Smart Super Market

Project Completion Report

The Smart supermarket is a project where the customer's mobile phone pairs with a supermarket trolley and sends the shopping list to the trolley. The smart trolley displays customized special offers and the availability of the items in the shop automatically. The customer picks up the items. The trolley keeps track of the barcodes and allows automatic checkout by sending the barcodes of the item to the Till. The Till processes the customer's bill by taking into account any discount or promotion. This will make shopping faster and easier.

Grace Ramamoorthy

Student ID: 10278389

12/1/2010

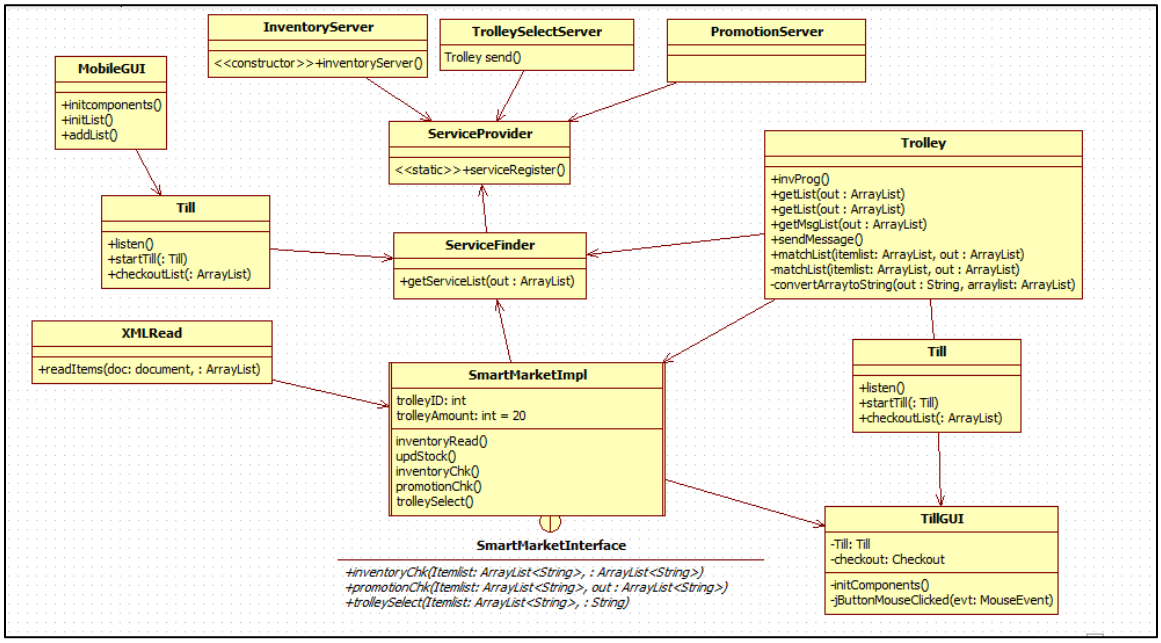
Smart Super Market

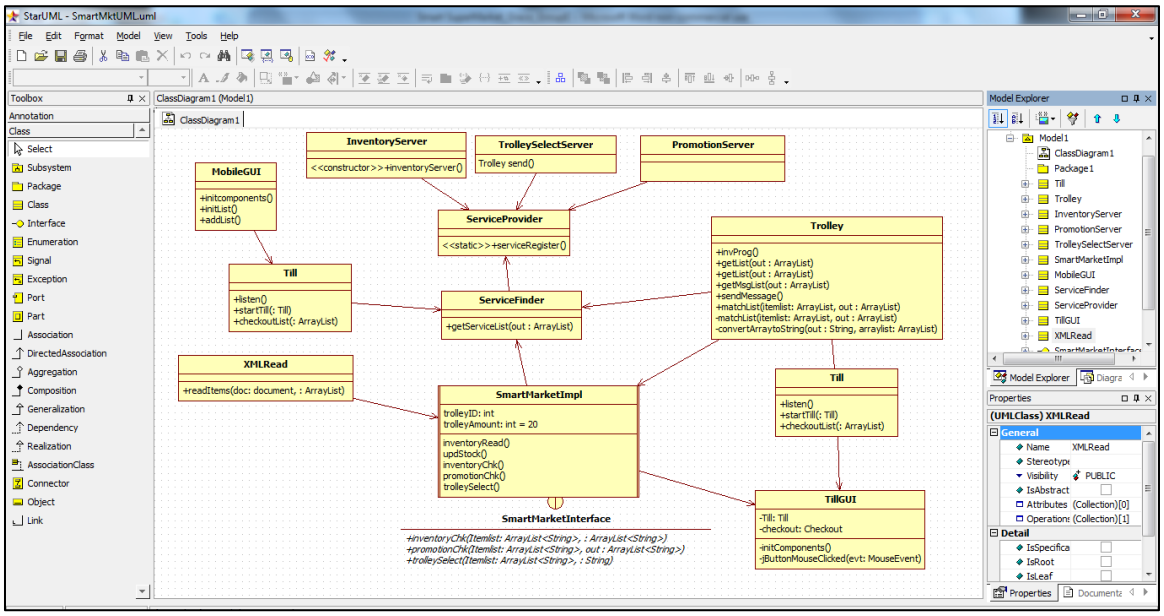
Project Completion Report

DESCRIPTION:

a. Customer walks into the smart supermarket with his shopping list on his mobile phone. As he enters the shop and gets a trolley for his shopping, he presses the **Send** button on his phone and it synchronises with the smart trolley and passes the shopping list to the trolley.
b. The trolley automatically sends the list to the Inventory to check for availability of items and sends back stock status to the trolley.
c. The list is also sent to the Special Offers database for any special offers available. A customized offer list will be displayed on the trolley screen. Any discount will be applied to the price.
d. As the customer adds items to the trolley, the inventory is updated. When the customer completes shopping, he clicks the **Final Send** button to send the final list to the Till. We decided to use the **Final Send** button to complete the shopping. This will allow the shopper to add any unlisted item that the shopper might pick up, or remove any listed item that the shopper might skip.
e. The Till then calculates the final payable amount by taking into account the discount applicable for the customer.

Mobile phone pairs with supermarket trolley to send the shopping list. The trolley displays customized special offers, availability of the items in the shop and sends list to Till. Till processes the customer's bill by taking into account any discount.





TECHNOLOGY:

a. We have used zero conf to locate services. We used a service provider to register trolley, inventory and special offer services. The mobile uses a service finder class to find a trolley and passes the shopping list (item names). The smart trolley uses RMI to pass the list to the smart market inventory service. The trolley uses the service finder to find the inventory service. This service also contacts the inventory server

using RMI. We used RMI here because the inventory server could be placed in a different address space and / or geographical location than the supermarket. The inventory details are stored in xml format and these could be derived from a separate inventory system. We decided to use xml to access inventory details so that the format would be flexible. As of now we are using four attributes – item id, item name, item price, and item quantity. *In future we could add more attributes, such as manufacturer's name, comparative nutrition status etc., that will make the shopping more comfortable*. XML will allow us to do that without much of a fuss. The data between the mobile, trolley, and the inventory service are passed as an array string. As of now, we just send the product name from the mobile to the trolley. However, *the next step is to add the quantity of items to purchase in the itemlist.* For example, a shopper might need to buy 2 cans of milk and 3 loaves of bread. An array String will allow such a change without much of code alteration.

b. The data between the inventory and the trolley is also an array string and it carries the item name, price, quantity available, and a status flag. The status flag would mention if an item is 'available', 'out of stock', or 'limited stock'. As of now we have set a flat threshold limit of 10 to flag an item as limited quantity. *In future, we could have a variable threshold limit for items based on their movement pattern*. Flagging an item as limited quantity if an item is low on stock protects us in the following scenario: there is a possibility that the shopper may not find the item on the shelf if other 'n' number of random shoppers (whose list may not have such an item. For example, a shopper may pick up an item that is not part of the shopping list) have picked up the item earlier. We also considered the scenario where a shopper has an item on the list but for some reason decides not to buy the item. In such a scenario, we realised that the item may not be taken back to the right aisle immediately and hence can remain 'out of stock' till the next refresh of the inventory server data.

c. The arraylist sent from the inventory serves as the base for communication between the trolley and the Till. However, we decided to allow some flexibility in shopping so that the shopper could add a few unlisted items or substitute some item that is 'out of stock' before finalizing the Final Send. In real life the shopper will scan the barcode of the item on the trolley reader.

d. The shopping list is also sent to the promotion server. We used the same RMI technology and the service finder to pass the list and check for any customized promotion that could be applicable to the specific shopper. We used the same xml file as the inventory to hold the sale status column and the new price. The sale column is updated based on the inventory stock. If the stock level is very high (for example is there is over 25 pieces of any item), the item sale status is set to 'Y'. In future, we could improve this to stream videos of promotion news or other offers. Based on the shopping list and the sales status of the items, the price of the item is adjusted.
e. We distributed the work equally component wise. Brain stormed technology issues and design strategies on Wednesdays and Fridays and worked on the code individually over the week. We synchronized code on Fridays after the class and tested them at every stage.

FAULT TOLERANCE:

The scenario of the customer pressing the send button twice while passing the shopping list to the mobile was considered and handled in the trolley. As soon as the shopping list is sent to the trolley, when the customer presses the send button again, a message is displayed that the list is already sent. If the trolley senses the same item second time in the list, it flashes a repeat item message and ignores it.

The link between the trolley and the mobile or trolley and inventory or trolley and till are based on requirement and so no connection was considered persistent.

The scenario of the inventory being unavailable when the customer comes in for shopping is still possible and in that case, the shopper will not be able to get the availability of items. Even then, he can shop and when done can press the 'Final Send' button to check his prices and to complete his payment. The assumption is that the inventory connection will not be down for more than this period.

WORK SCHEDULE:

We more or less divided the work based on components but were completely involved in the entire project. We collectively decided the technologies and the direction. Trolley was a central part and all of us had some contribution

in its creation and therefore will be very different in its design. For example the part of the program where the trolley identifies the mobile was written by Dawei, the trolley identifying the till by Rohan and trolley communicating with the inventory server by Grace and the communication between trolley and Promotion server by Patrick.

Personal Contribution:

I was assigned the trolley to inventory server program unit. This involved creating the entry for inventory in the service provider and service finder classes. I also created the SmartMarketInterface and Inventory_Server class. Creating an inv_Prog method in the Trolley class to locate the inventory service and to get the list from the trolley, to verify the list for duplicate items and then to send it to inventory check. For the inventory part, I created an xml file for the items: I decided on four tags – item id, item name, quantity in inventory and price. I created the XMLRead class and a readItems method to read the xml file and retrieve the Quantity and Price of each shopping item. I used the DOM parser to read parse the XML document. The question of using JAXB over DOM was considered. But, in our project since we have an XML document which is a byproduct from a 'big inventory' system we settled in for DOM instead of JAXB. We assume that we get a flat XML data file from the "Big Inventory System" –and hence it would be better to use the DOM Parser as any change to the .xsd file may not be replicated to our part of the super market sales system.

In the SmartMarketImpl class, I created an inventoryRead and inventoryChk methods to check the availability status of the items. This data is compiled as an arraylist and sent to the trolley to be displayed. The trolley GUI picks up this array list and displays on the screen. The arraylist is then used to update the inventory by reducing the item available by one for each of the shopping item. This is done separately so that the list could be recompiled, if needed. In this project, the xml update is done on a separate file to facilitate testing. So I created a ItemList.xml file for reading and ItemList1.xml to update so that we don't reach 0 for any item due to frequent testing.

It was an interesting project with unlimited scope but with limited time, we had to restrict the features. So we didn't delve deep into adding items in the trolley or barcode reading etc.,

The major cause of concern was integration of all the bits and pieces. The Trolley class and SmartMarketImpl class was written and coded by all of us and required careful integration all the time. Again to decide the format of outputs at each stage so that each one can work on their programs required lots of discussion and agreement. For example, the output from Dawei's trolley was the first input for my inventory. The output of my inventory was the input for Rohan. So we needed to touchbase more often for keep each other informed of the current situation. We created a common directory in the ConnectFiles and used that to upload our latest code from time to time. All of us had access to that shared folder and could at any time retrieve the latest code for any of the components. I created the UML file for the project soon after the prototype presentation and this was used to base our discussions and monitor progress.

UPGRADES:

If we had more time, we could have added items based threshold limit for inventory; more attributes in the inventory table; and added number of items to be purchased in the shopping list.

REFERENCES:

- *Intelligent Cokes and Diapers: Paper by Panos Kourouthanassis, Diomidis Spinellis and George Roussos.*
- *Comparing DOM and JAXB: http://mukulgandhi.blogspot.com/2009/02/dom-vs-jaxb.html*
- *http://www.javabeginner.com/java-swing/java-swing-tutorial*
- *http://cnx.org/content/m15092/latest/*

RUN THE PROGRAM:

1. To start the program, load all the source files in src directory, and compiled files in bin directory. Then copy the xml files outside the src directory.
2. Start rmiregistry.
3. Start ServiceProvider. Wait for the services to start and the Till GUI to pop up.
4. Start the MobileGUI to start shopping.
5. Press Send button on Mobile to send the list to the trolley.
6. Wait for Trolley GUI to pop up and the process to complete. Press CheckItems in Trolley to see the status of items in inventory.
7. Press Send button in Trolley to complete shopping and checkout.

8. Press Checkout in Till to refresh Till and start again.