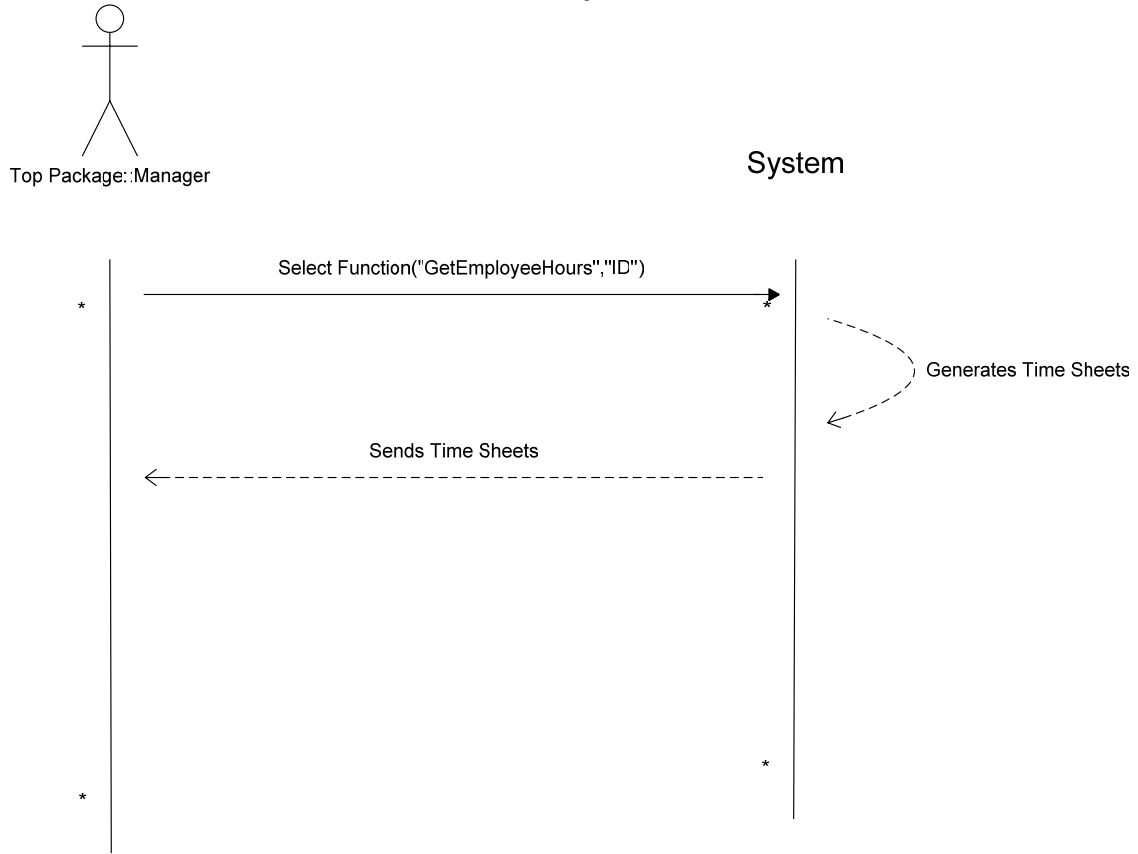
UC-5 DeletefromMenu



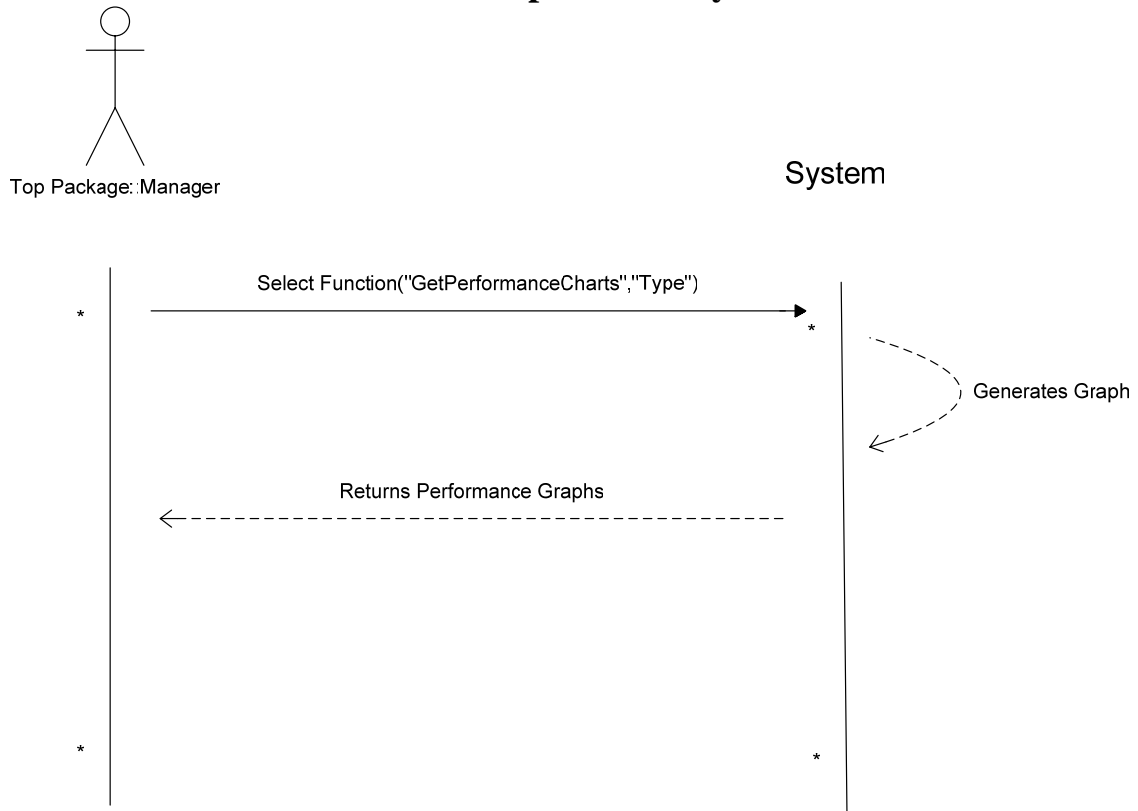
System



UC-6 Paycheck



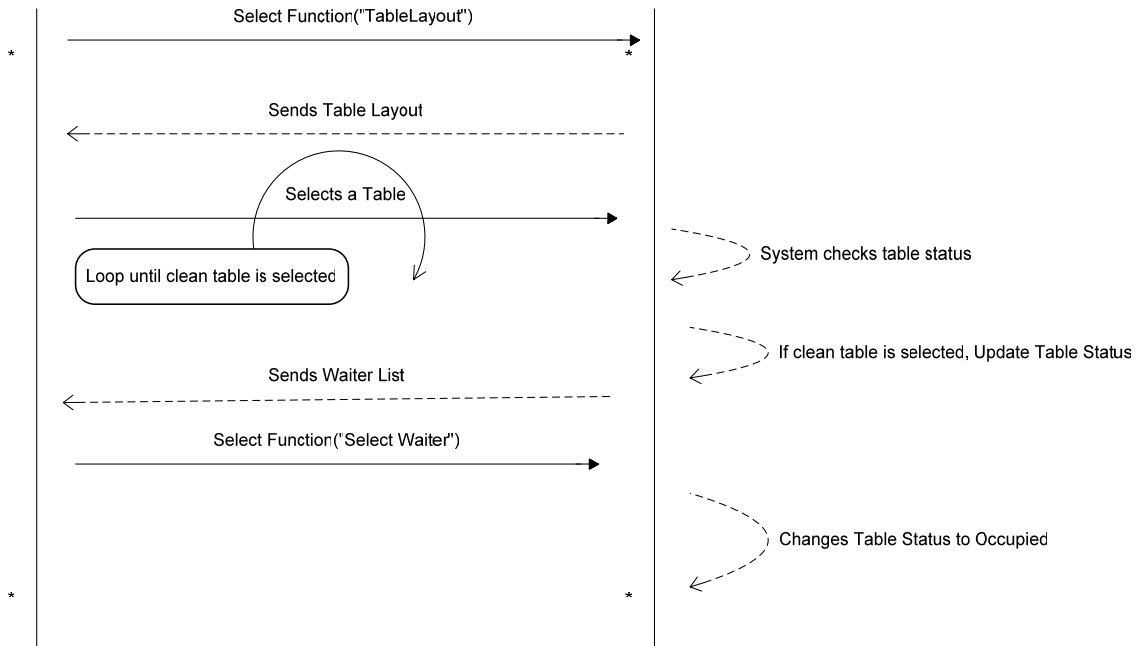
UC-7 Graphical Analysis



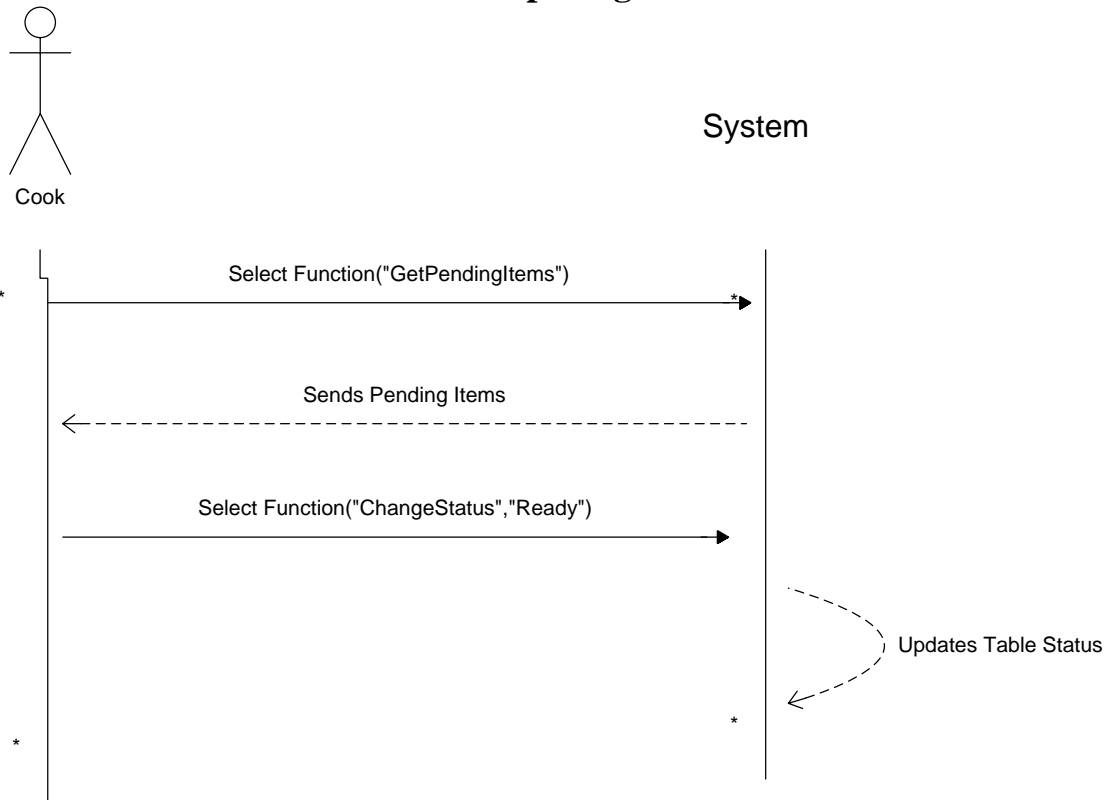
UC-8 SeatCustomer



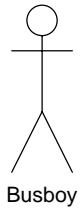
System



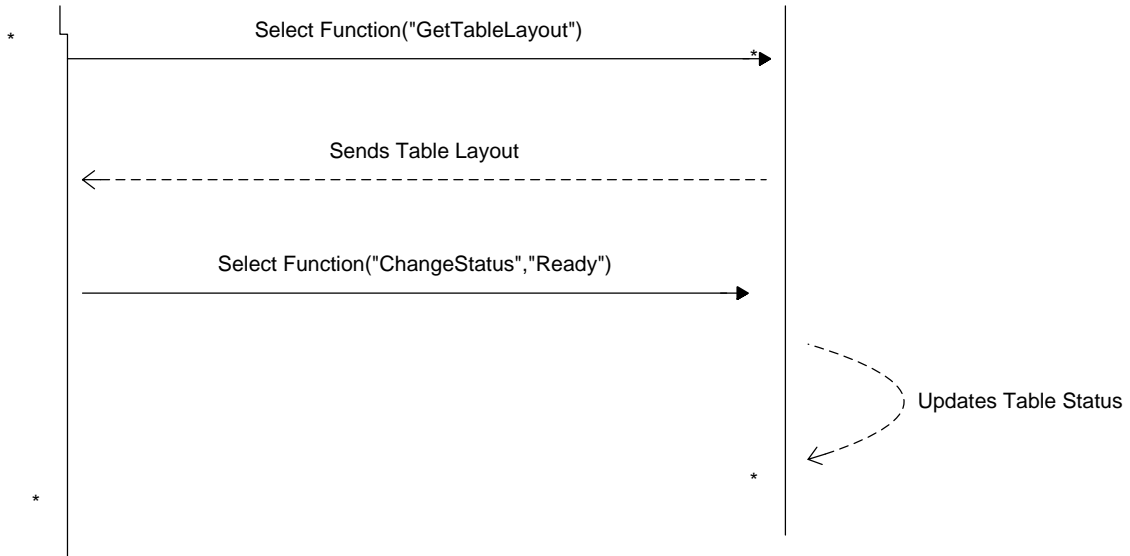
UC-9 PreparingOrder



UC-10 clean



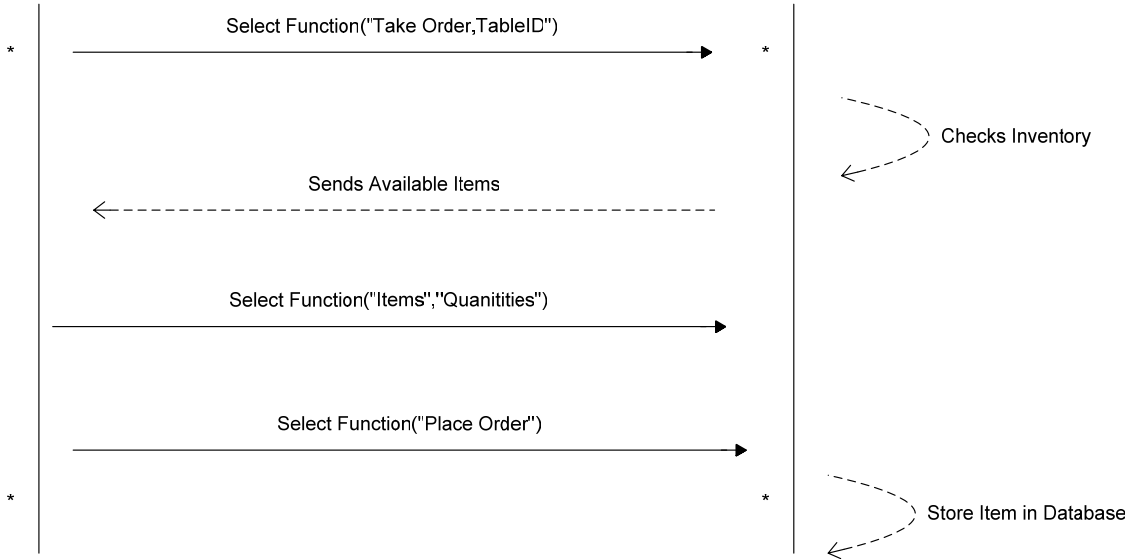
System



UC-11 order



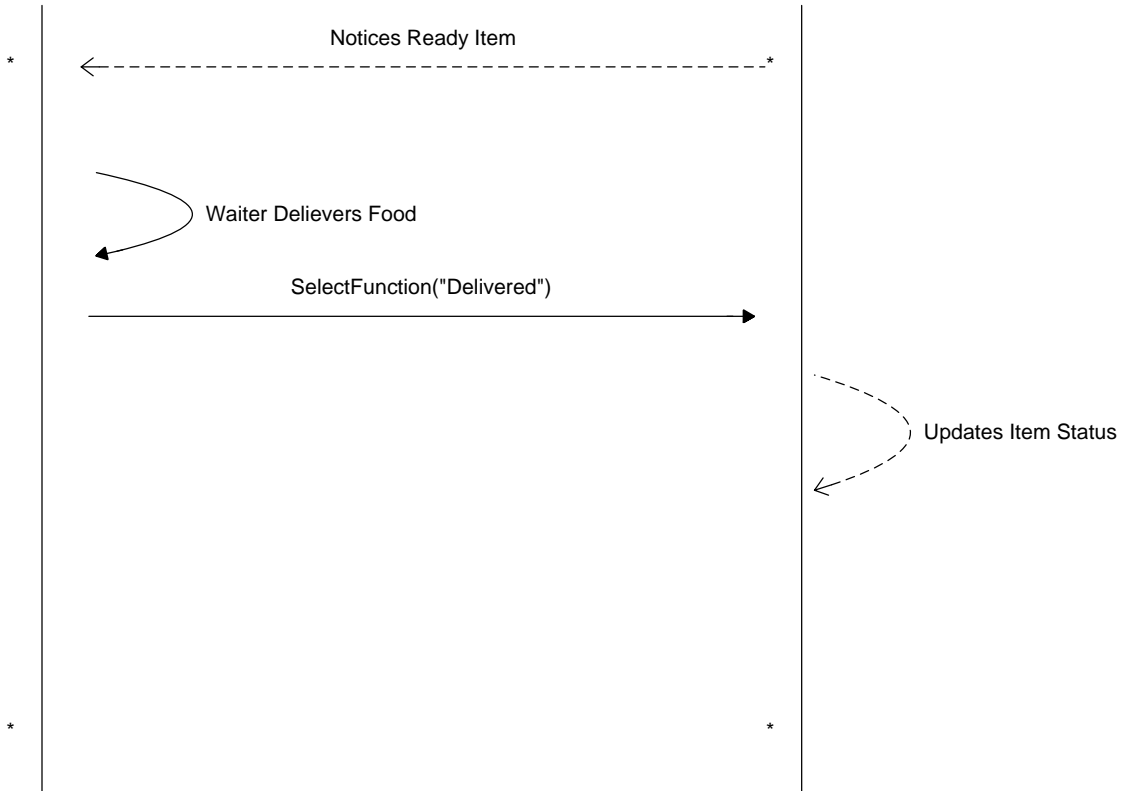
System



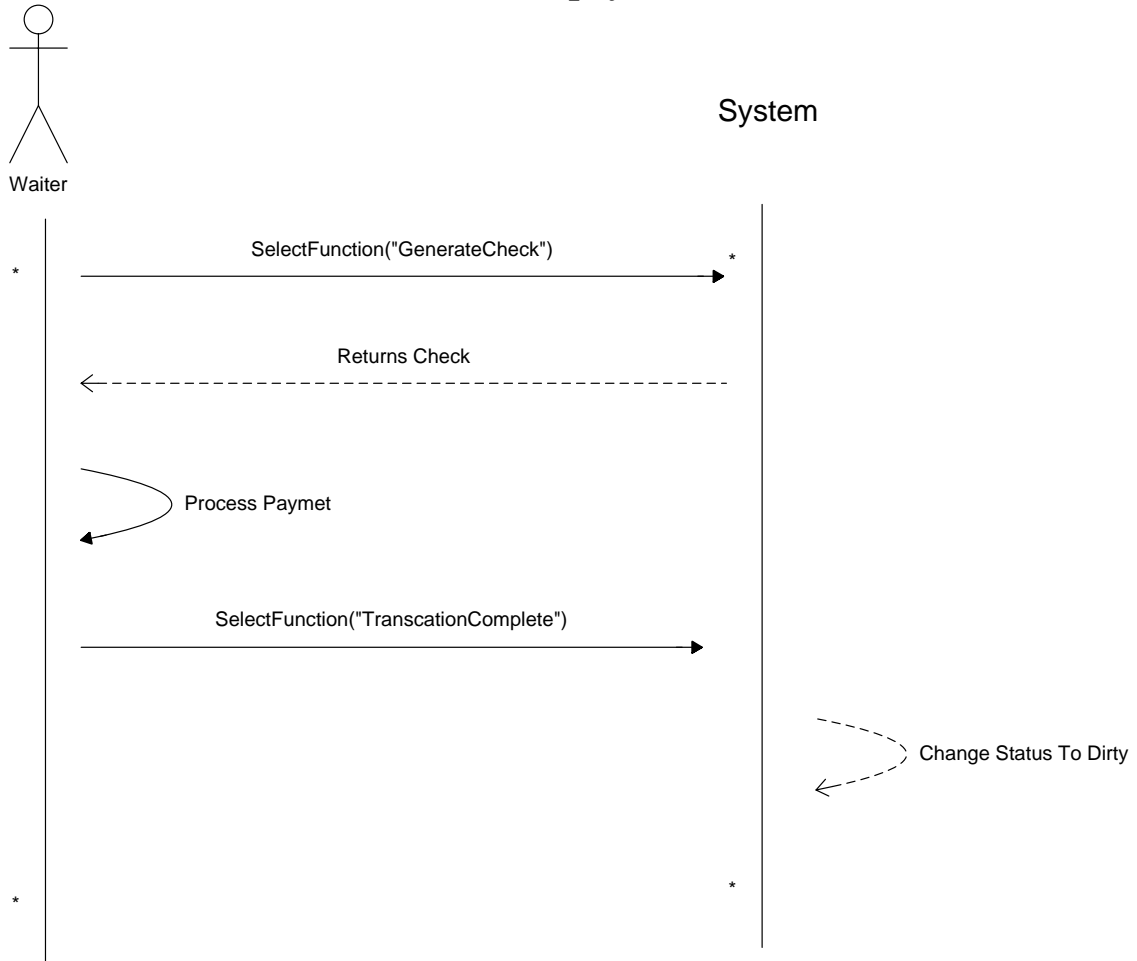
UC-12 deliver



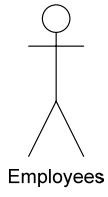
System



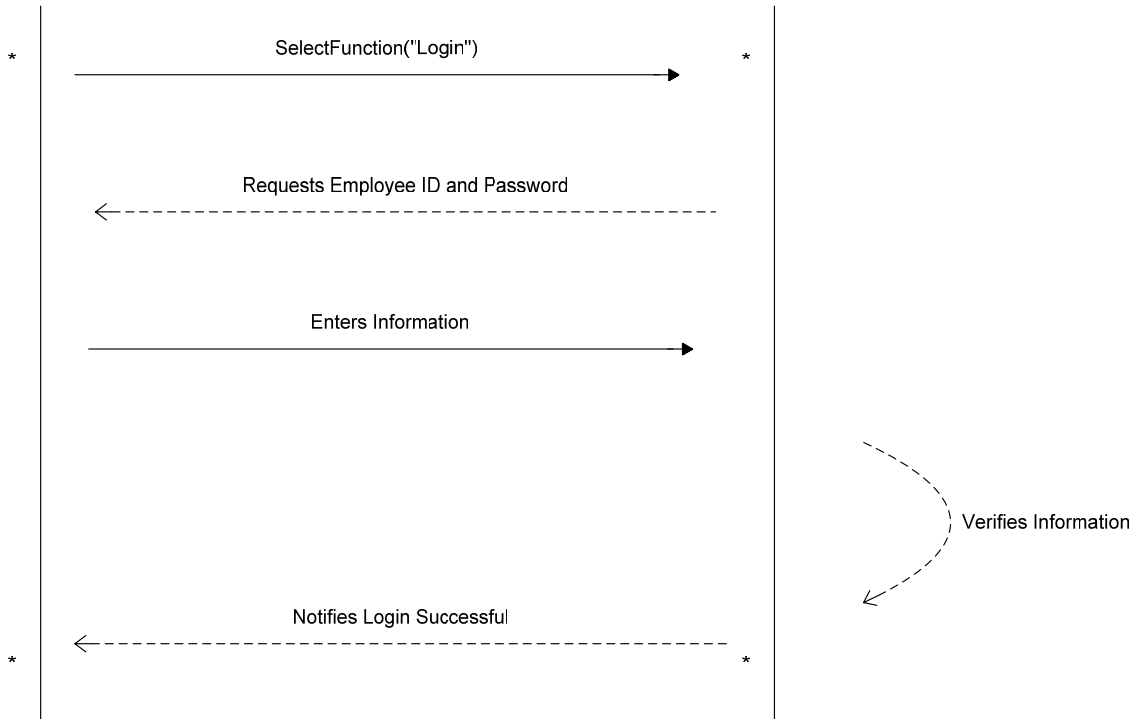
UC-13 payBill



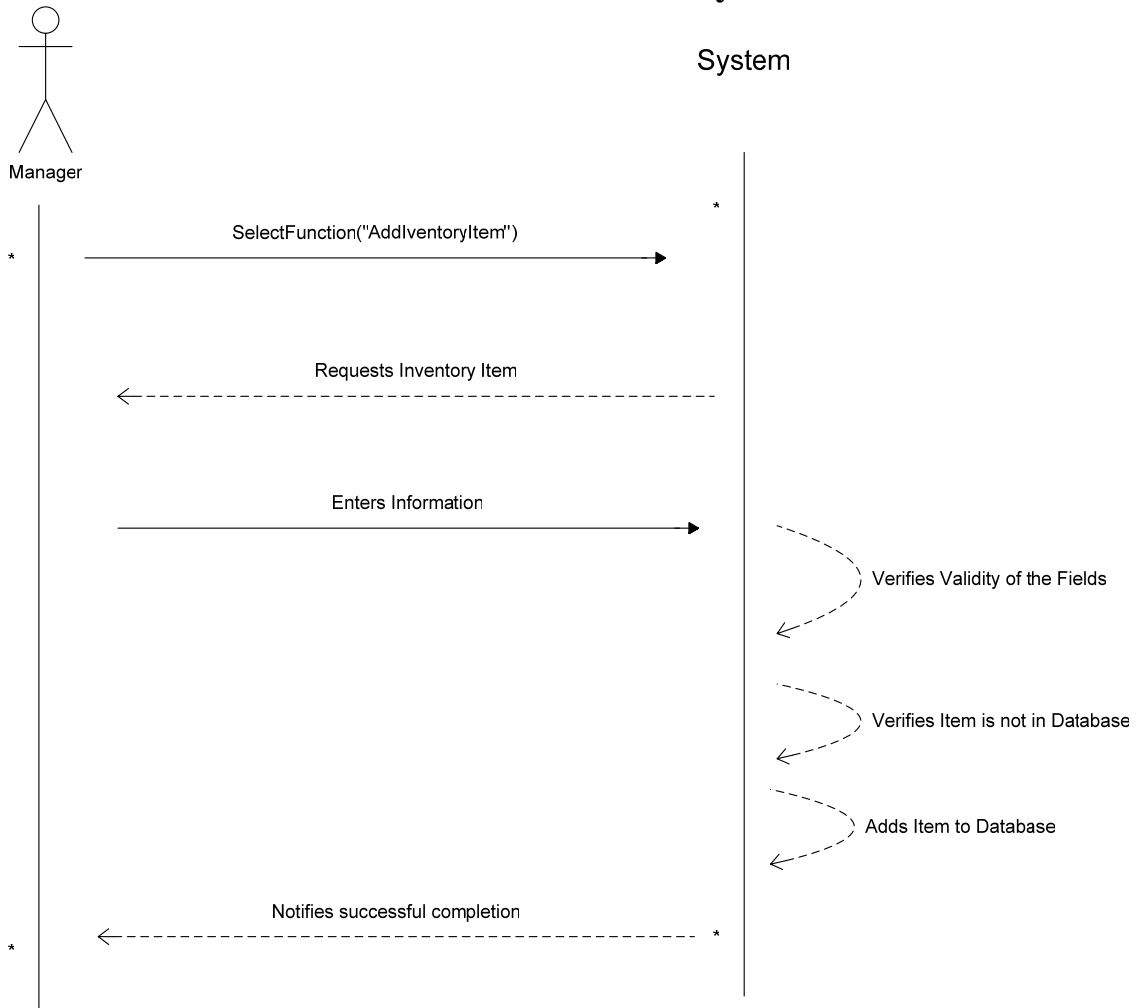
UC-14 login



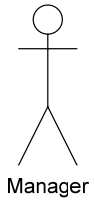
System



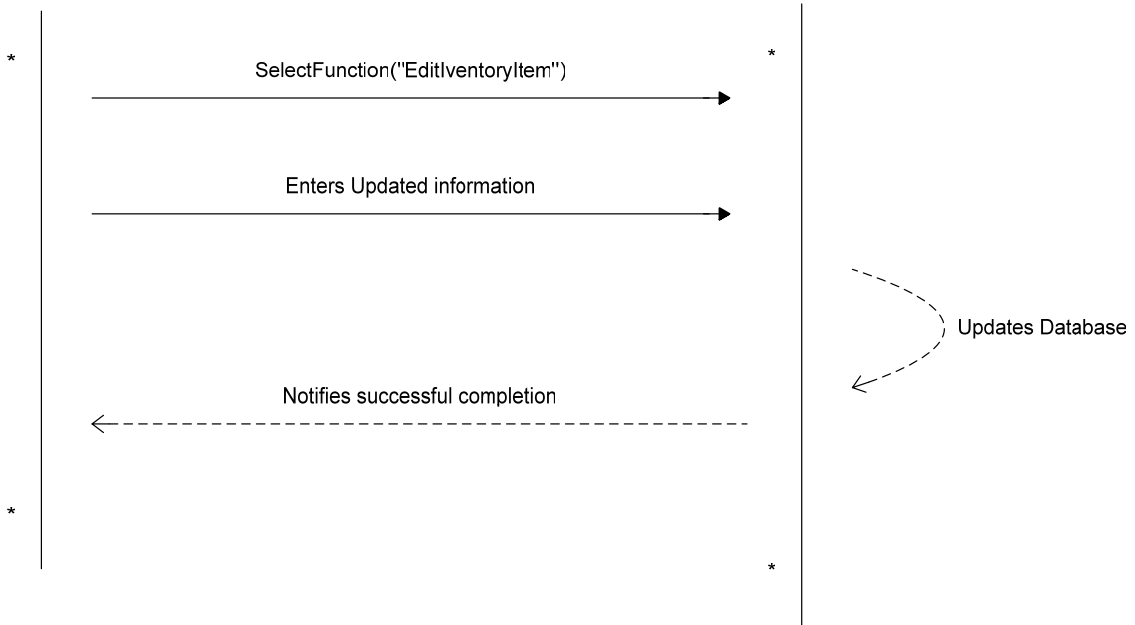
UC-15 addInventory



UC-16 UpdateInventory



System



7. Nonfunctional Requirements

Usability:

The system's user interface will be very simple and self-explanatory. The manager and the cook will interact with the system through touch-screen LCDs. They are very easy to adapt to and use. The users just have to touch the option they want. The waiters will interact with PDAs. They will each be able to access the tables that they are responsible for. The menus for the restaurant will show up as drop-down menus on the PDAs. The waiters only have to select the item that the customer ordered from the drop-down list.

Reliability:

The system is guaranteed to be reliable. All inputs to the system will be selections from options that the system will show on the screen. Since the system will only provide valid options, there is absolutely no possibility that invalid inputs can be entered. Each user will have a unique username and password. This removes the risk of unauthorized access to the system. Also, users can only access parts of the system that they require for their job. For example, a waiter cannot access the inventory management or the payroll feature of the system. Those parts among a few others can only be accessed by the manager. All these constraints will ensure the reliability of the system.

Performance:

The system will be used by many employees of the restaurant at the same time and can handle it without any errors. However, for this to happen, the server at the restaurant should be able to handle all the traffic without creating any problems. The restaurant should also have a high speed wireless connection for the system to perform its best. The waiters will be using PDAs and will only be able to connect to the system with a wireless connection. Since some tasks like placing and cooking and an order are sequential, the waiter has to update the system with the order first before the cook will see it. In order to for this to happen efficiently, the internet connection needs to be fast so that changes are reflected on the website instantly without any delay.

Supportability:

The system will support changes that the restaurant might need to make in future. The manager will have the ability to modify items from the menu. He will also be able to add or delete users to the system for layoffs, retirements or new hires.

Implementation Requirements:

We plan to utilize **PHP** for the front-end and **MySQL** as our database in the back-end. The server should have the ability to run both **PHP** and **MySQL**.

Interface Requirements:

The system requires both the manager and the cook to have touch-screen LCDs and all the waiters to have a PDA each. The Restaurant should also have a highspeed wireless connection for the PDAs to be connected to the system.

Operations Requirements:

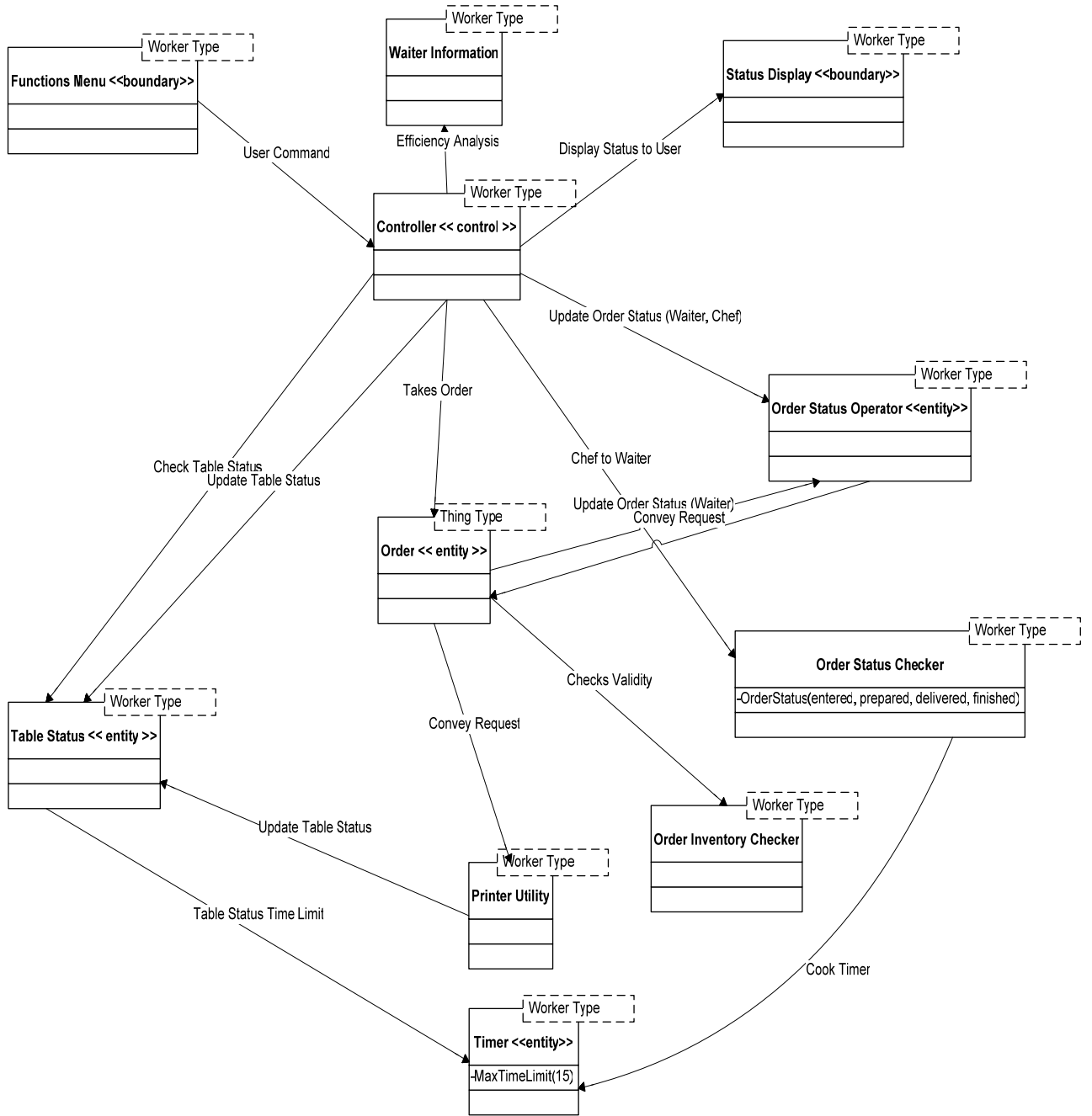
The restaurant manager will be the sole administrator of the system. He/she can modify the database and add or delete users from the system. The manager can also compile reports to analyze the restaurant performance in general or to see the best selling item or to compare different waiters.

Legal Requirements:

We will give each of our clients a license to use the software.

8. Domain Analysis

Domain Model



System Operations Contract

Name:	createAccount
Responsibilities:	Creates new employee profile and adds it to the database.
Type:	System
Exceptions:	If all the required fields are not complete, the employee information is not added to database.
Preconditions:	The employee being added does not exist in the database.
Postconditions:	New employee has been added to the database.

Name:	editAccount
Responsibilities:	Edits an already existing account
Type:	System
Exceptions:	The required fields are not complete or valid and the information cannot be updated.
Preconditions:	Employee information exists in the database.
Postconditions:	Employee information is updated and stored in the database.

Name:	deleteAccount
Responsibilities:	Delete an employee account from the database.
Type:	System
Exceptions:	Employee information does not already exist in the database.
Preconditions:	Employee information exists in the database.
Postconditions:	Employee information is removed from the database.

Name:	AddtoMenu
Responsibilities:	Adds a new item into the menu that the waiter selects from.
Type:	System
Exceptions:	None
Preconditions:	Item does not exist in the menu.
Postconditions:	Item is added into the menu and the database with the price.

Name:	DeletefromMenu
Responsibilities:	Deletes an existing item from the menu.
Type:	System
Exceptions:	None
Preconditions:	Item exists in the menu.
Postconditions:	Item information is removed form the menu and database.

Name:	Paycheck
Responsibilities:	Weekly paychecks are prepared, and the information is also stored into the database.
Type:	System
Exceptions:	None
Preconditions:	The employee information exists in the system; the number of hours are available.
Postconditions:	Information is calculated, returned, and stored into the database.

Name:	GraphicalAnalysis
Responsibilities:	Performance charts are displayed and stored.
Type:	System
Exceptions:	Information required to compute the graph is not available and no graph can be displayed.
Preconditions:	Information regarding the graph is available and stored in the database.
Postconditions:	Graph is displayed, and stored in database.

Name:	SeatCustomer
Responsibilities:	Table status is changed and waiter is assigned
Type:	System
Exceptions:	Table is already full and new customer cannot be seated there.
Preconditions:	The customer has not already been seated and the table is empty.
Postconditions:	Customer is seated, waiter is assigned.

Name:	PreparingOrder
Responsibilities:	Change the status of items, and notify waiter when ready.
Type:	System
Exceptions:	None

Preconditions:	Order has been made.
Postconditions:	Waiter is notified as soon as food status is changed.

Name:	clean
Responsibilities:	Table status has been changed to clean
Type:	System
Exceptions:	None
Preconditions:	Table is dirty
Postconditions:	Table status is changed to clean

Name:	Order
Responsibilities:	Order is placed and chef is notified
Type:	System
Exceptions:	None
Preconditions:	Items are selected by the waiter.
Postconditions:	Order is placed in a queue and chef is notified.

Name:	Deliver
Responsibilities:	Item status is changed by chef and waiter is notified that food is ready.
Type:	System
Exceptions:	None
Preconditions:	Food is waiting to be prepared by the chef
Postconditions:	Food is ready, chef changed status, and waiter is notified.

Name:	PayBill
Responsibilities:	Bill is prepared, and table status is changed to dirty
Type:	System
Exceptions:	None
Preconditions:	Table order has been completed, all items have been delivered.
Postconditions:	Payment is accepted, and table status is changed.

Name:	Login
Responsibilities:	Allow the user to enter his personal GUI interface
Type:	System
Exceptions:	Incorrect login information results in a failure to enter the system.
Preconditions:	Login information is available.
Postconditions:	User is either logged in or rejected entry based on their login information.

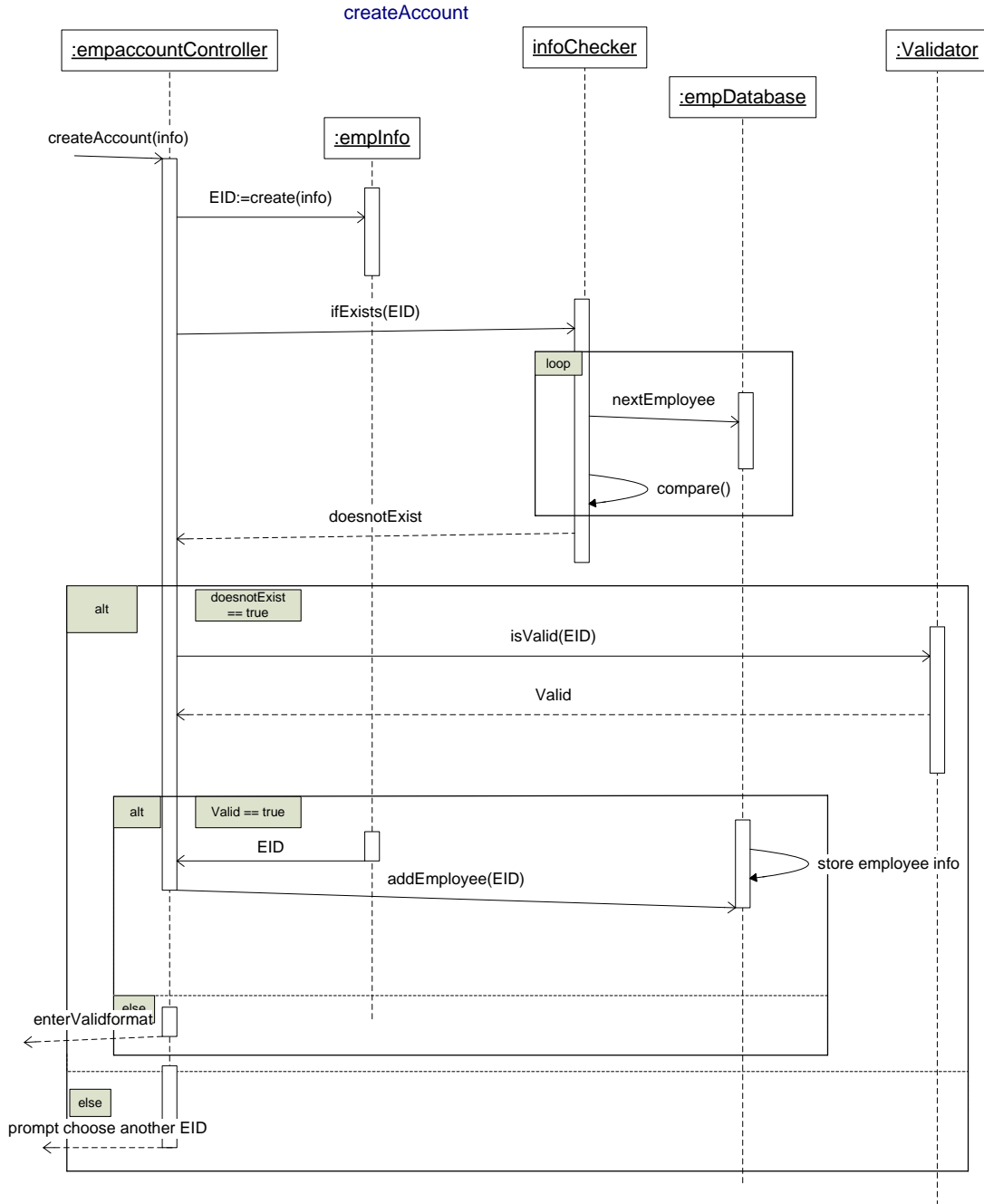
Name:	addInventory
Responsibilities:	To add inventory into the database.
Type:	System
Exceptions:	If item already exists, it cannot be added as a new item.
Preconditions:	New item information such as name and quantity is available.
Postconditions:	Item is added into the inventory and stored in the database.

Name:	Update/DeleteInventory
Responsibilities:	Updates or deletes the item based on user input
Type:	System
Exceptions:	If item does not exist in the database, it cannot be removed or updated.
Preconditions:	Item exists in the database already.
Postconditions:	If quantity input is 0, item is removed. If quantity input is an integer, the quantity is updated appropriately.

9. INTERACTION DIAGRAMS

Note: Descriptions for each diagram are present on the page after the interaction diagram. Please refer to it.

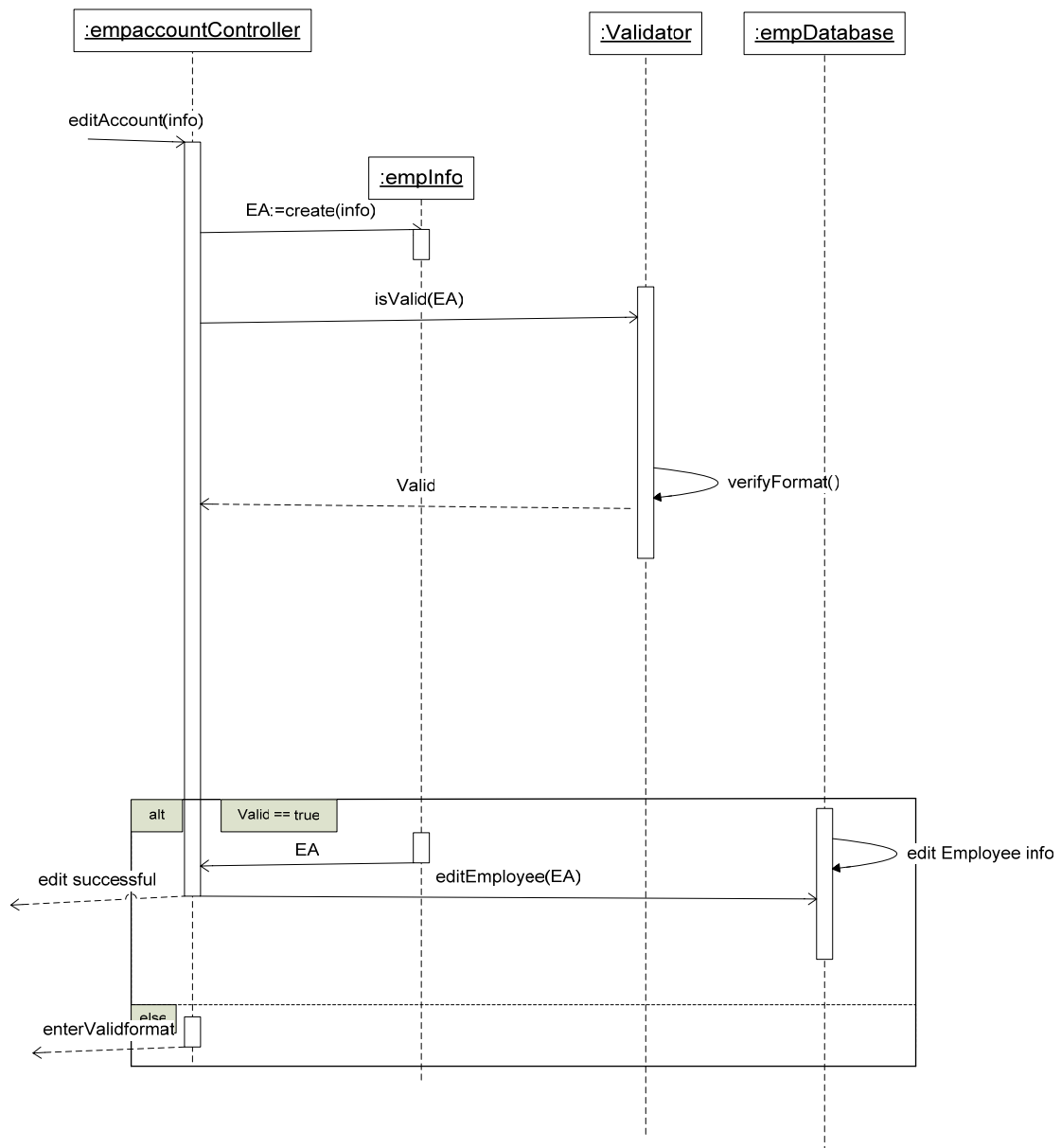
UC1 createAccount



Description of Interaction Diagram 1

1. Interaction Diagram 1: UCI

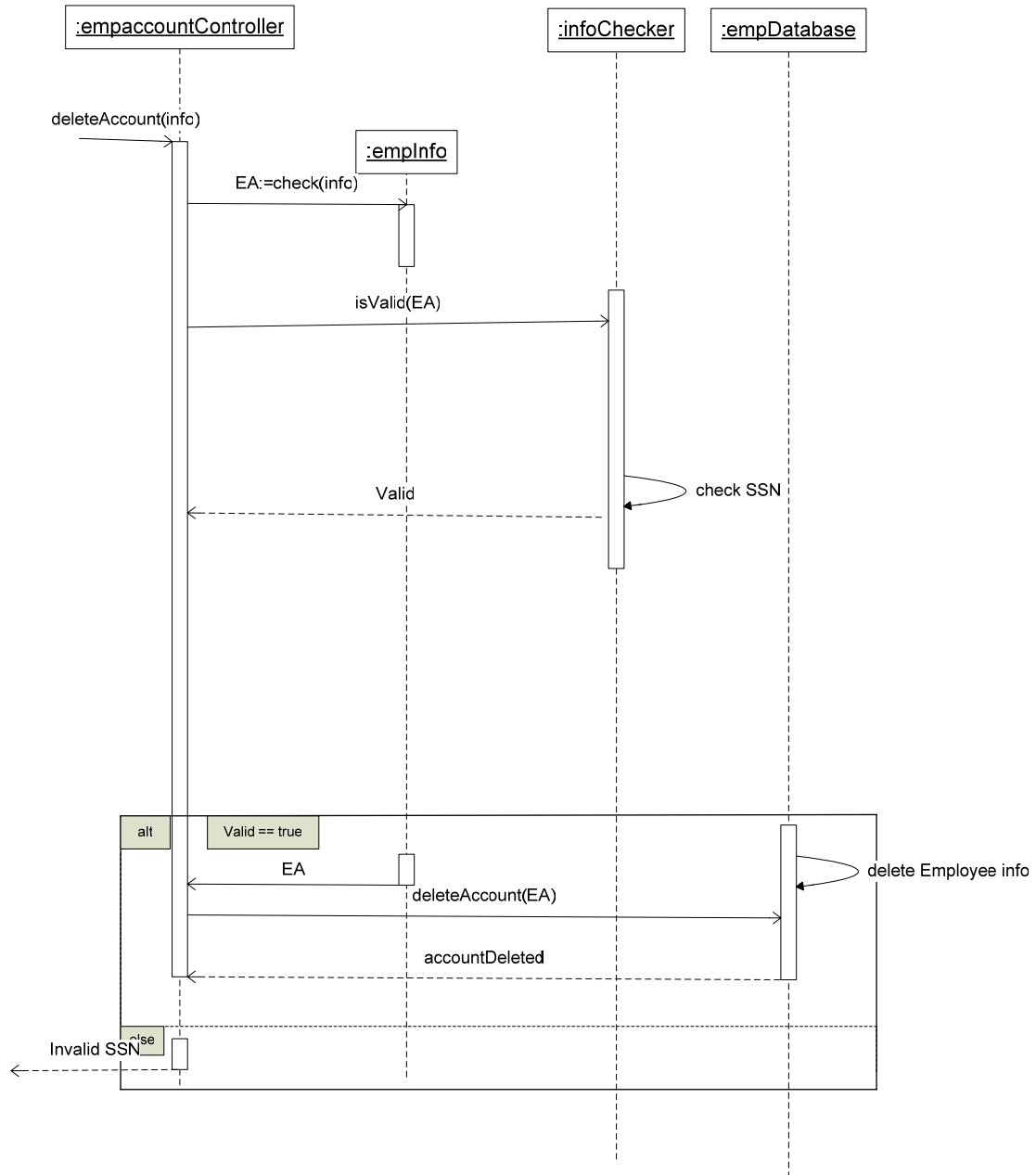
- (i) The manager has to create an account for a new Employee.
- (ii) Manager enters employee information such as Employee Social Security Number, last name, first name then passes the information to the empaccountController.
- (iii) empaccountController makes an instance of employee account called EA
- (iv) empaccountController makes a call ifExists () passing it to the InfoChecker to see whether the employee already exists in the EmployeeDatabase.
- (v) InfoChecker then makes a call to EmployeeDatabase to retrieve information of the employees in the database one by one and compares it with the information provided for the new employee.
- (vi) If the compare results in a false for every employee in the database then the InfoChecker sends a doesNotExist message back to empaccountController.
- (vii) Next the empaccountController checks whether the information is fed in the correct format by sending an isValid () signal to the Validator.
- (viii) If the Validator returns a true value then the employee is added to the database else the manager is prompted to enter the information in the correct format.

UC2 – editAccount

2. Interaction Diagram : UC2 editAccount

- (i) The manager can edit and update the accounts of the existing employees in the employeeDatabase.
- (ii) Manager enters employee information such as employee's SSN and passes it to the empaccountController.
- (iii) accountController then makes an isValid () call to the Validator to check whether the information was entered in the correct format.
- (iv) If the employee information was entered in the correct format the Validator returns a Valid () = true message to the empaccountController.
- (v) The empaccountController makes an editAccount () call to the employeeDatabase and edits/updates the database fields.
- (vi) A message is sent back to the user interface telling the user that the edit was successful.
- (vii) Otherwise if the employee account does not exist then ask the manager to enter the correct information.

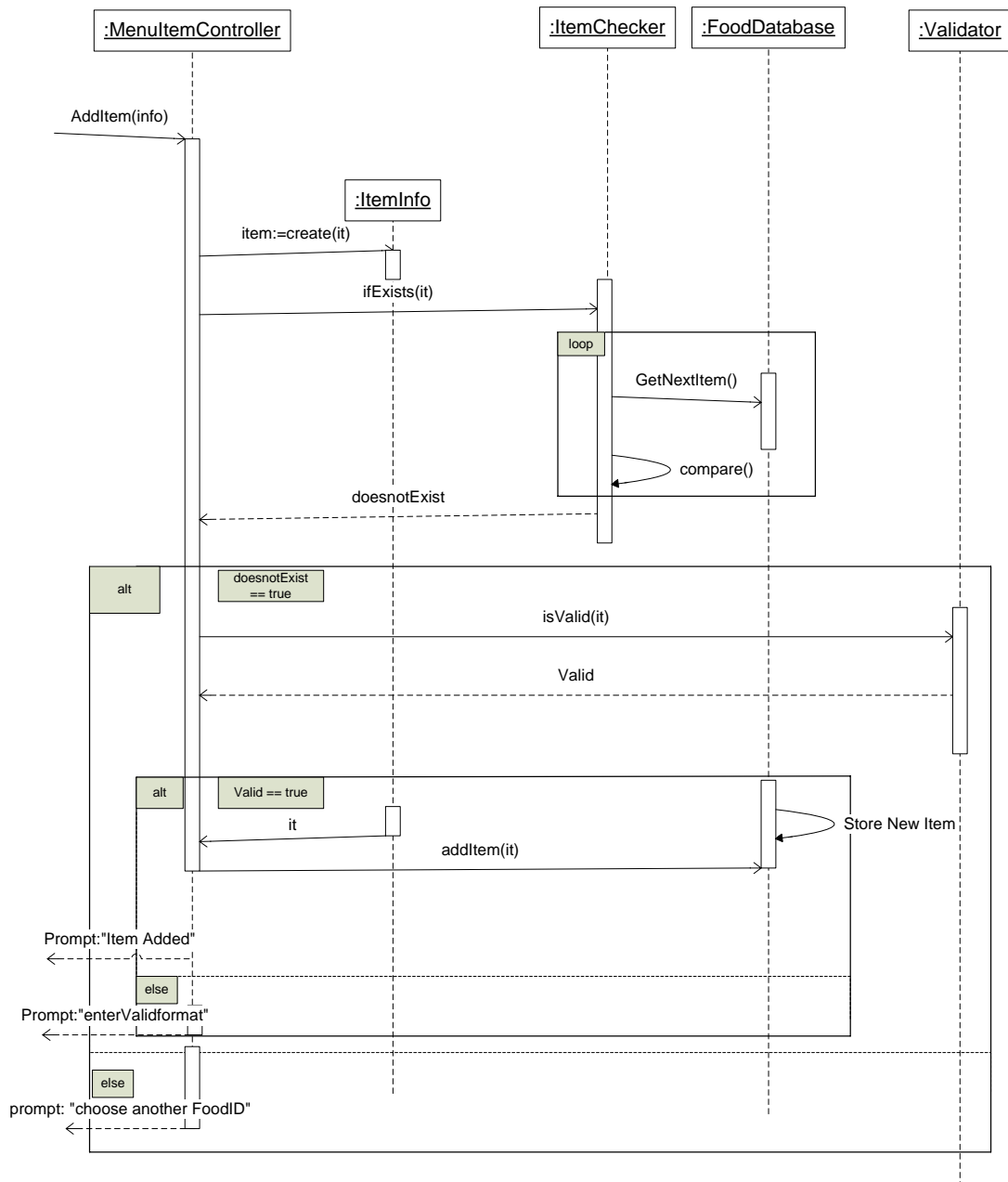
UC3 deleteAccount



3. Interaction Diagram: UC3 deleteAccount

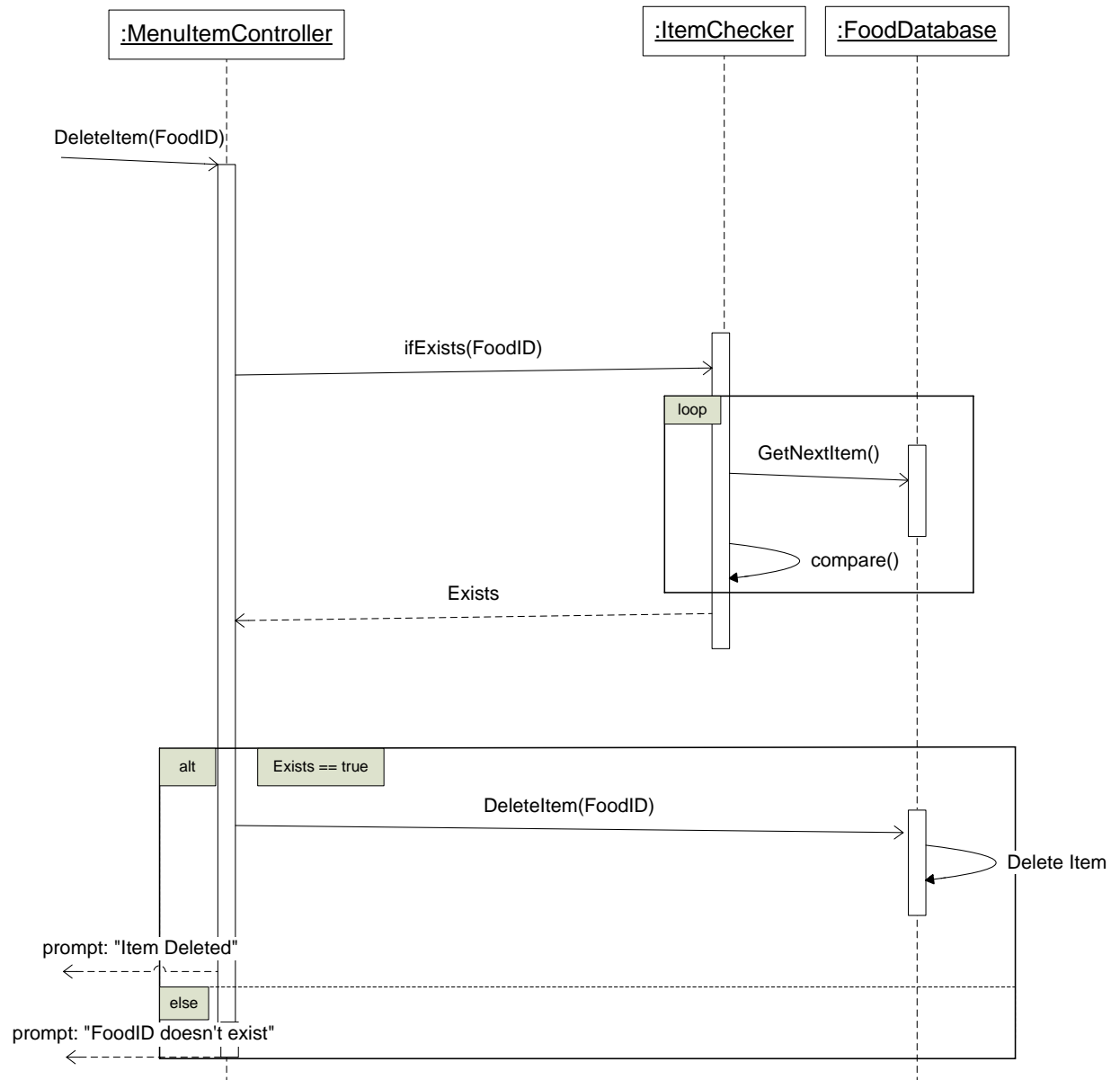
- (i) The manager enters the information for the employee whose account has to be deleted.
- (ii) The empaccountController receives this info which consists of the employee SSN and sends it to the infoChecker by sending a ifExists () signal.
- (iii) The infoChecker checks whether the employee SSN exists in the database or not by comparing it to the SSN provided.
- (iv) If it the SSN exists a Valid () message is sent to the empaccountController and the employee account is deleted from the database.
- (v) Once the account is deleted a message is sent to the user interface of the successful deletion.

UC4 – Add To Menu



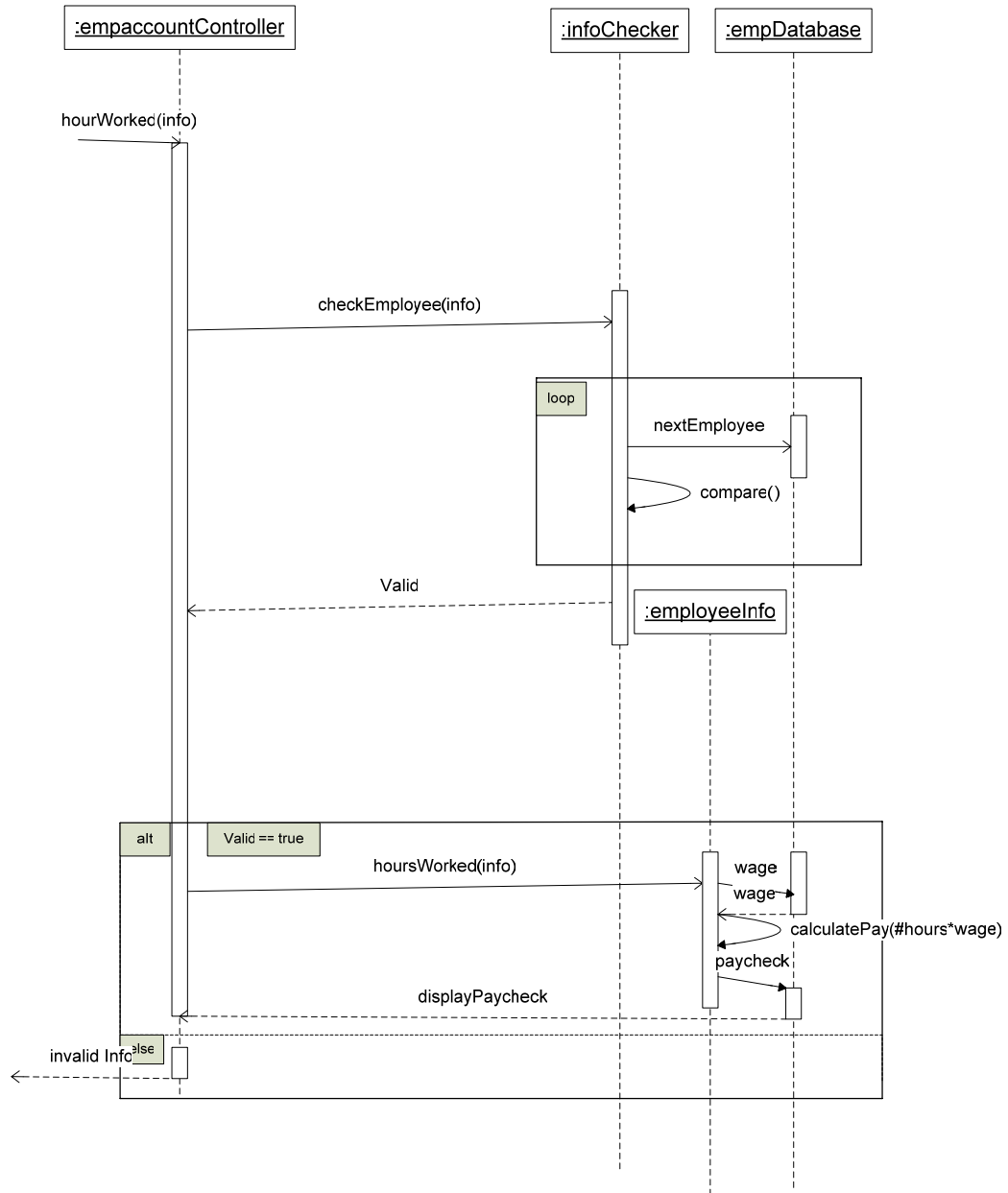
4. Interaction Diagram 4: UC4 – Add Item to Menu

- (i) The manager can add a new food item to the menu.
- (ii) Manager picks a FoodID for the new item and sends it to the MenuItemController along with the name, the type of food and its price.
- (iii) The MenuItemController makes an instance of Food Item called "it".
- (iv) The MenuItemController then makes a call `ifExists ()` and passes it to the ItemChecker to see if the food item already exists in the FoodDatabase.
- (v) ItemChecker then makes a call to the FoodDatabase to get information about the items in the menu one by one and compares it with the information provided for the new Item.
- (vi) If the comparison results in a match for any item already in the database, the signal "DoesNotExist" is set to false and a prompt "choose another FoodID" is displayed. If there is no match, the signal "DoesNotExist" is set to true and the MenuItemController makes the call `isValid()` and sends it to the Validator which checks for the format of the data. If the data is not in the correct format, the signal "valid" is set to false and the prompt " enter Valid Format" is displayed to the user.
- (vii) If the data is in the correct format, the signal "valid" is set to true and the MenuItemController makes a call `AddItem()` and passes it to the FoodDatabase, where the item is stored.

UC5 – Delete Item from Menu

5. Interaction Diagram : UC5- Delete From Menu

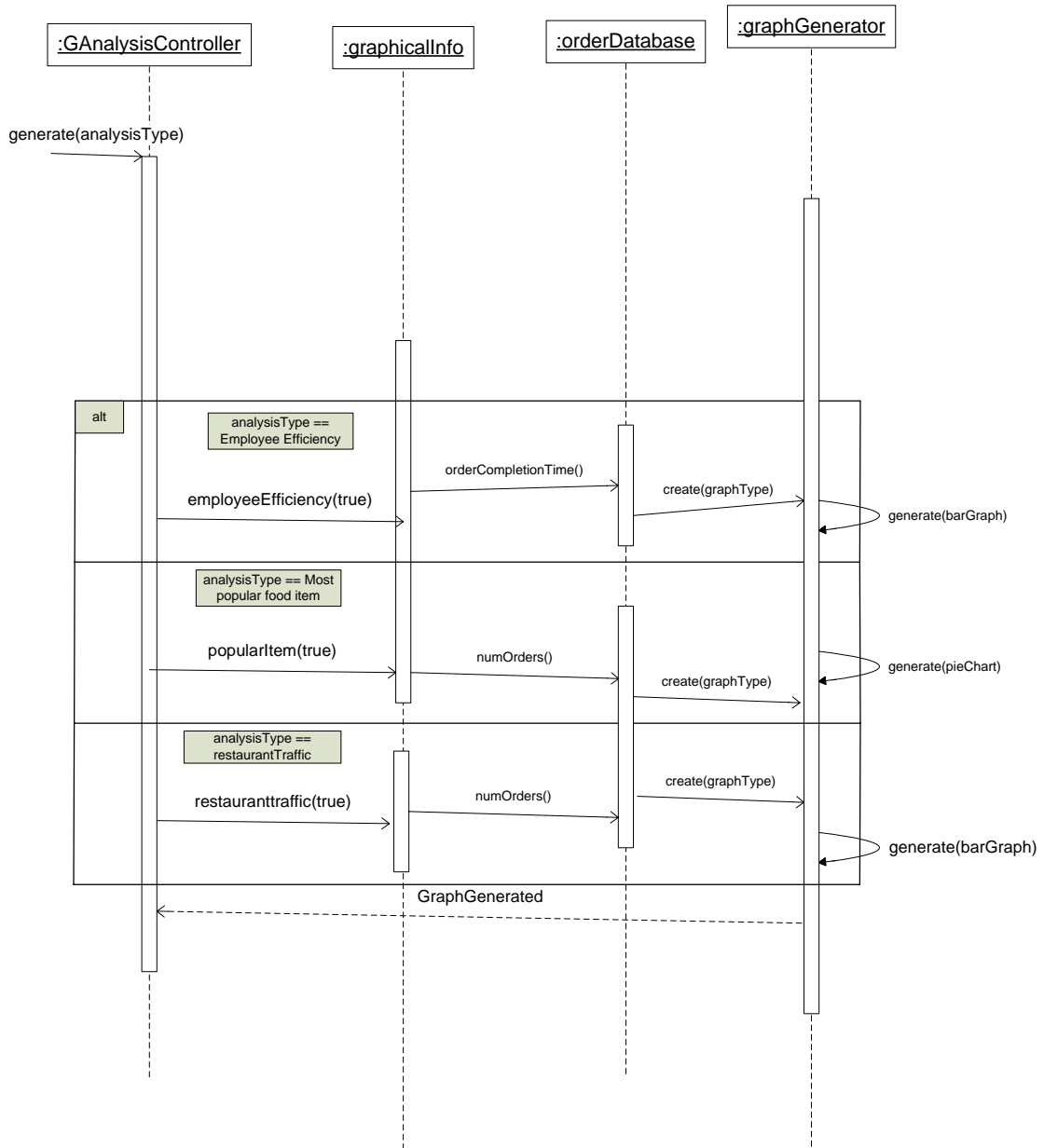
- (i) The manager enters the FoodID of the item that has to be deleted from the FoodDatabase.
- (ii) The MenuItemController receives the FoodID and makes a call, `ifExists()`. It sends it to the `itemChecker` which checks if the FoodID exists in the database by comparing it with each FoodID present in the database.
- (iii) If the FoodID does not exist, the signal “Exists” is set to false and a prompt “FoodID doesn’t exist” is sent to the user.
- (iv) If the FoodID does exist, the signal “Exists” is set to true and the item is deleted from the database.
- (v) Once the item is deleted, a prompt “Item Deleted” is sent to the user.

UC6 Paycheck

6. Interaction Diagram: UC6 Paycheck

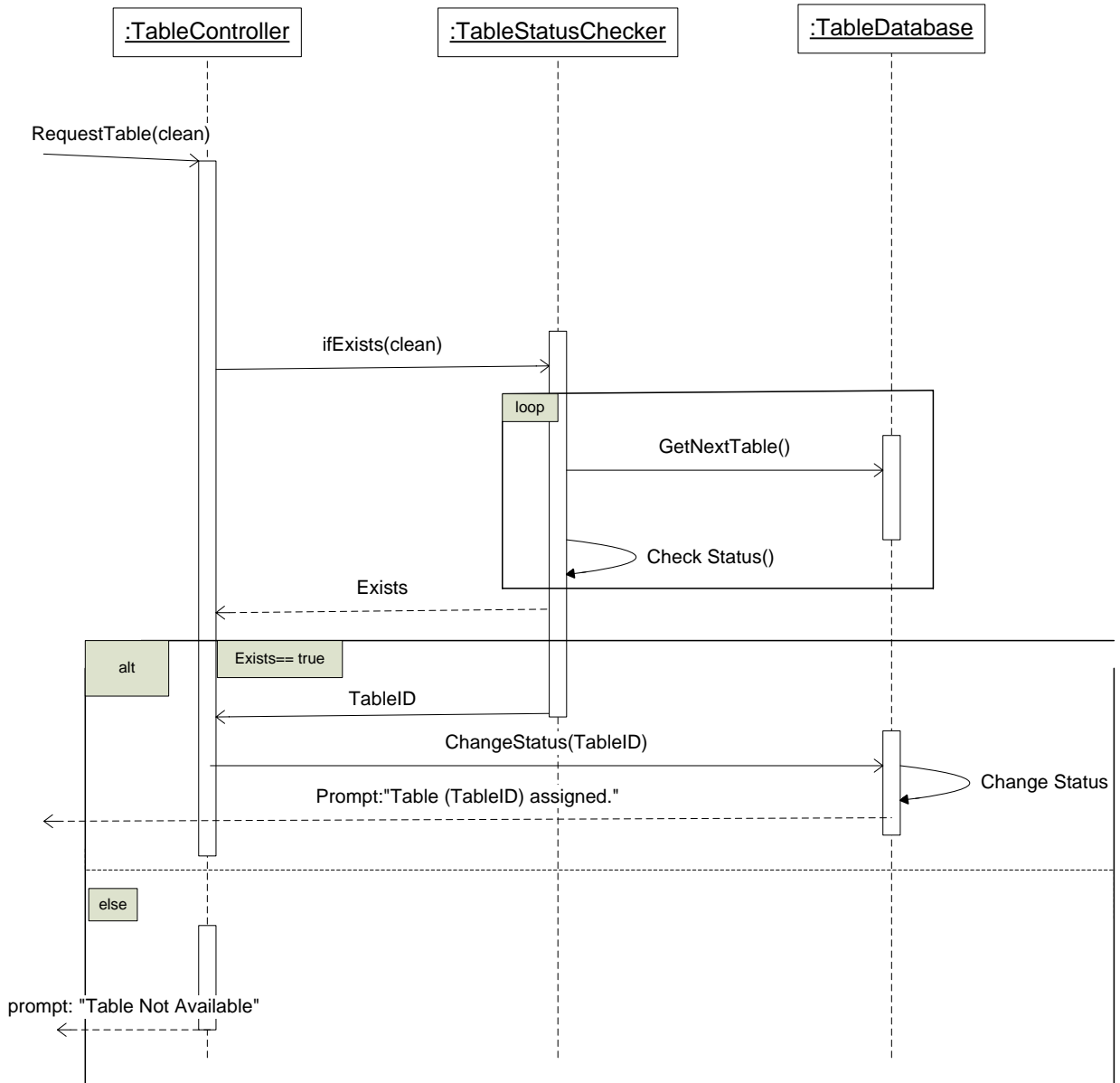
- (i) The manager cuts paychecks for the employees.
- (ii) The number of hours worked are provided to the empaccountController and employee information is checked to see whether the employee exists in the database or not.
- (iii) Next, if the employee exists in the database, the wage () signal is sent to the database to retrieve the wage of the particular employee.
- (iv) Finally, the wage is calculated and stored back in the employeeInfo and the paycheck is displayed to take a print out.

UC7 – Graphical Analysis



7. Interaction Diagram : UC7- Graphical Analysis

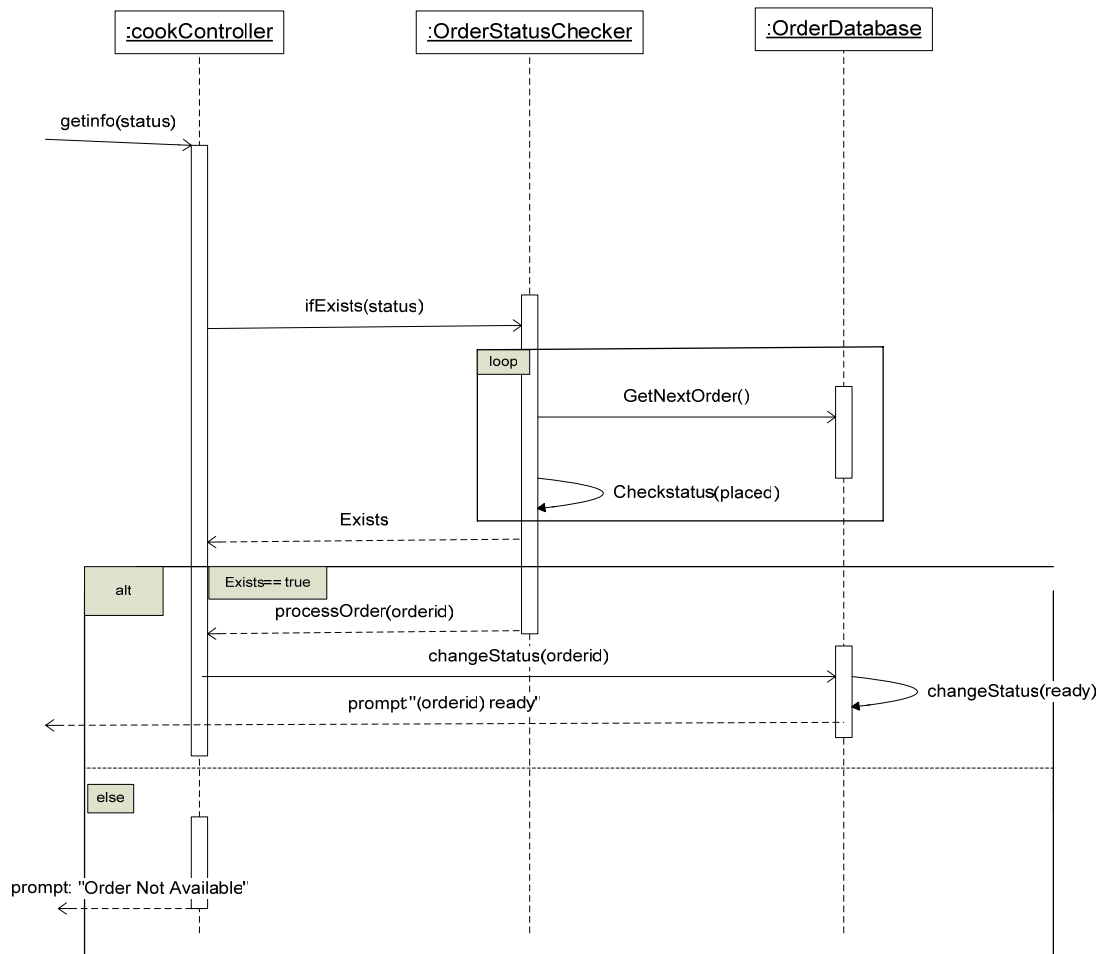
- i. The GAnalysisController gets the generate(analysisType) request to generate graphs for certain data that is collected by the system.
- ii. If the analysisType is for finding employee efficiency, the order database is looked up to find the turnaround time to serve an order by the particular employee is calculated and a bar graph is generated by sending the create(graphType) command to the graphGenerator class.
- iii. If analysisType is favorite food item then we need the number of orders placed for a certain food item in the order table and a piechart is created by sending create() to the graph generator.
- iv. If the analysis type is restaurant traffic for a certain day of the month, then the order database is looked up to find the total number of orders that were placed that day and it is then graphed for a certain period of time.

UC8 – SeatCustomer

8. Interaction Diagram : UC8- SeatCustomer

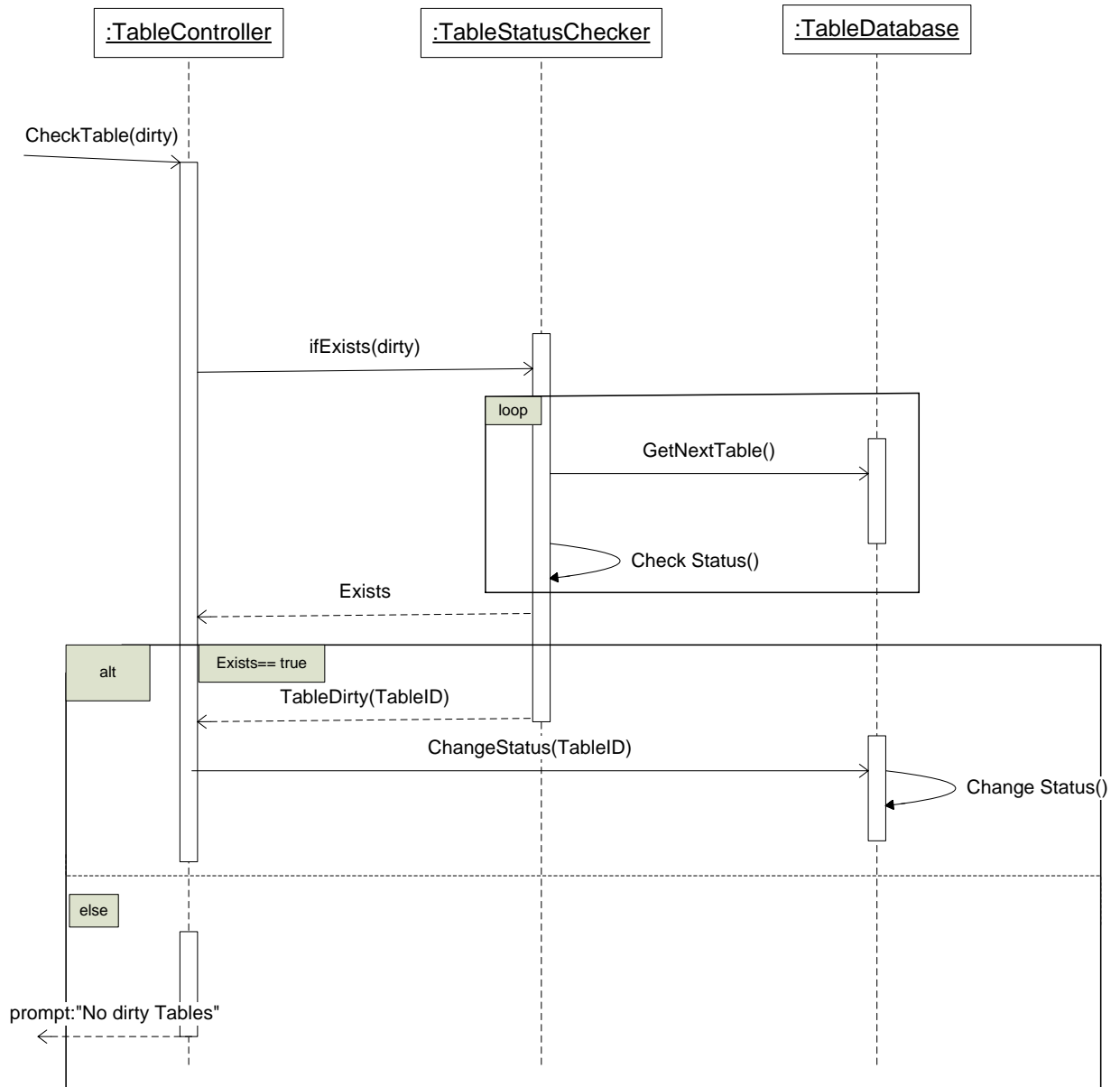
- (i) The host makes a call RequestTable(clean) to the Table Controller. Clean defines the status of the table.
- (ii) The Controller receives the required status and makes a call, ifExists(). It sends it to the TableChecker which checks the Table database to see if any table has the status “clean”.
- (iii) If such a table does not exist, the signal “Exists” is set to false and a prompt “Table not available” is sent to the host.
- (iv) If a clean table does exist, the signal “Exists” is set to true and the TableStatusChecker sends the corresponding table ID to the TableController.
- (v) The TableController makes a call ChangeStatus() and sends it to the TableDatabase with the TableID.
- (vi) The TableDatabase changes the status and the TableController sends a prompt “Table (TableID) assigned.”

UC9 – Preparing Order



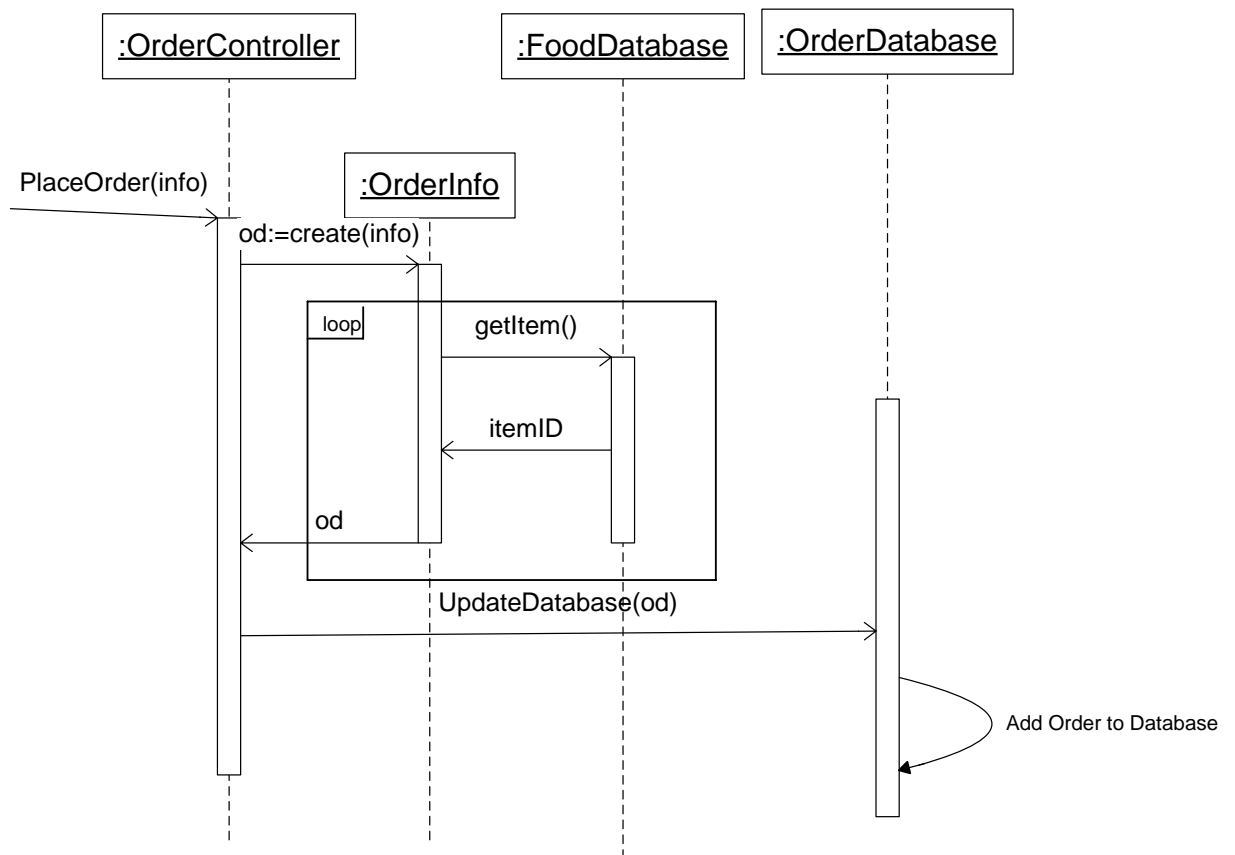
9. Interaction Diagram : UC9- Preparing an Order

- i. Cook controller gets information about the status of an order, whether the order is placed by the waiter and is still pending to be cooked.
- ii. The order status checker checks whether the order exists or not and gets the next order from the order database.
- iii. The OrderStatusChecker then checks what the status of the order is. If it is “placed”, the cook will process the order.
- iv. The OrderStatusChecker then changes the order status in the Order database to “prepared”.

UC10 clean

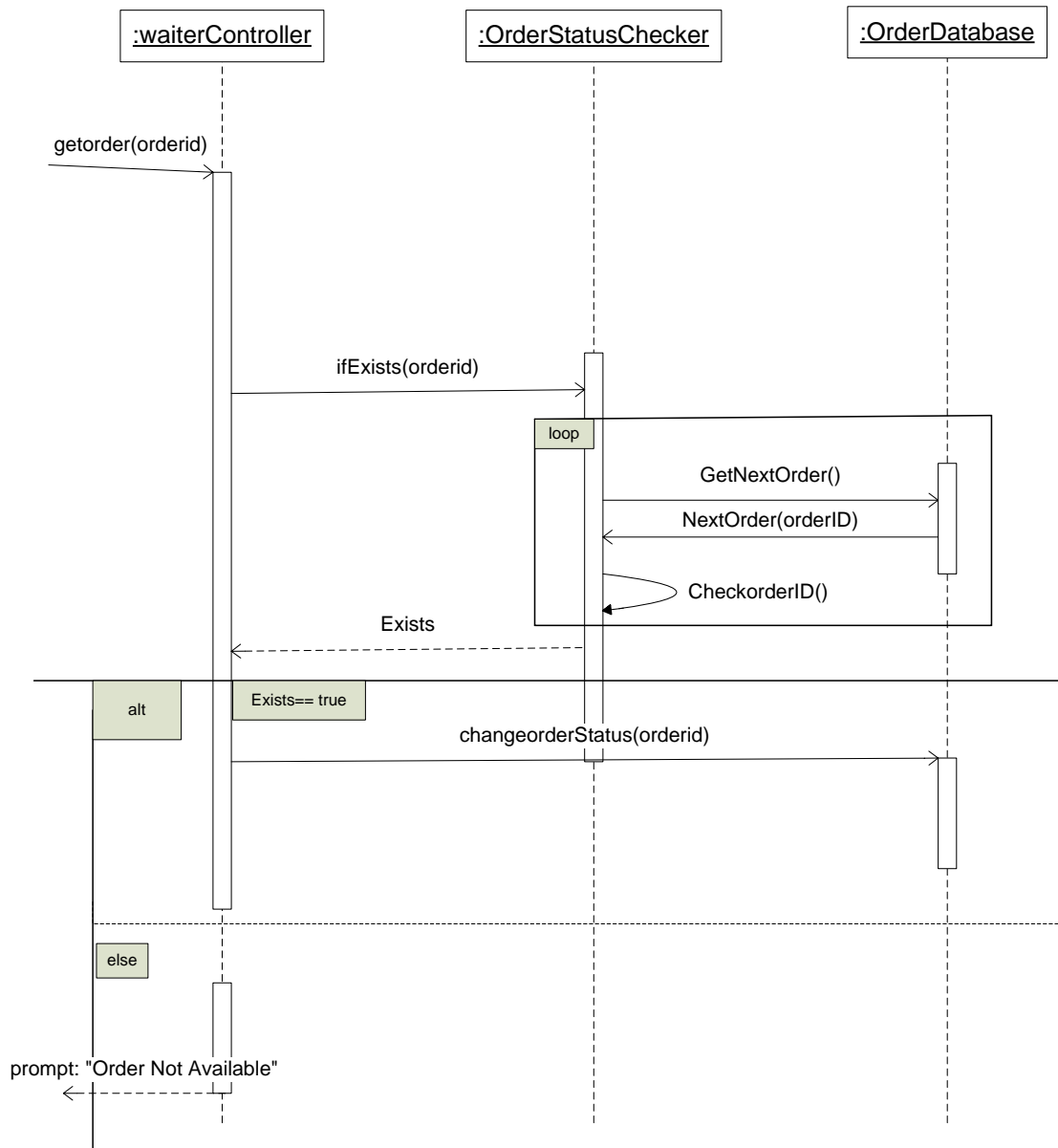
10. Interaction Diagram: UC10 clean

- (i) The busboy makes a function call `checktable()` that checks for dirty tables in the database and returns the first ID that it gets.
- (ii) The busboy then cleans the table and then changes its status to clean.
- (iii) If there are no dirty tables, the prompt, “No Dirty Tables” is returned to the screen.

UC11 Order

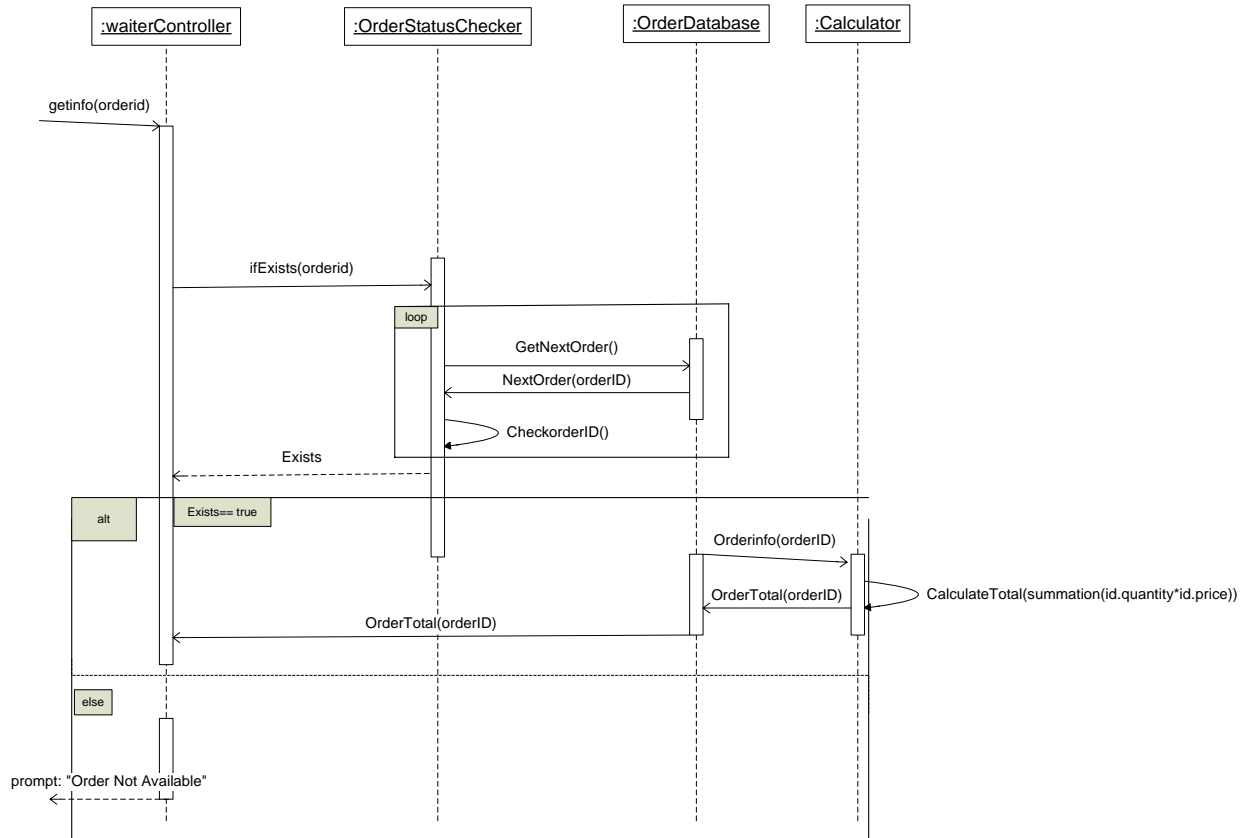
11. Interaction Diagram: UC11 Order

- i. The waiter makes a call PlaceOrder() and passes it to the OrderController.
- ii. The Controller then creates an instance of the order called “od” and sends it to the OrderInfo.
- iii. The instance collects all fooditemIDs from the foodDatabase.
- iv. The OrderInfo makes a call UpdateDatabase() and passes the instance “od” to the Database.
- v. The Order Database is then updated with the new order.

UC12 Deliver

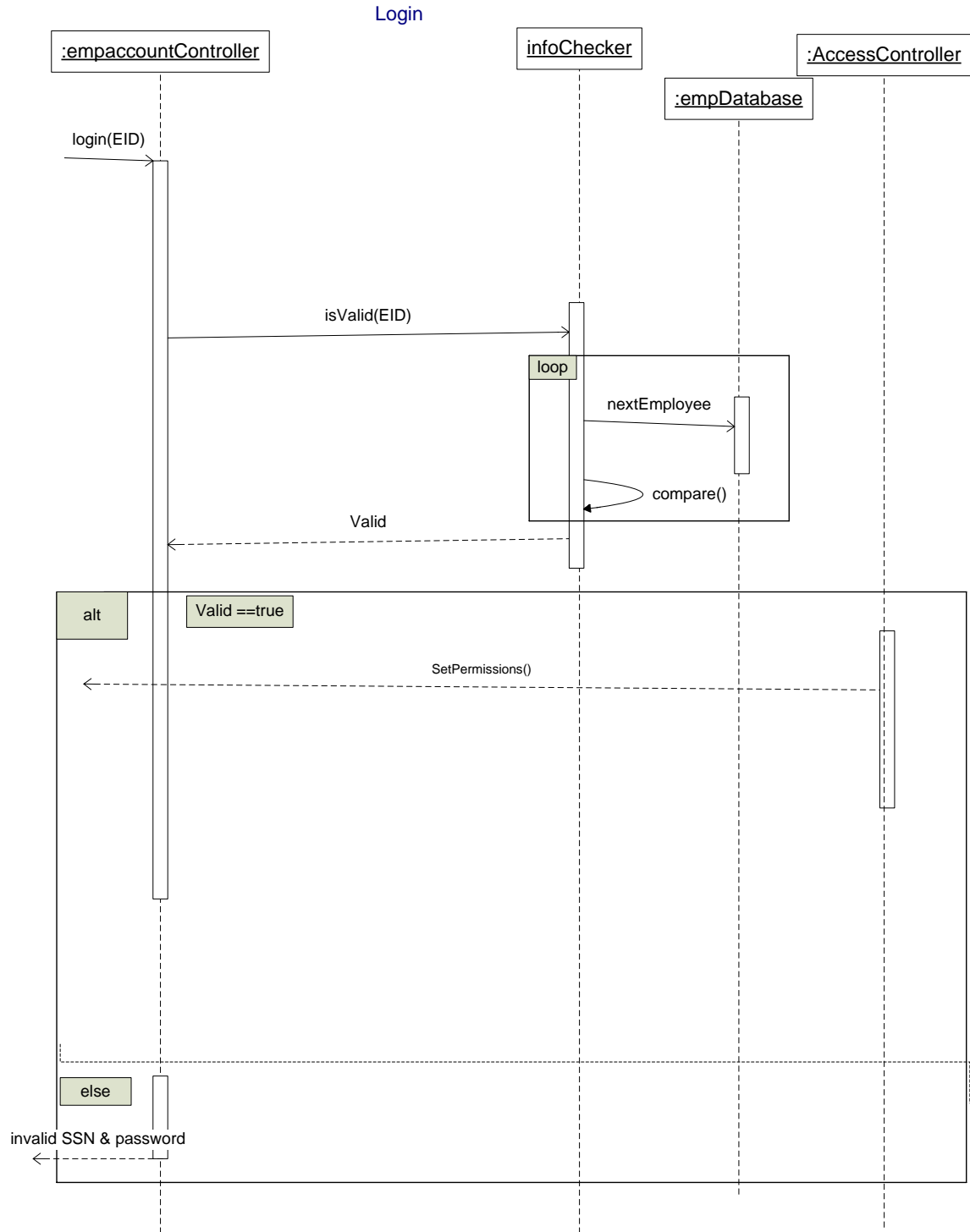
12. Interaction Diagram: UC12 Deliver

- (iv) After delivering the order to the table, the waiter makes a call to the waiter controller getOrderID()
- (v) The order status checker checks to see if the order exists in the database and if it does, return a “true” value.
- (vi) The waiter then makes a call changeorderstatus()

UC13 PayBill

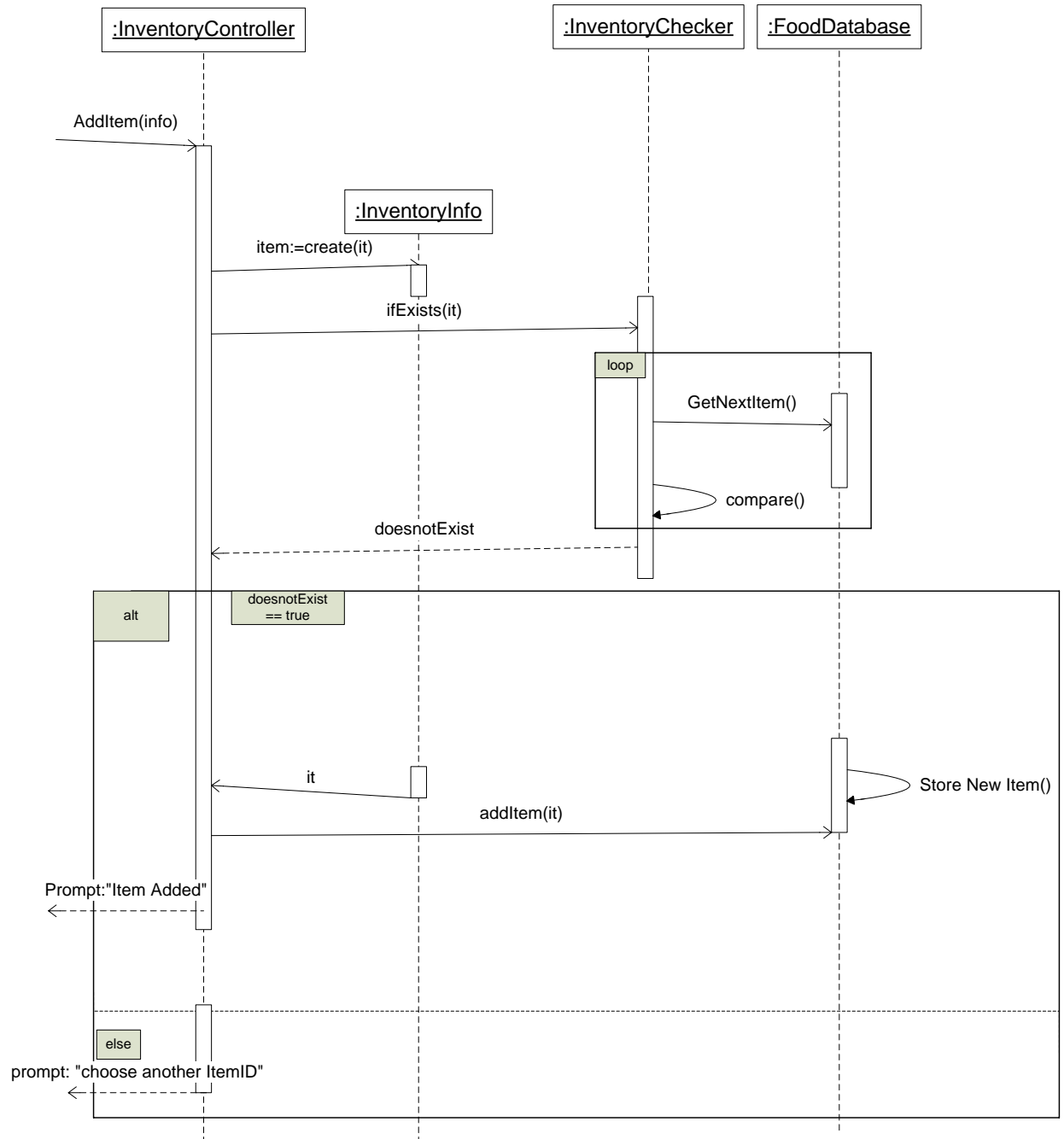
13 Interaction Diagram: UC13 PayBill

- i. The waiter collects the payment made by the customer for their order. The WaiterController gets information about a certain orderid in getinfo(orderid)
- ii. The OrderStatusChecker checks whether the order exists or not in the OrderDatabase.
- iii. If the order exists, the Calculator class calculates the order total and sends OrderTotal(orderid) to the waiterController.
- iv. If the order does not exist, an “Order Not Available” prompt is sent to the waiter interface.

UC14Login

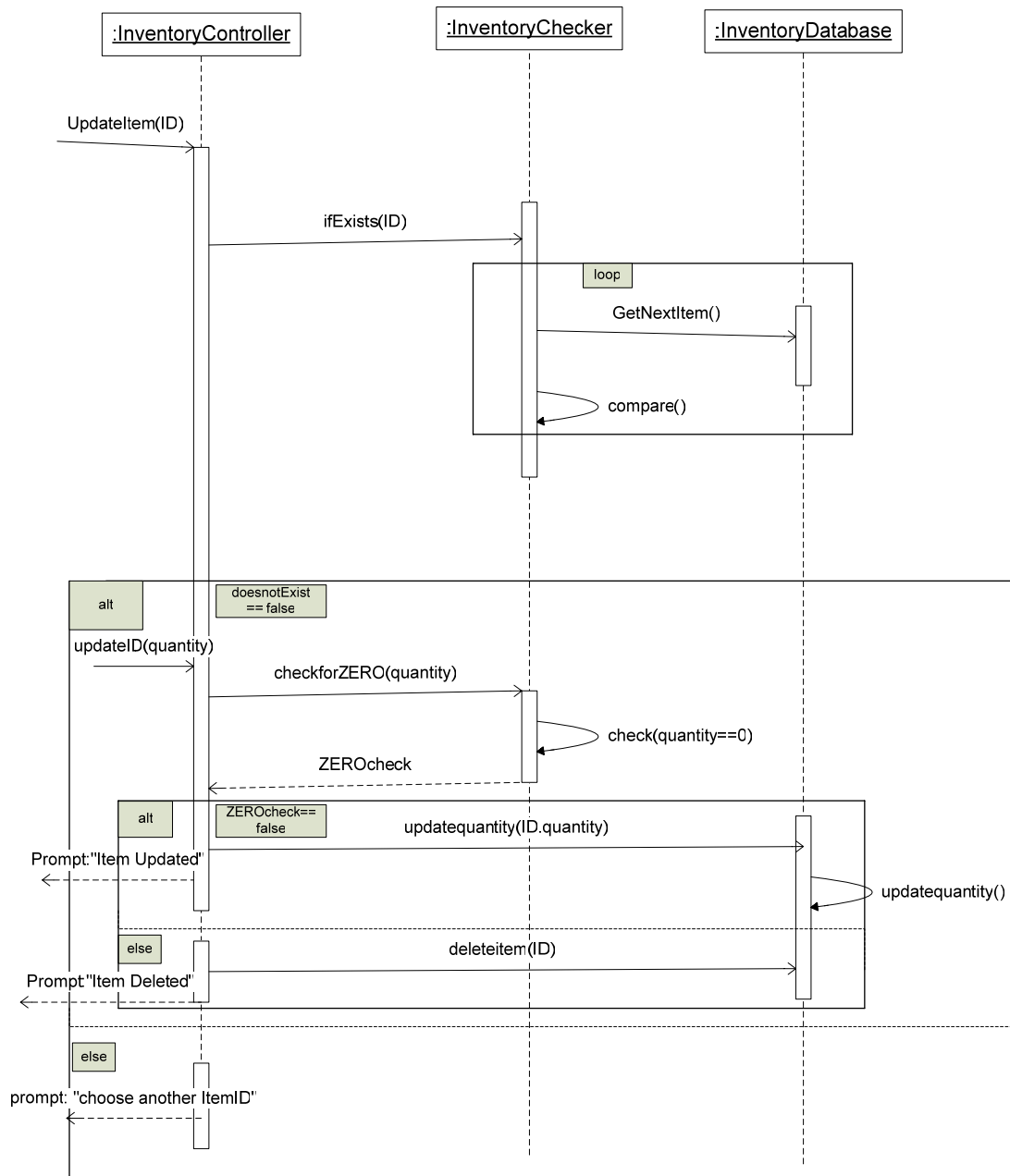
14 Interaction Diagram: UC14Login

- (vii) Employee provides login () information like his SSN.
- (viii) The empaccountController makes a call isValid() and passes it to the infoChecker which compares the SSN to each SSN in the employee database.
- (ix) If the SSN is not correct, the Controller sends back a prompt saying that the SSN and Password are invalid.
- (x) If the SSN and the password are valid. The valid signal is set to true and the information is then sent to the page access controller which sets permissions regarding what pages of the system that particular employee can access.

UC15 addInventory

15 Interaction Diagram: UC15 AddInventory

- (i) The manager can add a new item to the inventory database.
- (ii) Manager picks an itemID for the new item and sends it to the InventoryController along with the name, and quantity of the item.
- (iii) The InventoryController makes an instance of Item called "it".
- (iv) The InventoryController then makes a call `ifExists ()` and passes it to the InventoryChecker to see if the item already exists in the InventoryDatabase.
- (v) InventoryChecker then makes a call to the InventoryDatabase to get information about the items in the menu one by one and compares it with the information provided for the new Item.
- (vi) If the comparison results in a match for any item already in the database, the signal "DoesNotExist" is set to false and a prompt "choose another itemID" is displayed. If there is no match, the signal "DoesNotExist" is set to true and the InventoryController makes a call `AddItem()` and passes it to the InventoryDatabase, where the item is stored.

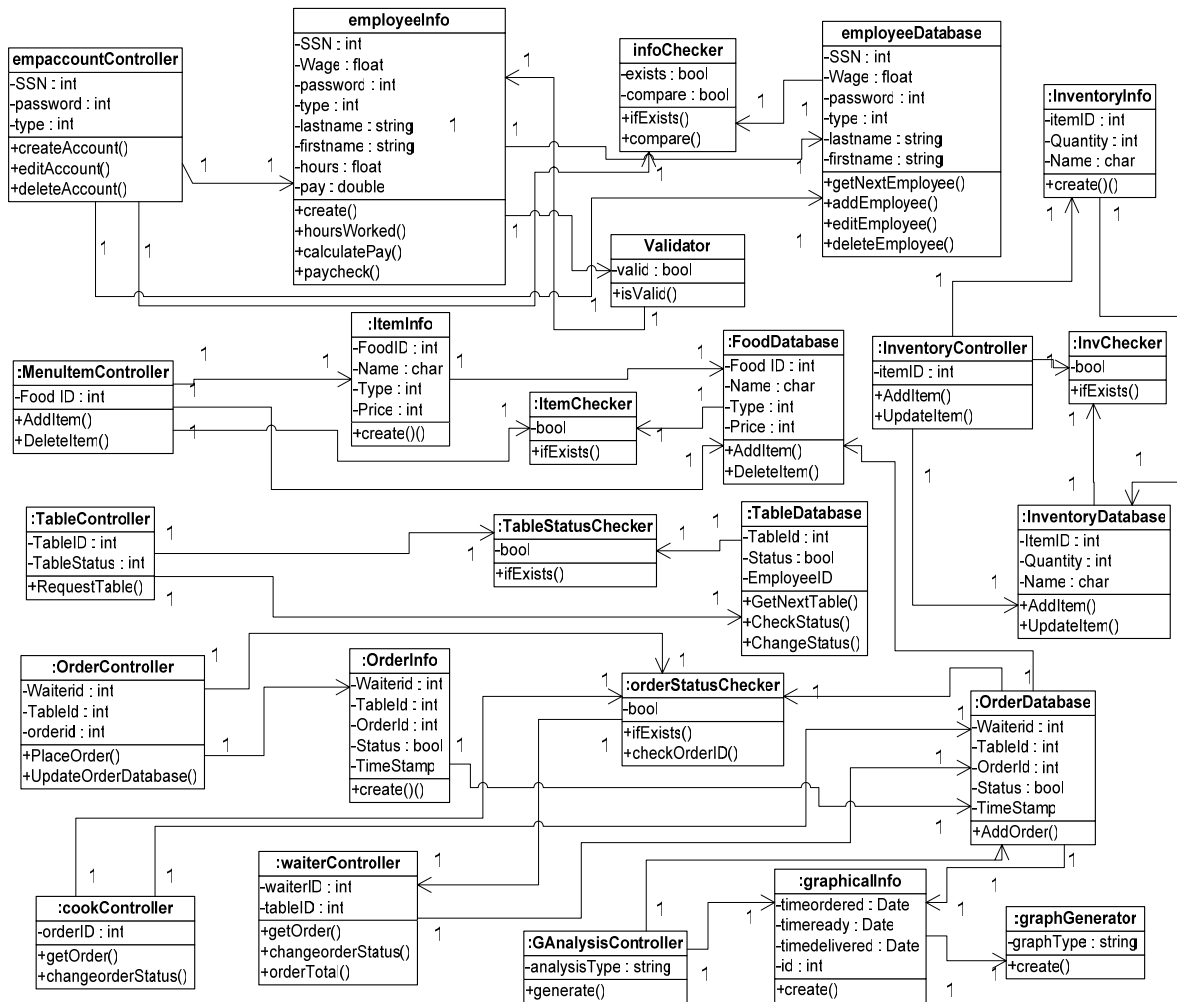
UC16 Update/DeleteInventory

16. Interaction Diagram: UC16Update/DeleteInventory

- (i) Manager makes a call Updateitem() and sends the ID of the item to be updated.
- (ii) The InventoryController then makes a call ifExists () and passes it to the InventoryChecker to see if the item already exists in the Inventory Database.
- (iii) InventoryChecker then makes a call to the InventoryDatabase to get information about the items in the menu one by one and compares it with the information provided.
- (iv) If the comparison results in a match for any item already in the database, the signal “DoesNotExist” is set to false.
- (v) The manager then enters the quantity to be changed. The InventoryChecker checks if the quantity entered is zero, if it is, the item is deleted.
- (vi) If it is not zero, the item is updated to the quantity specified.
- (vii) If the item does not exist in the database, a prompt “choose another itemID” is displayed.

10. CLASS DIAGRAM AND INTERFACE SPECIFICATION

Class Diagram



Description

Class diagrams are the mainstay of object-oriented analysis and design. UML class diagrams show the classes of the system, their interrelationships (including inheritance, aggregation, and association), and the operations and attributes of the classes.

The simple UML class diagram above is the conceptual model of the restaurant automation system we will be implementing. Classes are depicted as boxes with three sections, the top one indicates the name of the class, the middle list is for the attributes of the class, and the third one lists the methods. Some of the methods have a return type mentioned.

Some of the classes that will be used for our system are as follows:

- employeeAccountController
- Validator
- infoChecker
- employeeInfo
- employeeDatabase
- MenuItemController
- Iteminfo
- FoodDatabase
- ItemChecker
- OrderController
- Order Database
- Order Info
- TableController
- TableStatusChecker

Explanation of some of the Classes Used:

1. employeeAccountController: This class controls the use cases that are applicable to Employee Accounts. This controller is available to Manager, where the manager can Add new Employees, edit existing employee information and delete an employee from the employee data base.
2. Validator: This class is used to check the information entered either by the Manager of the employees is correct or not. It returns a Boolean value stating whether the information entered from the user interface is relevant or not for a particular operation.
3. infoChecker: This class is used to get employees information from the database and compare the information retrieved from the database to the information provided in the user interface. For example in case we want to delete an employee

- from our database the infoChecker checks whether the Social Security Number for the employee exists in the Employee Database or not.
4. employeeInfo: This class is used to create an employee's account, also to calculate the wage for that employee and cut a paycheck.
 5. employeeDatabase: Has methods that deal with the database for the employee information.
 6. MenuItemController: This class controls the use cases that are applicable to the menu/food database. This controller is available to Manager, where the manager can Add new Items, or delete existing items from the database.
 7. ItemInfo: This class is used to create a new item.
 8. FoodDatabase: Has methods that deal with the database for the food items available for the restaurant.
 9. ItemChecker: This class is used to get Food item information from the database and compare the information retrieved to the information provided in the user interface. For example in case we want to delete an item from our database, the itemChecker checks whether the FoodID for the item exists in the Food Database or not.
 10. OrderController: This class controls the use cases that are applicable to the orders database. This controller is available to the waiter and the cook. The manager can also access it, but usually does not need to add to it. The waiter adds new orders to the order table, and the cook sets the status symbol when the order is ready to serve.
 11. Order Database: Has methods to deal with adding new order information.
 12. OrderInfo: This class is used to create a new order object.
 13. TableController: This class controls the use cases that are applicable to the table database. This controller is available to the Host and the waiters. The manager can also access it but does not usually modify it. The host uses it to assign tables to new customers.
 14. TableStatusChecker: This class is used to get Table information from the database and compare the information retrieved to the information provided in the user interface. For example if the Host needs to assign a table for some new customers, he can use this to check for a clean empty table.

The classes have access specifiers placed in front of their methods which define whether the methods are private, public or protected. These access specifiers determine who can use the definitions that follow.

- + is for public: These members/methods are available to everyone.
- # is for protected: These members/methods are available to an inheriting class.
- - is for private: These members/methods are available to only the class they are created in and no other place.

For simplicity sake we have defined our methods as public. For a good design there should be low coupling and high cohesion. By coupling we mean that dependency of one module on other modules. A good design should have lower dependencies and hence low coupling. Our design has classes that are less dependent on other classes which can be seen by the number of dotted lines, which are very few in our design. Also another good design principle is to have high cohesion in the design. Cohesion is a measure of how well the lines of source code within a module work together to provide a specific piece of functionality. In our class diagram we have provided modules that will be expert at their functionality. High cohesion is preferred since it provides robustness, reliability, reusability and easy understandability for the system. Thus we have considered these software quality metrics in our design.

Therefore, the above design is the best suited for our purposes and would yield the results desired with high software quality.

Object Constraint Language

Contract CO1: createAccount

Operation: createAccount (SSN: integer, password: integer, wage: float, type: integer, lastName: string, firstName: string)

Cross References: createAccount UC1

Pre Conditions : The employee Social Security Number, wage, password, type, last name and first name should be provided in the correct format and using the correct type. If employee does not have an SSN then a unique identification number should be provided to the system to create his account.

Post Conditions:

- Employee SSN instance was created for a particular employee
- Employee password instance is created
- Assigned wage, employee type and employee name in the database fields.
- A message sent to user interface that the account creation was successful

Contract CO2: editAccount

Operation: editAccount (password: password, wage: float, type: integer, lastName: string)

Cross References: editAccount UC2

Pre Conditions: An account exists for the employee

Post Conditions:

- password: password , reset the employee password in the database
- wage: wage, update employee wage in the database
- type: type, update employee type in the database
- lastName: string, update the last name of the employee
- A message sent to user interface that the account has been updated

Contract CO3: deletingAccount

Operation: deleteAccount (SSN: integer, password: password)

Cross References: deletingAccount UC3

Preconditions: The employee should have an existing account in the database

Postconditions:

- SSN identifies employee whose account has to be deleted
- password: password to abandons the earlier password for that employee.
- Employee information fields in the database are deleted
- A message is sent to the user interface of the successful deletion of the employee account

Contract CO4: Add Item to Menu

Operation: AddItem (FoodID: integer, Name: char, Type: Integer, Price: Integer)

Cross References: Add Item To Menu UC4

Preconditions: The FoodID should not be in use by any other Food Item already present in the database.

Postconditions:

- FoodID: is unique so each item can be searched for by it. It can be passed as the search parameter when an item needs to be deleted from the database.
- A name, type and price has been assigned for each FoodID.
- A prompt showing the successful completion of the change is sent to the user.
- Only the Manager can add to or delete from the table.
- An order can be placed only from items in the Food Database.

Contract CO5: Delete Item From Menu

Operation: deleteItem (FoodID: integer)

Cross References: Delete Item From Menu UC5

Preconditions: The Item should exist in the database for it to be deleted.

Postconditions:

- FoodID identifies the specific item that the manager wants to delete.
- All Item information fields like name, type, price are deleted along with the foodID.
- That FoodID can now be used again when a new item is entered.
- A message is sent to the user interface of the successful deletion of the item.

Contract CO6: paycheck

Operation: hoursWorked (SSN: integer, password: password, hours: hours)

Cross References: paycheck UC 6

Preconditions: The employee should have an existing account in the database

PostConditions:

- Hours worked are multiplied by wage for that employee to calculate pay for the particular period
- Paycheck is created and stored in the database
- A print out of the paycheck is taken

Contract CO7: Graphical Analysis

Operation: generateGraph(timeOrdered:DateTime, timeReady:DateTime, timeDelivered:DateTime, orderid:int)

Cross References: Graphical Analysis UC 7

Preconditions:

- In the order table in the database, the information about the time the order is placed by the waiter, time the order is made ready by the cook and time the waiter takes to deliver it to the customer has been already entered by the respective employees.
- Also the order ids exist in the database for the food ordered in the restaurant

PostConditions: Graphs are generated for employee efficiency, favorite food item and the restaurant traffic.

Contract CO8: SeatCustomer

Operation: RequestTable (status: integer)

Cross References: SeatCustomer Table UC 8

Preconditions: The employee using this has to be the Host.

PostConditions:

- The request will only work if there is a clean, ready to use table in the database.
- If there isn't a clean, ready to use table, the request is denied.
- The host will have to send the request again if he still needs to assign a new table.

Contract CO9: Preparing Order

Operation: GetInfo(OrderID:integer, Status:Boolean, TimeStamp:DateTime, TableID: Integer, FoodID: integer)

Cross References: Preparing Order UC 9

Preconditions: The waiter must already have entered the order for the cook to be able to see it.

PostConditions:

- The status is set by the cook after the meal is prepared.

Contract CO10: Clean

Operation: CheckTable(Status:Boolean)

Cross References: Clean UC 9

Preconditions: There are dirty tables in the restaurant. There is a busboy.

PostConditions:

- The table has been cleaned by the busboy.
- The table status has been changed by the busboy after cleaning.

Contract CO11: Order

Operation: Order (CookID:integer, Status:Boolean, TimeStamp:DateTime, TableID: Integer, FoodID: integer)

Cross References: Preparing Order UC 11

Preconditions: Only the waiter can place orders. The Priority override field has to be unique to each order.

PostConditions:

- The status is set by the cook after the meal is prepared.
- The orders will be processed according to their status in the “priority override” field.

Contract CO12: Deliver

Operation: getOrder (OrderID:integer, Status:Boolean, TableID: Integer)

Cross References: Deliver UC-13

Preconditions: The food has been prepared by the cook, and its status changed to ready.

PostConditions:

- The waiter has delivered the food.
- The status should be changed by the waiter after delivery.

Contract CO13: PayBill**Operation:** GetInfo(orderID:integer, FoodID:int, Price: float, TableID: Integer)**Cross References:** PayBill UC 13**Preconditions:** The order has been delivered to the customer.**PostConditions:**

- The waiter takes the Payment from the customer.

Contract CO14: Login**Operation:** login (EmployeeID:integer, Password:string, type:string)**Cross References:** Login UC 14**Preconditions:**

- The employee has already been added to the database by the manager.
- The employee knows his/her username, password and job type.

PostConditions:

- The employee is taken to his or her page which links to all the pages needed to perform his/her job.

Contract CO15: addInventory

Operation: AddItem(itemID:integer, name:string, quantity:integer)

Cross References: addInventory UC-15

Preconditions: The itemID should not be present in the database.

PostConditions:

- The new item is added and the system updated.

Contract CO16: Update/Delete Inventory

Operation: UpdateItem (itemID:integer, quantity:integer)

Cross References: Update/Delete Inventory UC-16

Preconditions: The item has to exist in the database.

PostConditions:

- If the quantity entered is 0, the item has to be completely deleted from the database.
- If the quantity was a non zero number the quantity should have been updated to the number entered.

11. System Architecture and System Design

Architectural Styles

There are various architectural styles that we could have utilized in creating this application. The first thought was to use the Repository Architectural Style in which subsystems access and modify a central repository. Each of the subsystems are independent from each other and only interact with the data through the central repository. This style was quickly thrown out because of the high coupling found between the subsystems and the repository. We would not be able to change the repository or subsystems very easily. In the end, it is not portable and changes to the user interface would be difficult to implement quickly.

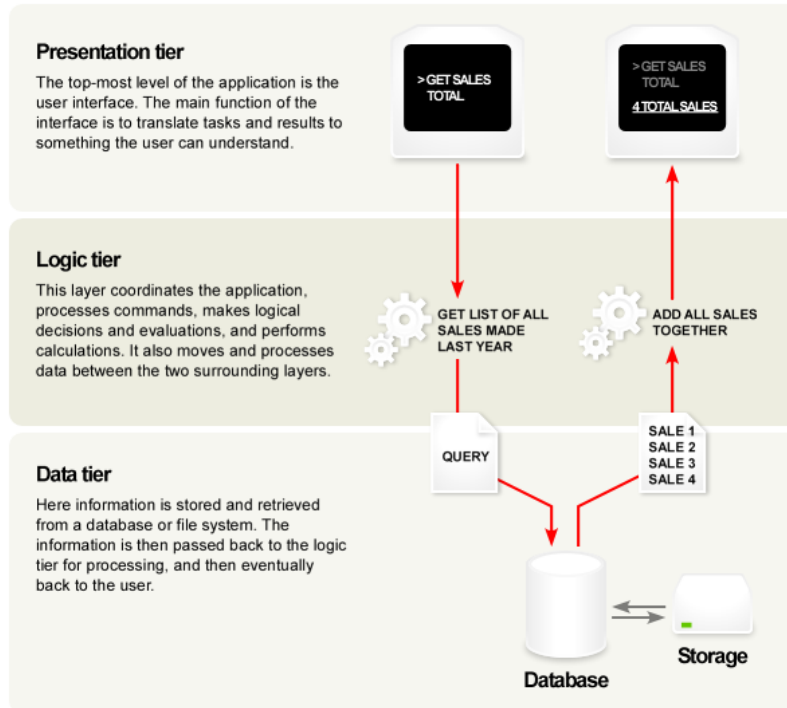
The second architectural style we investigated was the Model/View/Controller or MVC Architectural Style. In this architectural style, we separate all the subsystems into three categories. The model subsystems maintain the data of the application, the view subsystems display and format the data to the user, and the controller subsystems are responsible for managing the interactions between user and system. The model subsystem should be independent of the view and control subsystems. This is a very good possible architectural style for our application. It allows changes to the user interface to be very easily implemented. We will see later why this architectural style was not chosen.

The last architectural style we looked at was the Three-Tier Architectural Style; which is very similar to the MVC Architectural Style. The subsystems are once again, separated into three layers:

- Interface Layer – boundary objects that deal with the user (User Interface)
- Application Logic Layer – control and entity objects
- Storage Layer – the database

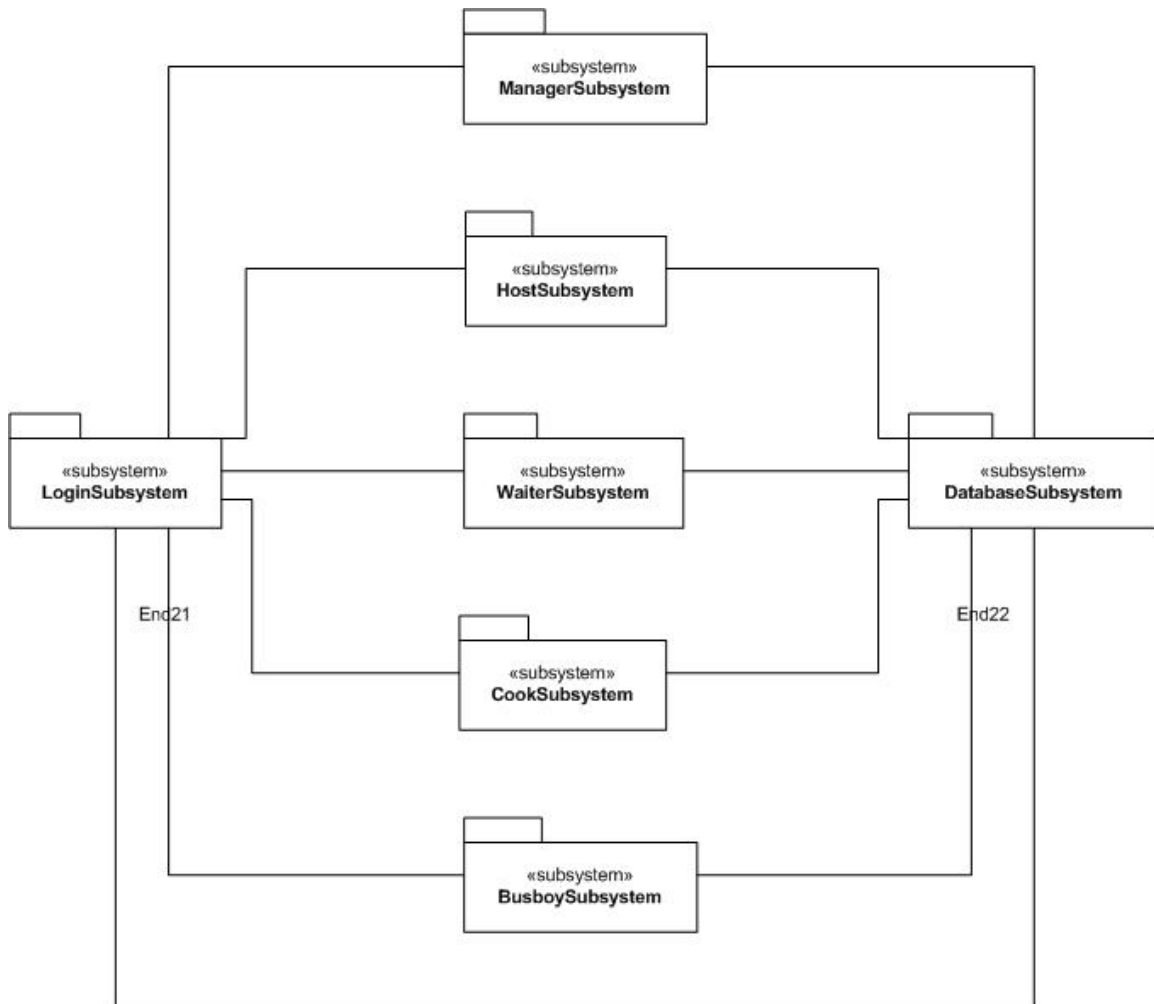
Once again, partitioning all the layers allows changes to be made to them very easily. We can change the User Interface very easily without having to make changes to the database or how we deal with the data.

The main difference between the MVC and Three-Tier Architectural Style is how the data is relayed back to the user. MVC follows a triangular path in which the model subsystems directly update the view subsystems. An example of the dataflow would be a user making a request through the controller which passes the request to the model subsystem which then updates the view. On the other hand, Three-Tier follows a linear path of events. All requests from the user have to pass through the interface which then goes through the application logic which then submits that request to the database. In other words, the user never has direct access to the database. The following diagram taken from Wikipedia (http://en.wikipedia.org/wiki/Image:Overview_of_a_three-tier_application.png) should help visualize how the Three-Tier Architectural Style works:



In the end, we chose to use the Three-Tier Architectural Style because we are developing a web based application built on PHP/MySQL. Natively, PHP is not a framework that is based on the MVC Architectural Style. It is much closer to the Three-Tier Architectural Style because all requests are sent to the server from the client. The server is responsible for interpreting the request and then forwarding it to the database. The database sends the data back to the server which then formats that information and sends it to the client. At no point, is the server circumvented in the interaction between user and data. This is a linear flow of events and not a triangular flow which the MVC Architectural Style utilizes.

Identifying Subsystems



- LoginSubsystem is responsible for keeping track of the Time-in and Time-out for the various employees. It will also send the employee to the relevant part of the application for their job type.
- ManagerSubsystem is responsible for presenting information and operations to the manager of the restaurant. From here, they can change various tables in the DatabaseSubsystem. The manager can also create various reports from the data that is stored in the DatabaseSubsystem.
- HostSubsystem is responsible for displaying the various tables in the restaurant and their states to the Host. The Host can seat customers at the various tables and change their status.
- WaiterSubsystem is responsible for providing the various operations to the waiter. He can place orders for the various tables, check on his order status, and tally up the bill.
- CookSubsystem is responsible for displaying the queue of orders to the cooks in the restaurant. It is a FIFO queue of orders and upon completion they change the status of the order for pickup.

- BusboySubsystem is responsible for informing the busboys when a table is ready to be cleaned up. When they are finished they change the status of the table to be ready.
- DatabaseSubsystem is responsible for providing access to the MySQL Database to the various other subsystems. It is also in charge of storing the data in a predetermined schema.

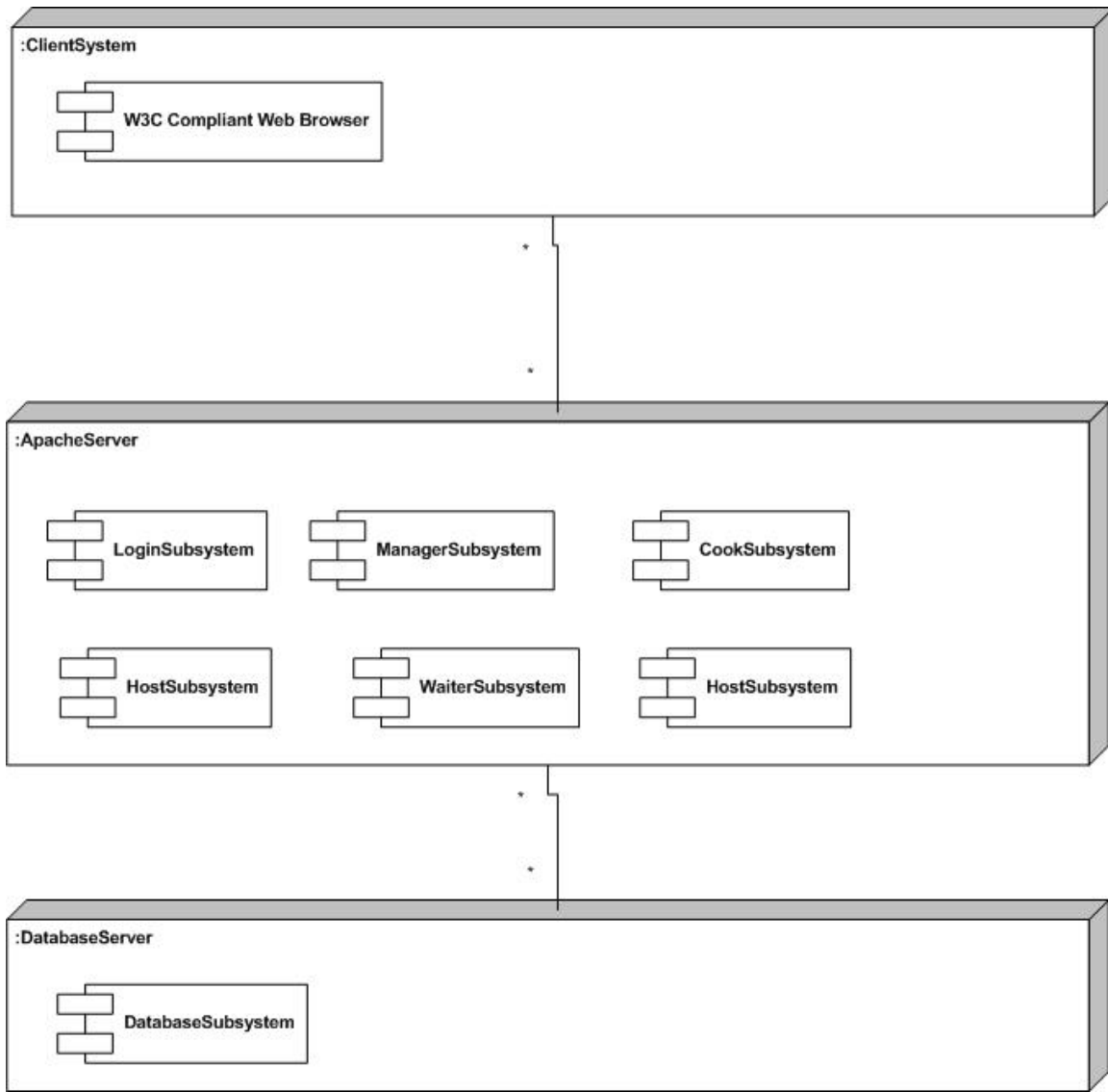
The various classes listed in the previous section are contained within these different subsystems. The subsystems are important for keeping functionally related objects together.

Subsystem	Classes
ManagerSubsystem	employeeAccountController, employeeInfo, MenuItemController
DatabaseSubsystem	Validator, infoChecker, employeeDatabase, ItemInfo, FoodDatabase, ItemChecker, OrderDatabase
WaiterSubsystem	OrderController
CookSubsystem	OrderInfo
BusboySubsystem	TableStatusChecker
HostSubsystem	TableController

Mapping Subsystems to Hardware

The application consists of three distinct virtual machines. The first one is the client computer which is running a W3C Compliant Web Browser. The client computer can be a normal computer, Pocket PC, or PDA. The second virtual machine is the Apache Webserver which is serving up the various HTML/PHP pages to the client computer across the network. The last virtual machine is the MySQL Database which serves up the data to the Apache Webserver.

UML Diagram:



Persistent Data Storage

Our application will definitely need to save data that will be persistent and accessible to various different clients at any time. The application will need to store various data including a list of employees, list of food, orders, tables, and payroll. The best way to organize the data is to utilize a relational database such as MySQL. The reason for this is that relational databases are good for when you have concurrent access and possibly multiple applications accessing the same data. In this case, we might have waiters, hosts, cooks, managers, and busboys accessing the database at once. We would also like the ability to extend upon the database with other programs. In addition, we will be running various queries for various reports such as most popular dish, payroll, and busiest hours. A good database schema is very important to the operation of the application. A bad schema will result in unneeded calls to the database which will result in poor performance. It is important that the schema is created and standardized. It will allow future expansions to the application.

The persistent objects are as follows:

- `_order`
- `Orderxfoodxaddon`
- `Food`
- `Addon`
- `foodXAddon`
- `employee`
- `_table`
- `Payroll`

The `_order` table keeps track of specific orders, what table they belong to, which waiter is taking care of it, its status, and many other things detailed in the following database schema. This is the base table for any orders that will occur in the restaurant.

The `orderxfoodxaddon` table is the items that are contained in an order. As the waiter adds items to an order, a new record is created with that item in this table.

The `food` table keeps track of all the items that the restaurant sells and includes a field for the price of the item. In addition, the `addon` and `foodXAddon` tables are used to list what addons are available and which ones are valid for specific food items.

The `employee` table is a listing of all the employees at the restaurant. We keep records of what type of employee they are and their wage. We also store their login password here.

The `_table` table is for assigning the specific tables to specific waiters and keeping track of the status of the table.

The `payroll` table is to keep track of individual employee's hours and to calculate the payroll for the week.

All these persistent objects are tables that can be found in the database schema. They contain various records that are important and should be available to users of the application.

The schema is attached:

DATABASE SCHEMA

Tables
employee
_table
_order
food
addon
ingredients
foodxaddon
orderdetail
orderxfoodxaddon
ingredientsxinventory
foodxingredients
payroll

Details to follow...

employee

Field	Type	Key	Default	Extra
id	int(5)	PRI		auto_increment
fname	varchar(20)			
lname	char(2)			
wage	float		0	
password	int(4)		0	
type	varchar(20)			

_table

Field	Type	Key	Default	Extra
id	int(5)	PRI		auto_increment
employeeid	int(5)			
status	varchar(20)			

_order

Field	Type	Key	Default	Extra
id	int(5)	PRI		auto_increment
timeordered	datetime		0000-00-00 00:00:00	
timeready	datetime		0000-00-00 00:00:00	
timedelivered	datetime		0000-00-00 00:00:00	
waiterid	int(5)		0	
tableid	int(5)		0	
status	varchar(20)			
total	float			

food

Field	Type	Key	Default	Extra
id	int(5)	PRI		auto_increment
name	varchar(20)			
price	float		0	
type	varchar(20)			

addon

Field	Type	Key	Default	Extra
id	int(5)	PRI		auto_increment
name	varchar(20)			
price	float		0	

ingredients

Field	Type	Key	Default	Extra
id	int(11)	PRI		auto_increment
name	varchar(20)			

foodxaddon

Field	Type	Key	Default	Extra
foodid	int(5)		0	
addonid	int(5)		0	

orderdetail

Field	Type	Key	Default	Extra
orderid	int(5)			
foodid	int(5)			
addonids	varchar(20)			
quantity	int(5)			
status	varchar(20)			
id	int(5)	PRI		auto_increment

orderxfoodxaddon

Field	Type	Key	Default	Extra
orderid	int(5)		0	
foodid	int(5)		0	
addonid	varchar(20)			

ingredientsxinventory

Field	Type	Key	Default	Extra
ingredientid	int(11)		0	
curr_quantity	int(11)		0	

foodxingredients

Field	Type	Key	Default	Extra
foodid	int(11)		0	
ingredientid	int(11)		0	
ingredient_quantity	int(11)		0	

payroll

Field	Type	Key	Default	Extra
id	int(5)		0	
timein	datetime		0000-00-00 00:00:00	
timeout	datetime		0000-00-00 00:00:00	

The fields are color coded according to secondary key restrictions
 For example in the table orderxfoodxaddon the field orderid is color-coded in orange
 the id field in the order table is also in orange. This means that the field orderid in this table
 is a secondary key based on the primary key of the orders table which is the id field

Network Protocol

Since this is a web based application utilizing PHP/MySQL; we will be running on an Apache server. All communication between the clients and server will be done over the HTTP protocol. Any client that can run a web browser will be compatible with the application. For example, the application can be easily extended to Pocket PC's/PDA's that have a built in web browser. They just need to go to the website and the browser will render the client appropriately for the browser. The application will detect the user agent and change the formatting appropriately for the browser type. For example, a PDA will have a smaller screen than a normal PC. So the website will need to be rendered in a smaller size.

PHP has a built in library for communicating with MySQL databases. It is much more efficient and faster than opening an ODBC connection. We will be using this built in library for communications between the application and its backend MySQL database.

Global Control Flow

Execution Orderness

This application is an event-driven system that waits for user input before doing anything. When an event occurs, it is dispatched to the system with the correct information. The advantage is a simpler structure and the centralizing of all inputs into a main loop. The difficulties arise when a sequence requires multiple steps to complete. Each individual user of the application can generate actions for each request. For the most part, the events in our application do not require multiple steps and can be easily completed at once. They do not depend on different actions from different users. Consequently, there are no timers or need for concurrency in the application.

Time Dependency

There are no timers in the application since it is an event-response system.

Concurrency

The system does not use multiple threads.

Hardware Requirements

The hardware requirements for the application are very low. The most important parts will be a fast hard drive for the MySQL database to reside on and a good network. The client part of the application is very portable since the only required piece of software is a W3C Compliant Web Browser. Microsoft Windows, *Nix, and Mac OS all have browsers that will work with the application.

What follows are the minimum specifications for the various computers and other hardware:

Server Requirements

Minimum CPU – P4/AMD Athlon 1.6GHz+
Minimum Disk Space – 2GB
Minimum Memory – 512MB
Minimum Network – 10/100MB Network
Required Software – Apache, PHP, MySQL

Client Requirements

Minimum CPU – P3/AMD Athlon 1.0 GHz+
Minimum Disk Space – 512MB
Minimum Memory – 256MB
Minimum Network – 10/100MB Network
Minimum Display – 800x600 Color CRT
Required Software – W3C Compliant Web Browser (Internet Explorer)

PDA Requirements

Minimum Network – 802.11b Wireless Network
Minimum Display – 240x320 Color Touch Screen LCD
Required Software – W3C Compliant Web Browser (Pocket Internet Explorer)

Other Hardware Required

10/100 Switch
802.11b Wireless Access Point

Optional Hardware

Touch Screens

12. Algorithms and Data Structures

Algorithms

Our system does not use any complex algorithms

Data Structures

Our system does not use any complex data structures

13. User Interface Design and Implementation

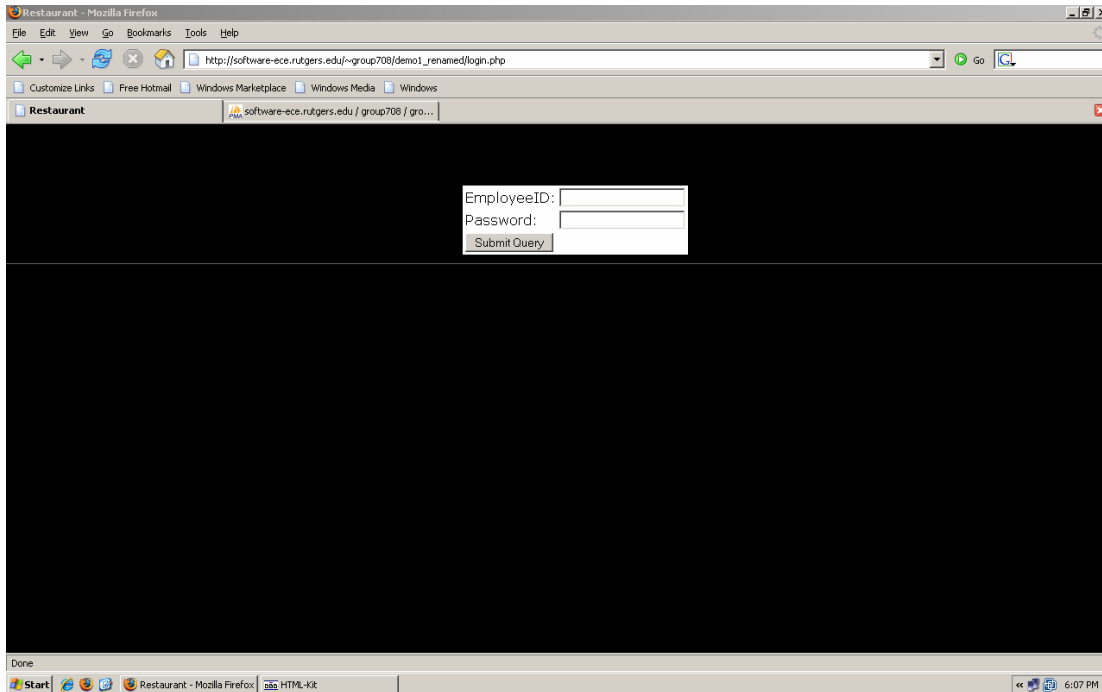
The primary goal of the interface is to be very simple and fast to use so that little time is wasted while inputting orders or preparing food. For the most part, the original interface mockups have been followed when being implemented in HTML. A majority of the choice menus will be dropdown so that waiters, busboys, hosts, and cooks will not need to use keyboards for most of the functions. Ideally, touch screens will be the main form of input. There will be no pictures or fancy graphics present so that the application will load quickly and lessen the load on the network and server. Simple customizations such as the name of the restaurant and color scheme will be easily changed to suit the customer. The color scheme can be controlled via CSS (Cascading Style Sheets) and will allow fast changes without needing to edit the underlying PHP code.

As we stated in the requirements section, the minimum supported screen resolution will be 800x600. The application will be formatted with this screen resolution in mind. All buttons and information should be viewable within the screen without excessive scrolling. In some cases, scrolling will be needed but for the most part will be restricted to administrative pages or non-essential data. Important functions, such as placing orders, are streamlined for the least amount of user effort. In order to simplify things, Internet Explorer can be run in Kiosk Mode.

A special interface for waiters will be implemented that can be utilized on wireless PDAs. The test unit will be a Dell Axim X5. This PDA has a screen resolution of 240x320 and uses a stylus for input. This interface will be very similar to the normal waiter screen. The application will automatically detect that the "User Agent" of the web browser is either Pocket Internet Explorer or Internet Explorer.

Table layout of the restaurant was an interesting issue that we discussed at length. We had two options available to us when implementing the table screen for the host, waiter, and busboy. Many of the groups decided to implement a layout screen that mimicked the actual layout of the restaurant. We; however, decided that the most efficient way to do this would be to have a static table layout which was numerically arranged. Our reasoning for this hinged on the fact that its faster to find tables if they are arranged numerically. Instead of having to pick out the specific table on a map of the restaurant you would just have to click that number table. In a real restaurant, all the tables are numbered and the waiters know the numbers for their assigned table. It is faster for them to find that number on the screen if it's in order rather than in a physical location type of order. This also reduces dependency on the physical layout of the restaurant or the need for the manager to constantly be there for any table layout changes. Simplicity was the main goal and this was achieved.

All users to the system will be presented with a Login Screen into which they will enter their Employee ID number and password. This screen will also serve as the Sign-In for payroll purposes. Every employee will also have a button to clock out of their shift. Both of these buttons will create records in order to calculate hours worked.



Manager

The manager will have access to four different functions and each will have its associated screen. There will be buttons for:

- Add/Edit/Remove Employees
- Manage Inventory
- Manage Payroll
- Reports Screen
-

The Add/Edit/Remove Employees screen will allow the manager to add new employees to the payroll. They can also change an employee's type so that they can fill a different role in the restaurant. In addition, they can remove an employee from the database.

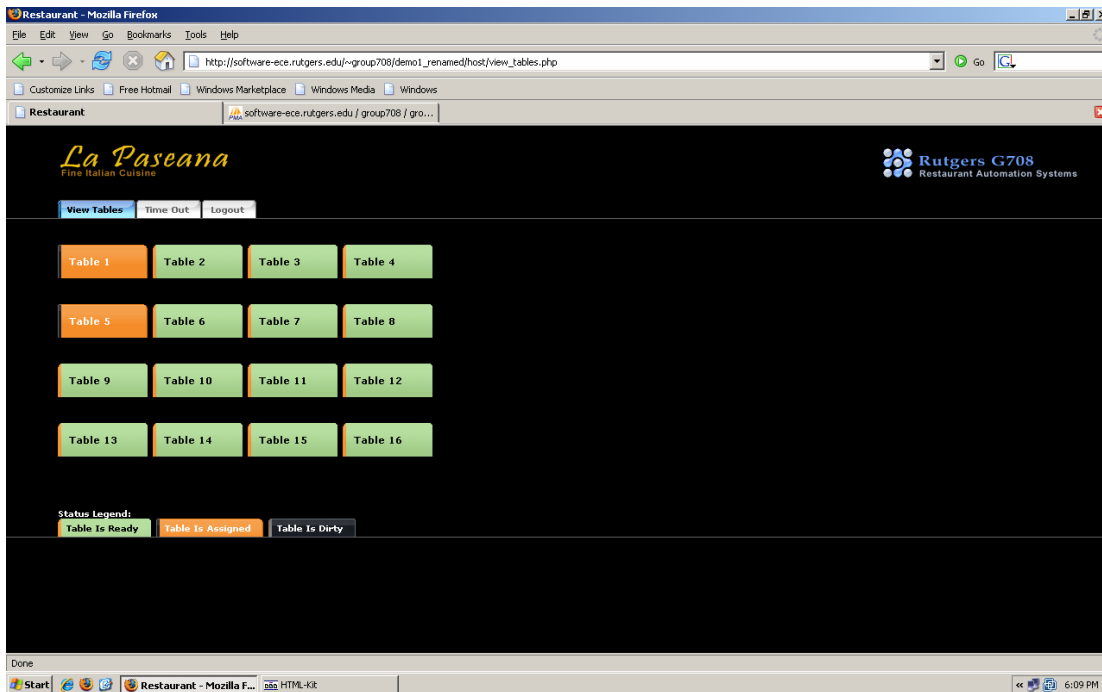
The Manage Inventory screen will have a list of items that the restaurant keeps on stock. It will have a box with the current amount. The manager can add/remove new items and change the amount.

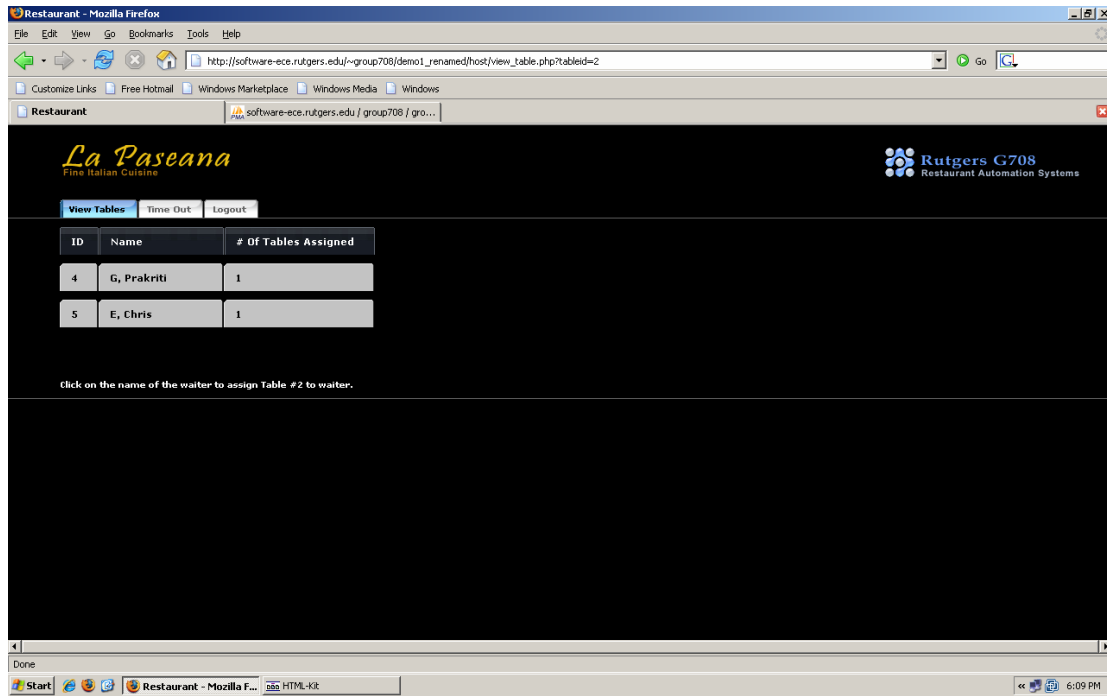
The Manage Payroll screen will show all the employees and how many hours they have worked during the week. In addition, it will have a field for their pay rate and it will calculate how much their paycheck should be at the end of the week.

The Reports Screen will show various charts and graphs of collected statistics such as hourly traffic flow into the restaurant and popularity of various dishes.

Host

The host has one screen which displays the various tables in the restaurant and their status. When they click on a table it will allow them to change the status of the table and seat customers.





The screenshot shows a Mozilla Firefox browser window displaying the La Paseana restaurant management interface. The browser's address bar shows the URL: `http://software-ece.rutgers.edu/~group708/demo1_renamed/host/view_table.php?tableid=2`. The page header includes the restaurant name "La Paseana" with the tagline "Fine Italian Cuisine" and the logo for "Rutgers G708 Restaurant Automation Systems". Below the header, there are three buttons: "View Tables", "Time Out", and "Logout". The main content area features a table with the following data:

ID	Name	# Of Tables Assigned
4	G, Prakriti	1
5	E, Chris	1

Below the table, there is a text instruction: "Click on the name of the waiter to assign Table #2 to waiter." The browser's status bar at the bottom shows "Done" and the system tray includes the Start button, taskbar icons for "Restaurant - Mozilla F..." and "HTML_KR", and a system clock showing "6:09 PM".

Waiter

The waiter has the most complex interface of the various users. Their main screen will display the various tables that they are responsible for in the restaurant. On the right side, they will have a Order Status screen which will alert them when to pickup orders from the kitchen. When they have delivered the order they just have to click on it and the status will change to Delivered. When they click on a table they will be presented with two options:

1. Place Order
2. Pay Bill

When they click on Place Order they will be see the Order Page and it is here that they can select various items from a drop down menu and input the quantity ordered. The order will be shown below that and by pressing the “-” they can remove items from the order. The order will be totaled at the bottom and they can place the order by pressing the “Place Order” button at the bottom right.

The Pay Bill button will change the status of the table to “dirty” and inform the busboy that the table needs to be cleaned.



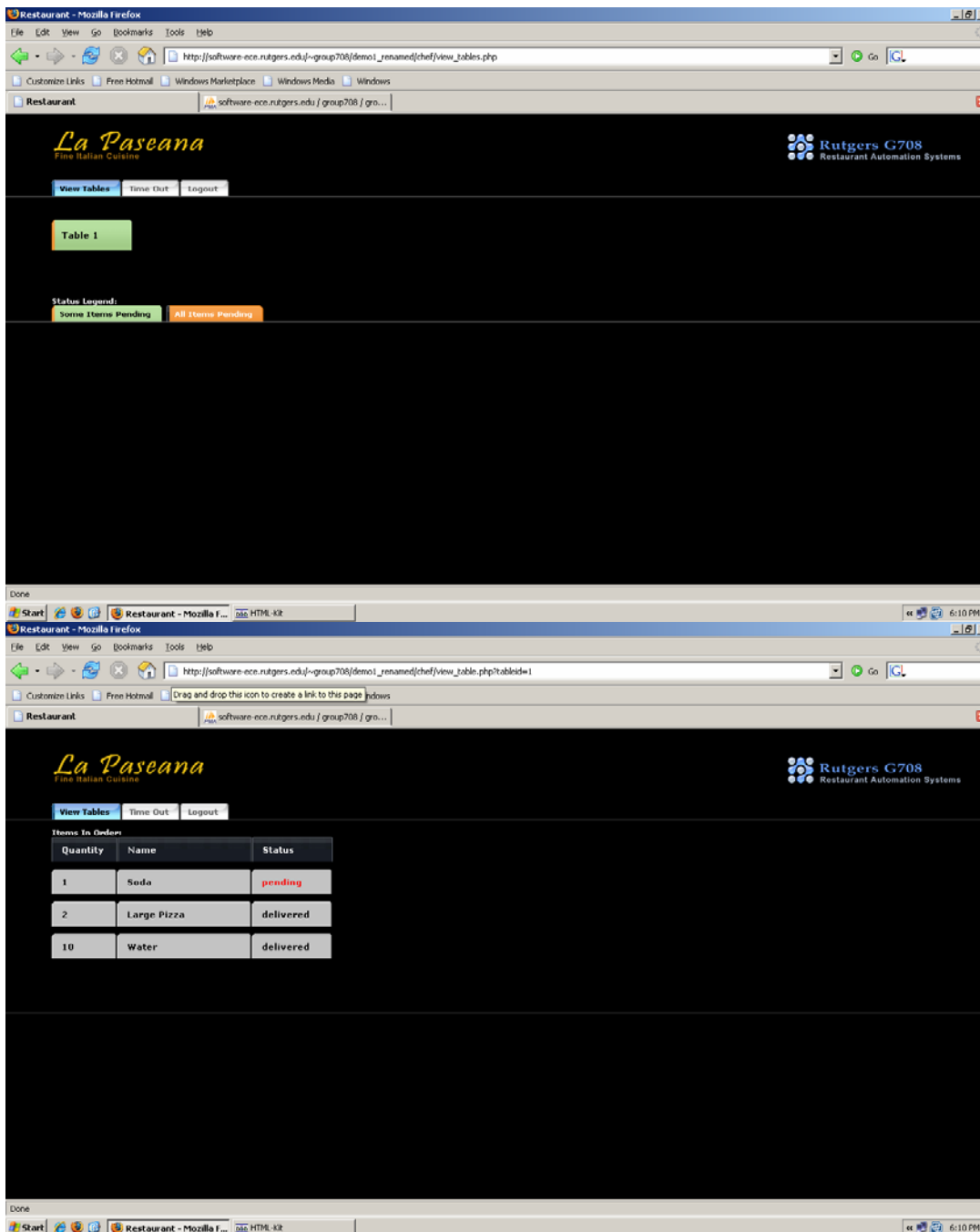
The screenshot shows a Mozilla Firefox browser window displaying a web application for a restaurant named "La Peseana". The browser's address bar shows the URL: `http://software-ece.rutgers.edu/~group708/demo1_renamed/water/view_table.php?tableid=5`. The page header features the restaurant's logo "La Peseana Fine Italian Cuisine" on the left and the "Rutgers G708 Restaurant Automation Systems" logo on the right. Below the header, there are navigation buttons: "View Tables", "Time Out", and "Logout". The main content area is titled "Add Item to Order" and contains a table with the following structure:

Type	Name	Quantity
Select One	Select Type First	1

Below the table, there is an "Add" button and a "Mark As Dirty" button. The browser's status bar at the bottom shows "Done" and the system tray includes the Start button, a taskbar with "Restaurant - Mozilla F...", and a system clock showing "6:08 PM".

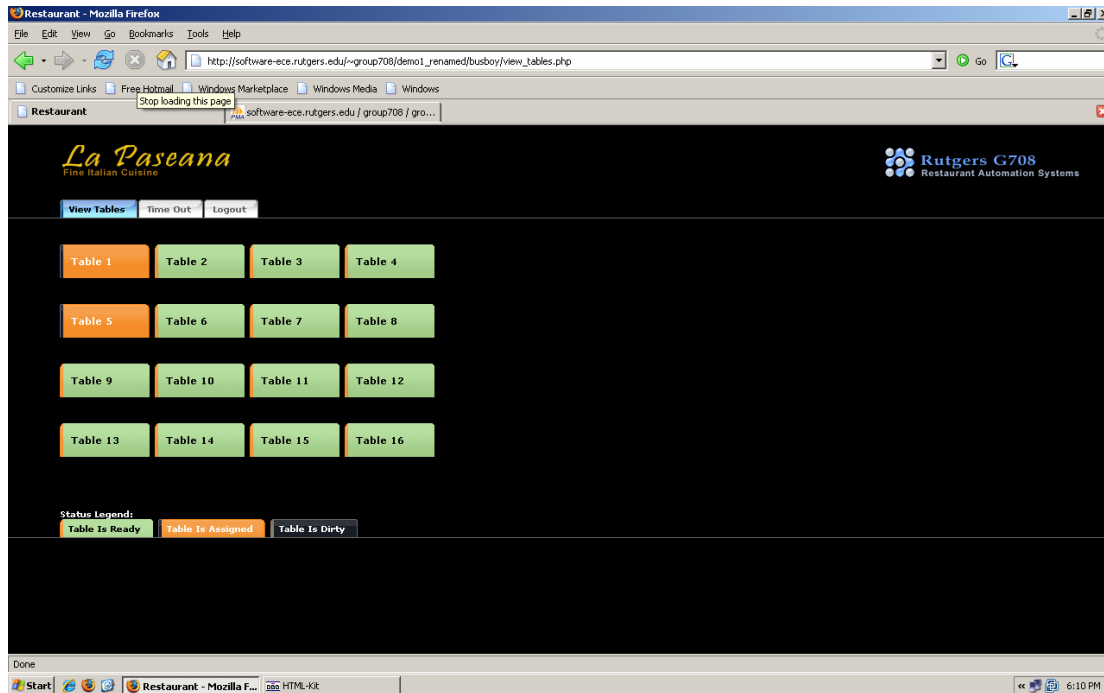
Cook

The cook's screen consists of a FIFO queue that displays the various orders and presents them with buttons to show the details of the order and a button to signal that the order is complete and ready for delivery. We will also keep track of how long it takes to complete the orders. Once the order is delivered it will be removed from the queue.



Busboy

The busboy's view is very similar to the host's view. The main difference is that there will be a list on the right showing the tables that need to be cleaned. This will also be a FIFO queue. By clicking on the table, the busboy will change the status of the table to “ready”.



User Effort Estimation

Placing an Order

- Navigation: total 3 mouse clicks, as follows
 - Click Appropriate Table Icon
 - Click “Place Order” Button
 - --- Complete Data Entry ---
 - Click “Place Order” Button
- Data Entry: total 3 mouse clicks and 1 keystroke multiplied by the number of items, as follows:
 - Select Item from Drop Down (2 mouse clicks)
 - Input Quantity (Default 1 otherwise press “#” key)
 - Click “Add to Order”
 - Repeat for each item

Signing In

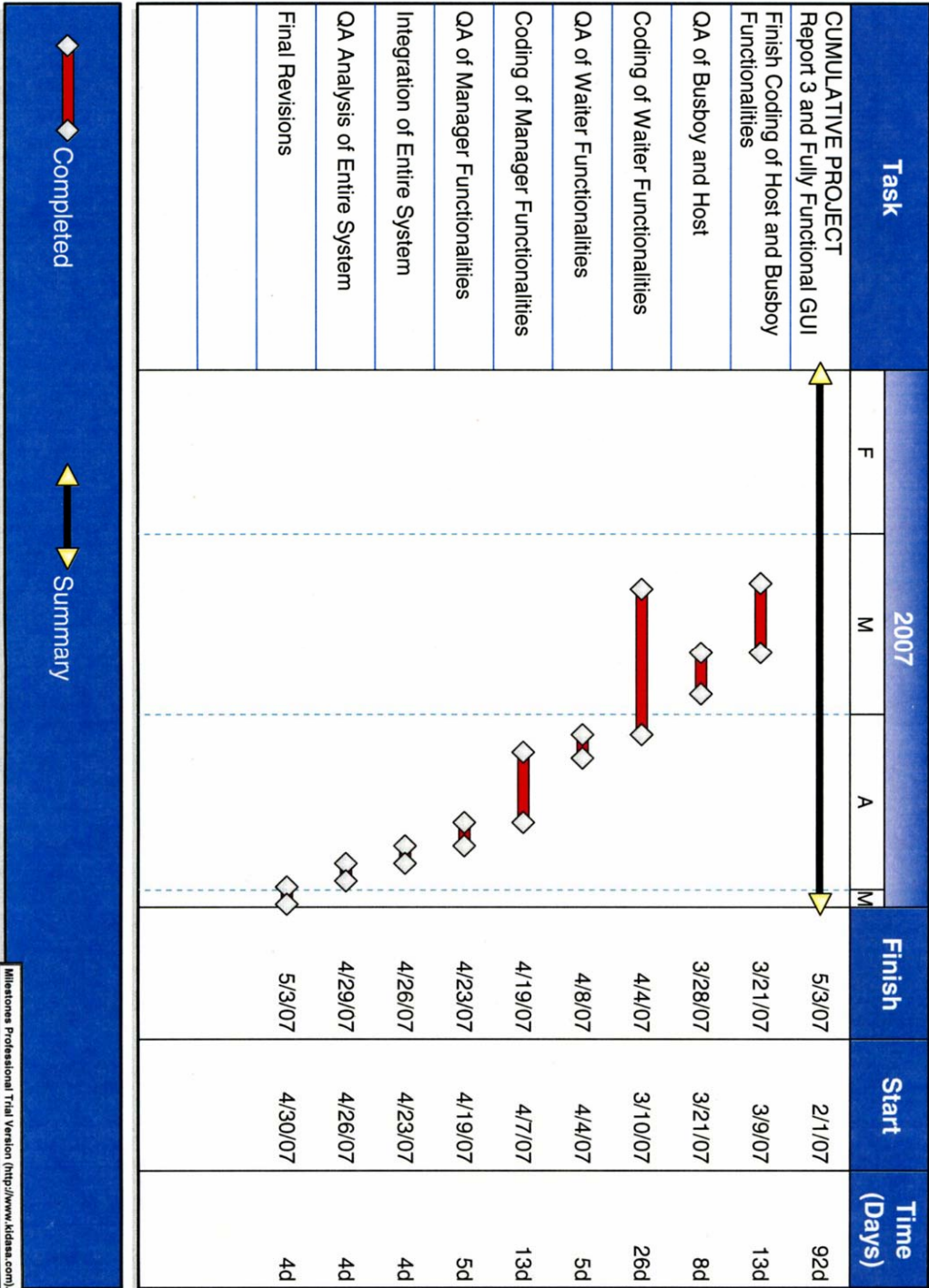
- Navigation: total 1 mouse click, as follows

- --- Complete Data Entry ---
- Click “Sign In” Button
- Data Entry: total 1 keystroke, as follows
 - Input “Employee ID”
 - Press Tab
 - Input “Password”

14. History of Work & Current Status of Implementation

Page 1 of 1

History of Work



Completed

Summary

- This restaurant automation project in itself is a very big accomplishment due to the fact that it gives us the sense of creating something from scratch.
- Although some of the group members had experience working with PHP and MySQL, none of us were proficient in either one. We overcame the difficulty this difficulty by researching and taking tutorials on the web in both PHP and MySQL.
- Debugging our code in order to work out software glitches was also another problem that consumed a lot of time. Working out the glitches was a good experience and taught us a great deal about dealing with software glitches.
- The interface that we designed for the website was done in CSS and even though a fancy interface did not bring us a better grade for the course, our group decided that a fancy interface would be necessary in order to make this product realistic for a touch screen interface that a restaurant would use because it would make things easier on the eye.
- Meeting the milestones was also another big accomplishment due to the fact that we had to work around the schedule of all the different members of the group.

15. Conclusions and Future Work

Challenge: The setup and implementation of the overall architecture was one of the most challenging tasks at the beginning of this project. The group has some trouble reaching a decision on the necessary functionalities required for different users of the restaurant automation model.

Solution: The used case and domain models discussed during the building blocks for this project. Using the fully dressed descriptions, we were able to implement use cases and modify them according to the individual needs of the host, waiter, manager, busboy, and chef. The domain model was crucial in tying everything together. It became evident that everything is interlinked and much of the code can be easily reused for different functions.

Challenge: When we began the implementation of the database and coding, the structure and organization of the class as well as their functionalities were very unclear. We were not sure of the required parameters and information required to implement the database. The various functionalities of different users caused some unnecessary repetitiveness in the database.

Solution: Using the class diagram and interface specification discussed in class, we were able to coordinate and implement an organized database containing only the necessary information and eliminating unnecessary repetitiveness. This also made the distribution of tasks easier within the group as the classes and functionalities required had now become very clear.

Challenge: The next challenge we faced in the group was communication among the users of the restaurant automation model (host, waiter, etc). We realized that when one

user enters the model and performs a certain task, it affects all other users. For example, when the host seats a customer, the waiter needs to acknowledge this and serve the table. When the waiter serves the table, the order needs to be sent to the chef. The chef then needs to alert the waiter when the food is ready, and finally, the waiter needs to alert the busboy at the end of the meal. This communication seemed tedious and caused some problems with shared access.

Solution: The Publisher-Subscriber pattern discussed during the course was a perfect implementation for this project. This way, all the users would be subscribed to their related tasks, and when one user performs a task, all other users are automatically updated.

Other Helpful Knowledge: The course did not focus much on the database implementation of the project. Although there was a general overview provided, there was no specific focus on the technical practices and habits for efficient database modeling. Knowledge from our current internships and jobs helped greatly with the implementation of this database model. This knowledge can be very helpful to many students that are unfamiliar with the implementation of a database.

Possible Directions for Future Work: This model can be easily redefined for personal needs of consumers (menu items, waiter functions, etc). However, a better implementation of the GUI for the PDA can be a future prospect due to their small screens and relatively slower operating speeds.

16. References

http://www.caip.rutgers/~marsic/books/SE/book-SE_marsic.pdf
<http://www.caip.rutgers.edu/~marsic/Teaching/CSE/report1.html>
www.wikipedia.org

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