Centre for Information Technology and Electronics

DTU Diplom
Lautrupvang 15, 2750 Ballerup

Bob's Shop

"Self-reliant" Customer Prototype

Group2 (EPRODE2-2 - Sept 2013). *Buddy group (Group 7 EPRODE2-2).*
Group2 (EPROPE2-2 - Sept 2013). *Buddy group (Group 1 EPRODE2-2).*

Supervisor(s): Agner Fog (EPRODE2-2)
Henrik Bechmann (EPROPE2-2)
Jens Jakob Thodberg (EPROPE2-2)

Student name	Student number		Signature
	IHK	DTU	
JiachenYin	120169	s127628	
Rudolf AntonFortes	130061	s137353	
Ebbe Valbak	130323		
RiccardoMiccini	130465	s137345	

Word Count: 13567

Table of Contents

1.Introduction

This project is was done for a a 2nd Semester Project at DTU Ballerup for the combined courses of EPRODE2-2 and EPROPE2-2 which started on Sept 2013.

In this project we will try to develop a prototype that could simulate a self service checkout solution that could be used for the project we were presented with in the Appendix we have included a copy of that original Project Description at ”13.3SELF-SERVICE SHOP PROJECTON PAGE 83”.

1.1. About this document

This document is meant to be a work in progress all ’notable’ changes to the solution, and findings should be documented here.

1.2. System description

We have decided to create a very simple user interface for the customer to ease the transition into the new shop. We have therefore decided to simulate all Customer interaction with the shop will occur at the shopping cart which will be equipped with a Terminal (RFID Scanner, micro-controller, LCD and keypad). When the user has completed their shopping experience they will approach the exit where a checkout terminal will scan all the products in the cart and the customer will be billed accordingly.

1.3. About Us

Our Group consists of Jiachen Yin (120169), Rudolf Anton Fortes (130061), Ebbe Valbak (130323) and Riccardo Miccini (130465).

Group chronology.

We started in a group of 3 students: Rudolf Anton Fortes, Riccardo Miccini and Xiaoshen. Shortly after which Jiachen Yin joined the group.

Approximately 2 weeks in Xiaushen left the group because he felt he would learn more from working with others.

Approximately 2 weeks after Xiaoshen left Ebbe joined the group.

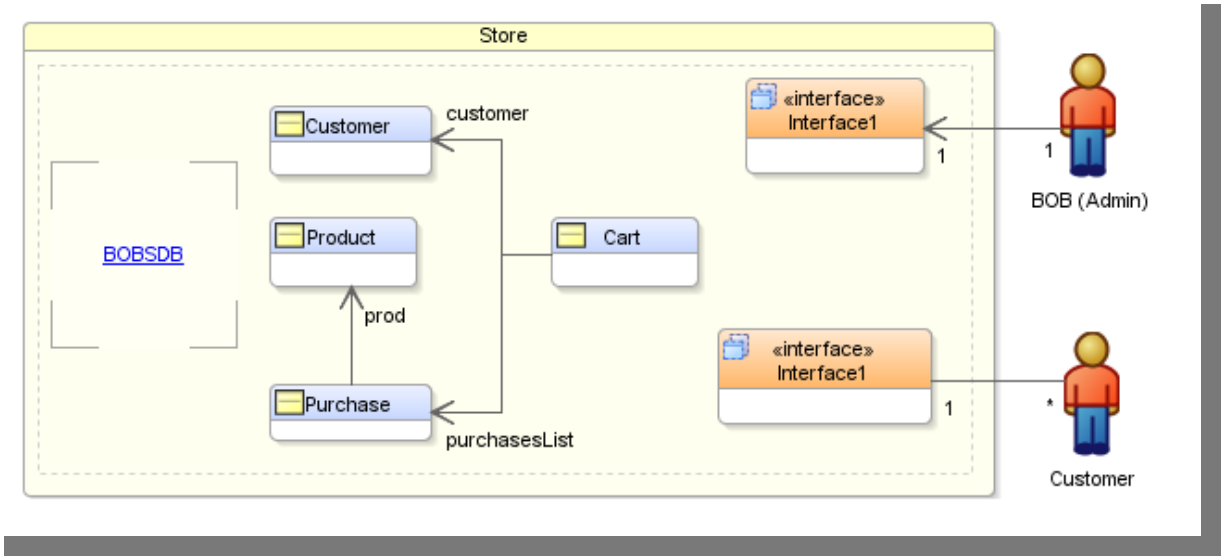
2.Analysis

The Following section will describe our Analysis of the project.

2.1. Problem Domain

This Problem Domain shows the major components within the scope of our project Bob (the administrator) and the Customer are the Human element to this solution. They will be provided an interface in order to access the system. The system will handle and process the actions relating to the customer the product's and the purchase's.

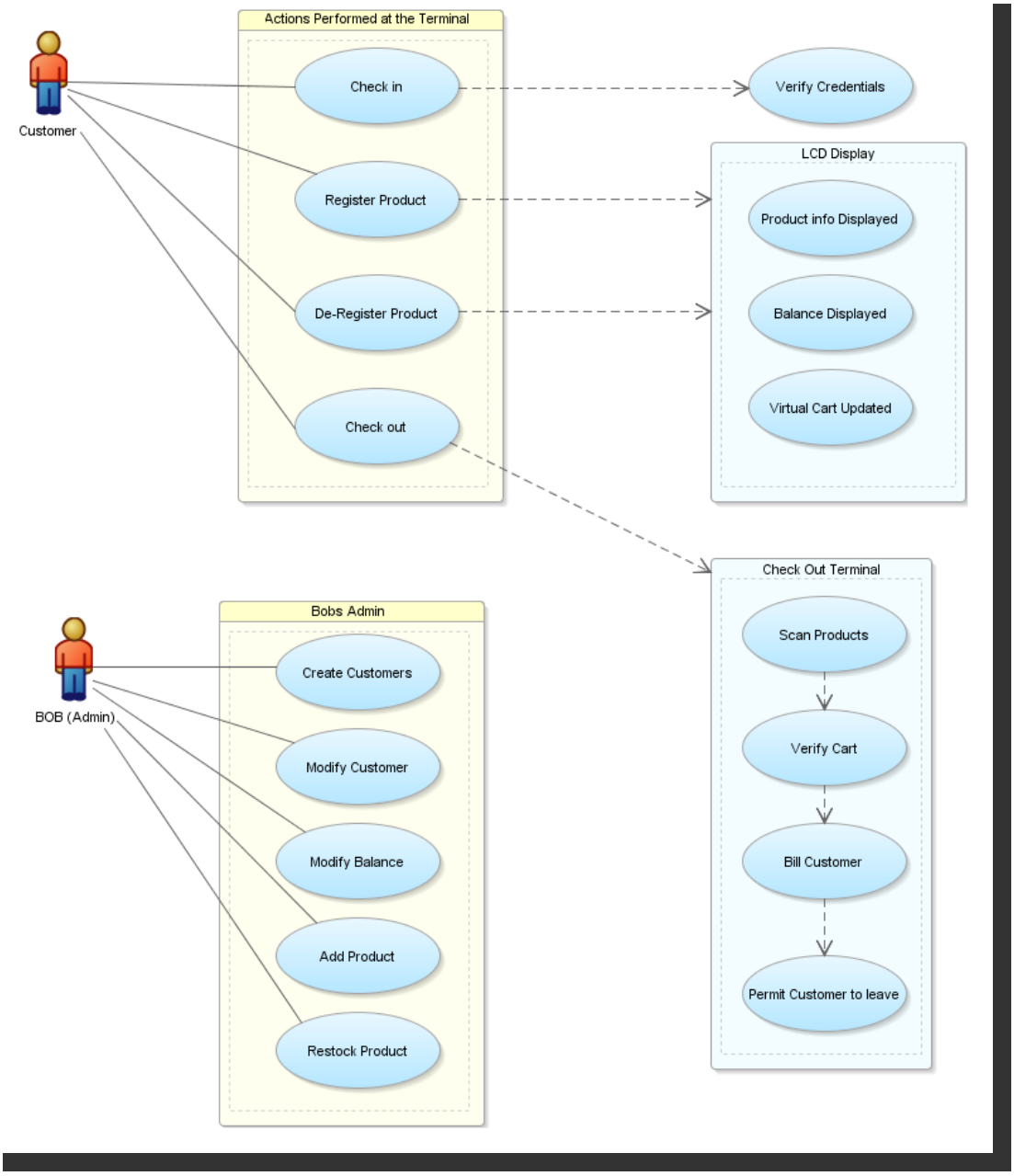
2:1Problem Domain



2.2. Use Case Model

This Use Case Model shows how the two actors should be able to interact with the system. They will be described in more detail in Requirement Specification at "6.3USE CASE SPECIFICATIONS ON PAGE 24".

2:2Use Case Model



3. Risk Management

This section will describe in brief the various Risk aspects of this project

3.1. Risk Management Description

This section will describe in brief the various Risk aspects of this project and how they are approached. This page includes a brief description on how to read the Risk Management Table.

We will be using a 4 stage scale for Impact and a 4 stage scale for likelihood.

	Likelihood ofoccurrence				
Scaleofimpact		High	Medium	Low	Negligible
	Critical				
	Large				
	Moderate				
	Small				

Risks may be accompanied with what area they will affect

	Cost	Description
F	Functionality	Loss of Functionality or services in the system
T	Time	Extra time spent amending Functionality or services
O	Other	Please annotate a number and describe below the table.

3.2. Risk Overview for the Project

Risk	Probability	Impact	Risk
Hardware			
Construction			
LCD	N	M	N-M
RFID	L	C	L-C
Keypad	N	M	N-M
Failure	L	C	L-C
Communication	L	C	L-C
Software			
Database	N	C	N-C
Loss of data	L	M	L-M
Functionality	L	M	L-M
Group			
Dynamics	L	M	L-M
Loss of a member	L	M	L-M
Lack of contribution	L	M	L-M
Other			
Programming	L	M	L-M
Documentation	L	M	L-M

3.3. Hardware

We minimized the impact of hardware failure by:

- spending a good portion of our time in the beginning working on the Construction of the Hardware.

We minimized the probability of hardware failure by:

- Creating duplicates of the terminal so that we would have a backup solution.

We did this a little late in order to reap the benefits of multiple coders having access to the board in the development stage but just in time for the Testing.

3.4. Software

We minimized the risks in the software section by:

- working on the base elements early on .

We minimized the probability of risks by:

- using Dropbox's in built feature of version control to be able to go back to a previous version of the software. This way loss was relatively minimal

3.5. Group

We tried to minimize the Group risks by:

- trying to keep each other updated with each others progress and knowledge with regular group updates.

This however ended up being cumbersome and not very fruitfull as the project scope was a lot more rapid prototyping and each weeks knowledge ended up being more and more individual.

3.6. Other

Programming: we tried to minimize some of the risks associated with programming by:

- Using each other to bounce ideas off each other.
- Trying to limit ourselves to types of products used so that we mimick a similar work environment
- Regular meetings to make sure one person was not stuck on a specific problem.

Documentation we tried to minimize some of the risks of lack of documentation and consistancy by:

- Documenting along the way as we went along.
- Making diagrams so that we both understood the code structure and managed to identify potential bugs.

3.7. Time

We tried to reduce the risks of running out of time by setting some small goals and deadlines along the way.

This was hard due to the nature of the project being rapid prototypeing and alot of proof of concept testing. But we kept specifically 3 milestones sacred.

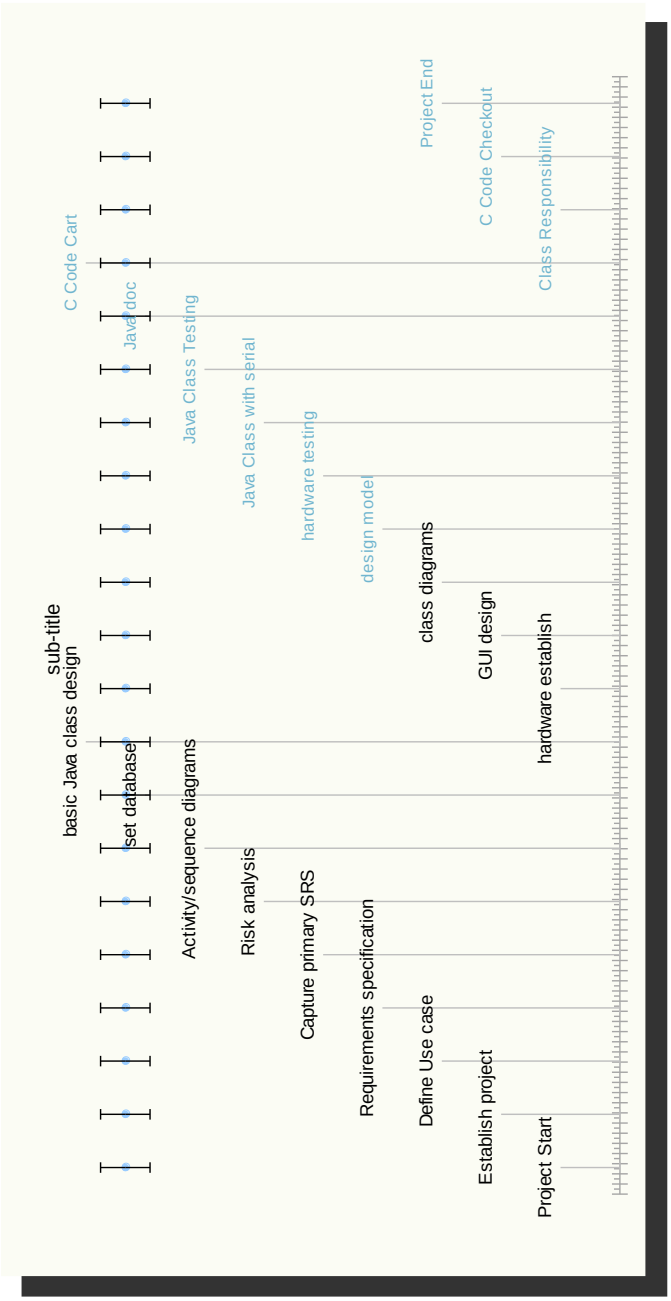
- The first Iteration because we had to hand in material to the teacher and it was a passing Criteria..
- The second was the DIG2 Design review where we prepared what we had and communicated with a buddy group for idea sparring.
- The other was the Documentation Cut off Date.. 2 weeks prior to the report hand in we would stop everything we were doing and Prioritize Documentation because that was another passing Criteria.

4. Milestone Plan

In this section we will highlight the various priorities of the Projects.

4.1. Project TimeLine Map

4:1Project Time Line



4.2. Project TimeLine Detailed

PROJECT DETAILS	
DATE	MILESTONE
02/09/2013	Project Start
05/09/2013	Establish project
10/09/2013	Define Use case
16/09/2013	Requirements specification
20/09/2013	Capture primary SRS
25/09/2013	Risk analysis
01/10/2013	Activity/sequence diagrams
08/10/2013	set database
10/10/2013	basic Java class design
15/10/2013	hardware establish
18/10/2013	GUI design
26/10/2013	class diagrams
01/11/2013	design model
05/11/2013	hardware testing
16/11/2013	Java Class with serial
20/11/2013	Java Class Testing
26/11/2013	Java doc
30/11/2013	C Code Cart
03/12/2013	Class Responsibility
08/12/2013	C Code Checkout
19/12/2013	Project End

4.3. Milestones

Keeping milestones in this project was next to impossible we had not really done anything like this before and we had little to no knowledge on how long each implementation were to take. We tried to keep up in class and do things in class as they came up. When we learned about the LCD we experimented with that when we learned about the keypad we tried that.

Here are some of the highlights that we aimed to achieve.

4.3.1. Microcontroller Part

- Proof of Concept
  - Card Reader
  - Keypad
  - Display
- Design
- Transfer Data
- State Machine / running solution combined
- Code
- Testing
- Report

4.3.2. PC Part

- Database
- GUI
- Data consistency
- J-Unit/TestNG Tests
- Testing
  - Usability
  - Functionality
  - Stability
- Report

4.4. Unified Process

We tried to follow the general guidelines of the Unified Process but in the end we ended following the course lessons more. As we learned new material in class we tried to incorporate these in our project. When we learned about design patterns we ”improved” our base elements so use the Static Factory Method. When we learned about the LCD we experimented with that etc.

The table below shows the different UP phases and iterations in which the complete PROG2-project has been divided into.

We in the end managed to do only one full itteration with some minor jumping from phase to phase.

UP Phases		
Inception	Elaboration	Construction
Phase 1	Phase 2	Phase 3
Iterations		

For each iteration the table below shows what tasks, deliverables and/or achievements the individual iteration consists of.

Iteration	Phase	Tasks	Deliverables
1 Start: 01/09/13 End: 19/12/13	1 Start: 01/09/13 End: 21/10/13	Establish project objectives (Vision) Identify critical project risks Capture primary System Requirements Define Use Cases Prioritize Use Cases	Project Report issue 1 including: Risk analysis Primary SRS Use Case Diagram and Use Case specifications Requirements Traceability Matrix.
	2 Start: 22/10/13 End: 18/11/13	Realize the basic core use case(s) Define Analysis Model Define Preliminary Design Model Run Unit Tests	Project Report issue 2 including: Analysis Model Class responsibilities Use Case realizations Design model Test cases
	3 Start: 18/11/13 End: 19/12/13	Complete use case realizations. Update: Analysis Model Class responsibilities Design Model Run Test cases. Perform acceptance testing	Project Report issue 3 including updated documentation for: Analysis Model Class responsibilities Use Case realizations Design model Test cases Three copies of the project report with CD containing NetBeans project should be delivered by 12:00 noon 19/12/13.

5.User manual

5.1. Micro-controller part

In the microcontroller / terminal User interface the user is updated with how they are supposed to interact with the device. All these messages will show up on the LCD display row 3.

The micro-controller's user interaction consists of a

- RFID card reader / writer : We only use the reader functionality
- LCD display: Customer is provided with feedback on what they are doing
- The Keypad: Customer is provides input. (values|Pin code|Exit Key)

5.1.1.The LCD Display

LCD Display: The LCD Display has 4 lines of 20 characters each.

Each line will represent some aspect of information provided to the customer.

		Character																			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ROW	1	User name (short)									Balance										
	2	Product name									Qty		X	Product Cost per unit							
	3	Messages by the system																			
	4	User Input																			

- The 1st row is dedicated for the user.
- The 2nd row is dedicated for the product.
- The 3th row will be used for messages sent by the system to the user.
- The 4rd row is dedicated for user input.

5.1.2.The Keypad

The keypad on the Terminal allows the user to type in their userpin when loggin in and update the quantity of a product when shopping.

1	2	3	F
4	5	6	E
7	8	9	D
A	0	B	C

The Digits [0-9] Provide the system with valid quantity values and pin codes

A: Enter Key
B: Backspace Key
C: Erase Key
F: Exit Key

5.1.1.The RFID card reader

When in the Idle state or shopping state it will register the cards and allow the user to either proceed to login or add a product to the shopping cart.

5.2. Pc Part

The PC software interface is only accessible by Bob and allows him to perform the tasks described by the functional requirements from 1.1 to 1.8.

The main screen consists of an overview of all the customers' data and products information. From there it is possible to create new costumer profiles, top-up their balances, or create and restock products.

The interface communicates with the database through a Data-connection class, and provides error messages when an action can't be performed (i.e. wrong data or DB connection issues).

5.2.1.Screenshots

Here we wil linclude all the Screen shots with a brief description of how Bob would utilize the solution.

Login Screen

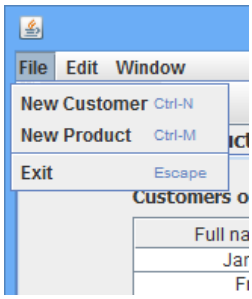
We start at the first Splash / Login page of the program.
Here bob will be allowed to login to his program. For tesing purposes we set the password to Blank the password is hardcoded for now but future implementation could add it to the database.

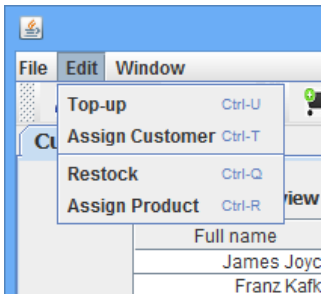
5:1Login Screen

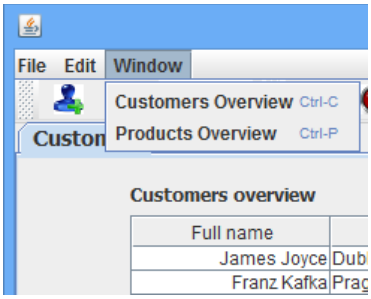
Menubar

These screenshots show the menubar it provedes bob with quick access to some of the main functionality.

5:2Menu Bar



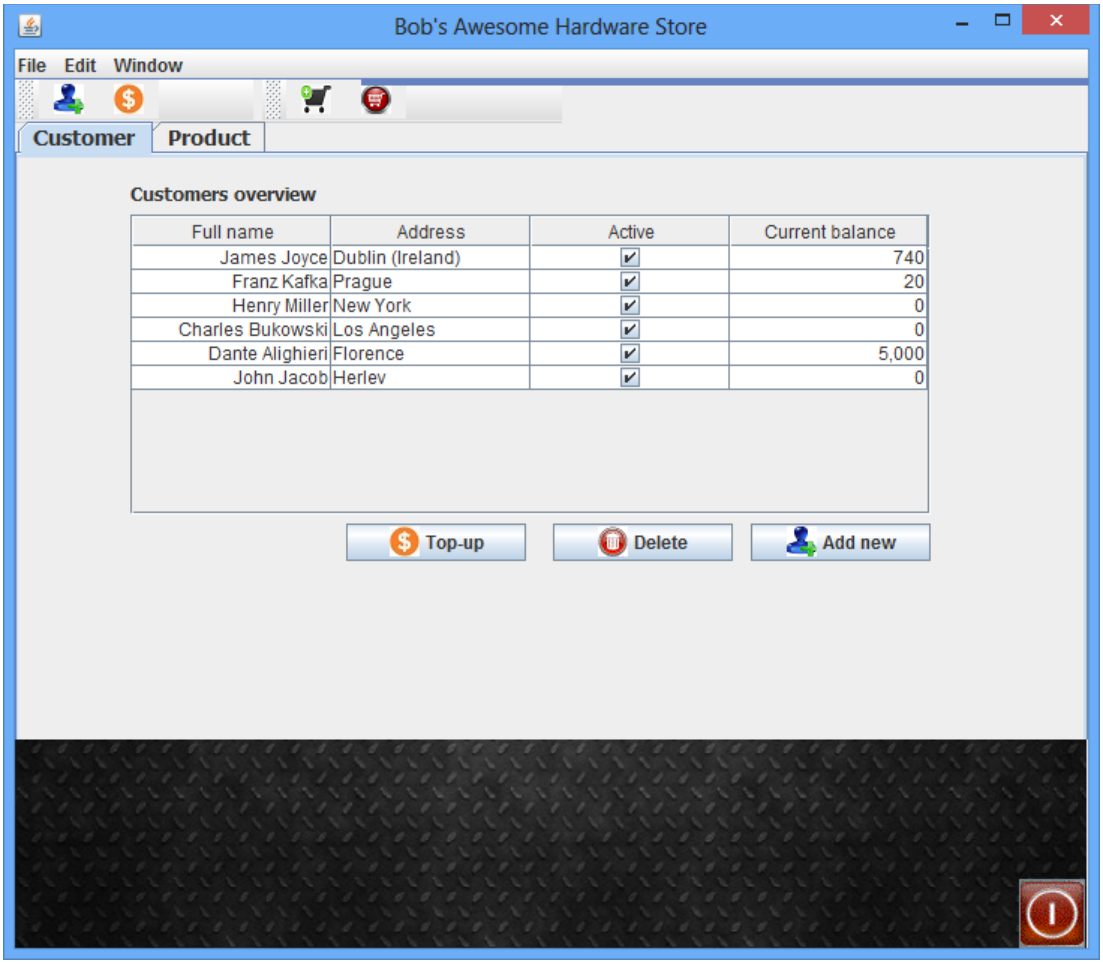




Main Program with Customer Panel

This is the First Page bob sees when he is logged on it gives him access to the the Customers. it allows bob to performall the *R1: Bob*Requirements specified for the ”customer”in the Requirement Specifications located at ”6.1FUNCTIONAL REQUIREMENTS ON PAGE 21”

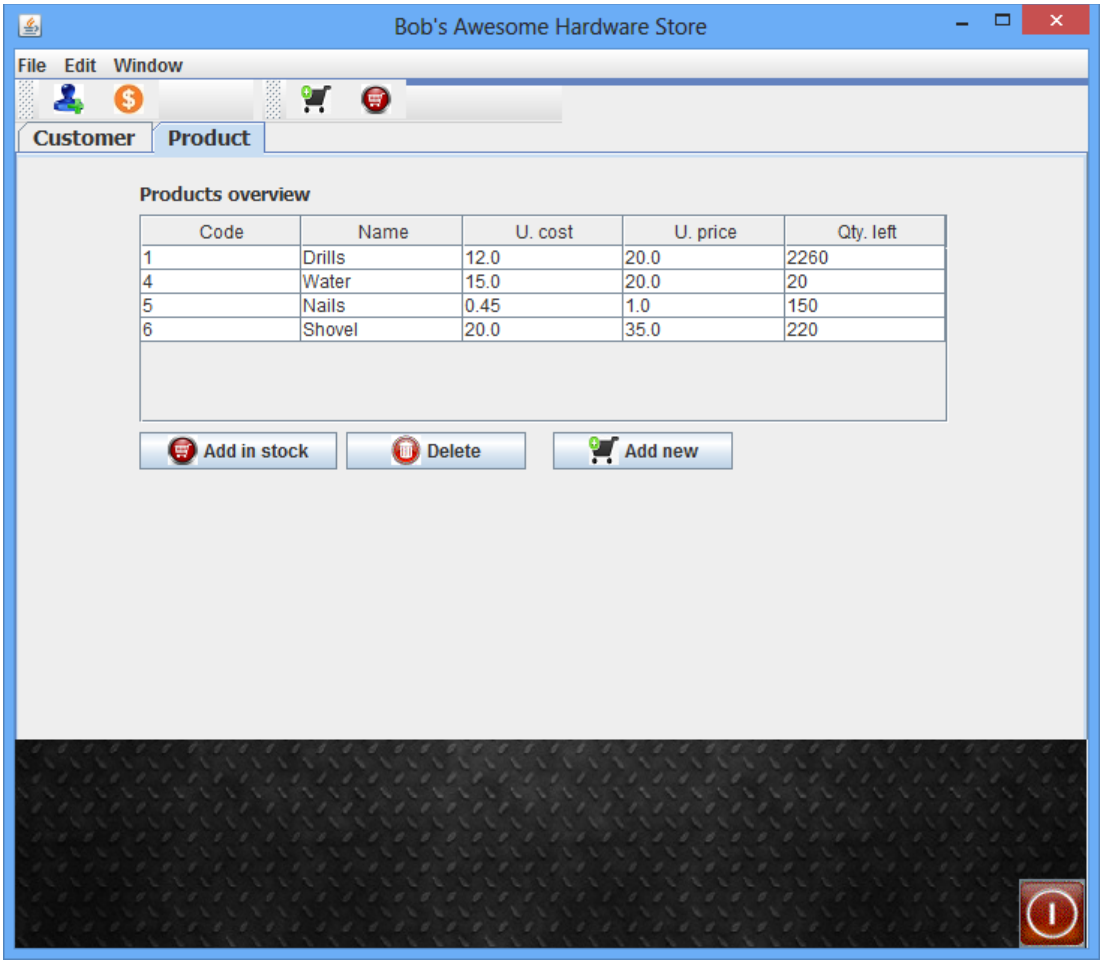
5:3Main-Active Customer Panel



Main Program with Product Panel

This Tabbed panel gives bob access to the the Products. it allows bob to performall the *R1: Bob*Requirements specified for the ”Product” in the Requirement Specifications located at ”6.1FUNCTIONAL REQUIREMENTS ON PAGE 21”

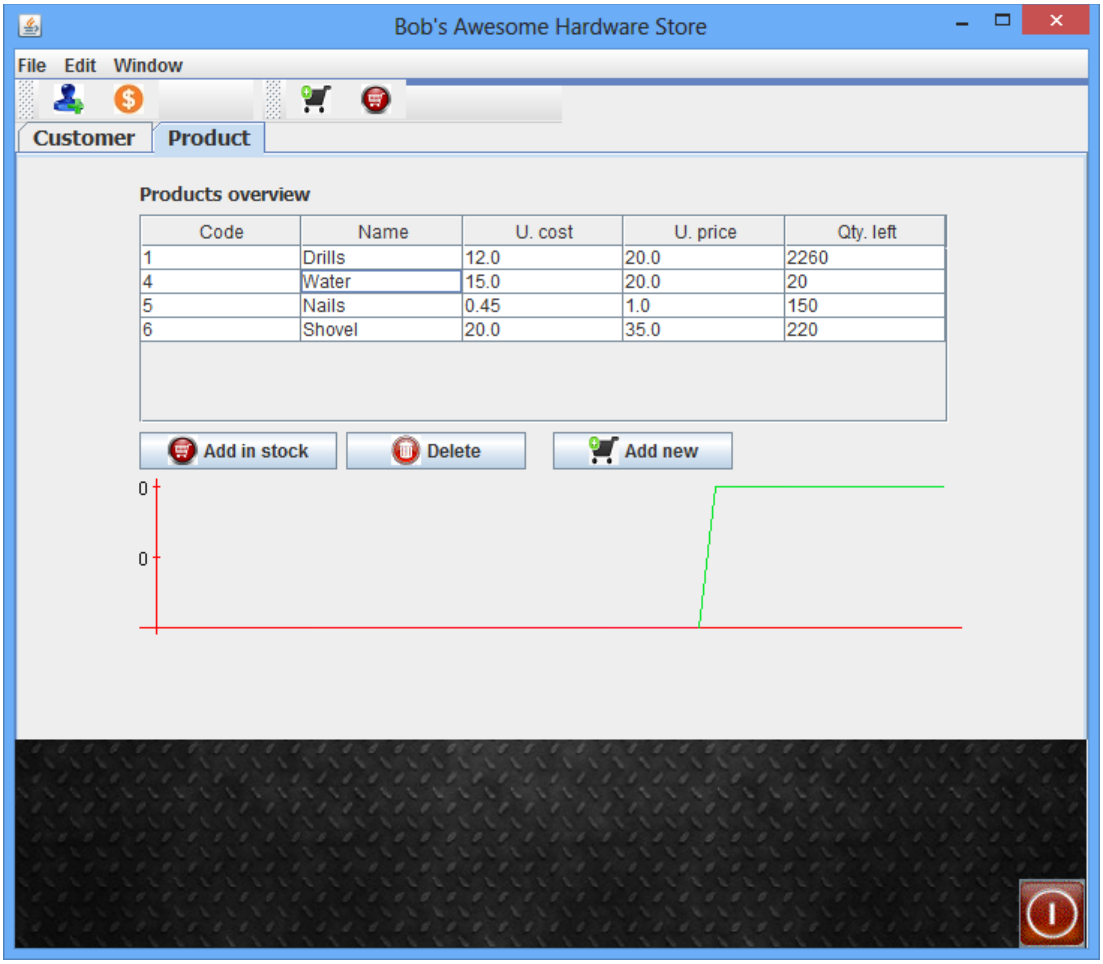
5:4Main-Active Product Panel



Main Program with Product Panel with Statistics

This screenshot shows the Product panel which below contains a graphical representation to the Stock of a speciific product chosen in the Table.

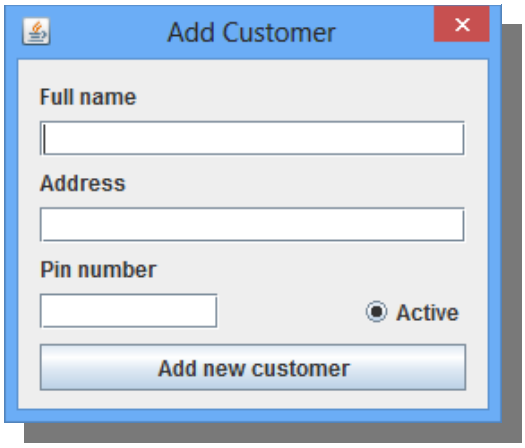
5:5Main-Active ProductPanel [with statistics]



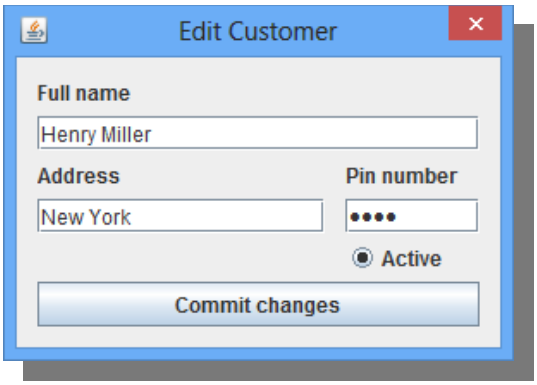
Dialog screen shots

The following screen shots show Bobs access to modify Customers and Products

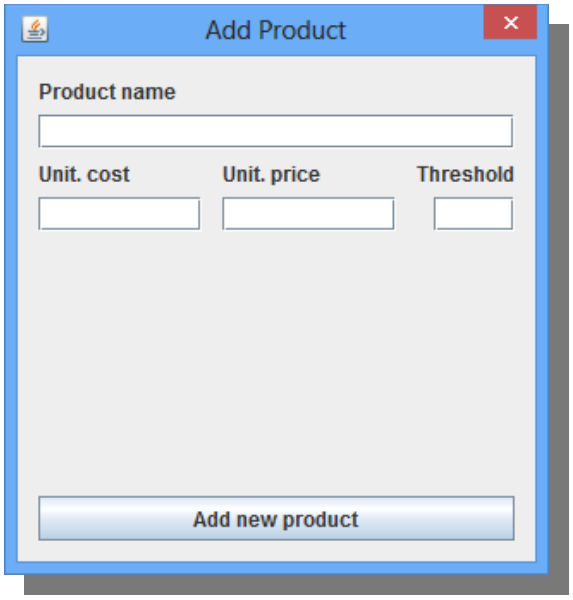
5:9Add Customer Dialog



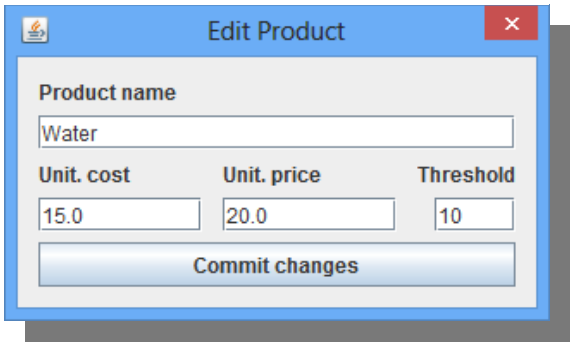
5:9Edit Customer Dialog



5:7Add Product Dialog



5:7Edit Product Dialog



6. Requirements specification

We decided to Approach this project as an Entire Entity and therefore our requirements will incoporate both the Micro-controller part and the PC Software requirements specification (SRS).

We decided however to split it into fucntional requirements and non-functional requirements where the functional requirements will be the basis for our tests of functionality and the non-functional will specify more about the scope, tendancies and generic guidelines for how the project should be approached.

6.1. Functional requirements

Functional requirements	
R1: Bob	
R1.1	Bob shall be able to create a Customer profile.
R1.2	Bob shall be able to modify the Customer profiles.
R1.3	Bob shall be able to add Products.
R1.4	Bob shall be able to modify Products.
R1.5	Bob shall be able to Restock Products (Order Request / Order Receipt).
R1.6	Bob shall be able to see statistics about sales and stock current state, pending orders and previous transactions
R1.7	Bob shall be able to update the pin code and unlock an account for a customer profile.
R1.8	Bob shall be able to amend the customers balance.
R2: Customer	
R2.1	The customer shall be able to check in (at the cart terminal)
R2.2	The customer shall be able to register a product (at the cart terminal)
R2.3	The customer shall be able to update the quantity of the last registered product
R2.4	The customer shall be able to De-register an unwanted product
R2.5	The customer shall be able to check out (abandon cart)
R2.6	The customer shall be able to “lock” the cart.

RX.X	Bob shall be permitted make suggestions concerning the GUI. “Amended – we kept this line in our report as in the previous semester the teacher changed some rules midway the project. This was done to give us an opportunity to adapt and change the project direction.”

R3: Cart terminal	
R3.1	The cart terminal shall be able to Verify credentials of the customer by pin code.
R3.2	PIN entry attempts limit. The system shall allow the customer to enter the correct PIN in no more than three attempts.
R3.3	The cart terminal shall Display the Product information of the previously Registered product.
R3.4	The cart terminal shall Display the balance of the account. Duration "Amended – we removed this feature and kept the display balance showing. " • trigger (only when the user checks in / else the display should read funds available or insufficient))
R3.5	The cart terminal shall send requests to update the Virtual Cart.
R3.6	The cart terminal shall display messages for Requirements R3.1, R3.2, R3.3, R3.4and R3.5.
R4: Checkout terminal / point.	
R4.1	The checkout terminal shall be able to scan products
R4.2	The checkout terminal shall be able to compare the scanned cart from requirement R4.1 to the virtual cart belonging to the customer.
R4.3	The checkout terminal shall be able to bill the customer
R4.4	The checkout terminal shall permit the customer to leave if their cart and credentials are valid.
R4.5	The checkout terminal shall communicate to the cart terminal to request the customer to verify credentials. "Amended" Work around-checkout Point has a keypad
R4.6	The checkout terminal shall communicate to the cart terminal to inform the customer why they may not leave. "Amended" Work around-checkout Point has a display

6.2. Non-Functional requirements

Non-functional requirements	
NFR1	The database for the hardware store payment system shall have a JDBC driver for Java 7
NFR2	The software for Bob's Admin terminal shall be written in Java 7
NFR3	All documentation and Communication Should be done in English
NFR4	The Project design should strive towards being modular.
NFR5	The Project architecture should be database centric.
NFR6	The communication should adhere to a client-server functionality.
NFR7	All the java shall adhere to a standard Format in our project using the Netbeans formatter <Alt Shift F>
NFR8	As much code as possible should adhere to some sort of code documentation standard. In java that would be javadoc and in c we will attempt to use doxygen.

6.3. Use Case Specifications

Here we wil describe the Use case specifications. We will not adhere to the standards set by Jens Jakob Thodberg in the Java class because we would much rather prefer to keep it simple and use visual diagrams to describe the flow.

We will use these Use cases as part of the Acceptance Test to make sure that our Program acts the way we have agreed. We have therefore chosen not to ammend them from the original ones we set out in Iteration one but will highlight those we failed to or chose to include in the project.

UC1:	Check In
7:1Activity Diagram-Check In	
Brief description:	The customer checks into the shop
Actors:	Customer
Preconditions: • The customer is registered • The customer has a valid card(simulated by a student card) • The customer has access to a "shopping cart" referred to in this document as a terminal(RFID scanner, Micro-controller and custom Build).	
Main flow: • The customer scans their card at the RFID scanner. • The customer enters a valid 4 digit pin number. • The customer receives a message on the Terminal whether their check in was successful or they should try again.	
Post conditions: The customer is checked allowing them to use the designed services.	
Alternative flows: The customer's user is blocked when more than 3 faulty attempts have been made. The user is locked: not permitting the user to check-in until a system overwrite performed by Bob.	

UC2:	(main) Register a product / (alternate) De-register a product
Brief description:	The customer registers or de-registers a product
Actors:	Customer
Preconditions:	• The customer is checked in (see **UC1**).
Main flow:	• The customer scans a product with the RFID scanner (simulated by card) • The customer is given the option (LCD message) of updating the total quantity of the product scanned (default is + 1) at the terminal (keypad)
Post conditions:	• The Virtual cart is updated with the product and the quantity of items requested.
Alternative flows:	• Flow 1: The customer scans a product with the RFID scanner (simulated by card) and updates the total quantity of the product scanned to "0". • Flow 2: The customer presses the C character on the keypad and then scans the product with the RFID scanner.
Note: Flow 2 is not yet implemented.	

UC3:	Check out
Brief description:	The customer checks out
Actors:	Customer
Preconditions:	• The customer is checked in (see **UC1**). • The customer's cart has registered products (see **UC2**). • The customer has sufficient balance to pay for the items in their cart.
Main flow:	• The customer goes to the check-out terminal. There all the items are scanned (simulated by the terminal). • The cart is verified (virtual cart compared with the scanned cart) • The balance is verified. • The customer is Billed (a receipt) • The customer is permitted to leave the shop
Post conditions:	• Receipt or order is created.
Alternative flows:	• should anything go wrong then bob would have to assist the customer: • This could be updating the customer's credit. • Asking the customer to put some of the items back. • Accepting a balance top up.

UC4:	Create Customers
Brief description:	Bob can create a customer profile
Actors:	Bob
Preconditions:	• Bob has access to the system. • Bob knows the Customers details.
Main flow:	• Bob logs on to the system • Bob creates a Customer profile using the GUI.
Post conditions:	• A customer profile has been created
Alternative flows:	- none-

UC5:	Modify Customers
Brief description:	Bob can alter a customer profile
Actors:	Bob
Preconditions: • Bob has access to the system. • The customer profile exists.	
Main flow: • Bob logs on to the system • Bob amends the customer profile using the GUI.	
Post conditions: • A customer profile has been amended	
Alternative flows: - none-	

UC6:	Modify Balance Customers.
Brief description:	Bob updates the Customers balance
Actors:	Bob
Preconditions: • Bob has access to the system. • The Customer Profile exists. • Bob knows the Customers details.	
Main flow: • Bob logs on to the system • Bob creates a receipt or an order with a special product that simulates type of payment (CASH / Credit card etc.).	
Post conditions: • A receipt is created to amend the balance.	
Alternative flows: - none-	

UC7:	Create / amend Products
Brief description:	Bob can create Products
Actors:	Bob
Preconditions:	• Bob has access to the system. • Bob knows the Product details.
Main flow:	• Bob logs on to the system • Bob creates or amends a Product using the GUI.
Post conditions:	• the Product has been created or amended in the system
Alternative flows:	- none-

UC8:	Restock Products
Brief description:	Bob can restock Products
Actors:	Bob
Preconditions:	• Bob has access to the system. • Bob knows the Product details.
Main flow:	• Bob logs on to the system • Bob shall have access to a GUI that allows him to keep track of the stock of his store and send in order requests for products soon to run out of stock.
Post conditions:	• Stock Maintained.
Alternative flows:	- none-

UC9:	Store overview
Brief description:	Bob needs a GUI so that he can oversee the status of the Shop.
Actors:	Bob
Preconditions:	• Bob has access to the system.
Main flow:	• Bob logs on to the system • Bob shall have access to a GUI that allows him to keep track of the stock, customers. This will allow bob to see when it would be prudent to update customers or reorder stock.
Post conditions:	• Store Maintainable.
Alternative flows:	- none-

6.4. Requirement Tracing

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	
R1: BOB										
R1.1										
R1.2										
R1.3										
R1.4										
R1.5										
R1.6										
R1.7										
R1.8										
R2: CUSTOMER										
R2.1										
R2.2										
R2.3										
R2.4										
R2.5										
R2.6										
R3: CART TERMINAL (RFID Scanner, Micro-controller and Custom Build)										
R3.1										
R3.2										
R3.3										
R3.4										
R3.5										
R3.6										
R4: CHECK-OUT TERMINAL / POINT. (simulated by (terminal as above)										
R4.1										
R4.2										
R4.3										
R4.4										
R4.5										
R4.6										

	Implementable
	Not-implementedThis feature was Described either for a real world simulation of the project and not realistic in this project prototype scope. Or Omitted due to time constraints.

7. Problem Solution

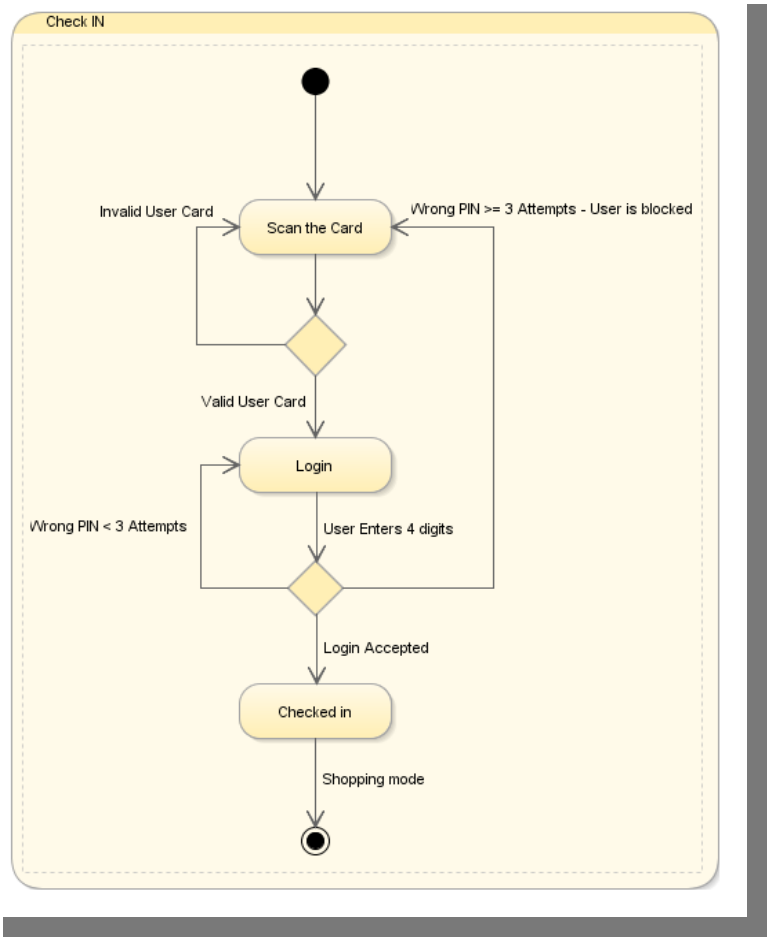
We have decided to create a very simple user interface for the customer to ease the transition into the new shop. We have therefore decided to simulate all Customer interaction with the shop will occur at the shopping cart which will be equipped with a Terminal (RFID Scanner, micro-controller, LCD and keypad). When the user has completed their shopping experience they will approach the exit where a checkout terminal will scan all the products in the cart and the customer will be billed accordingly.

7.1. Activity Diagrams

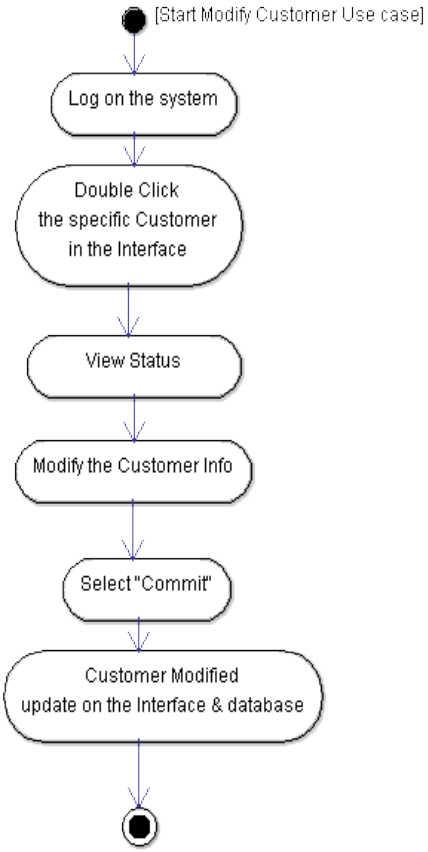
In this sub section we will include Activity Diagrams on how the ”Customer” can interact with the Terminal. The Usability of the Java is described as part of the User Manual ”5.2PC PART ON PAGE 15”

Each diagram should represent one of the Use cases related to the Customer interactions described in ”6.3USE CASE SPECIFICATIONS ON PAGE 24”.

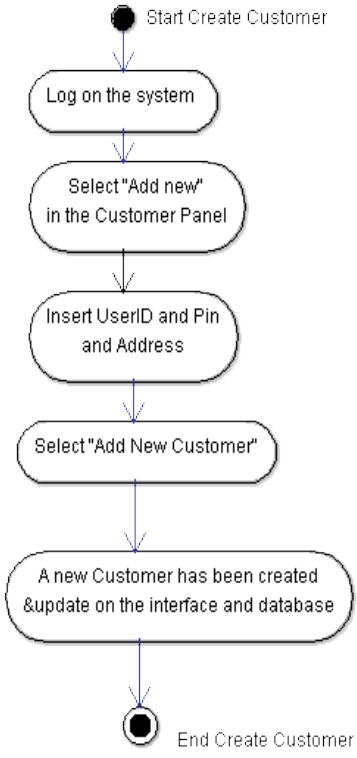
7:1Activity Diagram-Check In



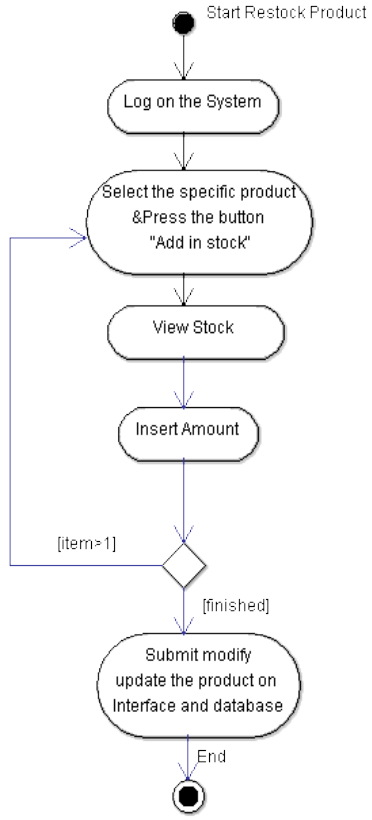
7:3 Modify a Customer



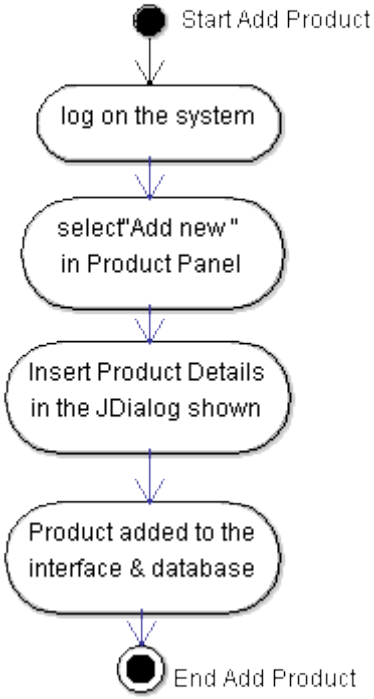
7:2 Add a Customer



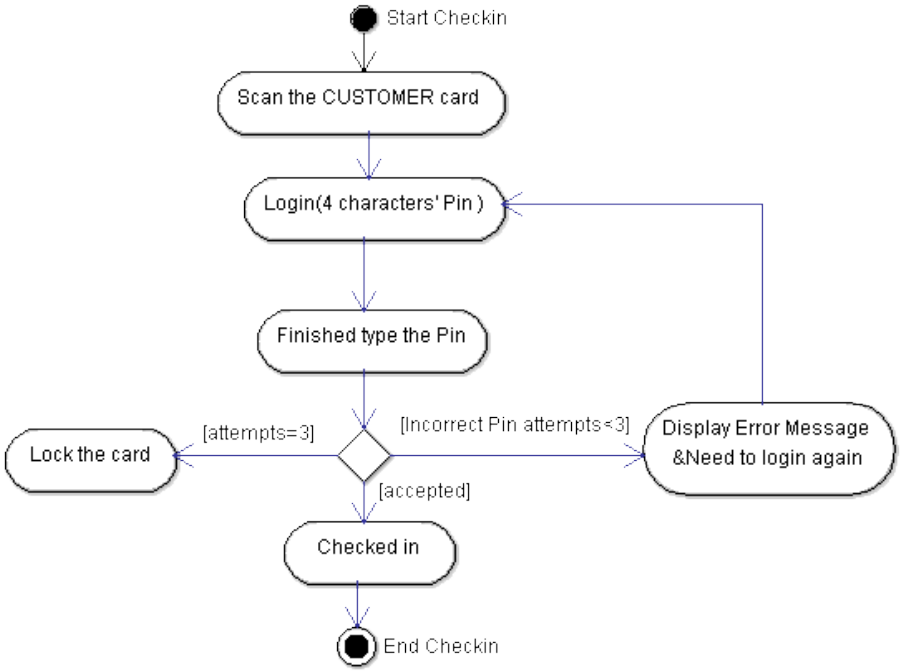
7:5 Restock A Product



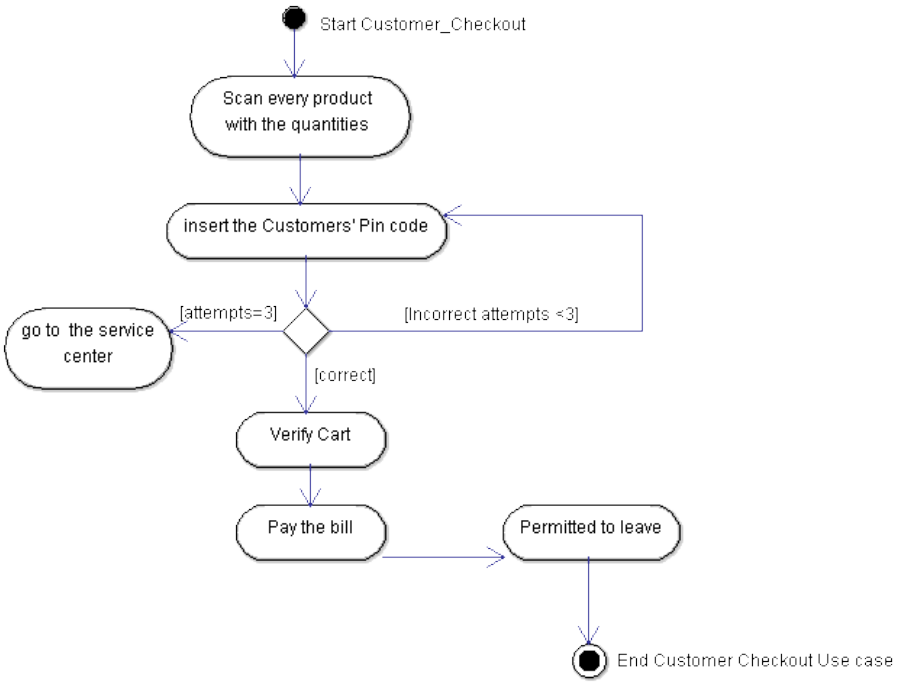
7:4 Add a Product



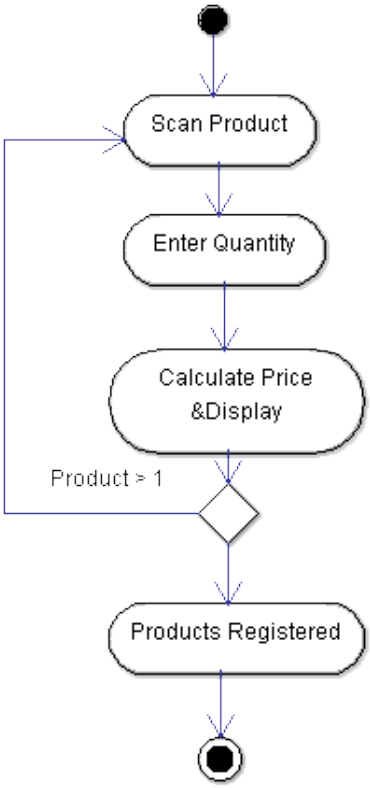
7:6Customer Check-In



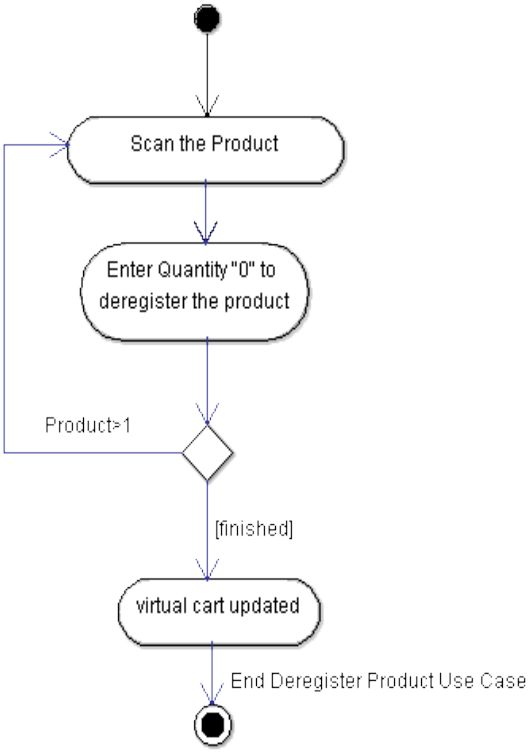
7:7 Customer Check out



7:8 CART - Register Product

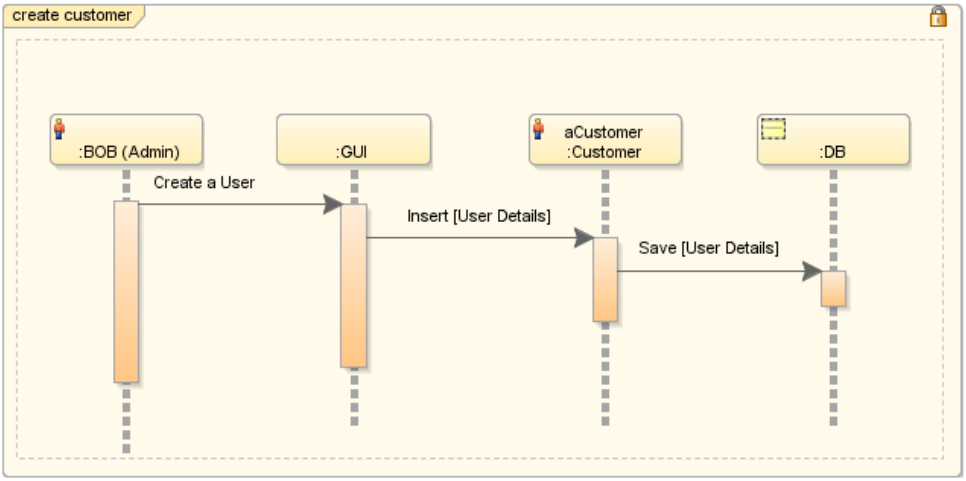


7:9 CART - De-Regiser Product

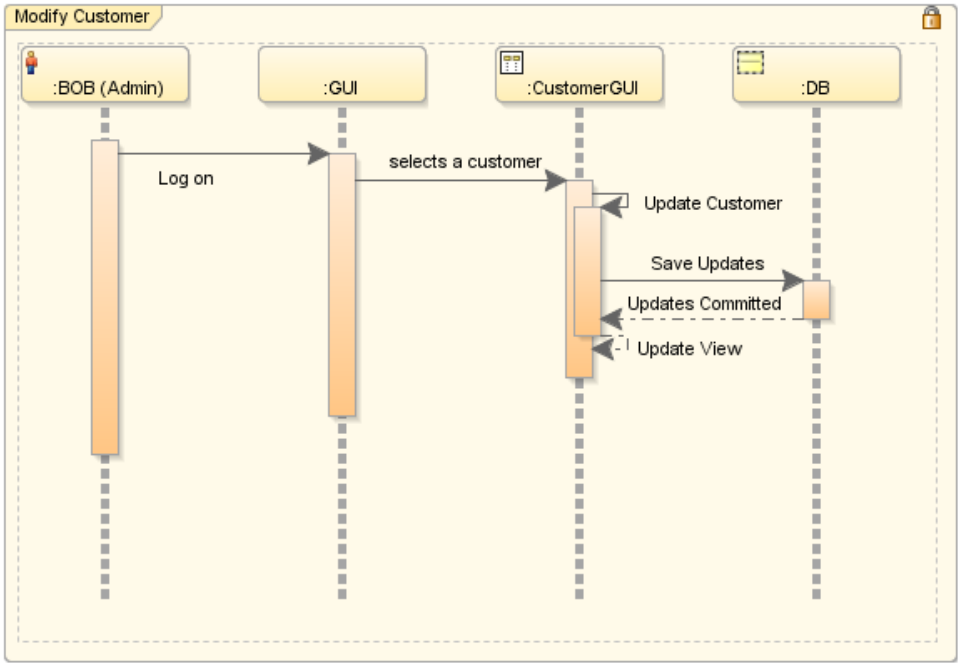


7.2. Sequence diagrams

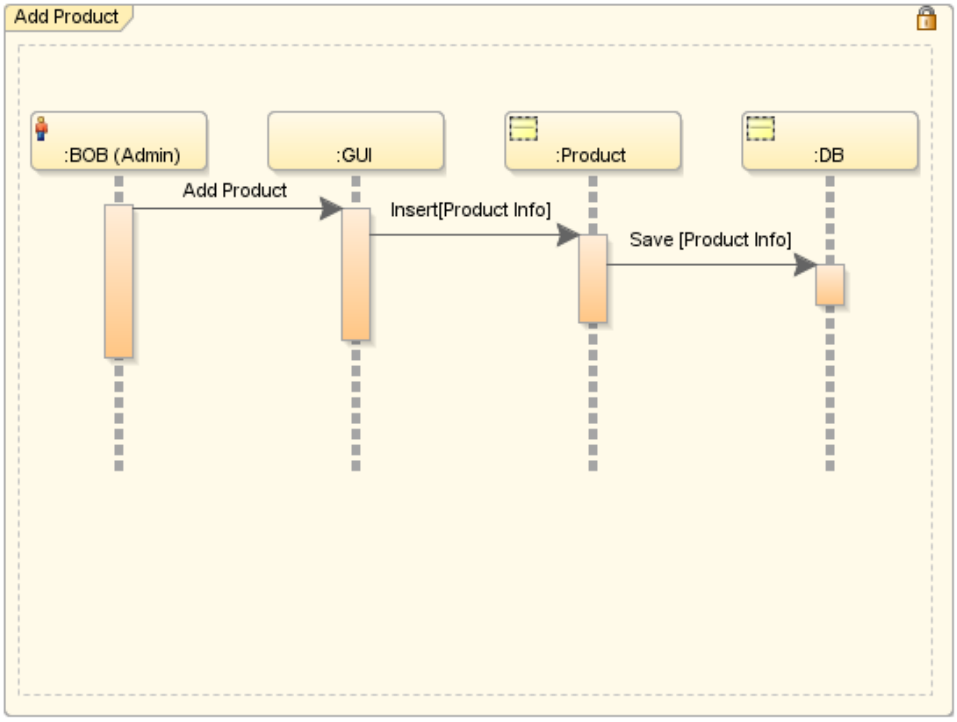
7:10Sequence Diagram: Bob - Create Customer



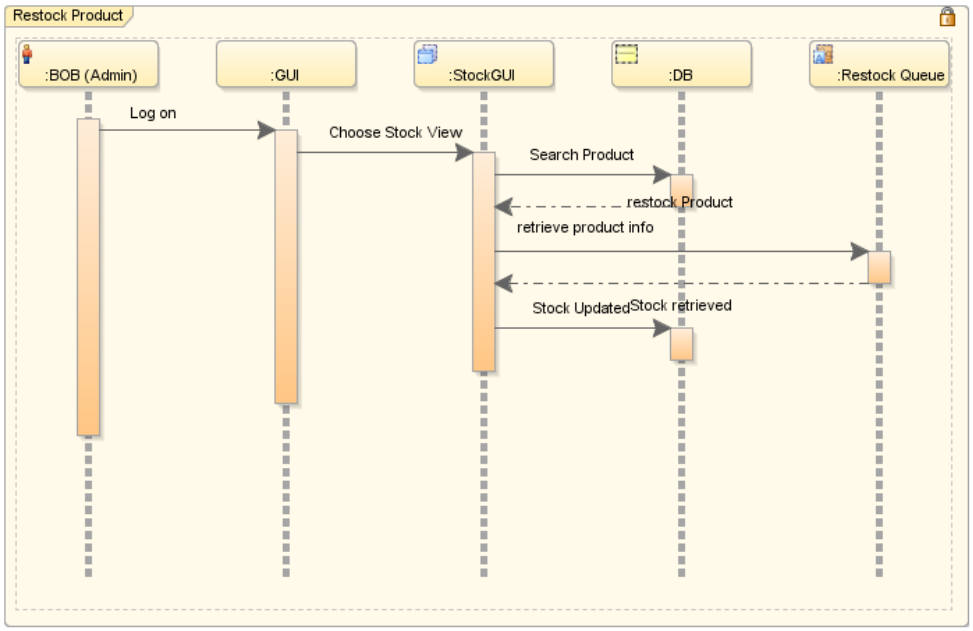
7:11Sequence Diagram: Bob - Modify Customer



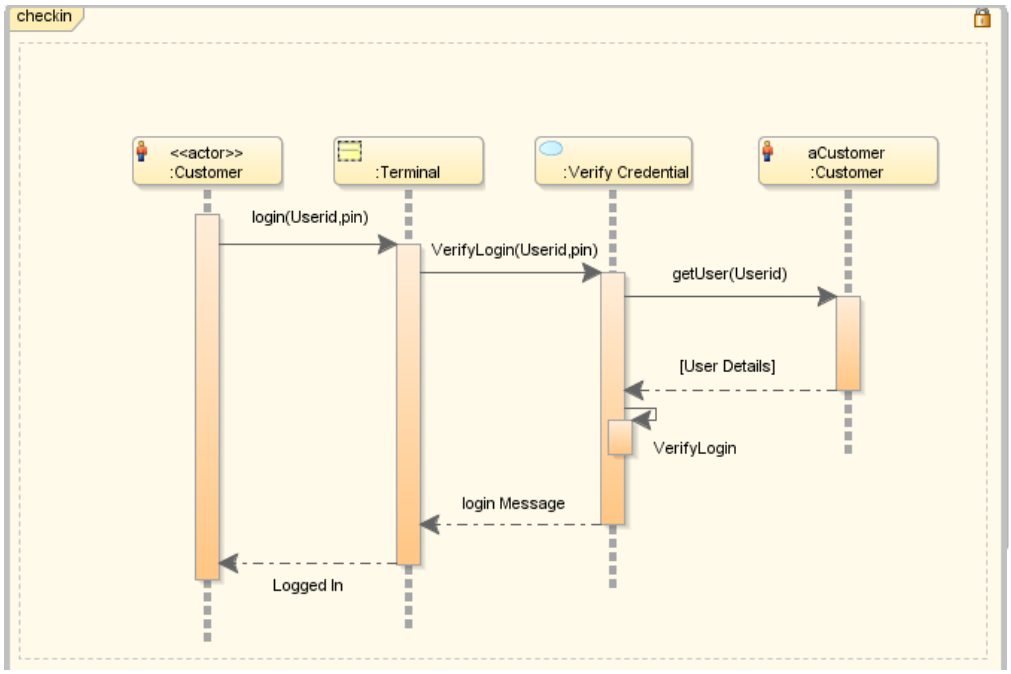
7:12Sequence Diagram: Bob - Add Product



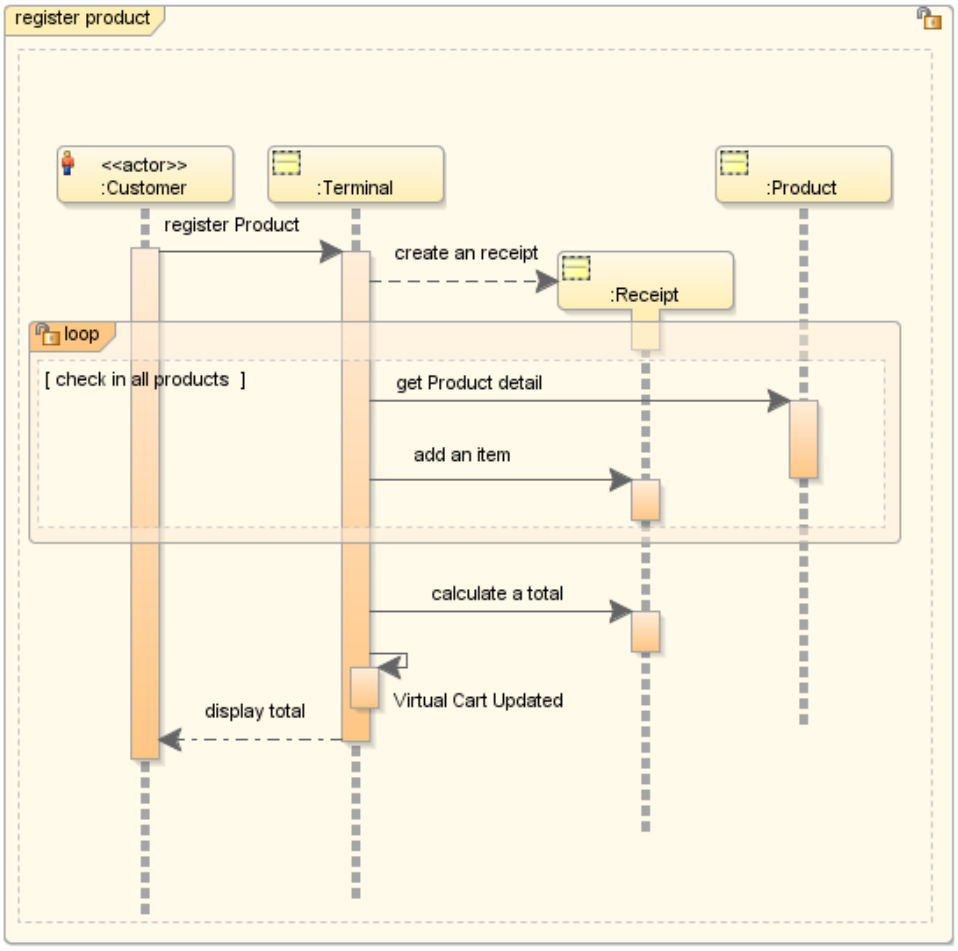
7:13Sequence Diagram: Bob - Restock Product



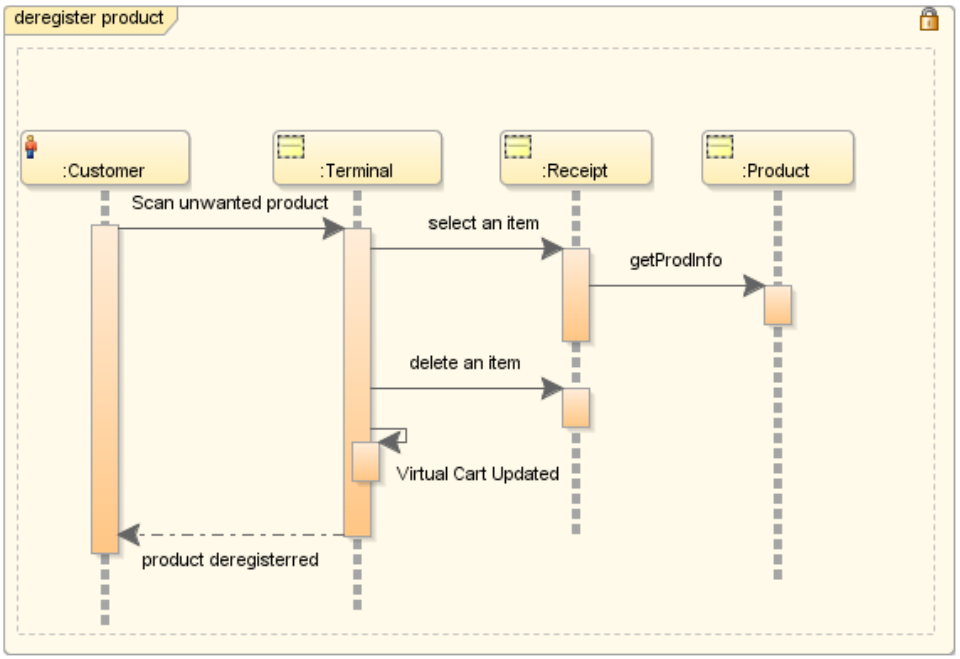
7:14Customer - Check in



7:16Sequence Diagram: Customer - Register Product



7:17Customer- De-Register Products



8. Problem Solution – Microcontroller

In This section will describe how we implemented and tackled the various features and aspects of the Microcontoller part of the solution.

8.1. Terminal

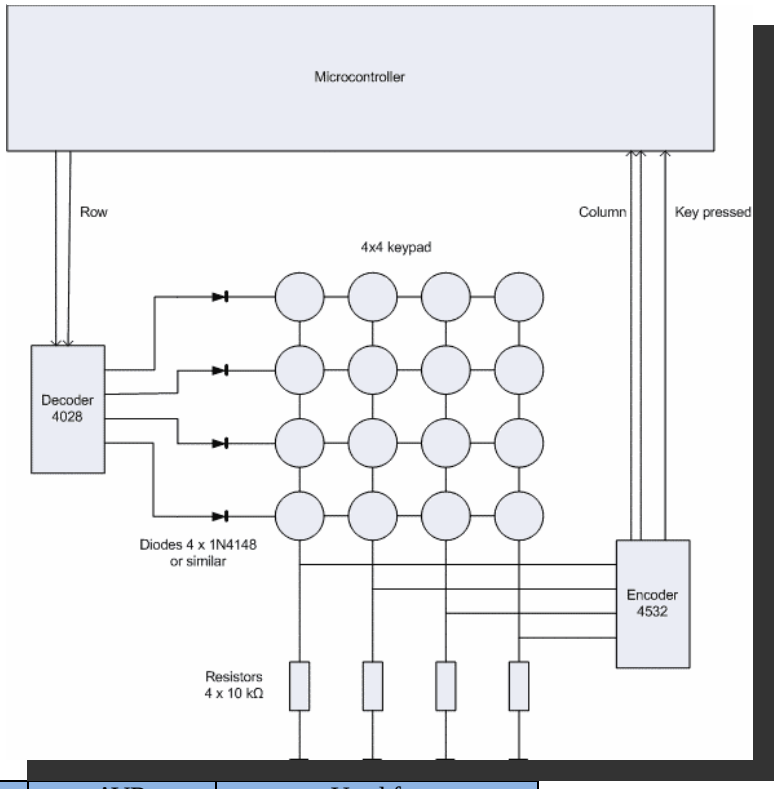
The first Priority when it came to the Microcontroller was to Design and build the ”Terminal”.

This was done by Rapid Prototyping.

First we read through the provided documentation on the various provided components: the RFID Scanner, the AVR microcontroller, the LCD display, the keypad and the vero board.

One of the main design assistances was this diagram located in the Teachers provided material it gave us a quick and easy to implement solution to reduce the quantity of pins alocated for the keypads allowing us to keep them free for other purposes.

8:1Keypad Schematic

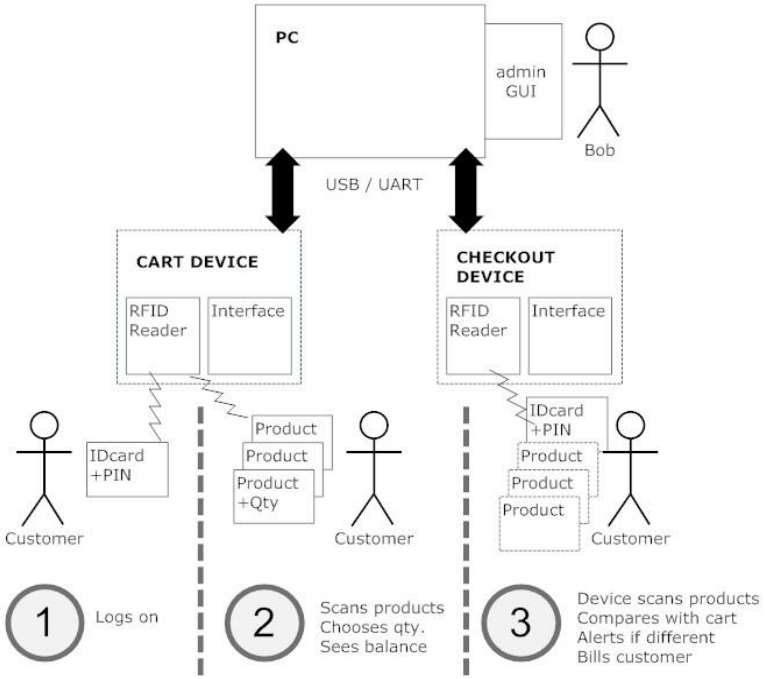


We read through all the relevant information in the Datasheets and Course documents. To come up with the following pin selection on the AVR 26 pin connector

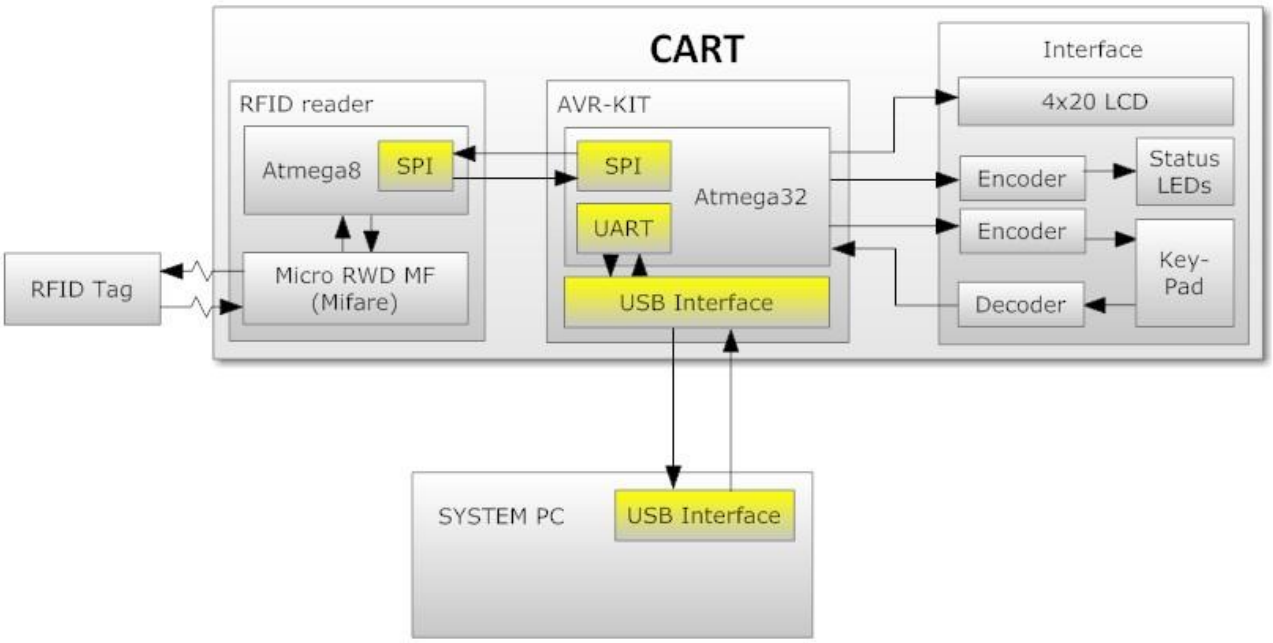
Used for	AVR	Pin	Pin	AVR	Used for
Column	Port A bit 0	1	2	Port B bit 0	Row
Column	Port A bit 1	3	4	Port B bit 1	Row
LCD RS	Port A bit 2	5	6	Port B bit 2	Interrupt 2 (Key)
LCD EN	Port A bit 3	7	8	Port B bit 3	
LCD D4	Port A bit 4	9	10	Port B bit 4	SPI SS / Start button
LCD D5	Port A bit 5	11	12	Port B bit 5	SPI MOSI
LCD D6	Port A bit 6	13	14	Port B bit 6	SPI MISO
LCD D7	Port A bit 7	15	16	Port B bit 7	SPI clock
	Port C bit 6	17	18	Port D bit 2	Interrupt 0 / Card present
	Port C bit 7	19	20	Port D bit 3	Interrupt 1 / SPI data ready
	Port D bit 6	21	22	Port D bit 4	
Timer 2 out	Port D bit 7	23	24	Port D bit 5	Timer 1 out
	Ground	25	26	VCC + 5 V	

The following diagram is a Overview of the Hardware / components involved in the Solution.

8:3Hardware Modules

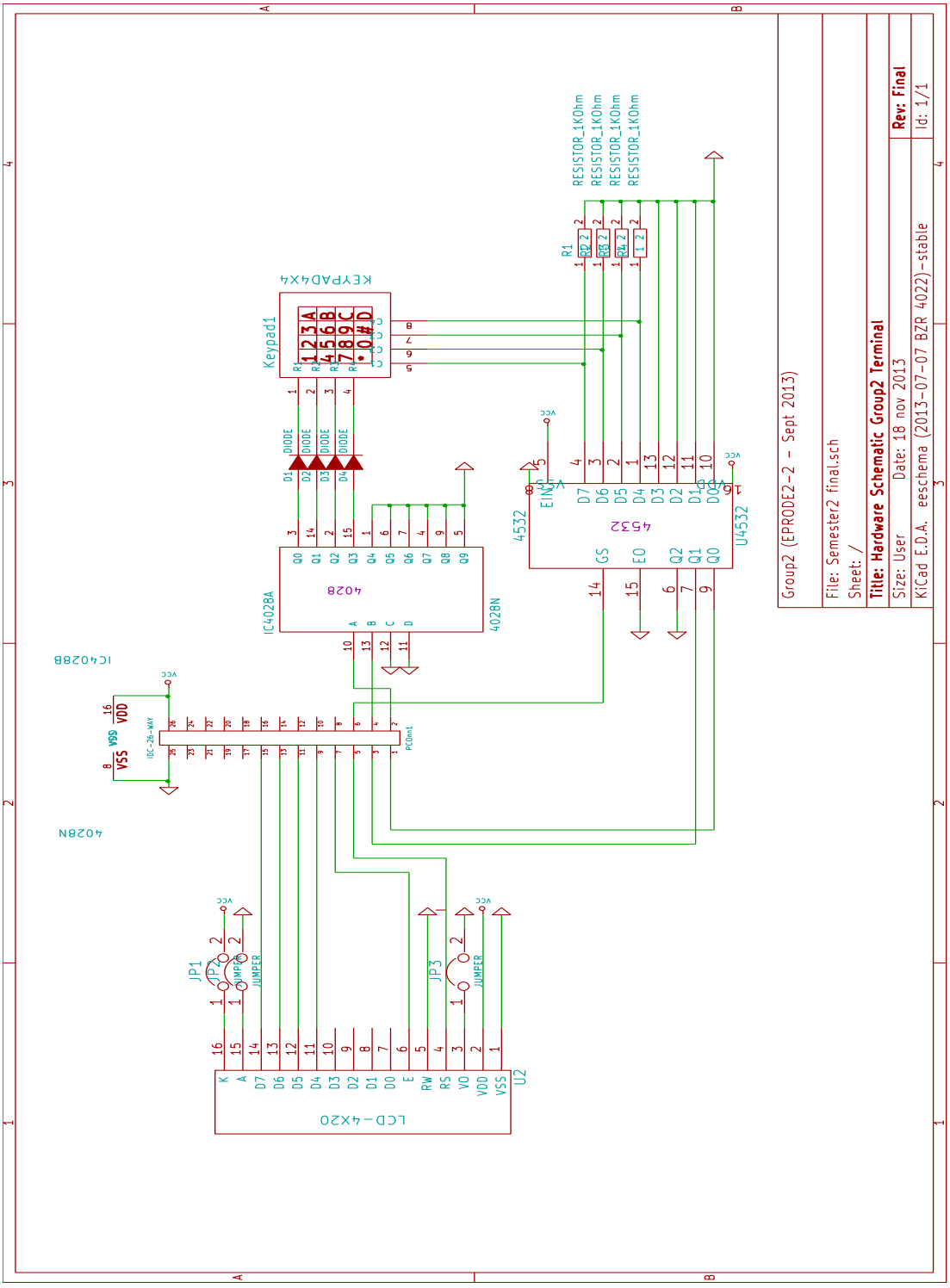


8:2 Hardware Block Diagram



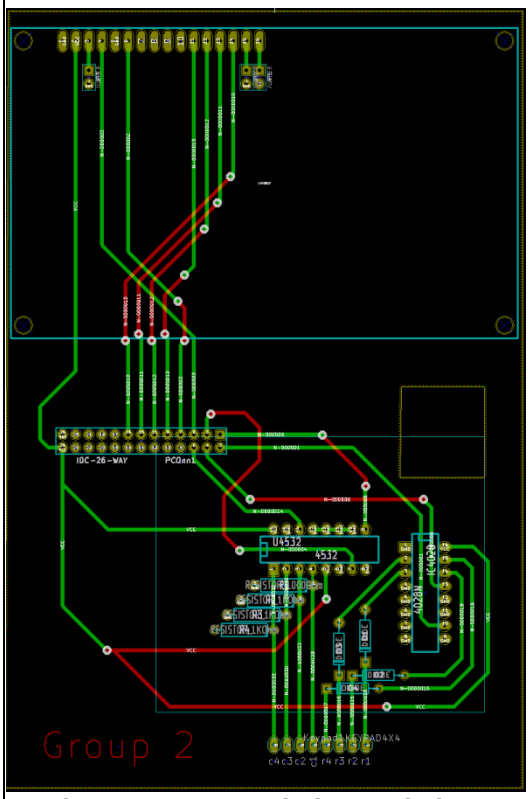
We then spent time creating the vero board prototype and attempting to create a PCB version of the board but after 2 attempts we had waisted too many resources to justify spending more time on it.

8:4Terminal Schematic



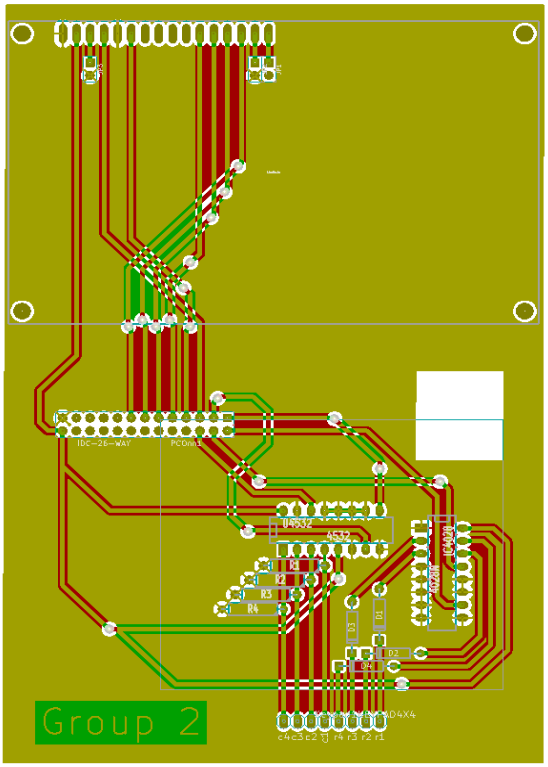
This was one of the latest attempts at making a PCB but the order in which the components were assembled caused some issues pluss the tracks for the wiring were so thin that they came off.

8:5 PCB layout



- the green represents the lines at the bottom of the PCB board
- the red represent the etched wires at the top
- the lime green represent the copper pads where components are soldered fitted

8:6 PCB Print Layout all layers



- This diagram represents the physical print but without the layer seperation all layers are combined for the documentation.

8.2. Software

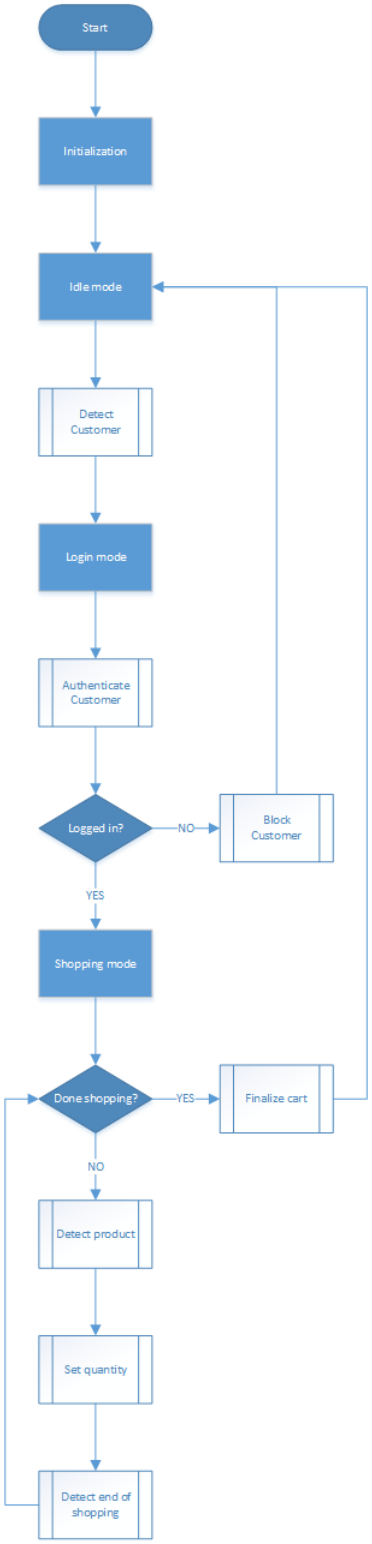
For now we have decided to code in C as it will be the most approachable for all the team members we may choose to alter this later to C++ should this seem like a beneficial solution but for now we will stick to C.

Below is a State Diagram that emulates how we designed our logic in the Main. This was done through a series of if statements and functions that were performed at states in the program. The transition to Checkout mode is done by switching carts and reusing a stripped version of the code where little input is provided only card scanning and user verification at the end.

8.2.1.Flow Chart Cart Main

This diagram shows the flow of the Shopping Cart's Main

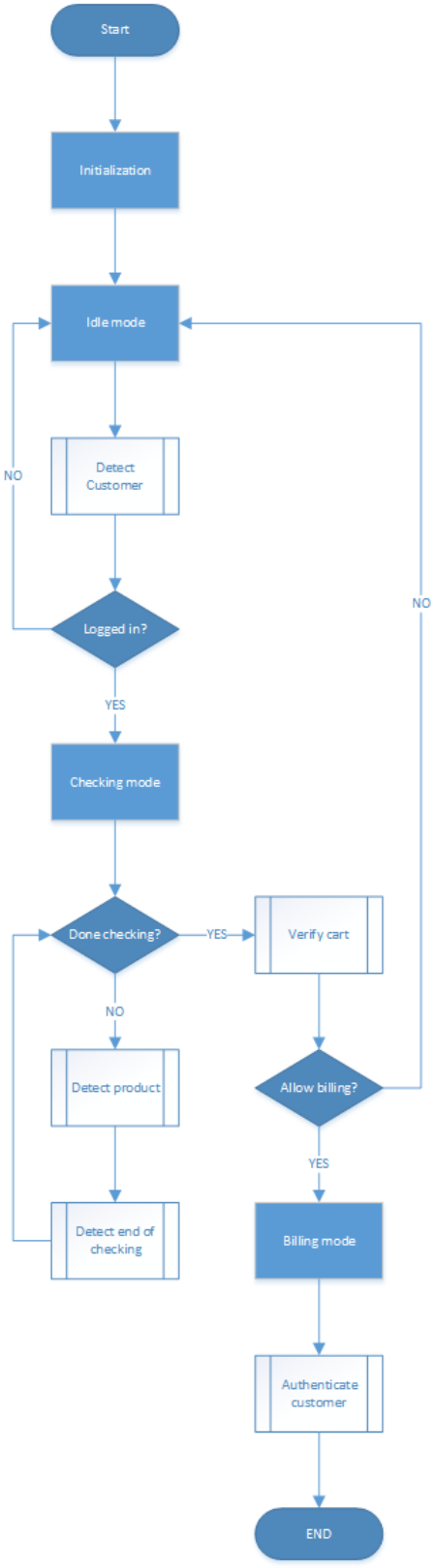
8:7 Flow Chart-Cart Main



8.2.2.Flow Chart Checkout Main

This diagram shows the flow of the Checkout terminals Main

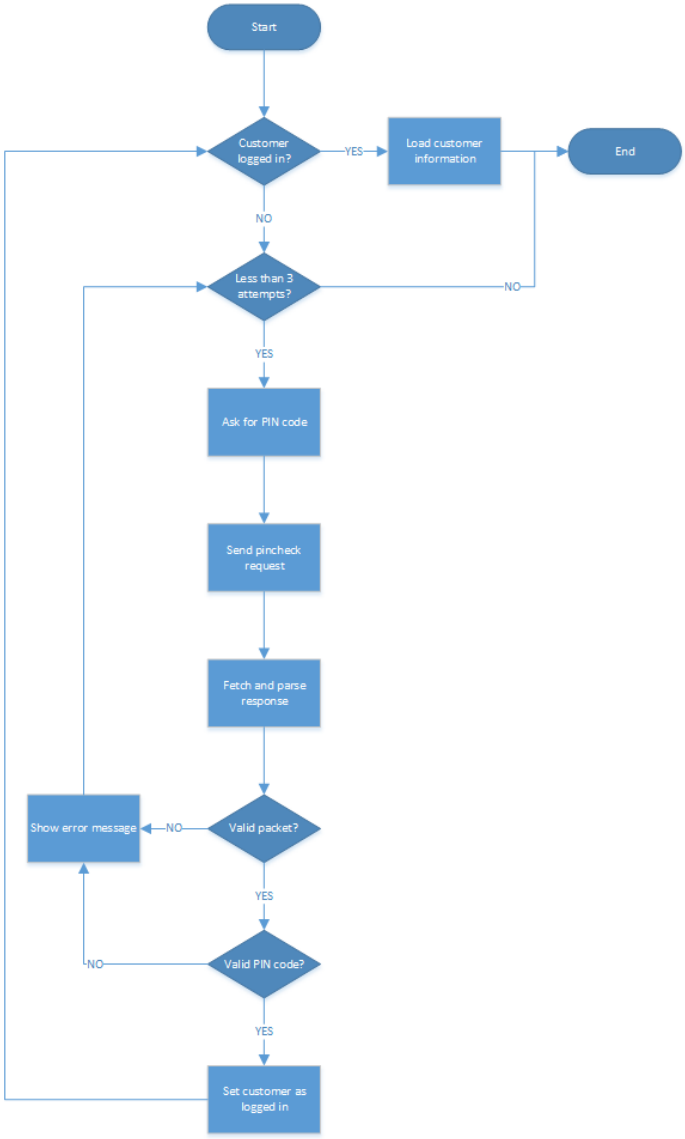
8:8 Flow Chart Checkout Main



8.2.3.Flow Chart Authenticate Customer

This diagram shows the flow of the Authenticate Customer Functionality.

8:9Flow Chart-Authenticate Customer



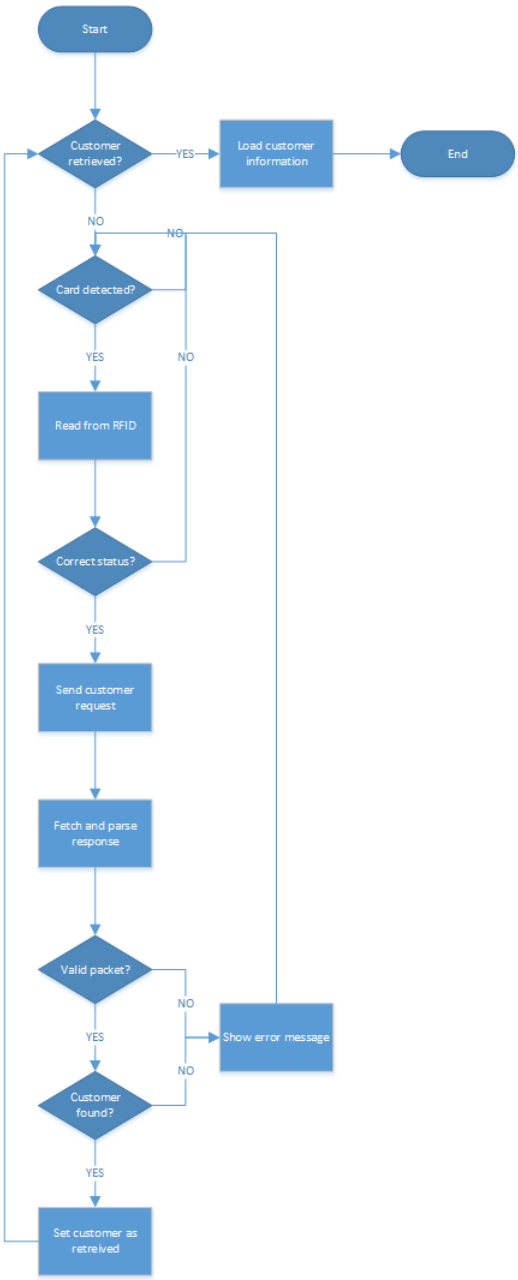
8.2.4.Flow Chart detect Product

The diagram on the right shows the flow of the detect Poduct Functionality.

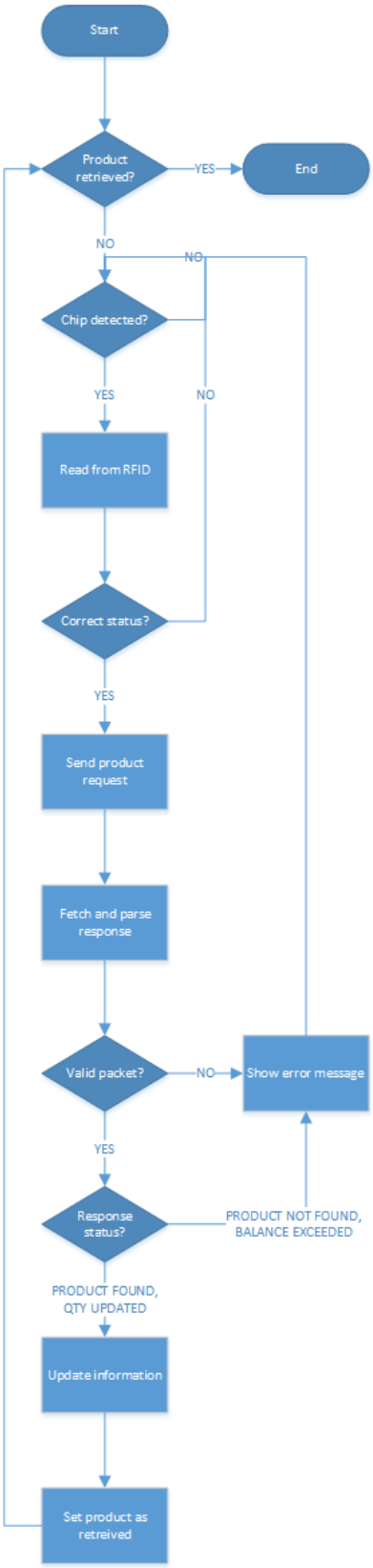
8.2.5.Flow Chart detect Customer

The Diagram below shows the flow of the detect Customer Functionality

8:11Flow Chart-Detect Customer



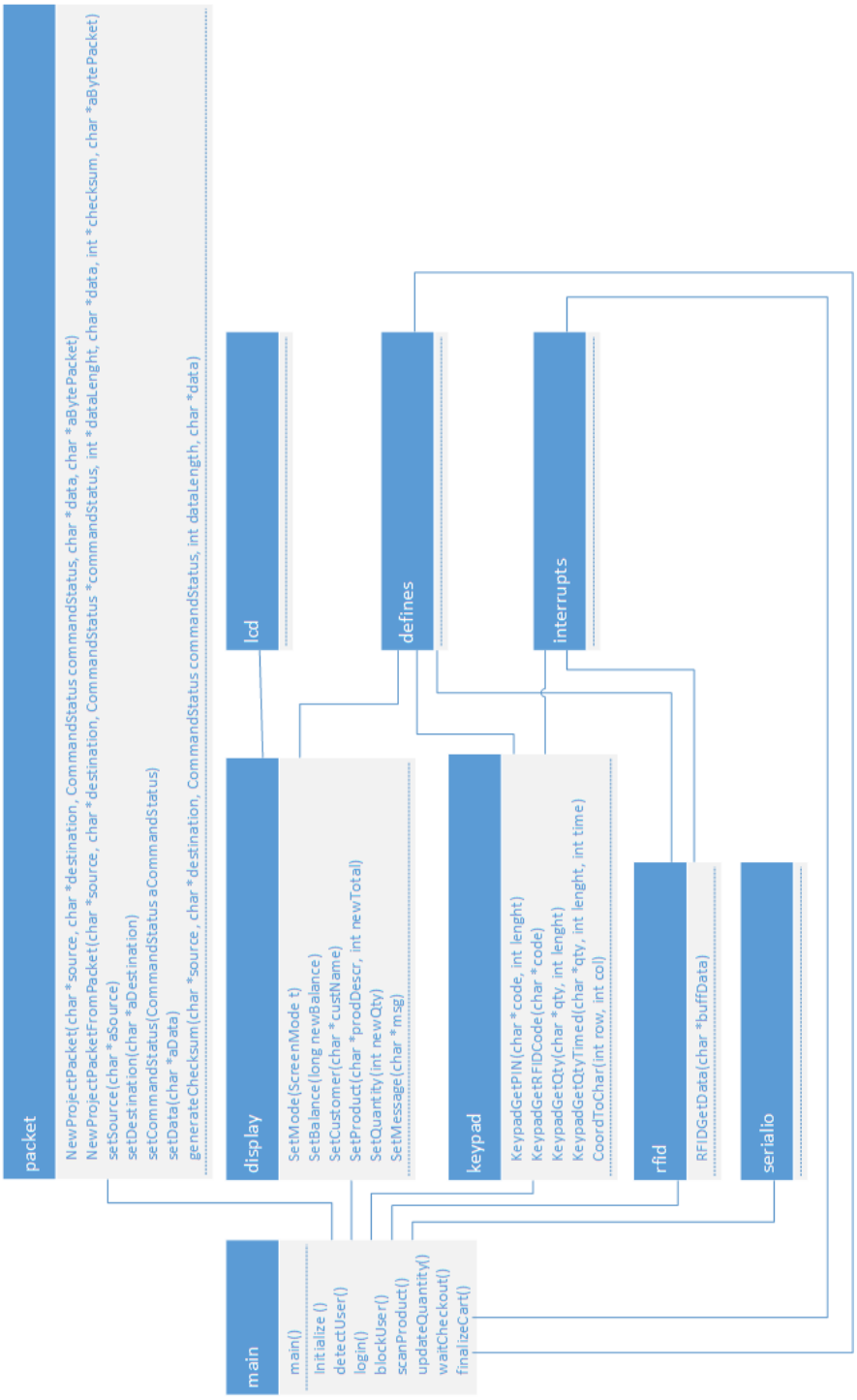
8:10 Flow Chart-Detect Product



8.2.6.Block Diagram-Code distribution Cart

The diagram shows the distribution of functionality in the C code. We represent the modules by their header name only.

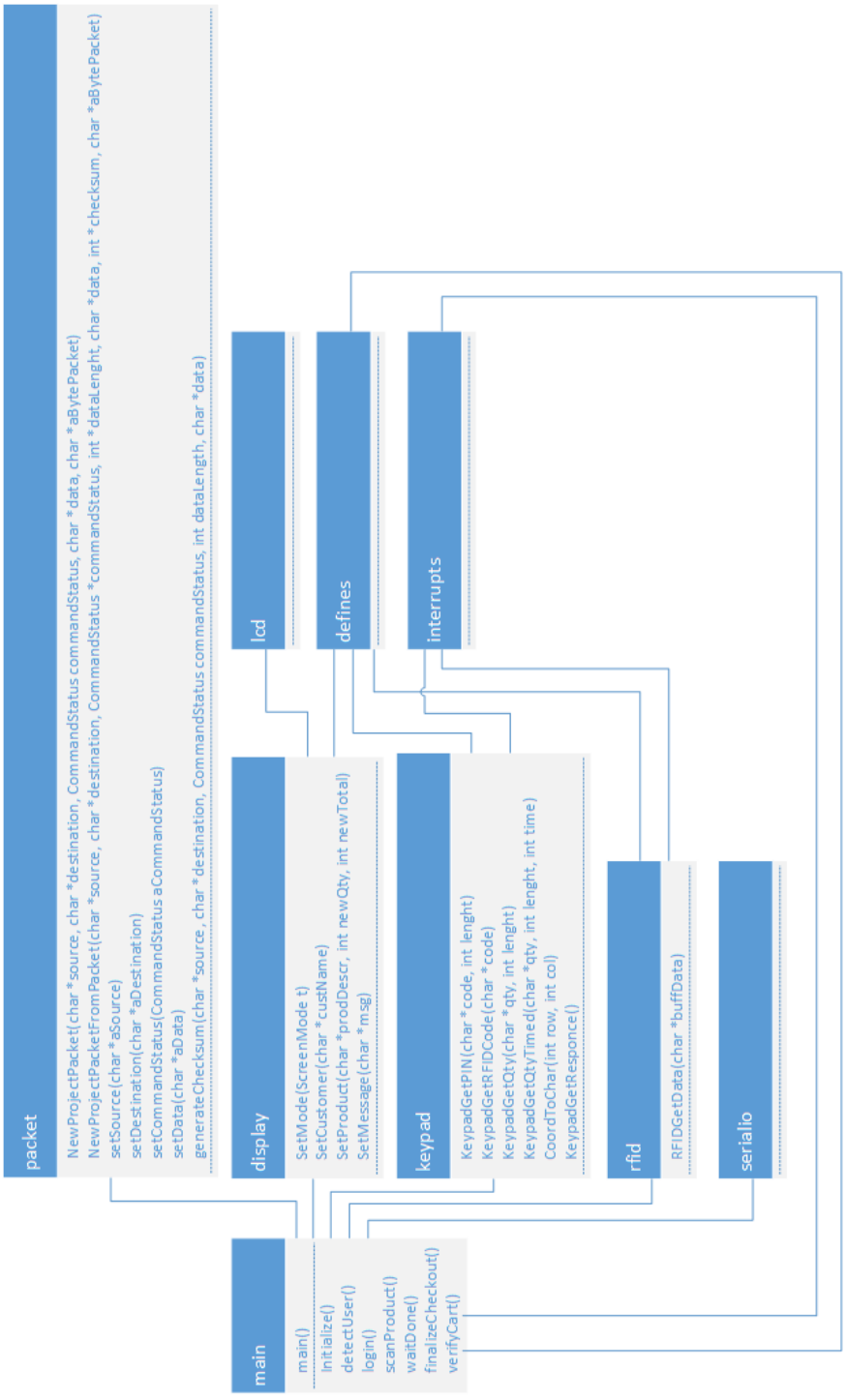
8:12Block Diagram-Code distribution-Cart



8.2.1.Block Diagram-Code distribution Checkout

The diagram shows the distribution of functionality in the C code. We represent the modules by their header name only.

8:13Block Diagram-Code distribution-Checkout



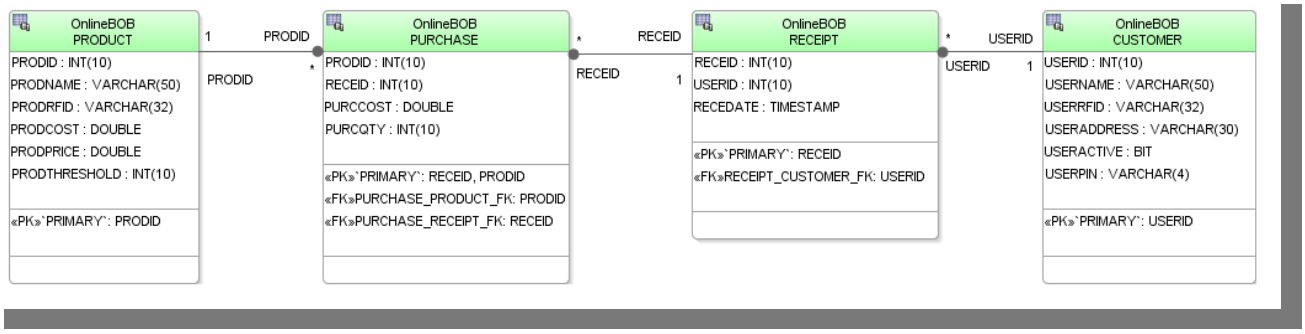
9.Problem Solution - PC part

This section will describe how we implemented and tackled the various features and aspects of the PC side of the solution.

9.1. Database Design

One of the first priorities we had in the PC part of this solution was sorting out the Database. We eventually opted for a simple database which would simulate interaction between the users the products(merchandize) and some of the basic functionality. These tables were chosen so that we could maintain some sort of cohesian to the between the different layers of code.

9:1Bobs Database



9.2. Software Design and Implementation

We tried to split the Project into different packets to deal with different aspects of the code.

Packet Responsibilities

- Bobsshop — This is where the main lies.
- Controller — Event manager for the Devices"COMPort"
- Data — Data Communication
  this design allows easy ammendment to other Databases or DataStructures.
  DataConn. Provides an universal interface to the Data.
- Elements — Base Classes
- Forms — Gui Interface Panels
- Graphics — Paint classes for Custom Graphs.
- Models — Table Models used in the Forms
- Protocol — Packet handling for serial communication
- Prototype — The Gui design for our Prototype
- Serial — The serial communication

9.3. Class Responsibilities

Here will follow a breakdown of all Classes used in the java program. The images included and all Class/Package diagrams will be located on the distributable media device(DVD) under ”DIAGRAMS\CLASS DIAGRAMS” should they not be legible in print.

9.3.1.Package Data

DataConnectionDB should be renamed to DataConnectionMySql because the current Database in use is an online Database located at www.freesqldatabase.com. We opted for solution because then we could all work on the same database rather then each their own local derbyDatabase.

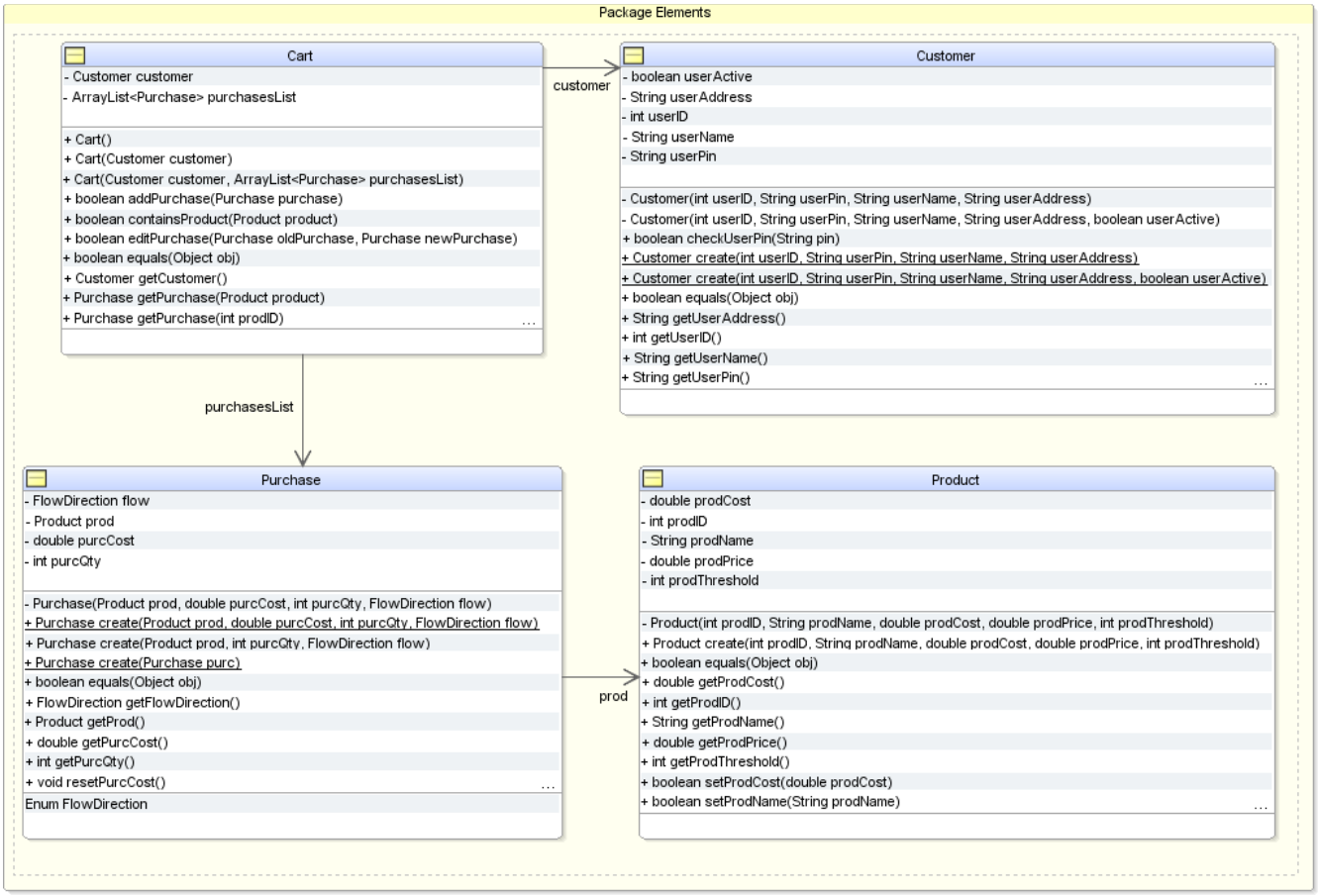
9:2Package Data

Class name: DataConnection<abstract>	
Superclass: Subclass:DataConnectionDB	
Responsibility: <abstract> get the arraylist of customer/ product/ purchase get the balance of the customer get the product quantity get the product trend modify customer detail modify product detail add the receipt which contains when and who bought what and how much cost.	Collaboration: Customer Product Purchase

Class name: DataConnectionDB	
Superclass:DataConnection Subclass:	
Responsibility: the connection with the database get the arraylist of customer/ product/ purchase get the balance of the customer get the product quantity get the product trend modify customer detail modify product detail add the receipt which contains when and who bought what and how much cost.	Collaboration: DataConnection Customer Product Purchase

9.3.2.Package Elements

9:3Package Elements



Class name: Product	
Superclass: Subclass:	
Responsibility: define product detail : prodName; prodPrice ; priceCost; prodID ; prodThreshold. getters and setters allowed to create products	Collaboration: CustomerInterfaceProductsInterface DataConnectionDB

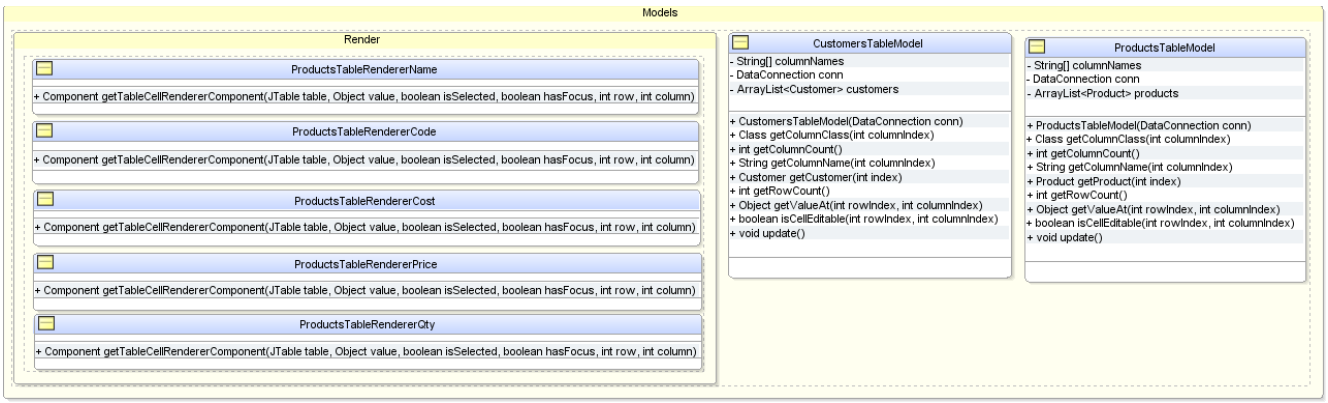
Class name: Cutomer	
Superclass: Subclass:	
Responsibility: define customer detail : userName; userID ; userAddress; userPin. getters and setters check if userPin is valid allowed to create customers	Collaboration: CustomerInterface ProductsInterface DataConnectionDB

Class name: Purchase	
Superclass: Subclass:	
Responsibility: define purchase detail : purcCost; purcQty; get Product detail; enum flowDirection{ Restock,(allowed to restock the product) Sell, (allowed to sell the product) Topup;(allowed to add user's balance) }. getters and setters check if userPin is valid allowed to create purchase	Collaboration: Product DataConnectionDB

Class name: Cart	
Superclass: Subclass:	
Responsibility: Define ArrayList<Purchase>: Add the Purchase Delete the Purchase Modify the Purchase check if ArrayList<Purchase> contains product. Get Customer detail: get the total cost by the customer	Collaboration: Purchase Customer CheckoutEventManager CartEventManager

9.3.3.Package models

9:4Package Models

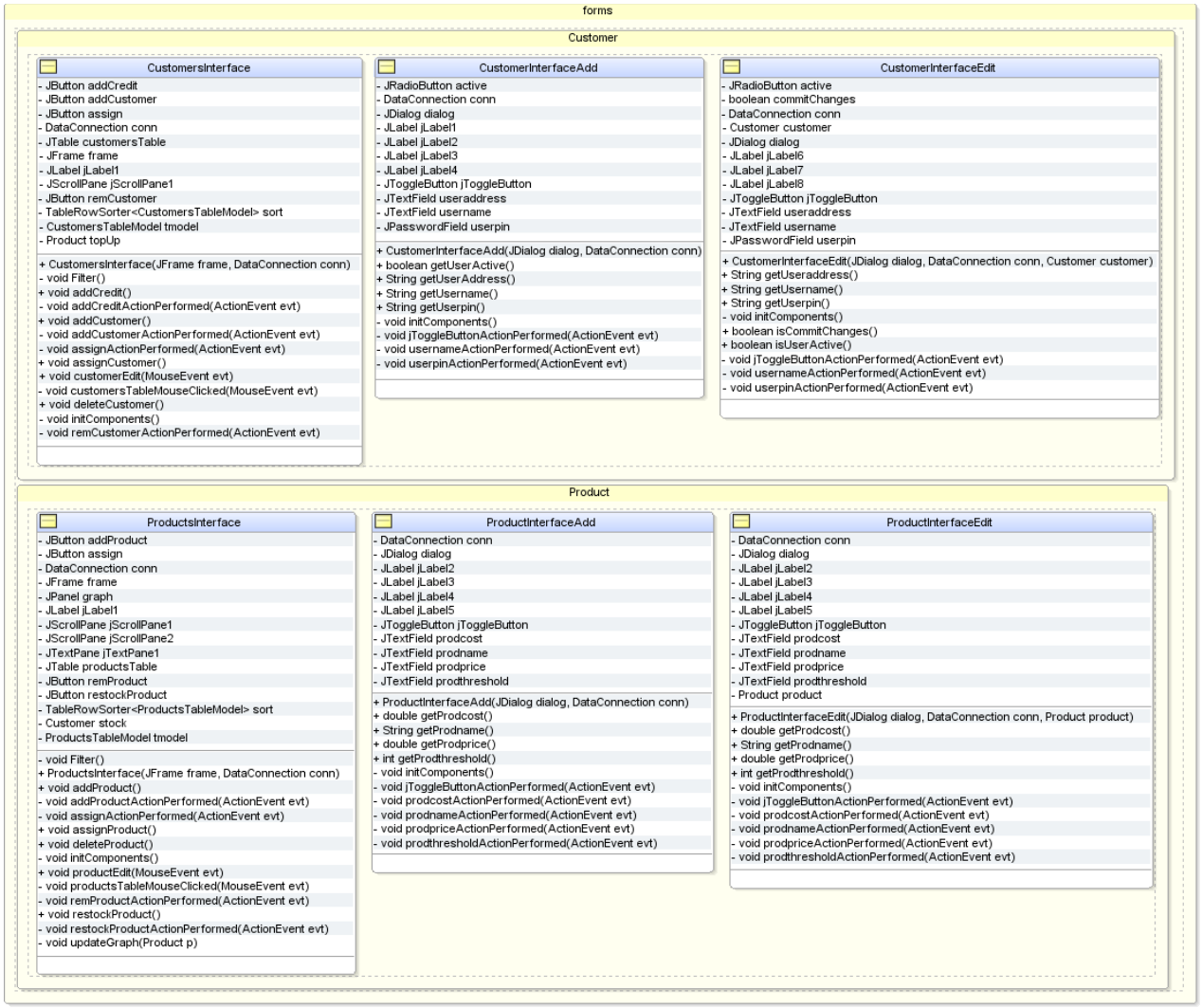


Class name: CutomersTableModel	
Superclass:AbstractTableModel Subclass:	
Responsibility: declare ArrayList<Customer> get customer info columnNames{ “Name”,"Address","Active","Current Balance" } Get Customer instance at index getters (colunms and row and ValueAt) update Customer info	Collaboration: Customer DataConnection

Class name: ProductsTableModel	
Superclass:AbstractTableModel Subclass:	
Responsibility: declare ArrayList<Product> get product info columnNames{ “Code”,"Name","Unit.Cost ","Unit.Price", " Quantity left", "Threshold" } Get Product instance at index getters (colunms and row and ValueAt) update product info	Collaboration: Product DataConnection Product

9.3.4.Package forms

9:5Package Forms



Class name: CutomersInterface	
Superclass:JPanel Subclass:	
Responsibility: shows the customer table on the customer interface which contains sorts of functionality that allow admin(Bob) to modify customer info on GUI	Collaboration: Customer DataConnection CustomersTableModel CustomerInterfaceAdd/Edit

Class name: CustomersInterfaceAdd	
Superclass:JPanel Subclass:	
Responsibility: The dialog on adding the customer ,which contains sorts of functionality that allow admin(Bob) to add customer to the customer table.	Collaboration: Customer DataConnection CustomersInterface

Class name: CustomersInterfaceEdit	
Superclass:JPanel Subclass:	
Responsibility: The dialog on editing the customer ,which contains sorts of functionality that allow admin(Bob) to edit customer to the customer table.	Collaboration: Customer DataConnection CustomersInterface

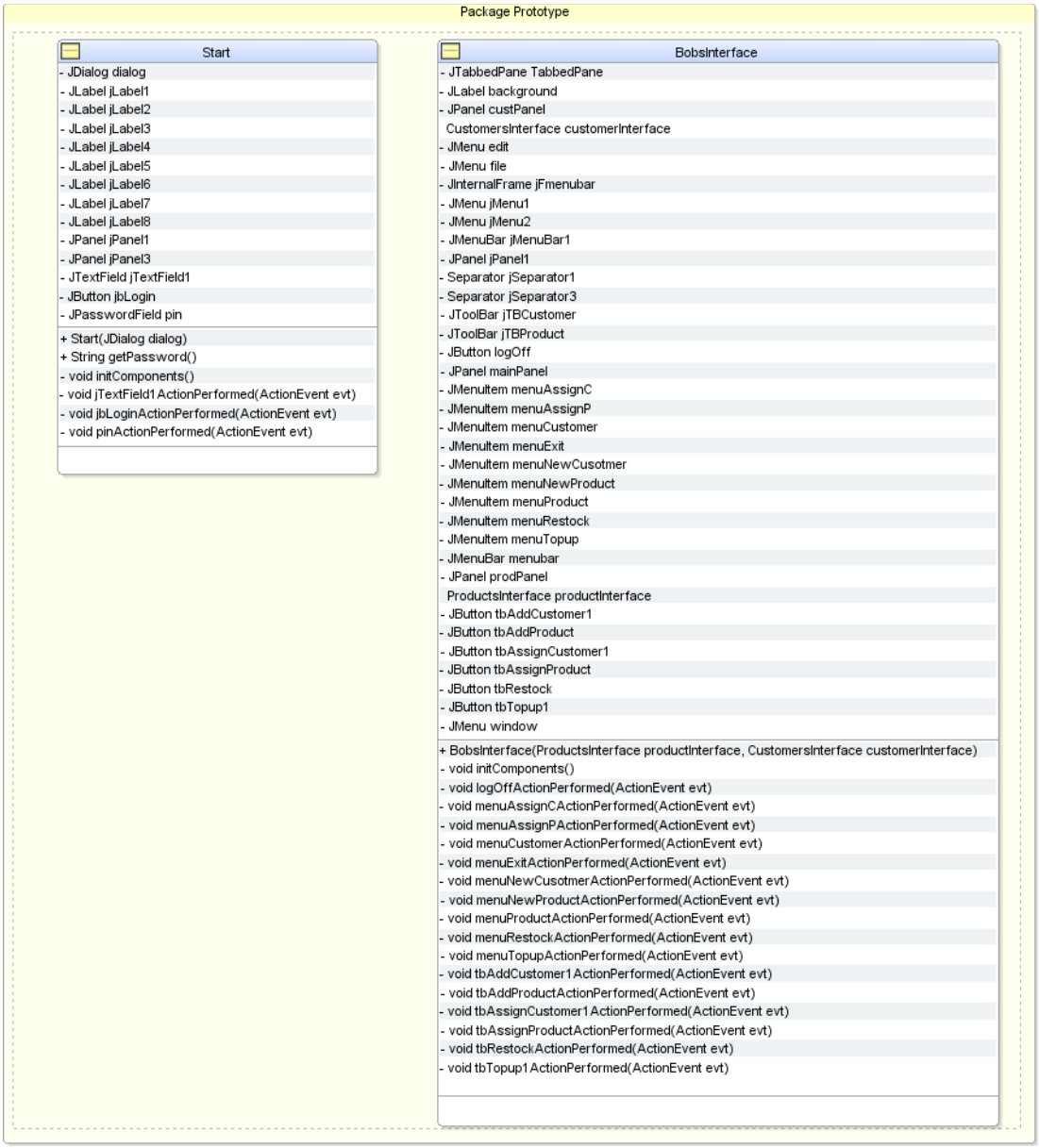
Class name: ProductsInterface	
Superclass:JPanel Subclass:	
Responsibility: show the products table on the product interface,which contains sorts of functionality that allow admin(Bob) to modify product info on GUI	Collaboration: Product DataConnection ProductsTableModel ProductInterfaceAdd/Edit

Class name: ProductsInterfaceAdd	
Superclass:JPanel Subclass:	
Responsibility: The dialog on adding the product ,which contains sorts of functionality that allow admin(Bob) to add product to the product table.	Collaboration: Product DataConnection ProductsInterface

Class name: ProductsInterfaceEdit	
Superclass:JPanel Subclass:	
Responsibility: The dialog on editing the product ,which contains sorts of functionality that allow admin(Bob) to edit product on the product table.	Collaboration: Product DataConnection ProductsInterface

9.3.5.Package Prototype

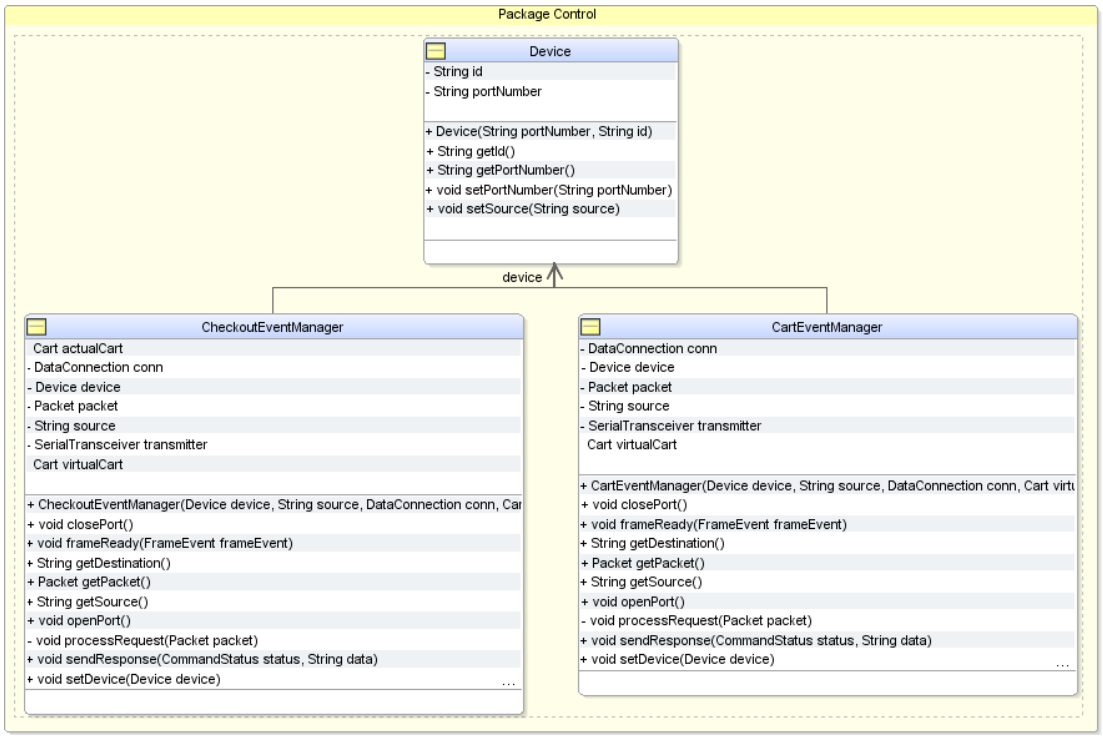
9:6Package Prototype



Class name: BobInterface	
Superclass:JPanel Subclass:	
Responsibility: The main interface has menubar, toolbar , several functionalities and also contains the products table and customer table ,which contains sorts of functionality that allow admin(Bob) to modify product/customer info.	Collaboration: ProductInterface CustomerInterface

Class name: Start	
Superclass:JPanel Subclass:	
Responsibility: The dialog which need admin to type the pin code to log on to show the BobInterface.	Collaboration: BobInterface

9.3.1.Package Control

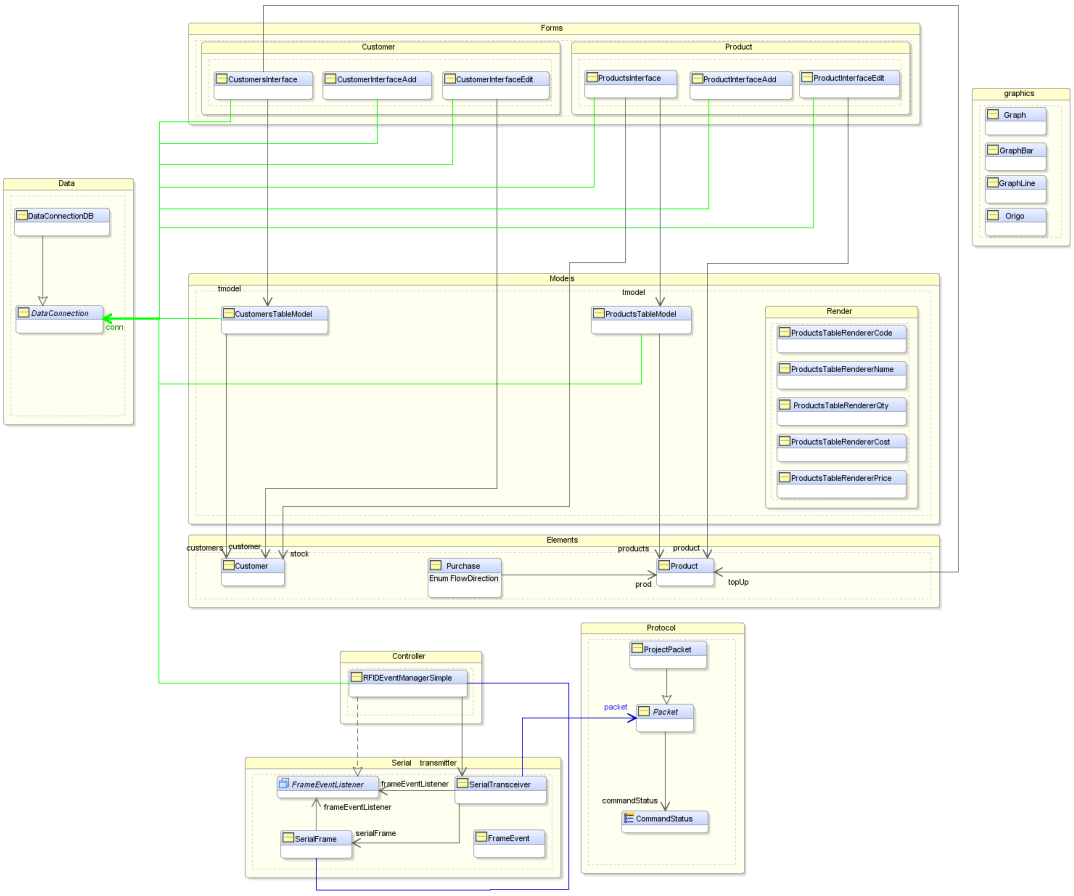


9.4. Class Diagram

The following pages will show a series of Class diagrams to give an overview of our solution larger views can be found on the DVD under Diagrams.

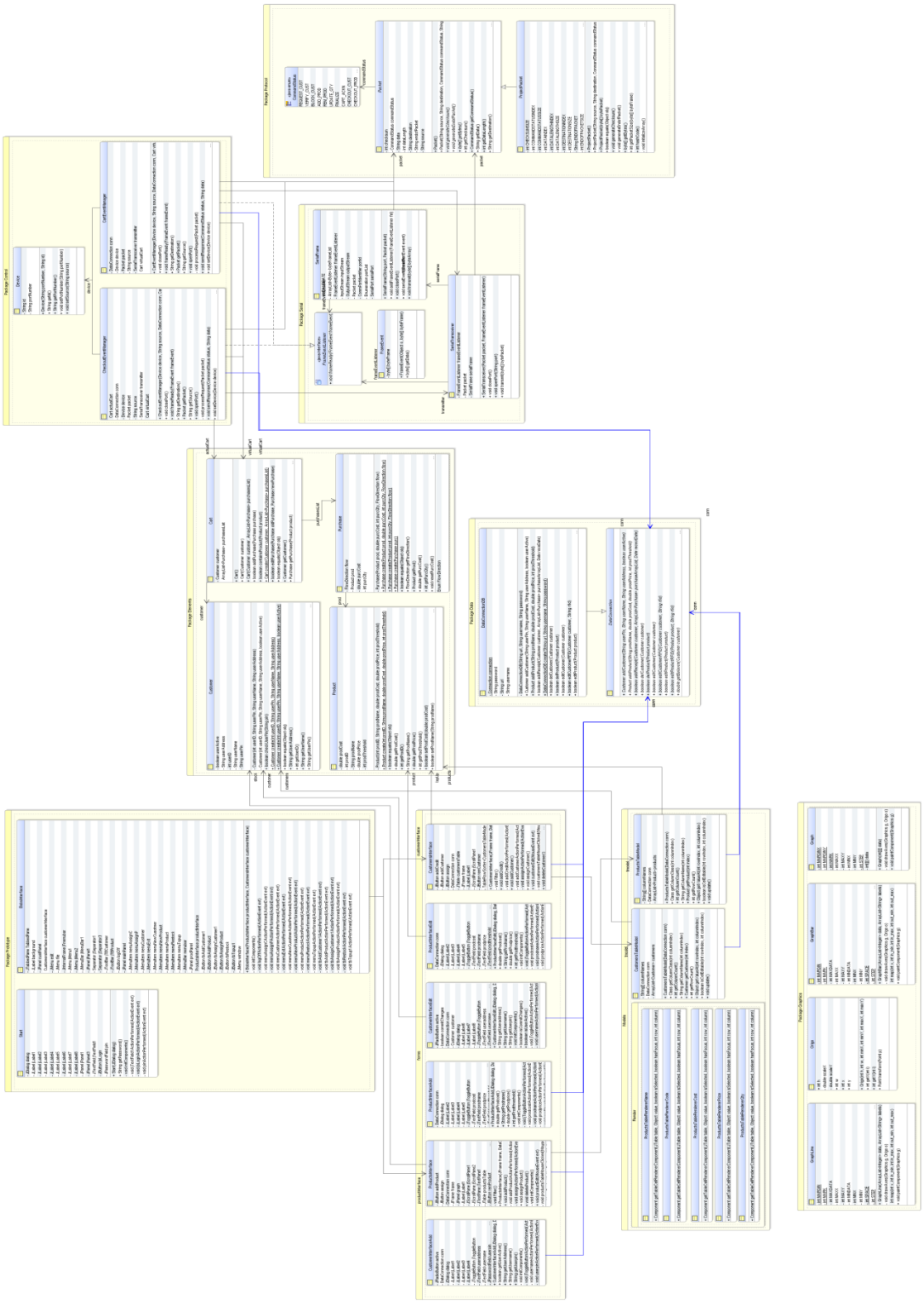
9.4.1. Total Class Diagram Phase1

The diagram below is an overview of the total Class diagram of the prototype at phase 1.



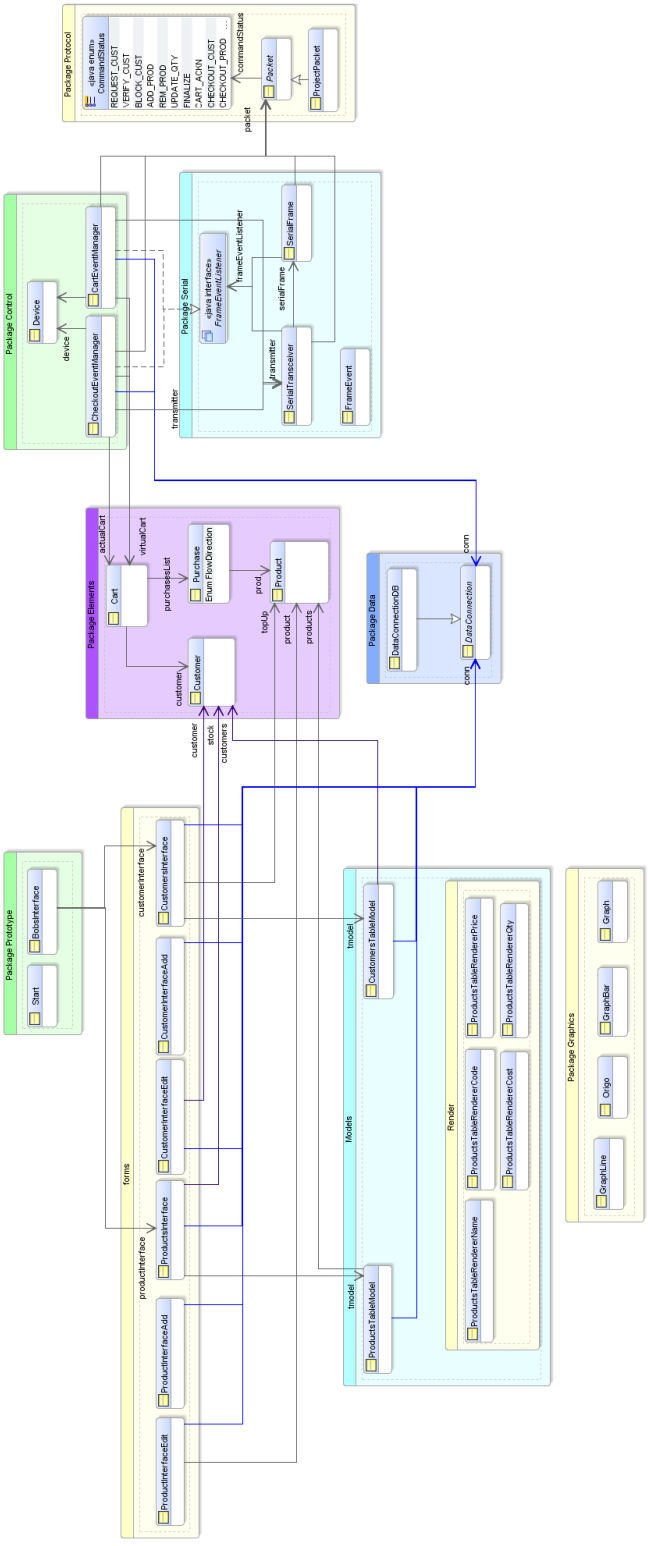
9.4.2.Total Class Diagram Now

9:7 Total Class Diagram



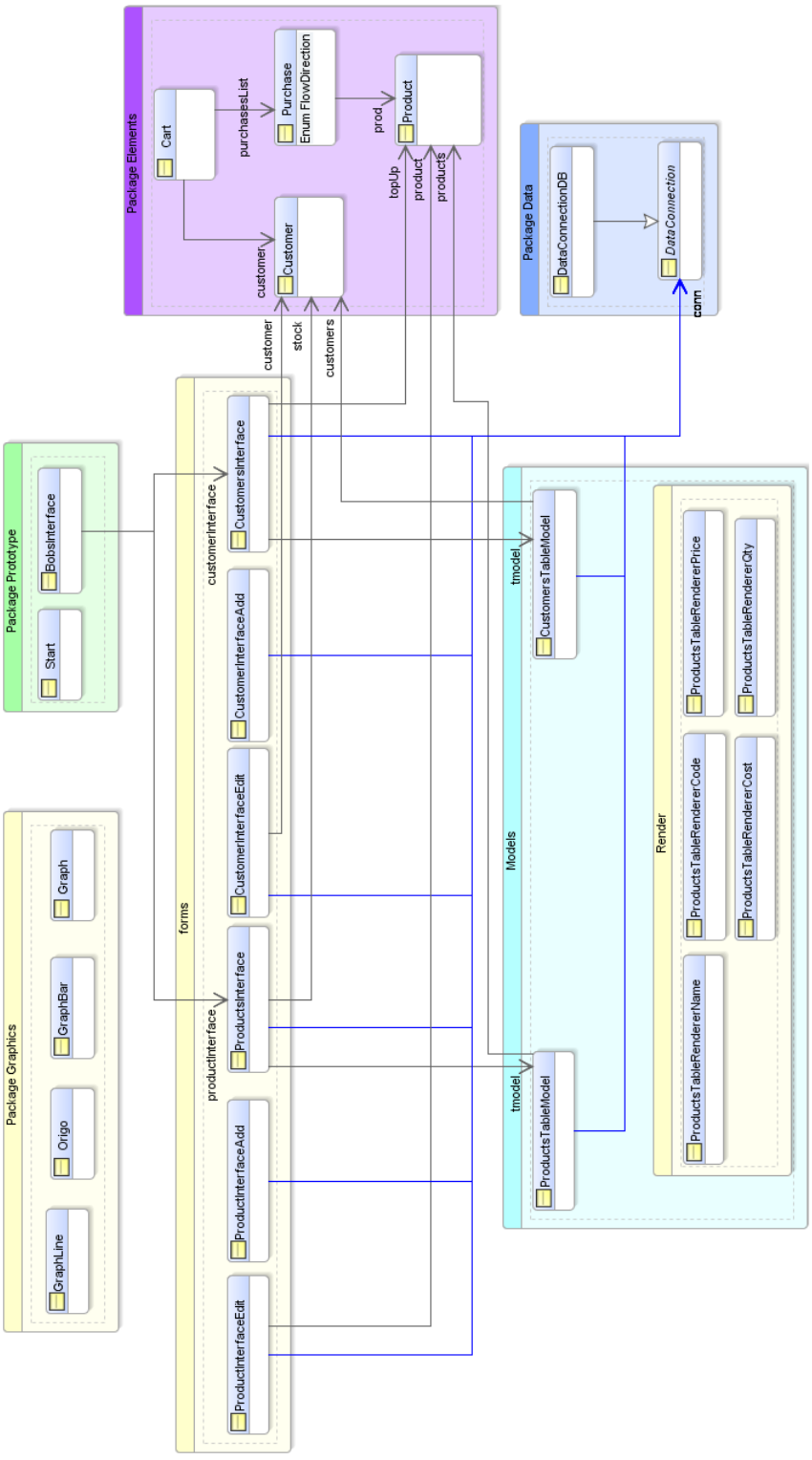
9.4.3.Class overview

9:8Total Class Overview



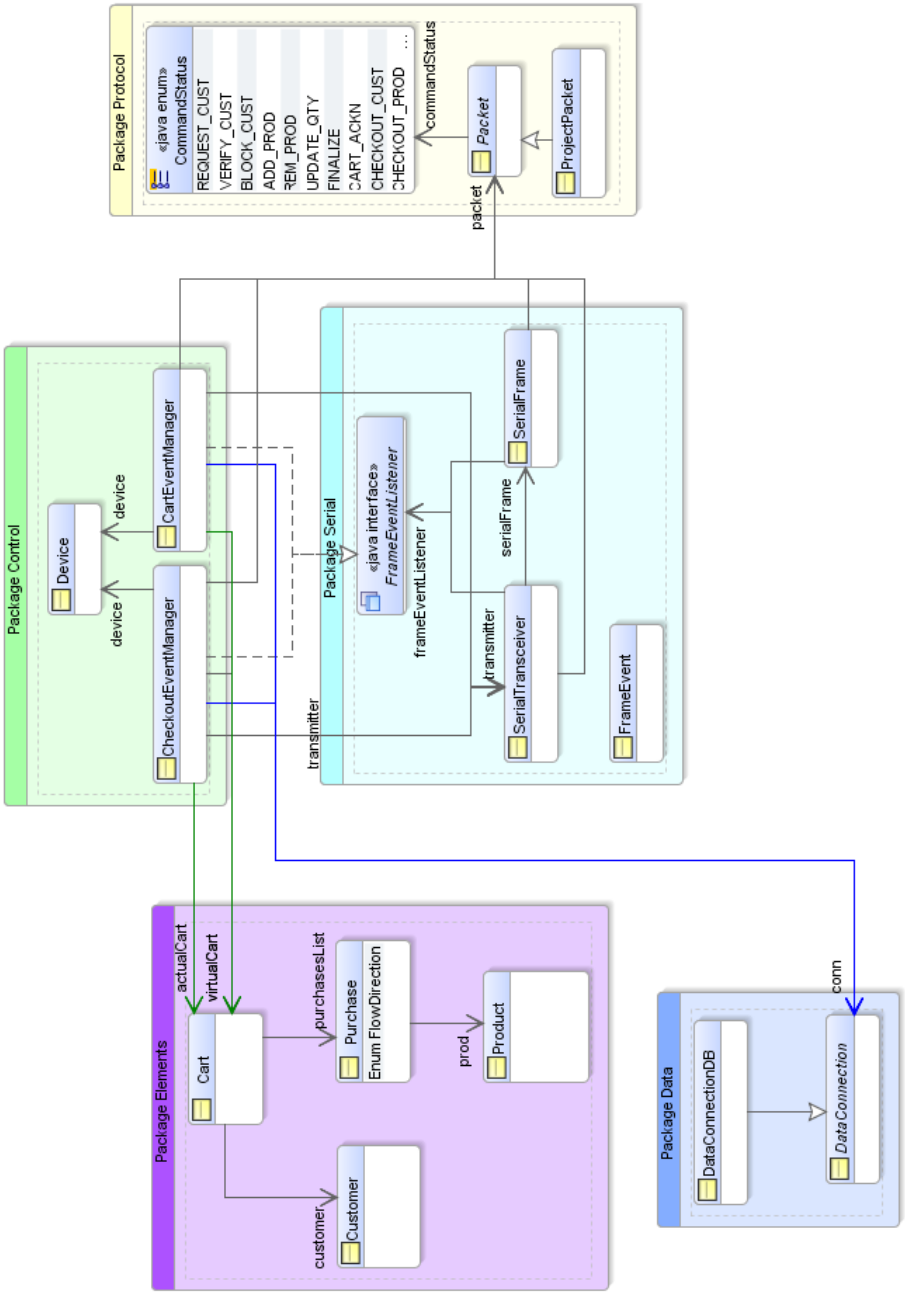
9.4.4.Class Overview GUI

9:9Class Overview GUI



9.4.5.Class Overview Server side

9:10Class Overview Server Side



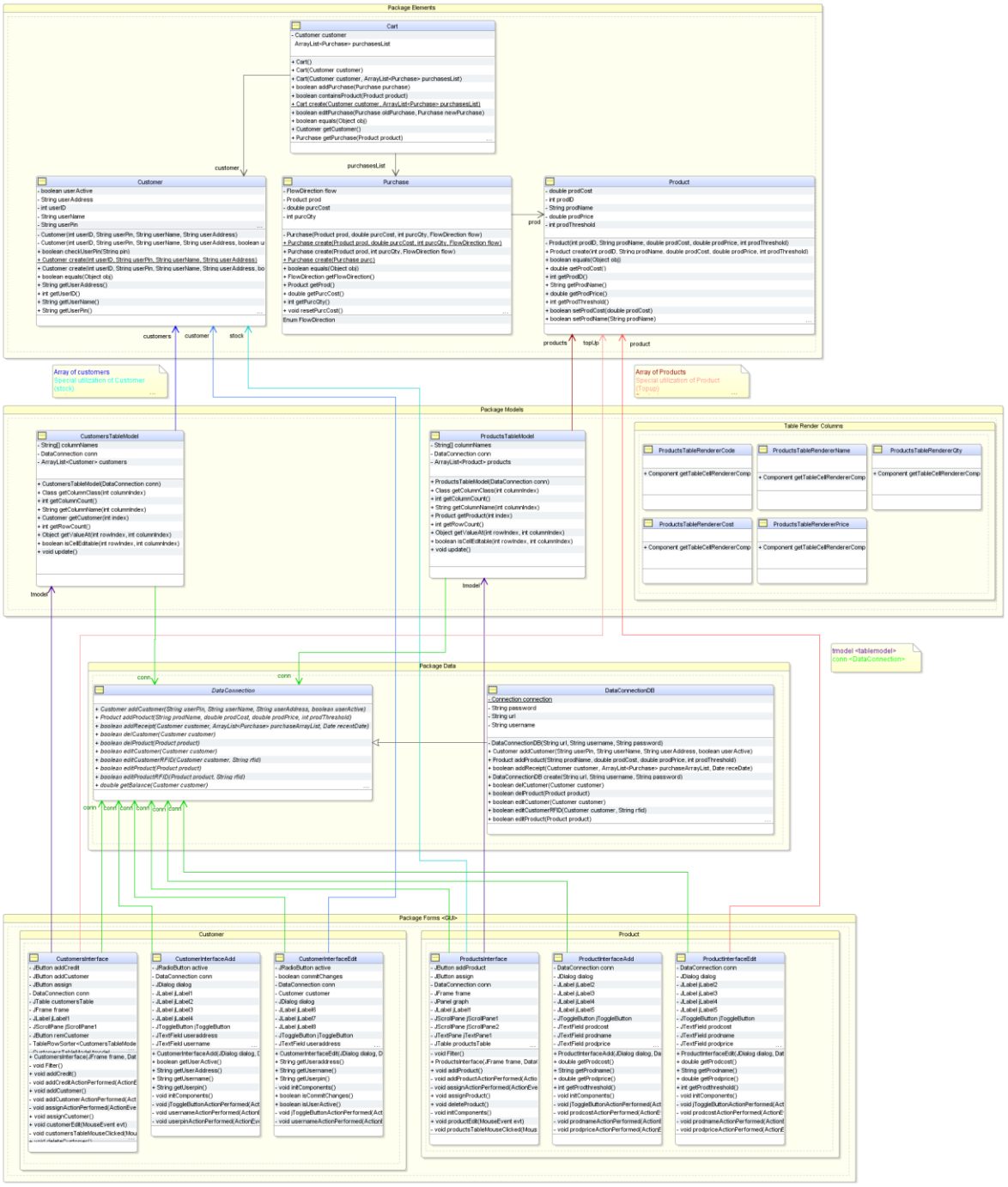
9.5. Design model

Each of the Elements in the Elements Packet are made using the Static Factory Method. We looked into improving some of the code with other design patterns but ran out of time to give it our full attention.

Our overall design for the GUI was a more emulated 3 tier solution with a database at the base the functionality / structure just above and then the GUI at the top.

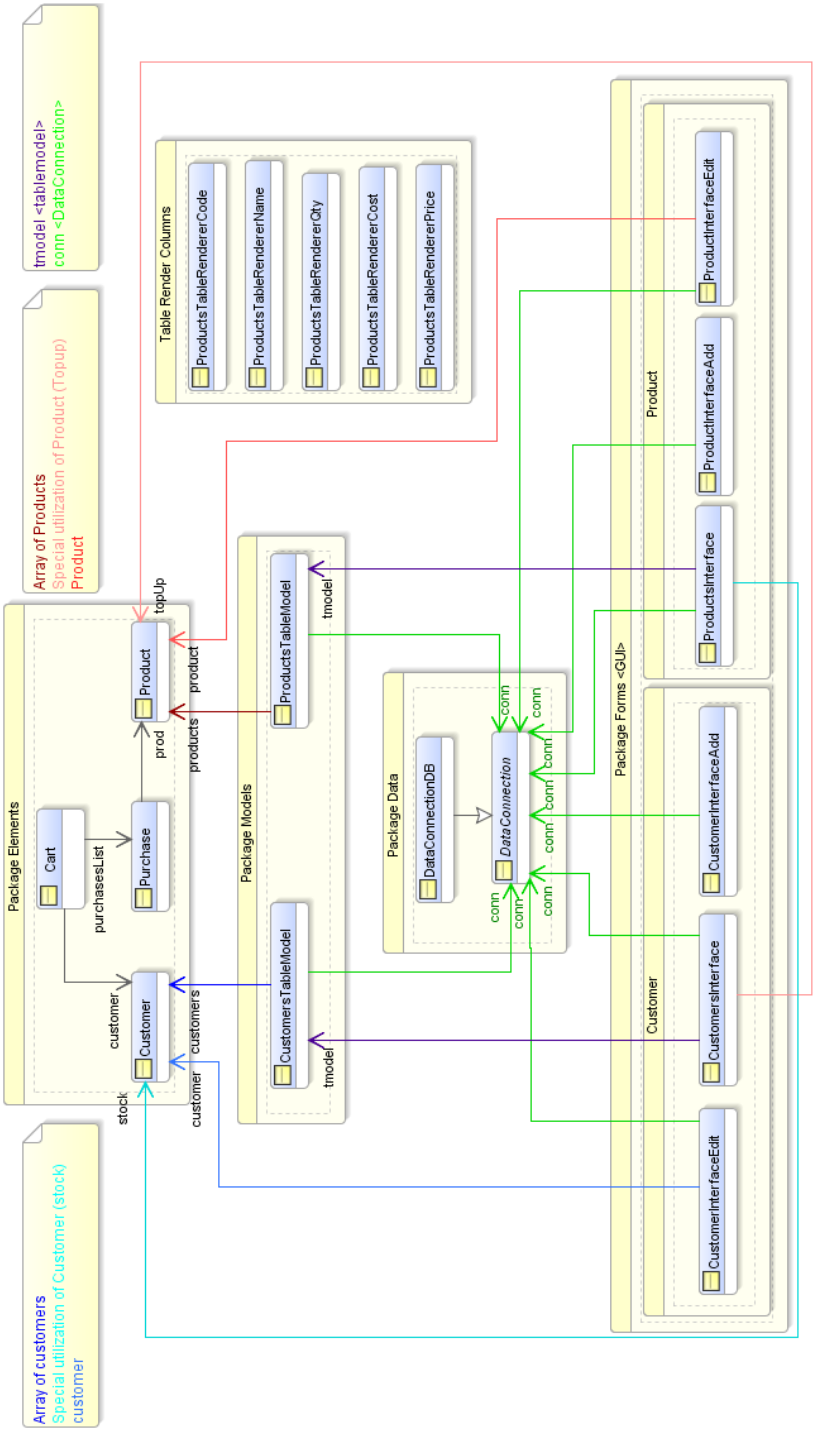
The next two figures will try to illustrate the Data flow and structure we tried to accomplish.

9:11Data flow Model Full



This image gives a better overview of the Data movement in the functionality of Bob's GUI. The Data Package in the middle show the Database Centric design we were going for. There are some special uses of the base elements. Topup is a special type of product that allows the customer to have credit added to their account without a purchase attached. The stock is where Bob buys stock without a sales price.

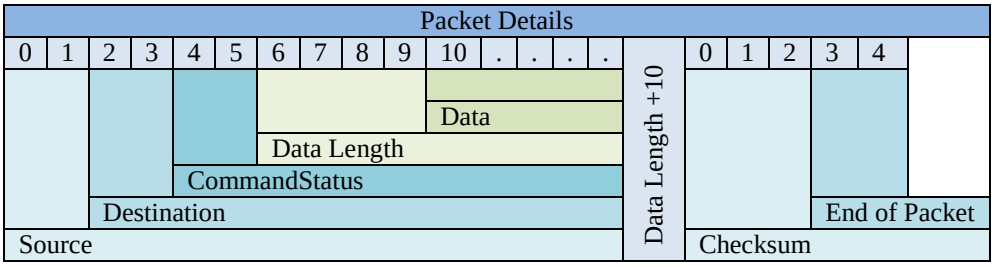
9:12Data flow Model simple



10. Problem solution - Communication

This was one of the trickiest issues when it came to this project.
We ported some of the design from the java to the c code. Where we encapsulated the protocol into a packet model.
And tried as much as possible to match both sides of the communication.

10.1. Packet Design



Each packet contains which is sent as an Ascii string. The table below shows the different ways the packet was defined in each of the programming languages to show the similarities in the structure.

Java	C
`privatefinal int SOURCEINDEX = 0;`	`#defineSOURCEINDEX0`
`privatefinal int SOURCESIZE = 2;`	`#defineSOURCESIZE2`
`privatefinal int DESTINATIONINDEX = 2;`	`#defineDESTINATIONINDEX2`
`privatefinal int DESTINATIONSIZE = 2;`	`#defineDESTINATIONSIZE2`
`privatefinal int COMMANDSTATUSINDEX = 4;`	`#defineCOMMANDSTATUSINDEX4`
`privatefinal int COMMANDSTATUSSIZE = 2;`	`#defineCOMMANDSTATUSSIZE2`
`privatefinal int DATALENGTHINDEX = 6;`	`#defineDATALENGTHINDEX6`
`privatefinal int DATALENGTHSIZE = 4;`	`#defineDATALENGTHSIZE4`
`privatefinal int DATAINDEX = 10;`	`#defineDATAINDEX10`
`privatefinal int CHECKSUMSIZE = 3;`	`#defineCHECKSUMSIZE3`
`privatefinal String ENDOFPACKET = "\r\n";`	`#defineENDOFPACKET "\0"`
`privatefinal int ENDOFPACKETSIZE = ENDOFPACKET.length();`	`#defineENDOFPACKETSIZE strlen(ENDOFPACKET)`
`privatefinal int HEADERLENGTH = SOURCESIZE + DESTINATIONSIZE+ COMMANDSTATUSSIZE + DATALENGTHSIZE + CHECKSUMSIZE + ENDOFPACKETSIZE;`	`#defineHEADERLENGTH( SOURCESIZE+ DESTINATIONSIZE+ COMMANDSTATUSSIZE+ DATALENGTHSIZE+ CHECKSUMSIZE+ ENDOFPACKETSIZE)`

10.1. Communication Protocol

The table below gives a overview of the types of messages that can be sent back and forth. In the C part the logic of what order messages are sent depends on the state of the program . in the Java it responds to initial requests on a ”server”.

Some messages are sent for debugging / tracing purposes only and are not used by the micro-controller. They still may be usefull for future FSM Oriented design.

Java	C

`//Cart => PC`
`//08 is not first priority`
`REQUEST_CUST("01"), //`
`VERIFY_CUST("02"), //`
`BLOCK_CUST("03"), //`
`ADD_PROD("04"), //`
`REM_PROD("05"), //`
`UPDATE_QTY("06"), //`
`FINALIZE("07"), //`
`CART_ACKN("08"), //`
`//Checkout => PC`
`//38 is not first priority`
`CHECKOUT_CUST("31"),`
`CHECKOUT_PROD("34"),`
`CHECKOUT_DONE("37"),`
`CHECKOUT_PINCHECK("32"),`
`CHECKOUT_ACKN("38"),`
`//PC => Cart`
`//'-ed' are not first priority`
`CART_CUSTOMER_FOUND("11"),`
`CART_CUSTOMER_NOTFOUND("21"),`
`CART_PIN_MATCHES("12"),`
`CART_PIN_DOESNTMATCH("22"),`
`BLOCKED("13"),`
`PRODUCT_FOUND("14"),`
`PRODUCT_NOTFOUND("24"),`
`REMOVED("15"),`
`UPDATED("16"),`
`BALANCE_EXCEEDED("17"),`
`FINALIZED("18"),`
`//PC => Checkout`
`//'-ed' are not first priority`
`CHECKOUT_CUSTOMER_FOUND("41"),`
`CHECKOUT_CUSTOMER_NOTFOUND("51"),`
`CHECKOUT_PIN_MATCHES("42"),`
`CHECKOUT_PIN_DOESNTMATCH("52"),`
`CHECKOUT_PRODUCT_ADDED("44"),`
`CHECKOUT_PRODUCT_NOTFOUND("54"),`
`CHECKOUT_CARTS_MATCH("47"),`
`CHECKOUT_CARTS_DONTMATCH("57"),`
`CHECKOUT_SUCCEEDED("99"),`
`//checksum issues`
`SEND_COMMAND_AGAIN("10"),`
`SEND_STATUS_AGAIN("09");`
	`typedefenum{`
	`//Cart => PC`
	`//08 is not first priority`
	`REQUEST_CUST=1,`
	`VERIFY_CUST=2,`
	`BLOCK_CUST=3,`
	`ADD_PROD=4,`
	`REM_PROD=5,`
	`UPDATE_QTY=6,`
	`FINALIZE=7,`
	`CART_ACKN=8,`
	`//Checkout => PC`
	`//38 is not first priority`
	`CHECKOUT_CUST=31,`
	`CHECKOUT_PROD=34,`
	`CHECKOUT_DONE=37,`
	`CHECKOUT_PINCHECK=32,`
	`CHECKOUT_ACKN=38,`
	`//PC => Cart`
	`//'-ed' are not first priority`
	`CART_CUSTOMER_FOUND=11,`
	`CART_CUSTOMER_NOTFOUND=21,`
	`CART_PIN_MATCHES=12,`
	`CART_PIN_DOESNTMATCH=22,`
	`BLOCKED=13,`
	`PRODUCT_FOUND=14,`
	`PRODUCT_NOTFOUND=24,`
	`REMOVED=15,`
	`UPDATED=16,`
	`BALANCE_EXCEEDED=17,`
	`FINALIZED=18,`
	`//PC => Checkout`
	`//'-ed' are not first priority`
	`CHECKOUT_CUSTOMER_FOUND=41,`
	`CHECKOUT_CUSTOMER_NOTFOUND=51,`
	`CHECKOUT_PIN_MATCHES=42,`
	`CHECKOUT_PIN_DOESNTMATCH=52,`
	`CHECKOUT_PRODUCT_ADDED=44,`
	`CHECKOUT_PRODUCT_NOTFOUND=54,`
	`CHECKOUT_CARTS_MATCH=47,`
	`CHECKOUT_CARTS_DONTMATCH=57,`
	`CHECKOUT_SUCCEEDED=99,`
	`//checksum issues`
	`SEND_COMMAND_AGAIN=10,`
	`SEND_STATUS_AGAIN=9`
	`}`
	`CommandStatus;`

10.1. Communication Checksum

Below is a Brief code comparison for how the Checksum is generated in each Programming language.

Java

```
int sum = 0;

for(int i=0; i<SOURCESIZE; i++) {
     sum+=getSource().charAt(i);
     }
for(int i=0;<DESTINATIONSIZE; i++) {
     sum+=getDestination().charAt(i);
     }
sum +=
Integer.parseInt(getCommandStatus().getCode());
sum += getDataLength();

for(int i=0; i<getDataLength(); i++) {
     sum+=getData().charAt(i);
     }
setChecksum(sum%1000);
```

C

```
intsum=0;
inti;
for(i=0;i<SOURCESIZE;i++)
   sum+=source[i];

for(i=0;i<DESTINATIONSIZE;i++)
   sum+=destination[i];

sum+=commandStatus;

sum+=dataLength;

for(i=0;i<dataLength;i++)
   sum+=data[i];

return(sum%1000);
```

11. Testing

Testing was a cumbersome and tedious task that took alot of resources from the project. It had some positive results:

- We managed to fix some of the unexpected results by finding that they existed. Fixing some of the logic and placement of code.
- The acceptance tests were more to see whether we could do with our solution what we set out to. And running them through systematically also gave us some bugs which were corrected as we did multiple itterations of the Test.

We did more testing then we have specified however the lacking documentation forced us to ommit this infomration.

11.1. Acceptance Tracing

	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9	TC10	TC11
R1: BOB											
R1.1											
R1.2											
R1.3											
R1.4											
R1.5											
R1.6											
R1.7											
R1.8											
R2: CUSTOMER											
R2.1											
R2.2											
R2.3											
R2.4											
R2.5											
R2.6											
R3: CART TERMINAL (RFID Scanner, Micro-controller and Custom Build)											
R3.1											
R3.2											
R3.3											
R3.4											
R3.5											
R3.6											
R4: CHECK-OUT TERMINAL / POINT. (simulated by (terminal as above)											
R4.1											
R4.2											
R4.3											
R4.4											
R4.5											
R4.6											

	Implemented
	Work Around
	Not-implemented

11.2. Microcontroller part

It was hard to test the Micro-controller without actual access to the internal workings of the microcontroller. We therefore were limited to a couple of proof of concept tests and an Acceptance test.

11.2.1. Proof of Concept-Keypad - Yin

LCD Display: The LCD Display has 4 lines of 20 characters each.

		Character																		
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Row	1																				
	2																				
	3	Messages by the system(Type Pin/Qty & press A)																			
	4	User Input(Type the Pin and Quantity)																			

Keypad:

A represents Enter;
B represents Backspace;
C represents Erase;
0-9 is the number from 0 to 9.

The Pin code has 6 character which sets to 147258.
The Quantity has 9 character which sets to 123456789.

Test:

This test is a simple one that test the keypad working. The LCD display the “Type Pin & press A” first and need to type the pin and after press Enter_Key A, the system will message the “Type Qty & Press A” and then need to type the quantity, and if there is something wrong with the typing, you could press B for backspace and press C for erase all and type again.

The source code for this is located on the DVD / Release Materialin the foler ”SourceCode\Tests\Keypad-Jiachen”

11.2.2. Display - Rudolf

Due to lack of access to the AVR boards it was hard to collaborate on the c. I there fore concentrated on trying to document our Solution this included the Doxygen that was included in this test and the final Shopping cart. This can be found on the DVD / Release Material.

When i did have access to the board and ample knowledge on some of the background information i needed to get aclimatized to the working environment. And this is a brief note of some of the expectations vs reality i found.

T-Rudolf1: Testing Display	
Tested by:	Rudolf
Purpose:	Familiarize myself with Atmel Studio and Create a small program
Preconditions	No example Ported from teacher code use the tool from scratch. Lcd.h and lcd.c copied from the teacher.
Test sequence1.1:	Create a project
Description what i learned. IDE	Atmel Studio 6 is a tool with a slow learning curve reusing theacher code was relatively easy but when creating code from scratch it took some time to figure out where the tool saved it's information aout the board and remove some of the preset conditions that came loaded with the atmega32 settings. Converting the default C project to a C++ Project required Closing the project manually editing the Project config file and ammending settings. This seems a little counter intuitive for an IDE. Figuring out how Doxygen worked was another aspect of this since it would prepare us for the semesters to come and uncommented or code commented without some sort of a standard is harder to distribute.
Description what i learned. IDE – UBUNTU / WINE	Installing Atmel Studio 6.1 even in wine is next to impossible (out of the box) and even if it was support for the USB simulation is next to impossible (out of the box). The only practical way of doing it would be to use another Tool such as CODE:BLOCKS and then a JTAG cable which would allow you to upload the hex file using much more supported upload solutions for UBUNTU.
CODE:BLOCKS	Really nice lighweight solution for coding c,c++ and there i came across native support for Doxygen which i then used to find out how to use in Atmel Studio since the rest of my Group was using that.

11.2.3. RFID - Yin

TC1: Testing RFID	
Tested by:	Jiachen
Purpose:	Make a small program to test RFID working
Test sequence:	Test RFID card reader.
Description what i learned. Atmel Studio 6	The one that teacher recommended each one to use for coding in C/C++, in this case I use C. This one is a little bit more complicate than the one in Java thatneeds to create everything, write the methods manually. But that would be a really good one for the beginner to learn something about.
Test Result	The simple test for RFID, which first of all asks the cards' UID after scan the card and then display the UID and the Command & Ack from the CPU on the LCD screen if you scan another card the amount of the cards would be updated on the LCD and if you scan the same card the will show the warning message and the amount will also be updated but except the same one.

11.3. PC part

11.3.1. Unit Tests

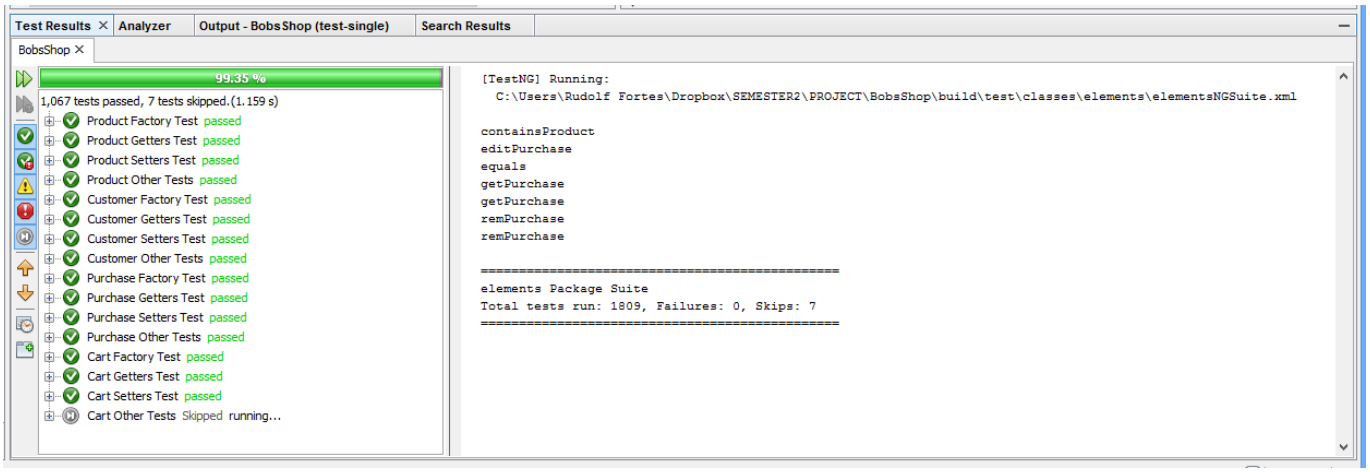
After working on the Unit tests using Junit. We ran into the problem that it was really hard to differentiate the tests by grouping them into sections and funcitonality. Doing some research found that TESTNG has much better support for such features. So we tested our Base elements using this. ”Newer” testing model called TESTNG.

We tested for logic consistancy and not for compiler errors. We did not test for limits of integer more so the fact that the correct results were provieded.

We tested the Elements package as most other packages were derivatives of java beans or provided by the teacher.

To run the test one merely needs to run the XML file in netbeans within the Netbeans code to get a catagorized test for each element. All the tests are grouped into

- Constructors / Factory Create Methods
- Getters
- Setters
- Other



The current setup is that Each Test generates its own test data based on a private variables these variables can then be reused by other Test classes but for future purposes they could be provided by CSV files. The Cart Test has some skipped tests partly to experiment with the possibility of those and because accurate testing was opted out due to time constraints and partially because they heavily rely on Purchase and Customer classes that are thoroughly tested. They are commented with Method not implemented when skipped.

TESTNG has some beneficial features.

12. Conclusion

12.1. Product oriented conclusion

Overall our Product works as a proof of concept and it does what we set out to do. The code is relatively easy to modify. **We have met our Basic goals**.

12.1.1. PRODE2_2

Our Shopping Cart & LCD test is fully commented using the Style of Doxygen and has a extracted documentation on the code which can be found in the REPORT MEDIA.

We managed to get to itteration 3 of our Terminal Design.

- 1st itteration was Pin tests using a breadboard
- 2nd itteration was experimenting with getting a working PCB
- 3rd itteration cut our losses and use a veroboard.

Future itteration would have been create the Terminal in modules using PCB's and have them seperated by ribbon cables where we could design the layout of the Terminal to our exact needs while talking to the Design students to get a functional prototype of the solution.

We have managed to reach itteration 1 in our C code.

- 1st itteration all code is working and proof of concepts are done

Future itterations could include porting the solution to c++ for improved documentation. Doxygen is a lot more geared towards object oriented C++. We would also be able to implement a State machine making our Solution more scalablle to add States and functionality.
We have experimented with the working environment enough to make a better solution next time.

12.1.2. PROPE2_2

We managed to get a working back-end and gui for bob. We have a well documented well illustrated solution. The Gui was designed with seamless functionality in mind the table models helped us achieve this. All of our code was done within the confines of Netbeans meaning that anyone with little Netbeans experiance would be able to find their way around this solution making it Scalable.

Our Main issue was the Development environment. Sharing the code gave the most problems. Eventually after many failed attempts we decided to use a Shared Online Database and a Library Folder located in the Dropbox Shared Folder. All Libraries were added as either native support or Relative to the Project.

Jdeveloper is a very well rounded tool that supports a wide range of Visual Support and gave us a great overview of where our code was and where it could be heading. The main drawback was that Netbeans Development just works better and more seamlesly. So we ended up copying the sourcecode once in a while and porting it into an Open Jdeveloper Project where we would make the diagrams with the copied code. This was time consuming but eventuall worth it.

12.2. Process oriented conclusion

When dealing with a fairly big and complex project, settling on a proper workflow is essential to accomplish the prefixed result within the given time. We first thought about embracing the "Scrum" development approach, but due to the lack of time and insufficient knowledge of the problems that lay ahead Scrum became more of a Burden then an asset.

We realized it would have been hard to keep up with this methodology. It requires too frequent meetings and alot more micro managment.Adopting Scrum will have been too timeconsuming rather then beneficial however a more organized project plan could have led in a more efficent time managing.

In particular when it came to assignment of roles, tasks. Deadlines and goals could more easily be adapted to by shuffeling tasks and switching tactics.

As the microcontroller developing had several aspects we needed to focus on (SPI communication, keypad handling, etc...), having different "projects" being mantained at the same time was the only solution that could allow different members to work individually and independently, without the risk of corrupting the overall result. This led to some difficulties with keeping code consistency among all the projects.

Gathering all the functionalities together was also a slightly more complex task because of this fractionation.Although the product features the possibility of adding extra carts and checkout points (with the only limit being the amount of available serial ports), it lacks a native interface for adding more. But as we were asked to only develop a prototype, scalability has not been one of our major concerns. Other possible improvements might regard data security (encryption, ashing, or hashing and salting for passwords), statistics and monthly sales report generator, and file-stored settings, should Bob decide to migrate his platform.

Much of our time was spent aclimatizing ourselves with the development environment. Both Atmel Studio and Jdeveloper have a really slow learning curve and learning to use these IDE's would make all of our future related projects alot easier.

12.2.1. Group evaluation

We had a hard time making this group work efficiently and there were many factors that played a part in this.

In the beginning we swapped members a few times before we settled on our final group members at that time our focus was on developing the hardware and the base elements of the java and we were meeting frequently in class and discussing the project at length. After we completed these collaborative aspects we tried to divide some of the work into areas.

This division was not done correctly we should have paired into groups of 2 rather then 4 individuals that way we would have a way of assisting each other and trying to work in groups of 3 based on who was in class didn't really work either. The most effective group would have been swapping working in groups of 2. if we had split up the tasks into pairs rather then rather then in individually it would have provided us with a more robust work environment and therefore given us a lot more effective and efficient time management.

Another major hinderence in the efficiency of the group was the diversity of skill in the group it was a balance of people with varying skill levels. Creating a balance of workload vs efficiency was really hard for both sides.

In the end we had to meet the deadline and we trimmed our solution to get the functionality to work and we "scrapped" many really interesting and creative features and design choices.

12.3. Future Projects

Allthough we managed to get a working project for the c programming many of the trimmed features lack scalability and work more as a proof of concept rather then a viable sellable product.

Future implementations should include C++ code rather than c with a statemachine. This was in process but as we reached towards the deadline we opted to scrap it and worked on porting our Integration tests into a functional working solution.

It would be interesting to approach this project as if you were a real company with real billable hours and restrict the project to the hours we would naturally be allocated during the lessons.

For the java it would be nice if we could make Jdeveloper our native Development IDE since the visual representation of our documenting made it alot easier to revisit code and see why things were the way they were.

One of the main benefits of opting for c++ would be that we would be able to utilize many of the design features that Object Oriented programming facilitates. A clear benefit would be improved utilization of Design patterns; why reinvent the wheel when a standard way of implementing it has been tried and tested. Another would be that doxygen has really nice support for Object Oriented code it would simplify the documentation process drastically if each class was made with that style of documentation standard. Testing a class would also be a lot easier since you are dealing with more finite boundries.

Last but not least project management software would be highly beneficial.

- A time tracking software would be highly beneficial such as the linux freeware hamster. This way we could accurately track what time was being spent on.
- We looked into several Scrum project managers and many of them seemed adequate.
- Microsoft Project has some great features but would be more appropriate to use an opensource alternative.
- Online versioning service/server to allow multiple threads of thought so that we could checkout the project and avoid cross programming problems.
- More elaborate use of Doxygen and Javadoc. We have functional readable code but these tools have many more features such as Diagram generation that we didn’t have time to experiment with.

12.4. Literature

Google is your friend. Many code references were lost in the development but some of the major inspirations are listed here .

- Agner Fog
- Henrik Bechmann
- http://homepage.hispeed.ch/peterfleury/group__pfleury__lcd.html
- https://code.google.com/p/phi-prompt-user-interface-library/downloads/detail?name=HD44780LCD.lbr
  Campusnet Ballerup Campus:
  - Front page / Electrical Engineering and Information Technology / EPRODE2-2. sem IT-E13
  - Front page / Electrical Engineering and Information Technology / EPRODE2-2. sem. elektro-E13
  - Front page / Electrical Engineering and Information Technology / EPROPE2-2. sem elektro-E13
  - Front page / Electrical Engineering and Information Technology / EPROPE2-2. sem IT-E13
- http://stackoverflow.com/
- http://www.cplusplus.com/
- http://www.dreamincode.net/forums/topic/160032-finite-state-machines/

13. Appendices

This section will include all of our Appendices and additional documentation.

13.1. About the Appendix

Our Appendix consists of two formats. The DVD / Media Device with our Code working projects and major Images. And the remainder of this document which will include some noteworthy information described by it's chapter.

MEDIA / DVD	
Diagrams	Diagrams And illustrations used to describe our solution.
Hardware Schematics	Kicad and Eagle. Schematics of our Terminal and a simple PCB layout Design.
Libraries	Various JAVA libraries we used in the Java aspect of the Project
SourceCode	The Sourcecode for our project please see "13.7SOURCE CODEON PAGE88"

13.1. Activity list

Group Assignments	
Group coordinator / Documentation Writer	Rudolf
Lead assistant	Yin
Code Monkey	Riccardo
Hardware	Ebbe

Task	Riccardo	Yin	Rudolf	Ebbe	Teacher
PROPE_2					
Database Design	X		X		
Base Elements Design	X	A	X		
DB – connection	L		A		
GUI	A	L			
Cart server	L				
Checkout Server	L				
Java infrastructure	L	A	A		
Frame / Model	L		A		
Serial					X
PRODE_2					
Breadboard testing	X	X	X	X	
Main / Infrastructure	L		A		
Display	L	A	A	A	
Keypad	L	A			
Protocol / Packet	L				
PCB design and schematic			L	A	
LCD					X
Serial					X
RFID research / timings				L	
Duplicate Prototype Build				X	
Documentation					
Jdeveloper		A	L		
Sequence diagrams		L			
Activity diagrams		L			
Flow Chart c Code	L		A		
Class Diagrams			L		
Doxygen	A		L		
Javadoc		A	L		
Writing	A	A	L		
Hardware Block diagrams				L	
Software Block diagram	L				
TestIng					
Acceptance Test	X	X	X		
JavaNG		A	L		
C Testing Cart & Checkout	L				
C Testing Full integration	L				
C Testing Display / Finite State Machine			L		
C Testing RFID	A	L			
C Testing Keypad	A	L			
Legend					
X	Load Shared equally				
L	Lead				
A	Assitant				

13.2. Diagram Appendix

13.3. Self-service shop project

Bob's hardware store needs modernization. There is always a long queue at the cash register. Bob wants a self service system where customers can buy things without standing in line to pay. You are going to develop such a system for Bob.

Bob is imagining that a self service system can work like this. There is a scanner on each shopping cart. The customer scans their customer card and enters a pin code. Each product in the shop has an ID chip attached. When the customer wants to buy a product, they scan the chip on the product and puts the product in the shopping cart. The products are automatically billed on the customer's account. There is no cash register and it is not possible to pay with cash. A scanner at the exit door scans everything that passes through and beeps if a product has not been paid for.

You have to construct a working model of a self service system for Bob's hardware store. The system should have a central database that contains information about prices and the

customer accounts. The central database should have a user interface that allows the administrator to manage customer accounts, change prices and get sales statistics.

The scanner should be built with an RFID card reader, a microcontroller board, and a Veroboard with LCD display, keypad and possibly other components at your choice. You can use RFID cards to simulate the ID chips in the products. It is possible to store information in the RFID cards.

The card reader terminal is connected to a PC through a USB cable. The wireless connection is only imagined. The central database and administration interface is implemented in Java on the PC in connection with an SQL database.

The project is made in groups of 2 - 4 students. The project work should include the following tasks:

• Design yourself service system. For example, you may consider these questions:

1. Should the customers pay in advance, or can they get credit?
2. Should identical products have the same ID number, or should each item have a unique ID?
3. Should the chip on a product remember that the product has been sold, or is this information stored in the database?
4. What should a customer do if they change their mind and wants to put a product back on the shelf before leaving the shop?
5. Should the system give a message if a customer has paid for a product by mistake and not taken it out of the shop?
6. Should the system tell the manager when a product is sold out?
7. What information needs to be transmitted between the terminal on the shopping cart and the database?
8. What information should be shown on the display on the scanner?
9. Any other features you think would be useful?

• Make a drawing and description of your system and make a requirements specification. The requirements specification must be approved by your supervisors.
• Make a project plan or time schedule based on a risk analysis so that the most critical risks are eliminated first. In other words, make the most important things work first, and do the features that are just "nice to have" later.
• Make regular group meetings where you decide who is responsible for each task.
• Build everything.
• Test everything.
• Write a report that documents all of these points.

13.4. Glossary

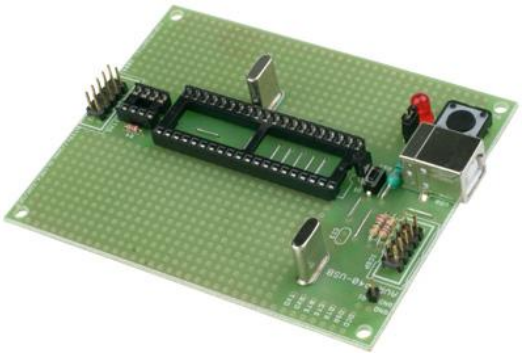
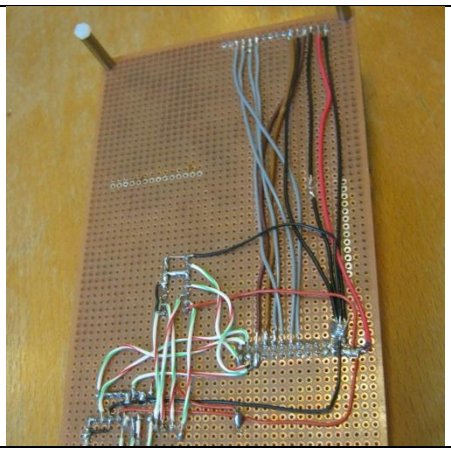
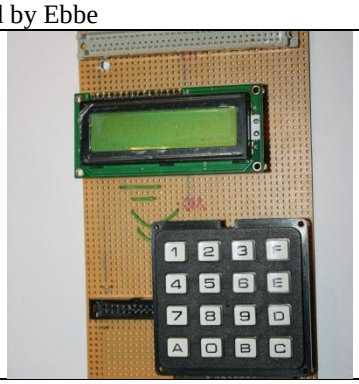
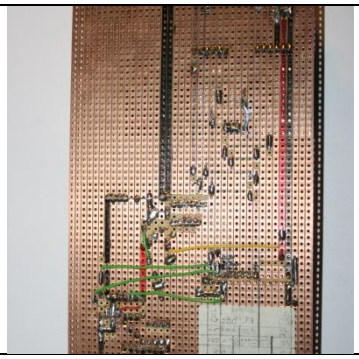
This section was meant to include a series of concepts and phrases and their definitions so that the reader would know what specifically we were referring to due to time contraints we have included only some of the basic conceptual conundrums.

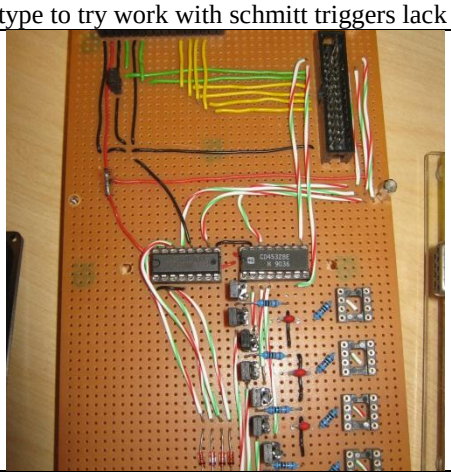
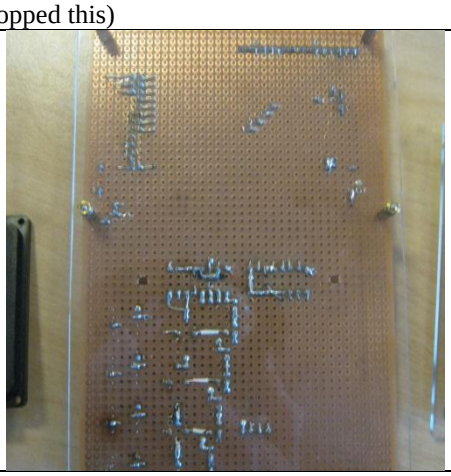
- User and Customer refer to the customer of the shop or the user of the shopping cart.
- The Admin may be reffered to as bob.
- The Virtual Cart represents what the Customer is shopping in the Shopping cart.
- The Actual Cart represents what is scanned in the cart at the checkout point.
- Terminal sometimes refers to the compilation of hardware we used that simulate the Shopping cart or Checkout point.
- The Shopping Cart or Cart refers to the devices which are utilized for the Customer to perform shopping (AVR, RFID, keypad, Display)

13.5. Source code author list

- Agner Fog
- Henrik Bechmann
- Our Sourcecode should be commented with authors listed in Javadoc / Doxygen format.

13.6. Hardware

AVR-Kit	RFID reader / writer
	
Interface prototype 1	
Created by the whole group	
	
Interface prototype 2	
Created by Ebbe	
	

Interface prototype 3	
Created by Rudolf (prototype to try work with schmitt triggers lack of time dropped this)	
	

13.7. Source code

The Source code will be Provided on a DVD or other Media device along with the report. It should be located under a Folder called SourceCode.

Folder Name	Language	Description
CART	C	Main Shopping Cart
doxygen-Cart	HTML / web	The Doxygen information retrieved from our Doxygen commented CART. ”Access the index.html page to view”
CHECKOUT	C	Main Shopping CheckoutTerminal
doxygen-Checkout	HTML / web	The Doxygen information retrieved from our Doxygen commented CHECKOUT. ”Access the index.html page to view”
BobsShop	Java	Netbeans Development Folder
BobsShop-JDEV	Java	Jdeveloper Code ported from netbeans for visual Documentation and Illustrations.
TESTS	C / C++	A series of Tests documented within this the Report
TESTS\CODEBLOCKS	C / C++	A series of Folder attempts of coordinating working between a linux and a windows environment with little notable outcome other then experiance.
Atmel_OLD_CODE	C / C++	A series of Old test / abandoned projects and experiments that have been updated since or simple dated backups..

13.8. Test results

13.8.1. Acceptance Test

Does the solution meet the SRS requirements we set out.

TC1: Testing R2.1 R3.1 R3.2 R3.5 partial R3.6	
Tested by:	Yin, Riccardo, Rudolf
Purpose:	Test the following requirements R2.1 R3.1 R3.2 R3.5 (R3.6 display messages for user feedback)
Preconditions	Plug in and start the cart Java software running
Test sequence1.1:	swipe the customer card
Description of the expected result:	Username shown if card is valid.
Test sequence1.2:	Swipe invalid customer card
Description of the expected result:	Card not valid splash screen displayed return user to start screen
Test sequence1.3:	Swipe blocked customer card
	Brief splash screen to enter pincode Card blocked splash screen return user to start screen
Preconditions	Valid customer card
Test sequence2.1:	Enter valid Pin
expected result:	Balance shown enter shopping mode
Test sequence2.2:	Enter Invalid Pin < 3
expected result:	Invalid pincode splash screen return user to start screen
Test Sequence 2.3	Enter Invalid Pin 3
	Inalid Pincode splash screen card blocked splash screen return user to start screen
Testers comments:	On hardware reset a blocked customer can be logged in. Bug 1 fixed now card marked as invalid. Splash screen invalid customer card. Without hardware reset a valid customer cannot log in after a customer has been blocked. Short splash screen to accept user Pin
R2.1	Working as designed - Bugs were fixed in first iteration of Test
R3.1	Working as designed
R3.2	Working as designed
R3.5	This was tested by debugging the java to ensure data was sent and it Worked as designed
R3.6	Working as designed
TEST ACCEPTED	

TC2: Testing R2.2 R2.3 R2.4R3.3 R3.4 R3.5 R3.6	
Tested by:	Yin, Riccardo, Rudolf
Purpose:	Test the following requirements R2.2 R2.3 R2.4R3.3 R3.4R3.5 (R3.6 display messages for user feedback)
Preconditions	Plug in the cart Java software running Customer logged in Customer has sufficient funds
Test sequence1.1:	Swipe valid Product 1st wait for timeout
Description of the expected result:	User is provided with an option to update the quantity rather then the default 1.
Test sequence1.2:	Swipe valid Product Nth wait for timeout
Description of the expected result:	User is provide with an option to update the quantity rather then the default +1
Test sequence1.3:	Swipe valid Product with update quantity User is provide with an option to update the quantity. User provides a quantity between 0 and 99
expected result 0:	Splash screen Product removed return the user to the main shopping screen
expected result 4:	the product information on the screen is updated with the quantity requested and then return the user to the main shopping screen
Preconditions	Plug in the cart Java software running Customer logged in Customer has insufficient funds
Test sequence2.1:	Swipe valid Product 1st wait for timeout
Description of the expected result:	Splash screen Insufficient funds. Product not added
Test sequence2.2:	Swipe valid Product Nth wait for timeout
Description of the expected result:	Splash screen Insufficient funds. Product quantity not updated
Test sequence2.3:	Swipe valid Product with update quantity User is provide with an option to update the quantity. User provides a quantity between 0 and 99
expected result 0:	Splash screen Product removed return the user to the main shopping screen
expected result 4:	the product information on the screen is updated with the quantity requested “4” and then return the user to the main shopping screen
Testers comments:	Once the balance goes below a Unit price for an product the customer may not amend the unit quantity. Remove product triggered some bugs concerning the balance count. Balance displayed Fixed.
R2.2	Acceptable Minor Bug (on hardware reset customer can login: Final Prototype only Bob should be able to reset the Cart.)
R2.3	Working As Designed
R2.4	Working after a bug was fixed. (Bug the Display was not updated when Product was removed.)
R3.3	Working as designed
R3.5	This was tested by debugging the java to ensure data was sent and it Worked as designed
R3.6	Working as designed
TEST ACCEPTED	

TC3: Testing R2.5	
Tested by:	Yin, Riccardo, Rudolf
Purpose:	Test Requirements R2.5
Preconditions	Plug in the cart Java software running Customer logged in Customer has sufficient funds
Test sequence1.1:	no shopping has been done
Description of the expected result:	Checkout ignored
Test sequence1.2:	Empty Shopping cart
Description of the expected result:	Checkout performed Cart in java is with an empty product list
Test sequence1.3:	Finalize with product
expected result:	Java has a cart with a valid product list
Testers comments:	This was harder to test on the Terminal. We had to check the Checkout Status by checking on the Carts at the Java end. (Some minor bugs-with the communication protocol were fixed)
R2.5	Working As Designed
TEST ACCEPTED	

Requirement R2.6 not tested because it was not implemented

TC4: Testing R4.1, R4.2 R4.3 R4.4	
Tested by:	Yin, Riccardo, Rudolf
Purpose:	Test Requirements 4.1
Preconditions	Plug in the cart Java software running Customer has ”Shopped”
Test sequence 1.1	Scan Products several products
Description of the expected result:	Products were scanned and added to the virtual cart.
Test sequence 1.2	Checkout Finalization was tested using empty carts different carts
Description of the expected result:	When carts were identical they proceeded to security pin and were greeted to exit the shop. When carts were different the customer was asked if they would like to exit the shop anyway. The actual cart is what they were billed for. We had to check this by looking at the database and Bob’s Gui but it worked.
Test sequence 1.3	Same as Test Sequence 1.2 but we looked at the Logs of Bob’s Gui in the runtime and compared the Database
R4.1	Working As Designed
R4.2	Working As Designed
R4.3	Working As Designed
R4.4	No Physical Barrier Desiged but Customer is bid farewell or not allowed to finalize shopping.
R4.5	Work around Display And keypad on checkout Terminal
R4.6	Work around Display And keypad on checkout Terminal
TEST ACCEPTED	

Requirement R4.5 & R4.6 - not tested because they were not implemented as initially specified. We created a Checkout point with the keypad and the display.

TC5: Testing R1.1	
Tested by:	Yin
Purpose:	R1.1 allow Bob to create the customer profile
Preconditions	Java software running
Test sequence1.1:	Click the button" Add new" in Customer Panel or in Toolbar or in the meanubar/File/New Customer
Description of the expected result:	JPanel "Add Customer" shown to allow Bob to create a new customer profile
Test sequence1.2:	Fulfill with the customer's detail
Description of the expected result:	Typing {"Full name","Address","Pin number"}
Test sequence1.3:	Click the button"Add new customer" in AddCustomer Panel
	The customer's profile created and shown on the customer table which is in the Customer Panel.
Testers comments:	Everything works fine

TC6: Testing R1.2 R1.7	
Tested by:	Yin
Purpose:	R1.2 Bob shall be able to modify the Customer profiles. R1.7Bob shall be able to unlock an account for a customer profile.
Preconditions	Java software running Customer profile created
Test sequence1.1:	Double-click the specific row in Customer Panel
Description of the expected result:	JPanel "Edit Customer" shown to allow Bob to modify a customer profile
Test sequence1.2:	Modify the customers' detail
Description of the expected result:	Changing the value of {"Full name","Address","Pin number"} also lock on unlock a user.
Test sequence1.3:	Click the button" Commit" in EditCustomer Panel
Description of the expected result:	The customer's profile modified and updated on GUI.
Testers comments:	Everything works fine

TC7: Testing R1.3	
Tested by:	Yin
Purpose:	R1.3 Bob shall be able to add Products.
Preconditions	Java software running
Test sequence1.1:	Click the button" Add new" in Product Panel or in Toolbar or in the meanubar/File/New Product
Description of the expected result:	JPanel "Add Product" shown to allow Bob to add new products.
Test sequence1.2:	Fulfill with the product's detail
Description of the expected result:	Typing {"Product name","Unit.cost","Unit.price","Threshold"}
Test sequence1.3:	Click the button"Add new product" in AddProduct Panel
expected result:	The products created and shown on the product table which is in the Product Panel.
Testers comments:	Everything works fine

TC8: Testing R1.4	
Tested by:	Yin
Purpose:	R1.4 Bob shall be able to modify products
Preconditions	Java software running Products created
Test sequence1.1:	Double-click the specific row in Product Panel
Description of the expected result:	JPanel "Edit Product" shown to allow Bob to modify the product's info.
Test sequence1.2:	Modify the products' detail
Description of the expected result:	Changing the value of {"Product name","Unit.cost","Unit.price","Threshold"}
Test sequence1.3:	Click the button" Commit" in EditProduct Panel
expected result:	The products modified and updated on GUI.
Testers comments:	Everything works fine

TC9: Testing R1.5	
Tested by:	Yin
Purpose:	R1.5 Bob shall be able to restock products
Preconditions	Java software running Products created
Test sequence1.1:	click the specific row in Product Panel and click the button "Add in stock"or click the row and then click the meanubar/Edit/Restock
Description of the expected result:	JDialog shown to allow Bob to restock the product.
Test sequence1.2:	Restock the product
Description of the expected result:	Type the amount need to be restock by the product .
Test sequence1.3:	Click the button" OK"
expected result:	The products quantity modified and updated on GUI.
Testers comments:	Everything works fine

TC10: Testing R1.6	
Tested by:	Yin, Riccardo, Rudolf
Purpose:	R1.6 Bob shall be able to see statistics about sales and stock current state, pending orders and previous transactions
Preconditions	Java software running Products created
Test sequence1.1:	click the specific row in Product Panel
Description of the expected result:	Diagram of the history of the product shown below the product table.
Testers comments:	Everything works fine

TC11: Testing R1.8	
Tested by:	Yin
Purpose:	R1.8 Bob shall be able to amend the customers balance.
Preconditions	Java software running Customer profile created
Test sequence1.1:	click the specific row in Customer Panel and click the button"Top-up" or click the row and then click the menubar/Edit/Top-up
Description of the expected result:	JDialog shown to allow Bob to amend the customers' balance.
Test sequence1.2:	Amend the customers' balance
Description of the expected result:	Type the amount need to be increased by the customer balance.
Test sequence1.3:	Click the button" OK"
Description of the expected result:	The customers' current balance modified and updated on GUI.
Testers comments:	Everything works fine

13.9. Tests – Microcontroller [Ebbe]

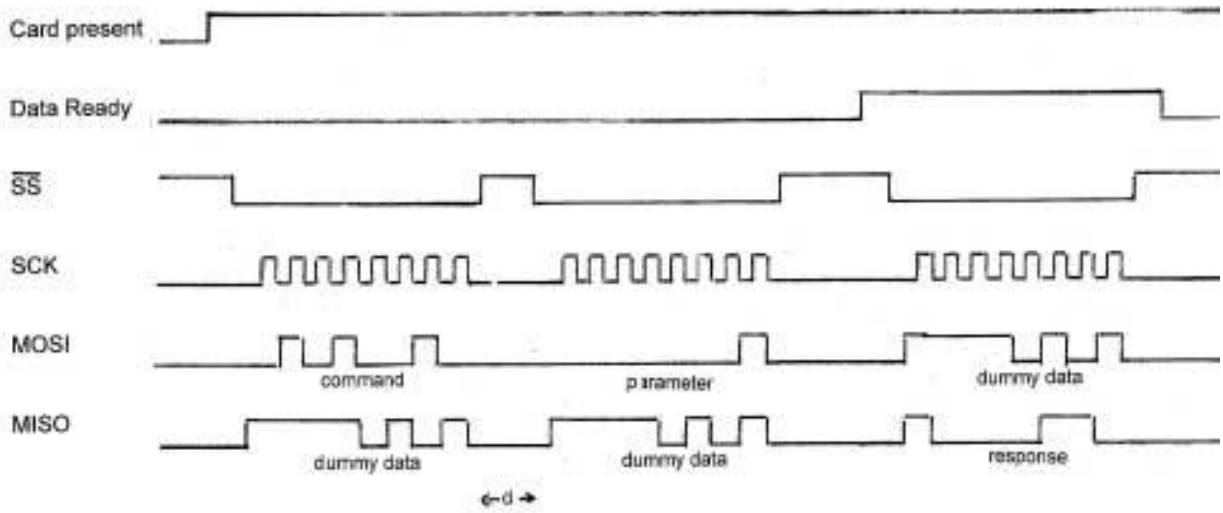
We found it difficult to realize a test for the hardware in the usual manner since the system under test is also our looking glass, the data looses credibility (unit test) and alternatively to create an entire system to test the system seemed wastefull of time.

13.9.1. Tests - RFID

We test the RFID reader for the essential function we require to meet our Overall Scope for the CART device:
When a card is detected, read the UID from the card to the microprocessor via SPI and send it to the PC for processing.

We encountered problems making the (reader ⇔ SPI) comply and send, so we tested on a lower level to make it work.

Therefore these tests first analyze the stages in the process of getting the UID from the card via SPI transfer to the Microprocessor outlined in the datasheet for the RFID card reader:

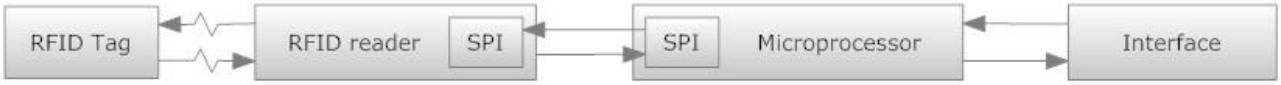


SPI sequence diagram from the RFID reader datasheet

13.9.2. Test Descriptions

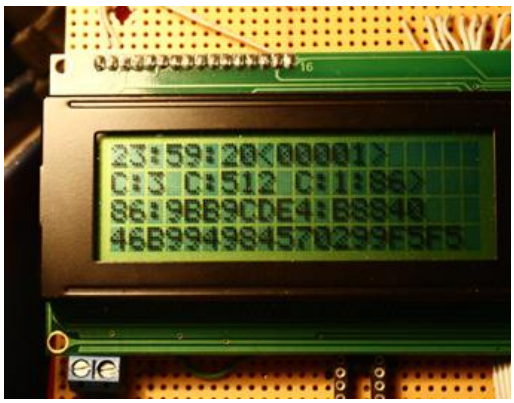
Scope :	Retrieve UID from Card	Show results on LCD display for verification.
Stage1	A valid card is detected	
Stage2	The SPI buffer has been written a valid command	
Stage3	Data is ready to be transferred	
All stages	When a card is detected, get the UID from the card via the reader.	

13.9.3. System Under Test

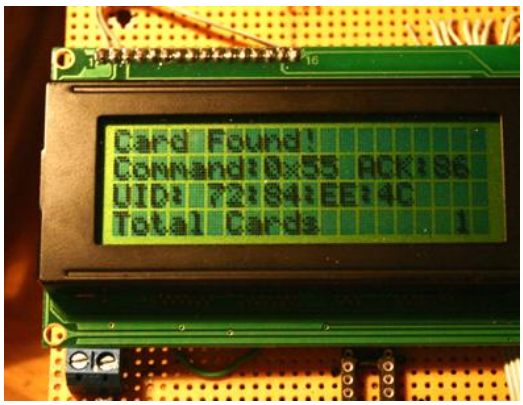


Event	Expected	Specific	Observed on display
Stage1	Send an interrupt to the Microprocessor	Set INT0: high	The variable Cvar1 increments
Stage2	Acknowledge commands via SPI:	Send: 0x86	First reply contains 0x86
Stage3	Signal when a command has been processed	Set INT1: high	The variable Cvar2 increments
All stages	Process the command "**Card UID"**	ACK + 4 bytes(UID) + 3 Dummy Bytes	We receive 8 bytes, this is repeatable

13.9.4. Display readout



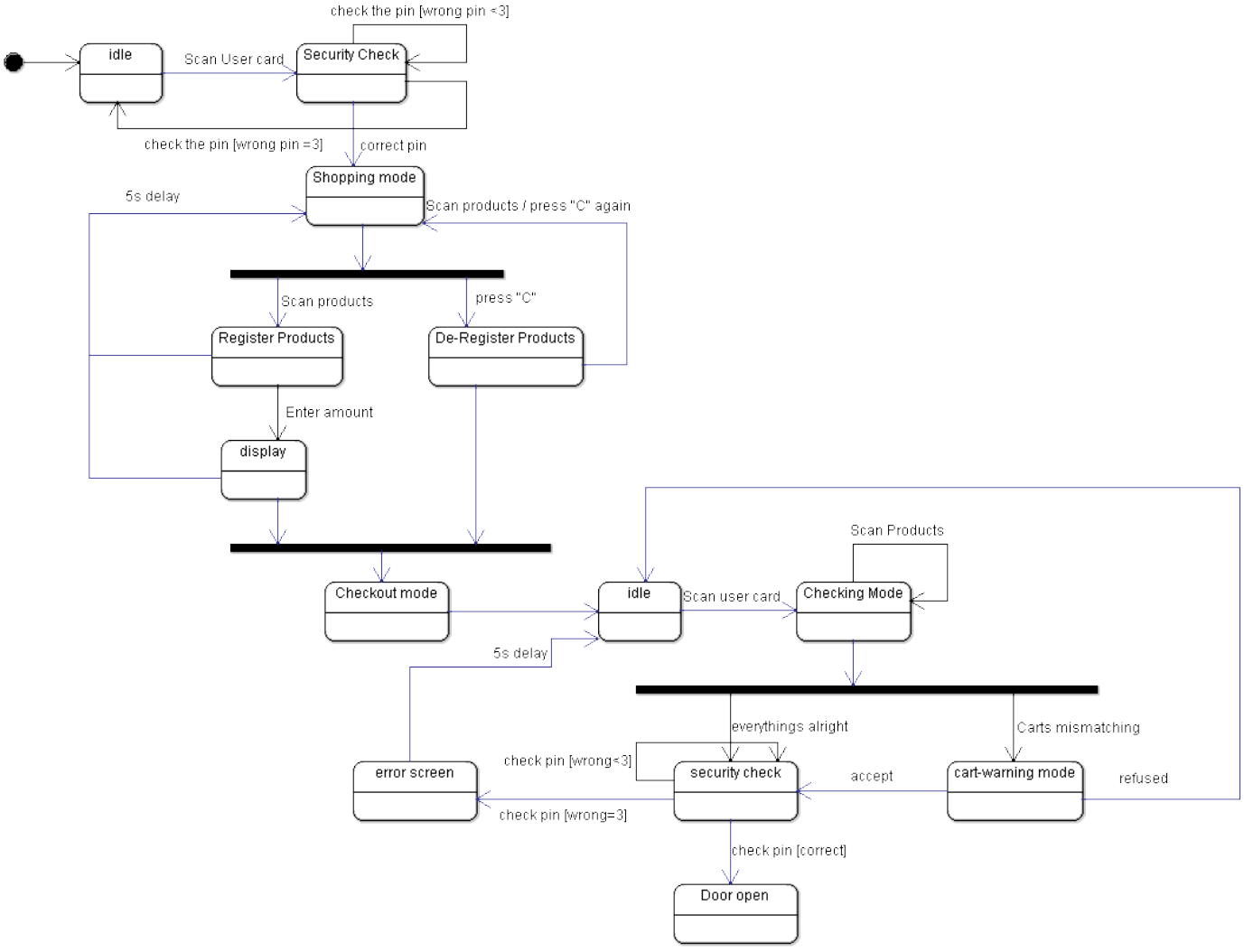
Testing Showing Cvars



Later test with command visible

13.10. C code brainstorm initial Design

13:1"Emulated" State Brain Storm



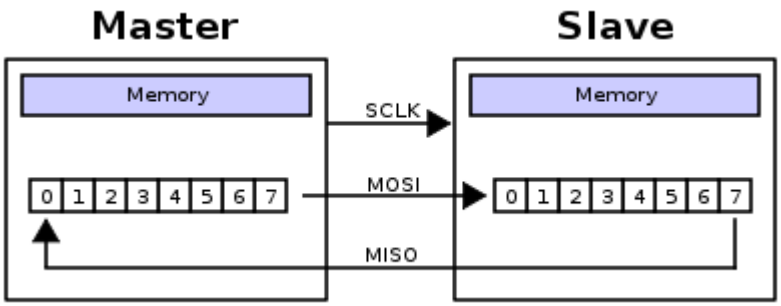
14. Reference Material [Ebbe]

14.1. SPI

Simple Peripheral Interface.

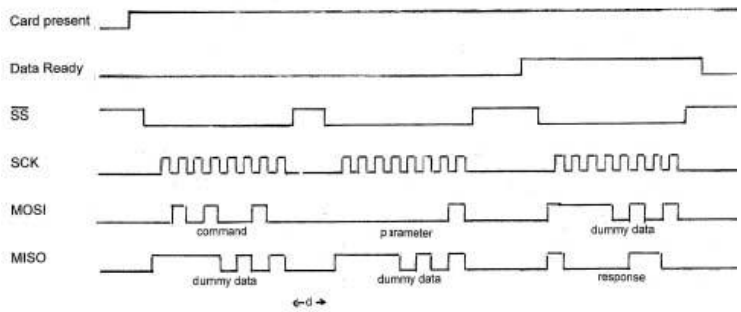
A serial protocol used for simple communication between digital circuits.

Data transfer works by exchanging 8bits of data in a ring buffer system using a common clock frequency



databuffer swapping between SPI units

The card reader is equipped with an ATmega8 microcontroller, set as SPI slave that handles all direct communication with the RWD module and forwards any data from the card to the SPI system.



timing diagram of the SPI transfer

it also has 2 extra control wires:

"Card Present" goes high when a card is successfully detected and is used as interrupt signal for the firmware.

"Data Ready" goes high when a command from the main microcontroller has been received, parsed to the RWD module, understood and return data is ready for transmission

Controlled by one master device that enable the slaves and sets the clock frequency.

In our case we only have one master(AVR kit) and one slave(RFID reader)

All data received from the Micro-RWD are passed on through SPI.

The clock rate is recommended to be 625 kHz (this is set on the master device)

Most significant bit of each byte comes first

sets the Data ready signal low when ready to receive commands

Sets the Data ready signal high when it is ready to send a response back.

The delay d between byte transfers must be between 2 μs and 5 ms.

14.1.1. Relevant SPI registers

Register	Name	Set	Function
SPSR	**SPI Status register**	0	Contains flag <SPIF> that indicate if a is swap is done.
SPDR	**SPI Data Register**	1	8 bits where the data is written to initiate data transfer, also where data is read when the swap is complete.
SPCR	**SPI Control Register**	0	defines the charateristics of the SPI system.
4	MSTR	1	Set as master
3	ClkPol	0	Low when idle
2	ClkPhase	0	Sample/setup
1	SPR1	0	Do not divide F_osc by 4
0	SPR0	1	Divide F_osc(10mhz) by 16 = 625khz

14.1.2. SPCR settings:

SPI master (AVR Kit)

bit	Name	Set	Function
7	SPIE	0	Interrupt not enabled
6	SPE	1	Enable
5	DORD	0	MSB first
4	MSTR	1	Set as master
3	ClkPol	0	Low when idle
2	ClkPhase	0	Sample/setup
1	SPR1	0	Do not divide F_osc by 4
0	SPR0	1	Divide F_osc(10mhz) by 16 = 625khz

Our code to use the SPI_do()

```
unsigned char SPI_do(char cData)
{
        PORTB &= ~(0x04);
        SPDR = cData;
        while(!(SPSR & (1<<SPIF))) {};
        PORTB &= ~(0x04);
        return SPDR;
}
```

15. Cart - prototype device [Ebbe]

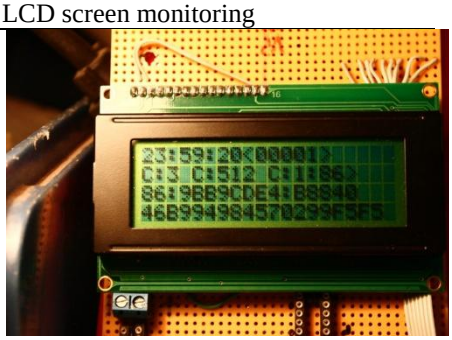
During the project, we created several interface boards to work with the firmware.
Being just boards and components, they would often break and subsequently consume our time tracking down errors.
This meant we created a couple of bughunting tootls for the boards we made… one of these was a 3 variable system of simple "++var1;" placed troughout the code, that to a certain degree tracked the state machine and would tell us via updating the LCD display if certain steps where missing or certain parts looped unexpectedly.

it was very usefull while getting the timings right for the RFID/SPI communication

Early testcode

```
if (CommandSent)
{
while (!SPI_dataready)
{
_delay_ms(100);
++Cvar2;
}
if (SPI_dataready)
{
while (i < 19)
{
CardData[i] = SPI_do(0xF5);
_delay_ms(1);
++i;
}
SPI_dataready = 0;
i = 0;
CommandSent = 0;
SPI_data_recieved = 1;
++Cvar3;
}
}
```

LCD screen monitoring



the trackervars outputs to second line
C:3 C512 C:1:86

This 3 var system, later became a usability focused actionLED system and is 3 LEDs on the front panel that indicates the users next expected move.

1: Scan ID card
2: Scan a product
3: Input via keypad.

We believe this helps new users and in general to reduce confusion using the device.

Controlling the LEDs was implemented using another encoder 4028, since we didi not know how much information we would convey with this system.

So currently 2 wires are used to control the LEDs.

Adding this to the system is very simple since we operate with ‘modes’ already… IDLE,LOGIN,SHOPPING

Swipe ID is for IDLE before login and At the checkout terminal

Scan product is on when in shopping mode.

Use Keypad is on when ever a function requires input.
PIN number, quantity input, add and erase.