Smart Refrigerator Report

April 29, 2012

This project will develop a prototype "Smart Refrigerator" system, which will monitor grocery items purchased by the user in order to reduce food waste and facilitate efficient shopping habits.

Steven Strapp - ses6498@rit.edu

Ben Reeves - bpr5171@rit.edu

Dustin Stroup - dxs2857@rit.edu

Contents

1 Overview

1.1 Needs Statement

The New York Times reports that an average American family of four will account for over 120 pounds of food waste per month and that 27% percent of all food available will be lost to waste [1]. In addition, other resources are lost due to inefficient shopping practices; forgetting common items or special trips made for recipe ingredients waste time and fuel. A system is required for shoppers both to ensure their purchases are used before expiration and to assist in planning of grocery shopping trips.

1.2 Objective Statement

The objective of this project is to design a prototype that will allow a user to track food items in order to reduce waste and improve shopping efficiency. The system will remind the user about items nearing their expiration date and track the frequency of purchased items. From this frequency calculation the system will suggest typical shopping lists. A mobile phone application will provide an interface to the unit to view or create shopping lists and to query inventory.

1.3 Description

A UPC scanner will be used to identify items added or removed from the refrigerator's inventory, and a database of UPC codes will translate from the scanned code to an item description. Two inventory databases will be maintained: one linking UPC codes to product descriptions and another to store items currently checked into the refrigerator and thier expiration dates. A central processing platform on the base station will be used to decode UPC information and to interact with the databases. This processing platform will provide a web interface accessible via a web browser or an Android mobile device. A display on the main unit will allow users to review the current inventory, with expiration dates, and will provide additional information when adding or removing items. The base station, web and mobile interfaces can also be used to display the current inventory and suggested shopping lists. The mobile application will interact with the same web interface but will provide a graphical interface optimized for smaller displays. The system will continually estimate the purchasing frequency for items and will use this information, combined with the expiration and purchase dates, to suggest shopping lists. In addition to shelf life, temperature is also a critical factor for food storage systems. To address this need, the system will incorporate a temperature sensor and the temperature information will be accessible through the base station application.

A high level system diagram which partitions major components is shown in Figure 1.

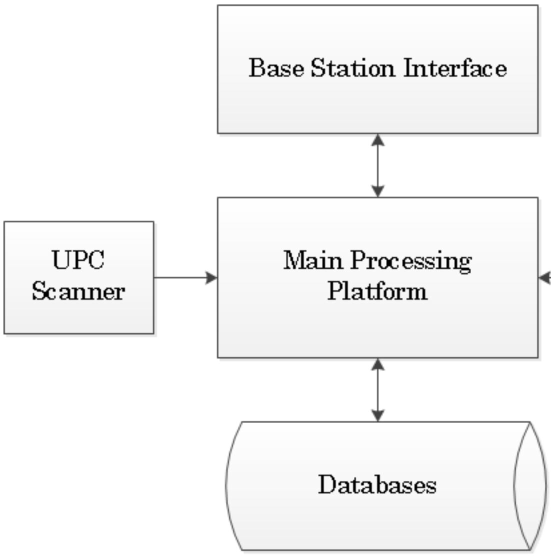


Figure 1: High Level System Diagram

2 Requirements Specification

2.1 Customer Needs

1. The system should provide an intuitive and easy to use graphical interface.
2. The system should require minimal user input.
3. The system should be able to scan product codes and identify corresponding items quickly.
4. The system should provide secure remote access.
5. The system should report items nearing expiration.
6. The system should provide access to the current inventory.
7. The system should provide a method to create and edit shopping lists.
8. The system should recommend shopping lists which accurately reflect buying habits.
9. The system should function as an add-on to an existing refrigerator or pantry.
10. The system should indicate if food products are stored at an unsafe temperature.

2.2 Engineering Specifications

Customer Need	Engineering Requirement	Justification
2,3	A. An off-the-shelf UPC scanner should be used to input items.	A UPC scanner can read product codes with a single click.
3	B. An internal UPC code database should be used to associate codes with items.	An internal database will remove delays associated with an Internet lookup.
1,4,6	C. The system should be Internet enabled and provide a web interface.	By providing a web interface any other Internet-connected device can access the system.
4	D. Remote access should be authenticated with user name and password.	User names and passwords are standard for access control.
2,5	E. An internal database will store default recommended expiration estimates for common categories of items.	Inferring expiration dates based on item category helps minimizes user input. It is well known how long some products take to expire.
1,5	F. The user interface will provide a method for updating default expiration estimates.	Default estimates will not account for condition of product on arrival and may need to be updated.
1,5	G. Interface will provide a visual indication to the user which items are closest to expiring.	The goal of the system is to reduce waste due to expiration.
1,6	H. From the base station, web and mobile interfaces the user will be able to view an inventory list.	The user needs access to the current inventory in order to use items and shop effectively.
7,8	I. A database will be devoted to storing recommend shopping lists produced by the system.	User may wish to retain generic shopping lists for future use.
8	J. Recommended shopping lists will reflect purchasing history and expiration dates of current inventory.	Recommendation policy must suggest items relevant to the user in order to be useful.
7	K. Custom shopping lists, created either from the base station or the mobile interface, can be added to shopping list database.	Inefficient shopping practices can be prevented by storing shopping lists and the system can not anticipate all required items.
9	L. The system will be self-contained and no modifications will be required to existing appliances.	Similar systems are commercially available but require costly replacement of existing appliances.
10	M. The system should measure temperature within the refrigerator.	Temperature measurements will allow the user to quantitatively determine if food storage conditions are safe.

3 Concept Selection

3.1 Evaluation of Existing Systems

Many refrigerator systems are currently available that offer integrated displays and internet connectivity. LG, Electrolux, and Samsung all offer refrigerators with large LCD displays that provide access to calendar applications, recipes, weather forecasts, and music and photo sharing services. The principle shortcoming of these devices is the elevated price and the need to completely replace existing appliances. As a more affordable alternative, tablet mounts are available for refrigerators as well. However, these systems do not offer tracking of the refrigerator's contents and do not attempt to reduce waste or improve efficiency. In April of 2011, LG demonstrated a "Smart Fridge" with goals closer to the proposed system. The sensors and algorithms used were not disclosed but the product objective is similar: tracking user purchases and providing a mobile interface to the refrigerator's contents while shopping [2]. Our system will provide a much more inexpensive alternative and will be more flexible; the system proposed will not be strictly limited to refrigerators and can be used as an add-on to an existing system.

Many patents exist on inventions related to the smart refrigerator system as a whole and its goal to reduce waste, but do not attempt to reduce user input. Patents 2004/0085225 A1 *Methods and Apparatus to Monitor the Inventory of a Food Storage Unit*, 2010/0148958 A1 *Expiration Warning Device of Refrigerator*, and 2011/0109453 A1 *Apparatus for Warning of an Expiration Date* all treat the goals of the overall system but rely on the user to enter expiration dates manually. More advanced systems, as in Patents 7,861,542 B2 *Refrigerator Including Food Product Management System* and 2011/016555 A1 *Refrigerator and Control Method Thereof*, use radio frequency identification (RFID) tags attached to foods to read expiration dates, with user input as a fallback. The prototype designed will improve the simple user-intensive method of the first group but without the added scope of radio frequency identification used in the second group.

3.2 Concepts Considered and Chosen

Many of the system design choices are easily derived from the engineering requirements; a UPC scanner with a standard USB interface is a clear choice for input of product codes and a mobile application is an obvious interface choice for a system catering to an on-the-go shopper. However, the choices of implementation platform and main base station display present more alternatives. Expiration date recognition is also a potential shortcoming of the system; ideally image processing could be employed to read expiration dates. However, the difficulty and computational complexity of applying image processing significantly extends the scope of the project and places additional performance constraints on the processing platform used. An evaluation of different expiration date recognition systems is tabulated in Table 1. The different evaluation criteria, ease of use, feasibility and accuracy, are at odds, and each criterion was given equal weight during concept selection.

Ease of use is one of the most critical system requirements; a system relying completely on input from the user will not be acceptable to consumers. However, feasibility and limiting processing performance required are important secondary objectives. Accuracy is critical to the goal of reducing waste due to expiration, but there is inherently some variability even in reported expiration dates. Image processing presents too much additional scope and too many additional requirements in exchange for marginal gains. As long as a predictive system learns from user input and anticipates that items will be purchased in different conditions, this scheme should be

Table 1: Comparison of Expiration Date Systems

	Method			
	User Input of expiration dates	Image to Text Recognition	Predictive Strategy without itemMaster	Predictive Strategy with itemMaster
Ease of Use	- - -	+	+ + +	+ + +
Feasibility	+ + +	- - -	- - -	+ + +
Accuracy	+ +	+ +	+	+
Total	2+	0	4+	7+

sufficient. One additional risk posed by the predictive system is the problem of deciphering text descriptions in order to assign an appropriate prediction. This risk has been mitigated by using the ItemMaster UPC database. Many websites, such as the Food and Drug Administration or community based resources like `www.stilltasty.com`, provide "rule of thumb" predictions for expiration dates. However, the system must associate a product description with a rule of thumb, which, after investigation, appears to be a difficult classification problem. The ItemMaster UPC database provides not only an association between a UPC code and a text description but also provides a GS1 category. There are a modest number of GS1 categories applicable to this system, each of which can be assigned a rule of thumb to initialize the prediction system.

The problem of predicting shopping habits will be formulated as a problem of predicting the probability that the user will purchase a product again after N days from the last purchase. A product will be added to the shopping suggestions at the peaks in the probability density function, after which the process would reset. To evaluate modeling strategies, receipts were retrieved for a three month interval from a single user. An initial attempt was to assume that the large number of factors influencing shopping habits could be approximated as normally distributed. However, for the data tested, this approximation was very poor; the data considered were either multi-modal or contained a single mode with outliers. In all cases considered, the distribution was shifted to the point where the most likely suggestion time was actually positioned in an interval not supported by any of the samples. A more advanced approach, a non-parametric distribution estimate, was considered next; this method outperformed the simple normal approximation, but appeared to interpolate more than necessary and appeared to be the most computationally complex method considered. However, SciPy, a Python scientific computing library provides non-parametric distribution approximation functions which are optimized and implemented in C. Database storage is another area where a non-parametric method excels; all other methods would require some parameter, or possibly a variable number of parameters, be stored in the database or be recalculated every prediction.

A final approach clustered the data points, approximated each cluster with a normal distribution, and summed these distributions. With this strategy, each mode can be captured without the influence of outliers. The accuracy of the methods considered were evaluated both qualitatively, by looking at the resulting probability density functions, and also quantitatively, by considering performance on the example sets. Overall, clustering to produce a sum of Gaussians appeared slightly more accurate but was the most difficult to store in a database. The probability metrics used are tabulated in Table 2. The goals of this subsystem, maximizing the probability of accurate recommendations and minimizing the probability of unsupported recommendations, ease of computation, and the ability to be stored in a database, were given equal weight during concept selection. Despite its slightly worse performance, the non-parametric distribution estimate was chosen since it required no storage in the database and was provided in the Python scientific computing library.

Table 2: Comparison of Distribution Estimate Performance Metrics

	Trial	Method: Normal Approximation	Method: Non-Parametric Distribution	Method: Clustering to produce sum of Gaussians
$\sum$ Log Probability Observed Habits (Goal to Maximize)	1	-38.3394	-35.9682	-34.7721
	2	-20.5647	-17.0897	-15.6641
	3	-47.8101	-44.9658	-43.9845
	4	-29.1931	-19.6762	-24.4915
Evaluation		- - -	+	+ + +
$\sum$ Log Probability Habits Not Observed (Goal to Minimize)	1	-36.7898	-38.4187	-50.6578
	2	-188.514	-225.002	-318.926
	3	-62.2909	-63.8609	-69.9759
	4	-29.6667	$-\infty$	-86.0767
Evaluation		- - -	+	+ +
Ease of Computation		+ + +	-	- -
Ability to Store in Database		+	+ + +	- - -
Total		2-	4+	0

The choice of the base station main display and processing platform are linked, but dictated mainly by the processing platform. For example, if a personal computer were used a standard LCD monitor may be appropriate, whereas if a tablet were chosen as the main processing engine the interface would be provided automatically. The most strongly considered option was to use a simple micro-controller or BeagleBoard to handle the processing load and to use a modest sized LCD display. Comparisons of different processing platform methods and different user interface choices for the base station are shown in Tables 3 and 4, respectively. The evaluation criteria for the processing platform and user interface were given equal weight; though since the processing platform and user interface concepts were related, the criteria of the processing platform were given higher priority than the user interface.

Table 3: Comparison of Main Processing Platforms

	Method: Personal Computer	Method: Tablet (Combined UI and Processing)	Method: Micro-controller	Method: Beagleboard-xM
Processing Resources	+ + + +	+ +	+	+ + +
Cost	- - -	+	+ + +	+ + +
Size	- - -	+ +	+ + +	+ + +
Total	2-	5+	7+	9+

Table 4: Comparison of Main User Interface Displays

	Method		
	LCD PC Monitor	Tablet	LCD with BeagleBoard-xM
Integration with Unit	- - -	-	+ + +
Ease of Use	+ + +	+ + +	+ +
Size of Display	+ + +	+ + +	+ +
GUI Quality	+ + +	+ + +	+ + +
Size of Unit	- - -	+ + +	+ + +
Total	3+	12+	13+

Evaluating both the interface choice and the processing platform choice together eliminates the personal computer; a personal computer cannot be integrated without significantly increasing the form factor of the system. A personal computer also greatly simplifies the system and strays away from an implementation tailored to this prototype. A tablet based interface was considered a very feasible alternative; however the cost and tailorability of the system are again concerns. A micro-controller based system is more appropriate for a small and specialized solution, with the principle concern being quality of the graphical interface produced compared with the other two methods. Considering both choices together, the Beagleboard-xM with an LCD display to inspect items visually as they are checked in and view inventory appears preferable. A Beagleboard, running a full Linux environment, will maintain a small form factor while still providing a high quality interface and sufficient processing resources.

4 Design

Consideration of the these concepts, as well as the high level system diagram presented in Figure 1, clarifies the separation of tasks while implementing the project. One group of tasks will contain the mobile interface and also development of an interface control specification to enumerate the commands provided over the web interface. A second task group will consist of configuring the internal databases on the Beagleboard, the expiration date warning system, and the shopping list suggestion algorithm. The final group of tasks will consist of interfacing the processing platform with the scanner, Ethernet interface, temperature sensor, and main user interface. The development of the base station interface will be distributed over the last two task groups. This division of work is also evident in the full system diagram shown in Figure 2.

The majority of software developed will run on the Beagleboard, which will be the main processing platform for the Smart Refrigerator system. The Beagleboard will run the Angstrom operating system, a lightweight embedded Linux. A full Linux environment will be very conducive for software development and will greatly simplify connection with peripherals; the keypad or keyboard, as well as the UPC scanner, will be able to simply "plug-and-play." The temperature sensor will require more effort, particularly to ensure the input and output voltage levels meet the specifications of the sensor and do not damage the Beagleboard. The Digilent Pmod TMP2 temperature sensor that will be used requires a 3.3V or 5V power supply and will output voltages as high as the supply voltage. The Beagleboard's I^2C interface supplies and receives voltages up to 1.8V only. To resolve this discrepancy, a BOB08745 Logic Level Converter will also be incorporated to interface between the Beagleboard and the temperature sensor.

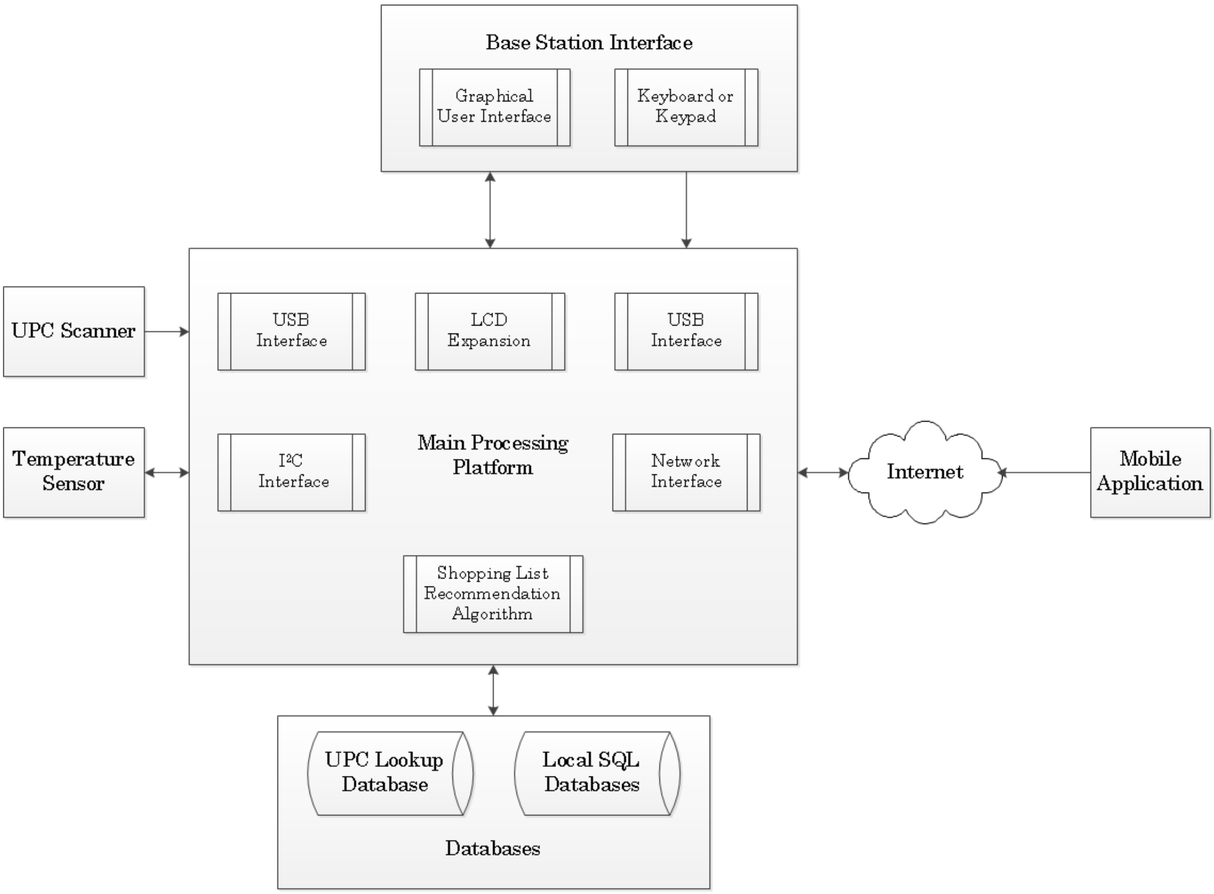


Figure 2: Full System Diagram

A more detailed diagram of the Beagleboard subsystem is shown in Figure 3. The figure shows all external connections to the Beagleboard as well as an internal separation of sub-components. The base station code will be inspired by the Model, View, Controller paradigm. A distinct module of code will create the user interface displayed on the Beagleboard's touchpad and will relay user interface events to a separate controller module. The controller sub-component will coordinate the various input events generated by the system. Input from the UPC scanner and keyboard will pass through this module and will then be transferred to the view. The controller is a necessary middleman in this process, since scanned UPC codes and user inputs must be shared with the model as well. The controller will also be responsible for handling input from the temperature and humidity sensor; this interaction will occur through a dedicated I^2C driver. The controller must also interact with the database structures and handle events from the network interface. The model itself will be distributed among the remaining modules; the content needed by the model will be stored within the databases, and the principal modeling task will occur within the expiration date and shopping list prediction sub-module.

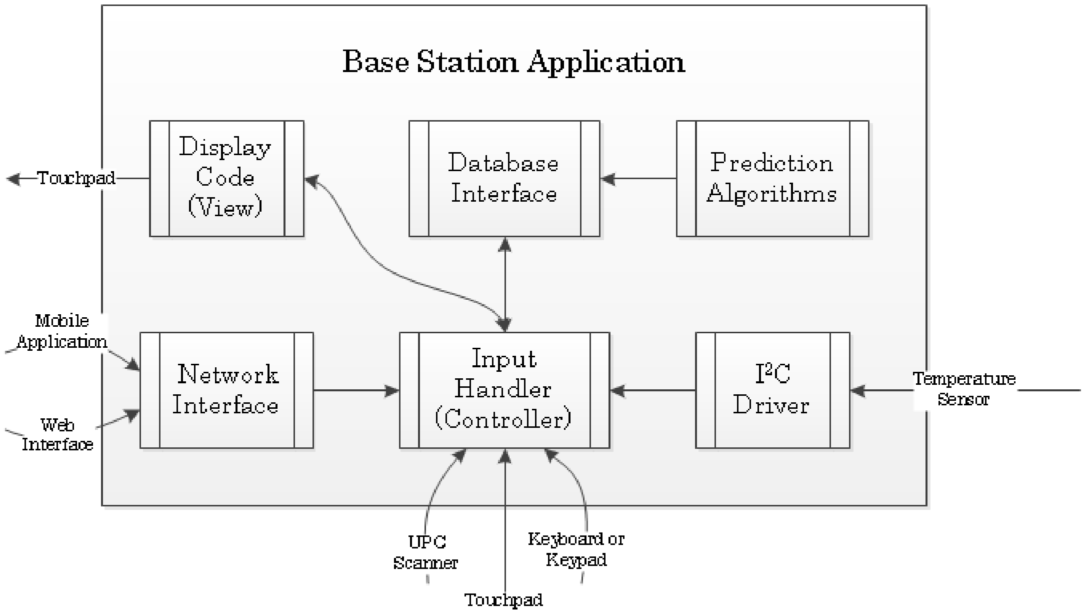


Figure 3: Beagleboard Subsystems

4.1 Base Station Code Package

The base station application will be developed in Python using the TkInter user interface framework. Java was also given consideration as the primary language, since this could potentially increase consistency with the mobile application. However, Python is notoriously quick to develop with and was able to prototype the user interface rapidly. Effort will still be made to maintain consistency between the two user interfaces. To facilitate ease of use and a fluid user experience between the two applications, the interface layout should be preserved, and aesthetic differences should be minimized. Screen captures of the base station graphical user interface are shown in Figures 4, 5, 6. When designing the interface layouts, the constraints of a touch screen interface were considered; all buttons and tabs are intentionally large and easy to click. The product entry tab, shown in Figure 4, is the default tab, and provides feedback to the user while scanning items. The check in and check out buttons function as radio buttons to indicate whether the next scanned item will be interpreted as a new purchase or an item being removed from the current inventory. Since this tab is also be the default, a list of items closest to expiring is shown at the bottom to remind the user. The shopping list tab provides a straight-forward view of past shopping lists, organized by descending creation dates. The suggested list button will produce a new recommended shopping list. The current inventory tab simply lists items currently checked into the refrigerator and provides a reset function to clear the current inventory.

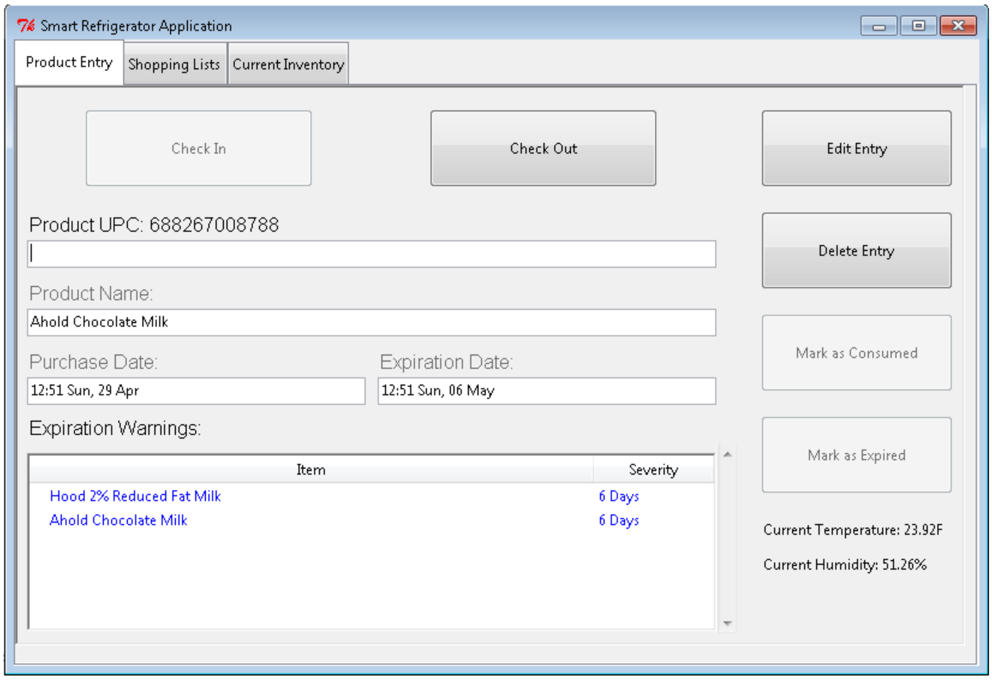


Figure 4: Product Entry Tab Layout

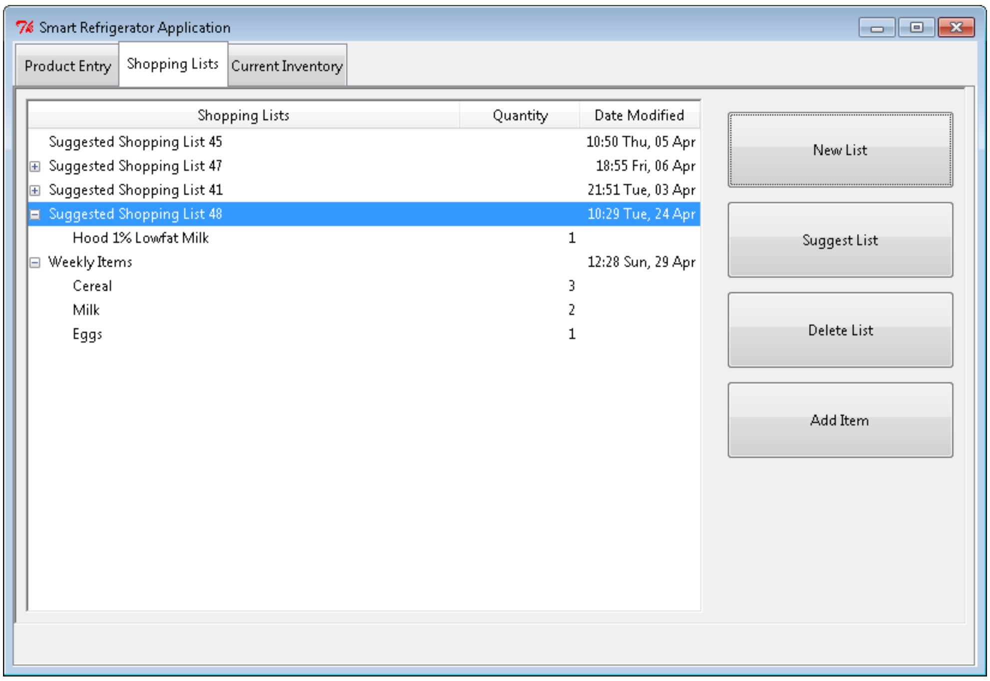


Figure 5: Shopping List Tab Layout

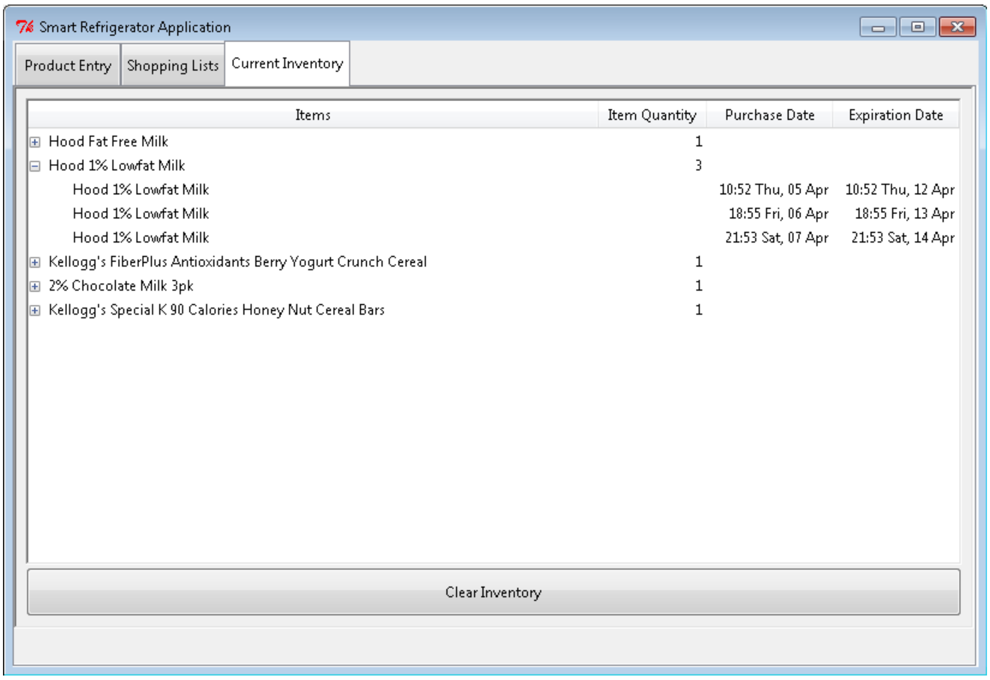


Figure 6: Current Inventory Tab Layout

4.2 Mobile User Interface

The mobile interface will be divided into similar tabs, which will limit the amount of data exchanged between the Beagleboard and mobile phone. The mobile application will be updated on a "need to know" basis only. If additional items have been added to the inventory or new shopping lists have been created, the mobile application will not be notified until the user has opened the application and navigated to the appropriate tab. It is anticipated that the largest data exchanges will need to occur for the shopping lists, since they will require information from multiple items and additional top-level information about the lists. The mobile application will further improve data efficiency by not loading shopping list items until a list has been selected. Navigation to the shopping list tab will generate a query for the top-level information about new or updated shopping lists, but will not delve into the lists themselves to retrieve item information. Only once a list has been selected will complete updated information about its items be retrieved. This implementation will create additional latency while using the application; faster strategies include a single large update of all data upon launching the application or a background fetch of information while running the application. However, given the high costs of mobile data plans and the lack of widespread WiFi at most grocery stores, limiting the amount of data passed across the network interface is an important consideration for usability.

The mobile application should also be tolerant of interrupted connections, since telecommunication networks often have spotty coverage. To mitigate the impact of dropped connections, updates should be transferred from the Beagleboard to the mobile application in a sequence of short communications; a large number of small messages will be more robust than a small number of large messages. This structure will facilitate the "need to know" distribution of information as well. The processes for handling updates to the current inventory and updates to shopping lists are conceptually similar and are illustrated with a common state machine in Figure **??**. Upon launching the application, the update system will enter a waiting state. The update system will remain waiting until a tab is selected or some other action is performed which requires a check for updates, such as expanding a shopping list to view the items within it. The mobile application will then request the number of relevant updates from the base station software. The state machine will pend until the number of updates is returned and will then enter a state which evaluates whether there are updates remaining. The system will alternate between pending to receive updates and checking whether additional updates are available. The system will return to the waiting state when all the "need to know" information has been retrieved.

4.3 Database Definitions

The various databases will be implemented as SQL databases on the Beagleboard. SQL databases will certainly be reliable, and testing will only ensure correct interaction with the databases. Figure 7 illustrates the partitioning of data into the multiple databases stored on the Beagleboard. The most intuitive database is the Item Database, which will store the total set of UPC codes and corresponding text descriptions. A quantity field will provide not only the quantity information, but also implicitly provide a check for whether an item is in the current inventory or not. The item database will also store the most recent purchase date for each item. However, only the last purchase date will not be sufficient for the shopping list prediction algorithm, so a separate item history table will be created. The history table will be indexed by UPC, and each element will contain a list of purchase dates for that particular item. Also, a shopping list database will be created; in this database each shopping list will be assigned an identification number and name. The shopping lists will also include a flag to indicate whether each shopping list was manually created or generated by the shopping list creation algorithm; the user may find this information useful when evaluating shopping habits. To coordinate between the database of items and the shopping lists themselves, an intermediate linking database will be used to store the actual items included on a shopping list. This database will be indexed by the shopping list identification number, and each element will contain a list of item UPC codes.

Extensibility – The Smart Refrigerator system presented is only a first step toward tackling food waste; integration of the concepts in this prototype into a larger context could provide enormous utility. Possibly the expiration date and purchasing prediction systems could be improved and applied in commercial domains. Both small restaurants and national chains could benefit from more accurate shelf life predictions. The shopping list suggestion algorithm may also be improved by aggregating data from multiple users.

Manufacturability – The Smart Refrigerator will be a mainly software system and will therefore be easy to reproduce; the same code package could easily be mass-produced. However, the Beagleboard is a very general processing platform. If this prototype system were manufactured commercially, a more tailored and application specific platform could be desirable. The system does not have a true need for a complete Linux environment, even though it is ideal for rapid development of a prototype.

Reliability - The components developed and tested externally, such as the Angstrom operating system, will likely be very robust in comparison with the modules developed specifically

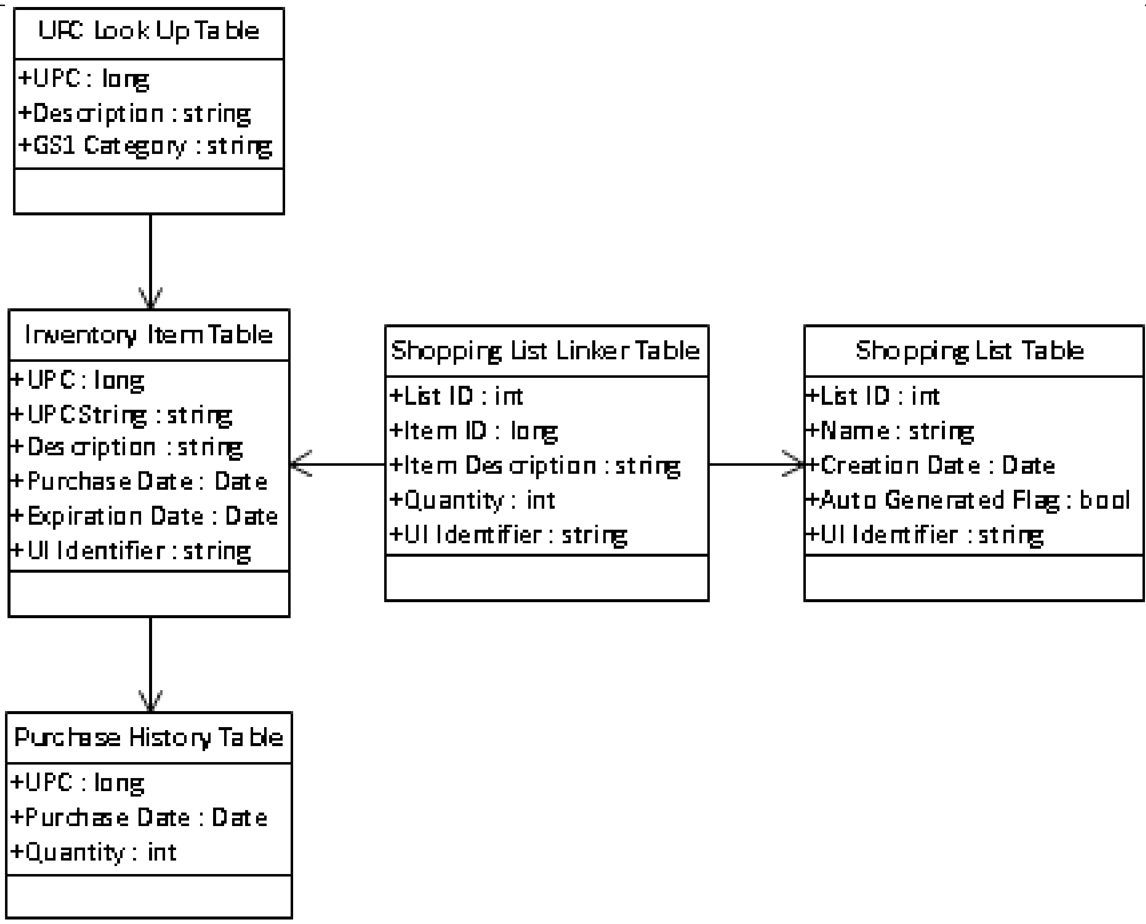


Figure 7: Separation of Information in Databases

for this prototype. The user interfaces and prediction algorithms will be tested extensively, though in both cases it is difficult to eliminate the possibility of error. The mobile interface may not encounter interrupted connections in testing, and defects may go unnoticed. The shopping list recommendation algorithm cannot be exposed to all shopping habits in testing, and therefore may perform poorly in some untested scenarios. Fortunately, the system will not incur any damage from a failure and will not immediately endanger the user in event of a failure.

Background – Experience with Linux based operating systems throughout the computer science sequence has been helpful. The computer science sequence and software engineering also have provided a valuable introduction to user interface development. The project does not contain significant hardware design, though skills learned in Interface and Digital Electronics may be useful in interfacing with the temperature and humidity sensor.

Multidisciplinary Aspects – This prototype system requires coordination between hardware and software and requires a mixture of Computer Engineering and Software Engineering skills. However, multidisciplinary projects often carry a mechanical connotation and this system does not require integration of any mechanical components.

5 Considerations

The Smart Refrigerator system proposed is a step toward promoting sustainability and good stewardship of natural resources. Both the New York Times articles mentioned in the statement of needs, and other reports [1, 3], indicate that approximately 27% of all food available for consumption is lost to waste. A study published by the UN Food and Agriculture Organization declares the global percentage is even higher, totalling 1.3 billion tons or 33% overall [4]. The system designed will increase awareness about expiring items, with the goal of reducing these figures. Hugh Collins from AOL News speculates that food waste is dismissed subconsciously; many foods are cheap, and the average consumer does not think about the aggregated cost of these small wastes [3]. By providing reminders to the user, the Smart Refrigerator can remedy this source of waste by keeping the user aware of all their purchases. Expiration of food products themselves is not the only source of waste involved with grocery shopping. Making unnecessarily frequent trips to a store for forgotten or unexpected items also wastes resources. The shopping list recommendations provided by the Smart Refrigerator will hopefully mitigate waste here as well. A final consideration of the system is health and safety, by providing reminders about expiring products the risk of eating expired products will hopefully be decreased.

6 Cost Estimates

We have submitted a proposal to the ARM student design contest requesting a BeagleBoard-xM and power adapter. Our team received a BeagleBoard-xM at no cost, however the ULCD7 Lite and power adapter were not provided and were purchased. We also already own many of the principle system components; the dorm room refrigerator, android smart phone, and keyboard will not need to be purchased.

Table 5: Cost Table

Part	Retail Cost	Our Cost
BeagleBoard-xM	$149	$0
BeagleBoard-xM Power Adapter	$26.33	$26.33
Dorm Room Refrigerator	$100	$0
Android Smart Phone	$100	$0
ULCD7 Lite Display	$151.50	$151.50
UPC Barcode Scanner	$40.67	$40.67
USB Keyboard/Keypad	$10	$0
Digilent Pmod TMP2 Temperature Sensor	$42.55	$0
Total Cost	$620.05	$271.05

7 Testing Strategy

Testing of the Smart Refrigerator will be divided into unit testing of the various subsystems and then top-level integration testing once the sub-systems have been connected. Some components used within the system, such as the Angstrom operating system and SQL database implementation, which have undergone extensive testing prior to use in our system, will be tested only to ensure proper configuration. The principle subsystems tested will be the base station user interface, mobile user interface and network interface, expiration date and shopping list prediction algorithms, and integration with the BeagleBoard.

7.1 Base Station User Interface Testing

The main testing focus will be on the user application, both the software running on the base station as well as the web and Android interfaces. Unit testing will be performed during development of each component, as well as integration testing of the final application. This subsection will focus on top-level testing of the base station user interface as a module, with tests particularly directed at the engineering specifications and user requirements. Tests directly motivated by the requirements specification and engineering specifications are listed below, and a test procedure is tabulated in Table 6.

- The user interface is required to be easy to use and intuitive; in order to verify this someone not involved in the project should contribute to top-level testing of this subsystem. This also can be tested quantitatively; tests should be performed to ensure the most used items are presented on the default tab, and the most frequently used controls are the most accessible.
- The user interface will provide access to the current inventory, which will be stored using an SQL database. The principal test effort at this step will be verifying integration of the display with the database, not verifying the storage of items themselves.
- The user interface will provide both read and write access to shopping lists, also stored using an SQL database. Testing of this feature will again focus on the ability of the interface to query and modify database entries, not on the database implementation itself.
- The user interface must provide a method to update expiration estimates. Testing of this subsystem will not verify that the update is reasonable or correct but simply verify that this user interface action triggers an update from the expiration prediction subsystem.
- To achieve the principal goal of the system, the user interface must provide a notification of items about to expire. Testing of this subsystem will not verify that the expiration estimate is reasonable or correct, but simply that if triggered by the expiration prediction subsystem the user interface will display an indication.

Table 6: Base Station User Interface Test Cases

Test Writer: Steven Strapp			
Test Case Name:	Base Station Interface Top-Level Unit Tests	Test ID #:	Base-01
Description:	Verify that the base station user interface meets the requirement and engineering specifications. Some, such as usability will be evaluated qualitatively and are difficult to outline in this way.	Type:	White Box
Tester Information			
Name of Tester:	Steven Strapp	Date:	04/29/12
Hardware Ver:	1.0	Time:	10:50
Setup:	User interface subsystem should be entirely integrated with prediction subsystems and SQL databases. System should begin without shopping lists or inventory. System date should be made mutable to facilitate quick simulation of expiration.		

Test	Action	Expected Result	Pass	Fail	N/A	Comments
1	Enter test product code	Switch to inventory tab, entered product should be shown. Inventory should be otherwise empty.	X			
2	Wait for test product to nearly expire	Interface should display a notification indicating expiring item.	X			Item added to list of items near expiration on main tab.
3	Use interface to indicate product has not yet expired	Verify that prediction sub-system is triggered to update its expiration estimate for this product.	X			
4	Create fake shopping list	Verify that list becomes accessible through base station and Android interface	X			
5	Modify items on fake shopping list	Verify that changes are retained and visible through base station or Android interface	X			

7.2 Mobile User Interface and Network Interface Testing

The web and mobile interfaces will have their own set of tests, focused on basic functionality and interoperability on various platforms. The web interface will be tested on the most popular browsers (Google Chrome, Firefox, and Internet Explorer), as well as some of the most popular mobile platforms (Android, WebOS, and iOS). The Android interface will need to be tested on various versions of the operating system. At a minimum, major versions between 2.1 and 4.0 will be tested.

Table 7: Mobile App Tests

<table>
<tr><td colspan="9">Test Writer: Ben Reeves</td></tr>
<tr><td colspan="2">Test Case Name:</td><td colspan="4">Downloading large database updates over an intermittent network connection</td><td>Test ID #:</td><td colspan="2">Mob-01</td></tr>
<tr><td colspan="2">Description:</td><td colspan="4">Ensure that the database is correctly downloaded even if the device's network connection is interrupted. This could be due to loss of service, a disabled network adapter, or the device powering down.</td><td>Type:</td><td colspan="2">White Box</td></tr>
<tr><td colspan="9">Tester Information</td></tr>
<tr><td colspan="2">Name of Tester:</td><td colspan="4">Ben Reeves</td><td>Date:</td><td colspan="2">4/19/12</td></tr>
<tr><td colspan="2">Hardware Ver:</td><td colspan="4">1.0</td><td>Time: 10:30</td><td colspan="2"></td></tr>
<tr><td colspan="2">Setup:</td><td colspan="7">System should have a fresh install of the application and no previous copies of the database downloaded.</td></tr>
<tr><td>Step</td><td>Action</td><td>Expected Result</td><td>Pass</td><td>Fail</td><td>N/A</td><td colspan="3">Comments</td></tr>
<tr><td>1</td><td>Initiate download update of the database</td><td>System should connect to the server and begin downloading.</td><td>X</td><td></td><td></td><td colspan="3"></td></tr>
<tr><td>2</td><td>Sever device's network connection</td><td>System should pause the download upon sensing the interrupted connection.</td><td>X</td><td></td><td></td><td colspan="3"></td></tr>
<tr><td>3</td><td>Reconnect device to the network</td><td>System should resume download of the database</td><td></td><td></td><td>X</td><td colspan="3">Download is restarted, not resumed.</td></tr>
<tr><td>4</td><td>Allow update to complete</td><td>System should download the remaining portion of the database</td><td>X</td><td></td><td></td><td colspan="3"></td></tr>
</table>

Table 8: UI Usability Test

Test Writer: Ben Reeves						
Test Case Name:		UI Usability Test			Test ID #:	UI-01
Description:		Ensure that the both the web and mobile versions of the User Interface are accessible and intuitive.			Type:	White Box
Tester Information						
Name of Tester:		Ben Reeves			Date: 4/19/12	
Hardware Ver:		1.0			Time:	11:45
Setup:		System should be representative of one which is in active use; that is, its database should contain both shopping lists and grocery items associated with them.				

Step	Action	Expected Result	Pass	Fail	N/A	Comments
1	System is given to a user unfamiliar with its operation and submitted to stress testing	User should experience little difficulty navigating the application and experience no bugs, freezes, or crashes.	X			

Table 9: UI Interoperability Test

Test Writer: Ben Reeves							
Test Case Name:		UI Interoperability Test				Test ID #:	UI-02
Description:		Ensure that the both the web and mobile versions of the User Interface are fully compatible with popular browsers.				Type:	White Box
Tester Information							
Name of Tester:		Ben Reeves & Steven Strapp				Date:	4/20/12
Hardware Ver:						Time:	10:40
Setup:		System should be representative of one which is in active use; that is, its database should contain both shopping lists and grocery items associated with them.					
Step	Action	Expected Result	Pass	Fail	N/A	Comments	
1	Interface is accessed via Mozilla Firefox and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X				
2	Interface is accessed via Google Chrome and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X				
3	Interface is accessed via Microsoft Internet Explorer and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X				
4	Interface is accessed via Android 2.3 and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X				
5	Interface is accessed via Android 4.0 and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X			Web interface has small break in table with default browser, not critical.	

7.3 Shopping List and Expiration Prediction Test

Testing of the expiration prediction and shopping list prediction subsystems will be difficult if the system's date can not be adjusted artificially; testing should occur over a few minutes rather than a series of days. For expiration date testing, the system's date should be easy to change artificially, so products appear to expire very quickly. The intelligence of the system can then be tested by providing feedback that test products expired more or less quickly than expected, and evaluating the updated predictions. By simply accelerating the rate with which the system changes date, this subsystem can be tested without adding specific test products with low shelf lives and without changing the algorithms to update more frequently. A set of tests is listed for this subsystem below.

- Enter a product code and verify that the expiration date system is initialized with recommended "rule of thumb" value.
- Provide feedback indicating that a product expired before estimate and validate that the estimate is decreased. Also validate the opposite case: providing feedback that a product had not yet expired on estimated date should increase the estimate.
- Enter a product code and advance the system time until the product is nearly expired. Verify that the prediction subsystem has indicated to controller that the product is nearing expiration.
- Rescan a product code after regular intervals, indicative of uni-modal shopping habits. Verify that the system recommends the product should be purchased again on this mode. Date may be artificially advanced to facilitate quick testing.
- Continue from the previous case and add outlier shopping habits. Validate that system continues to recommend purchasing the product again on the mode.
- Enter various products with different purchasing habits. Verify that the system recommends purchasing the products with the highest probabilities.
- Rescan a product code after regular intervals, then add significant variation. Observe that system attempts to track the variation in habits.
- Rescan a product code after varying intervals, indicative of bi-modal shopping habits. Verify that the system recommends purchasing the product on both modes.

7.4 Integration with BeagleBoard

Preliminary testing will focus on the BeagleBoard itself and its ability to interact with the desired peripherals. The system will require an LCD screen, a USB barcode scanner, a network connection, a keypad, and temperature/humidity sensor. Basic functionality of these components will be tested thoroughly during development, as well as during final system testing.

The SQL database used to store all data for the system will be tested once the core of the user application has been coded. Test scripts will be written to populate the databases with fake data in order to ensure that the database is configured as desired, and to verify that the user application is properly communicating with the database alongside the web interface.

It is difficult to outline exactly what testing will be required for the processing platform, since it is unclear what compatibility issues will arise that would not be presented by a conventional platform, where ideally the system would be entirely "plug and play." However, listed below is a baseline sequence of tests.

- Verify that the BeagleBoard, with power adapter, can power all peripheral devices reliably so that no sporadic failures occur; this will be performed as an endurance test.
- Verify that MAC address of Ethernet interface can be statically assigned and the BeagleBoard can be pinged reliably; this will be performed as an endurance test, cycling power or disconnecting the board multiple times.
- Verify that the BeagleBoard can reliably interface with the USB scanner and USB keypad; these tests should be performed by writing to a text editor or another program external to the user interface to isolate failures.
- Verify that the BeagleBoard consistently receives accurate temperature and humidity measurements from the sensor, via the general purpose input/output pins. The measurements should be verified with an external sensor.
- Verify that the touchscreen display accurately records users clicks and controls the pointer; tested outside of the user interface to isolate failures.
- Verify that touchscreen accurately displays the graphical user interface without artifacts or distortion consistently, and ensure all controls on the display are accessible.

8 Risks

The risks unresolved in the prototype system are best grouped by subsystem. Some unresolved risks are simply due to unknown availability of parts, others require further exploration, and still others are more fundamental risks.

Availability of LCD Display – The ULCD7 Lite is the preferred choice for the display to accompany the Beagleboard. The 7-inch resistive touchscreen is designed to work with the Beagleboard and has drivers built into the Angstrom Linux distribution. However, the touchscreen does not seem to be stocked by online suppliers and our request through the Arm Developer Day proposal process is still pending. Since the Beagleboard provides a DVI-D and S-video output, this risk could be mitigated using a traditional computer monitor if the preferred touch screen is not available. The availability of these alternate interfaces also facilitates development while waiting for the desired part. However, since the base station display is a critical system component, if this risk is not resolved shortly one of the less desirable alternatives will have to be selected.

Mobile Application Data Exchange – Exchanging inventory and shopping list data with the mobile application presents two risk areas: the amount of data plan usage required and interrupted network connections. These risks are partially mitigated by the design, which exchanges data in small increments on a "need to know" basis only; however the degree to which these risks are truly problematic is not know. For development, the smart phone can be connected with WiFi instead of a costly data connection; however, the data requirements should be tested early in development to ensure this risk does not become a latent problem. Interrupted connections are considered in the design, but the effectiveness of the design will be difficult to measure since it may be difficult to repeatedly interrupt the data connection.

Practicality of Prediction Algorithms – The shopping list prediction algorithm outlined appears optimal for the test data considered. However, the analysis performed did not consider a large sample of shoppers, and the conclusions drawn may not be appropriate for a larger population. Also, the prediction algorithm outlined will not perform well with only a few samples; it is unclear how unacceptable this warm-up period will appear to end users. The strategy presented is also quite heavy-handed and may be superfluous for this prototype system. The system will not serve as an effective shopping aid without an accurate prediction system, but a prototype may be acceptable with a much simpler system requiring significantly less effort.

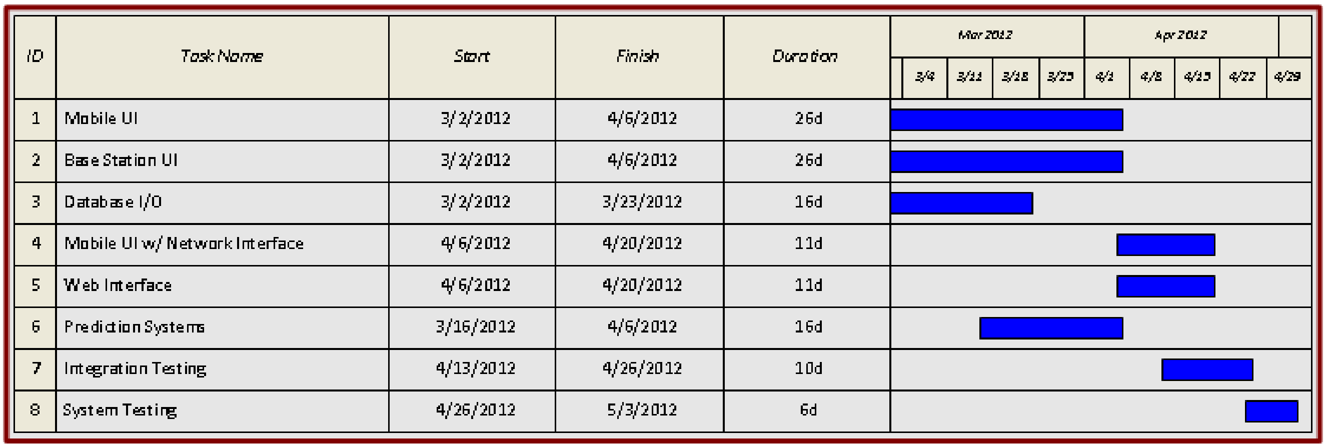
Efficiency of Database Access – Efficient practices for database access were not known or considered when partitioning information into separate tables. Efficiency was also not considered when developing the schema for the databases. For example, using the same database to store all possible items and the current inventory may be an inefficient choice. This strategy requires looking over all elements and extracting only those with non-zero quantities. If the system were extended to large commercial applications, or if the Beagleboard were replaced with a more cost-efficient and less powerful alternative, this organization may become a significant risk.

9 Milestones

Milestone	**Scheduled Date**	**Assigned**	**Modified Date**	**Comments**
BeagleBoard procured	February 10, 2012	SS	NA	Complete
Angstrom operating system running on board	February 24, 2012	DS	NA	Complete
Peripherals properly interfacing with board	March 02, 2012	DS	April 27, 2012	Complete. Touch screen Angstrom image works for both touch screen and temperature sensor.
Basic mobile UI, suitable for debugging	March 09, 2012	BR	NA	Complete
Basic base station UI, suitable for debugging	March 09, 2012	SS	NA	Complete
Database I/O configured	March 16, 2012	DS	March 23, 2012	Complete. MySQL databases configured, can be accessed with Python application.
Testing and integration of temperature and humidity sensor	March 16, 2012	SS/DS	April 27, 2012	Unable to interact with one wire sensor using Beagleboard GPIO and level shifter. Replacement I^2C temperature sensor from Dr. Mondragon works correctly.
Database and web server hosted by Beagleboard	March 16, 2012	DS	NA	Complete
Beagleboard touchscreen display procured	March 16, 2012	DS	NA	Complete
Mobile application integrated with web server	March 30, 2012	BR	April 20, 2012	Complete
User profiling and statistical analysis	March 30, 2012	SS	April 6, 2012	Complete

Milestone	**Scheduled Date**	**Assigned**	**Modified Date**	**Comments**
Shopping lists, item modification, basic settings	March 30, 2012	SS		Complete. Added context menus and hidden “administrator” panel.
Updated base station UI	April 6, 2012	SS		Complete, awaiting review by team or outside testers.
Updated mobile application	April 6, 2012	BR	April 20, 2012	Complete
Integration testing and system verification	April 13, 2012	BR	April 27, 2012	
Web interface development	April 20, 2012	SS		Complete. Does not look flashy but is effective and functional.
System testing and demo preparation	April 20, 2012	SS	April 27, 2012	
Project Website	May 3, 2012	SS		Complete
Project Demo	May 3, 2012	Team		
Imagine RIT Demo	May 5, 2012	Team		

9.1 Gantt Chart



ID	Task Name	Start	Finish	Duration
1	Mobile UI	3/2/2012	4/6/2012	26d
2	Base Station UI	3/2/2012	4/6/2012	26d
3	Database I/O	3/2/2012	3/23/2012	16d
4	Mobile UI w/ Network Interface	4/6/2012	4/20/2012	11d
5	Web Interface	4/6/2012	4/20/2012	11d
6	Prediction Systems	3/16/2012	4/6/2012	16d
7	Integration Testing	4/13/2012	4/26/2012	10d
8	System Testing	4/26/2012	5/3/2012	6d

Figure 8: Project Gantt Chart

10 Perspective

The design of the final project adhered to the original proposal, however some of the implementations themselves changed. Some aspects of the system, such as the web user interface and database organization, were not thoroughly enough defined in the proposal and were not clear until the actual implementation. Difficulties and deviations from the original proposal are organized into subsections below.

10.1 Base Station Code Package

The most significant change in the base station code package was the programming language used; the original proposal declared C++ along with the Qt user interface framework, whereas Python and the Tk framework were eventually used. Python is much quicker to develop and the Tk allows for very quick creation of user interfaces. Python was also an advantageous choice for the prediction subsystems, the scientific computing library SciPy provides clustering and kernel density estimator code; both of which were considered as options for the shopping list recommendation system. The prediction system itself was revised after considering implementation with MySQL databases. The clustering strategy initially chosen during conception selection became undesirable in that it required clustering information to be stored in the database, and the number of cluster were not known a priori. Alternatively, the clustering could have been performed again at every prediction step however this was computation undesirable. Once database storage was considered in the concept selection a non-parametric estimate appeared optimal instead.

The user interface design followed the mock-ups very closely. The only significant change was to remove the expiration warning pop-ups and replace them with a list of high risk items on the main panel. Once the GUI was created and experimented with it became clear that by only showing a pop-up it was too easy to forget the warning after the pop-up had been cleared. A list of items near expiration on the main window prevents users from forgetting items that had been identified as nearing expiration.

10.2 Web Interface

The specifications for the web interface were not clearly defined in the original proposal, and expectation for this interface varied between team members. The expectation and implementation for this interface was not resolved until initial implementation. The web interface was implemented as a simple HTML page with embedded PHP snippets to query the databases hosted by the Beagleboard.

10.3 Mobile Interface

The initial plan outlined in the proposal was for the mobile application to interact with the Beagleboard through the same web interface. However, part way through development it was suggested that it might be simpler to interact directly with the remote databases on the Beagleboard. Attempts were made to connect to the databases with an object-relational mapping without any success, and it was later discovered that Android does not support remote database access. Instead, the design reverted back to the original plan. Another small PHP snippet on the Beagleboard's web server was included to respond to HTML post requests, access the database, and return result to the Android application. The Android application would issue a post request indicating which tabel should be accessed, the server side PHP would perform this query and return results as a JSON string. The Android application parsed the JSON string and converted the returned values into Java objects. Aside from this indecision, the Android develop proceeded as expected.

10.4 Beagleboard Integration

A number of integration issues arose while using the Beagleboard. The first issue encountered was with the Python base station code package. The shopping list prediction system used the SciPy library, however prebuilt binaries for this library are not available for ARM architectures. Compiling this package from source was unexpectedly difficult and required resolution of many dependencies. Next, the temperature sensor initially purchased, an RHT03, could not be interfaced with the Beagleboard. The RHT03 used a custom one wire interface that enforced very accurate timing requirements; the Beagleboard, running a non real-time operating system was not able to meet these requirements. Attempts were made to use more accurate timers or interrupts from the GPIO pins but neither of these methods accurately retrieved information from the sensor. Instead, an I^2C temperature sensor was obtained from Dr. Mondragon which worked trivially. Finally, the ULCD7 Lite touchscreen was initially only partially functional with the Angstrom image used; the display worked but would not respond to touch. Investigation revealed that additional modules needed to be compiled in with the Angstrom kernel. Instead of pursuing this option, the Angstrom image provided for the temperature sensor was used and the components developed on the earlier image was reinstalled on the touch screen image. This strategy required repeating installations and configurations but also appeared to be the option that was simplest and most likely to succeed.

References

[1] Martin, Andrew. "One Country's Table Scraps, Another Country's Meal." New York Times. N.p., 18 May 2008. Web. 26 Jan 2012. http://www.nytimes.com/2008/05/18/weekinreview/18martin.html?pagewanted=all.

[2] Ridden, Paul. "LG launches first Smart-Grid appliance: the Smart Fridge." gizmag. N.p., 27 Apr 2011. Web. 29 Jan 2012. <http://www.gizmag.com/lg-smart-fridge/18502/>.

[3] Collins, Hugh. "Study: US Food Waste Is a Huge Energy Drain." AolNews. N.p., 02 Oct 2010. Web. 29 Jan 2012. <http://www.aolnews.com/2010/10/02/study-american-food-waste-is-a-huge-energy-drain/>.

[4] Ramaker, Rob. "Food waste is hard to combat." Resource. N.p., 26 Jan 2012. Web. 29 Jan 2012. <http://resource.wur.nl/en/wetenschap/detail/food_waste_is_hard_to_combat/>.