Functional Specification and Management Plan

Date:04/02/2016

Editors:
Tal Melamed
Brandon Mabey
Hadong Tao
Tania Akter

Table of Contents

Revision History

Feb 16, 2016. Version 1.0. Document creation, functional, user interaction, and management content.
March 3, 2016. Version 1.1. Incorporating customer comments, adding UML diagrams, and expanding system description.
March 13, 2016. Version 1.2. Adding technical design and testing plan.
March 18, 2016. Version 1.3. Formatted

Executive Summary

The purpose of this document is to describe the functional specifications of Servr, a system that handles the transport of full and empty carts to and from shoppers in various brick and mortar stores. The systems aims to make the shopping system more efficient by allowing users to send the chosen goods to the checkout stations while shopping for other goods thereby increasing store profits and customer satisfaction.

Servr delivers empty carts to users which make a request, taking their full carts to the checkout station. To do this, it must pinpoint the customer's location relative to the cart and keep track of obstacles in the store for which the cart must avoid. A mobile app will be the customer's main point of interaction for the system, allowing them to call and dismiss carts, provided that the pre-conditions set out in this document hold. The application will communicate with a central control station located within the premises which manages both the carts and users endpoints.

In addition, the minimum system to be implemented is specified, including possible implementations. Possible additions to the system if development progresses at a faster than expected pace are also defined in this document.

Functional Specification

Project Summary

The “Servr”shopping assistance system is a robot management and deployment system coordinating assistance to department store shoppers. Shoppers with a cart filled to maximum capacity can issue a request to the system which will dispatch a robot to bring the shopper an empty cart. The shopper, receiving the empty cart, can send the old cart to the checkout station and continue shopping for more items.

Needs and Objectives

The following needs and objectives are considered for the purposes of this document:

Needs

- Deliver an empty cart to the user’s location promptly after a request.
- Deliver the user’s cart to the checkout station.
- Avoid collision with obstacles and other objects in the store.
- Scan location identifiers within the store

Objectives

- Allow customers to purchase more goods than a single cart can carry.
- Improve checkout speed and reduce checkout lines.
- Increase store profits.
- Increase customer satisfaction and loyalty.

Significant Features

Customer request and dispatch interface

The shopping centre is populated with position information nodes at regular intervals. Users can request an empty shopping cart at these nodes by triggering an interaction between the nodes and the user’s mobile smartphone running the system’s mobile app. Triggering the interaction causes the central control system to dispatch an empty cart to the node’s location.

Each shopping cart is equipped with a module that the user can interact with to link the cart to their ownership and send the cart to checkout. Upon receiving an activation from

the user, the module is assigned a unique identifier from the user. This identifier is used by the user to claim the goods later at checkout.

Robot shopping cart pathfinding system

The shopping cart robots drive along predefined paths as instructed by the pathfinding algorithm in the central control system. The central control system has a predefined library of paths to and from any position. Upon receiving a dispatch request, after processing all current and pending traffic, the system selects a viable path preventing collisions with other robots and assigns it to the requesting robot. The robot will follow the path it is assigned. Collision with shoppers and other objects is prevented by detecting obstructing objects with a rangefinder that every shopping cart is equipped with.

Robot shopping cart maintenance system

A storage area is outfitted with robot charging stations. Robots will reside in these stations when not in use to ensure full charge ready status

Central Control

The central control system is responsible for all decision making and robot path generation. The control system coordinates shopping cart request and dispatch orders. It stores the data of all shopping cart locations and customer information such as checked total and owned carts. The system runs as an application on an in-store computer connected to the internet.

Hardware and Performance

Servr is expected to utilize smart mobile phones as the main interface for customer interaction. The mobile application will wholly contain the user interface for all customer interactions. In addition, the response time of the system progress is under 0.1s to give the impression that it is reacting instantaneously. For actions that cannot be completed within 0.1s, a loading bar or spin circle will be shown within that time range to ensure that the user understands that system is working as expected.

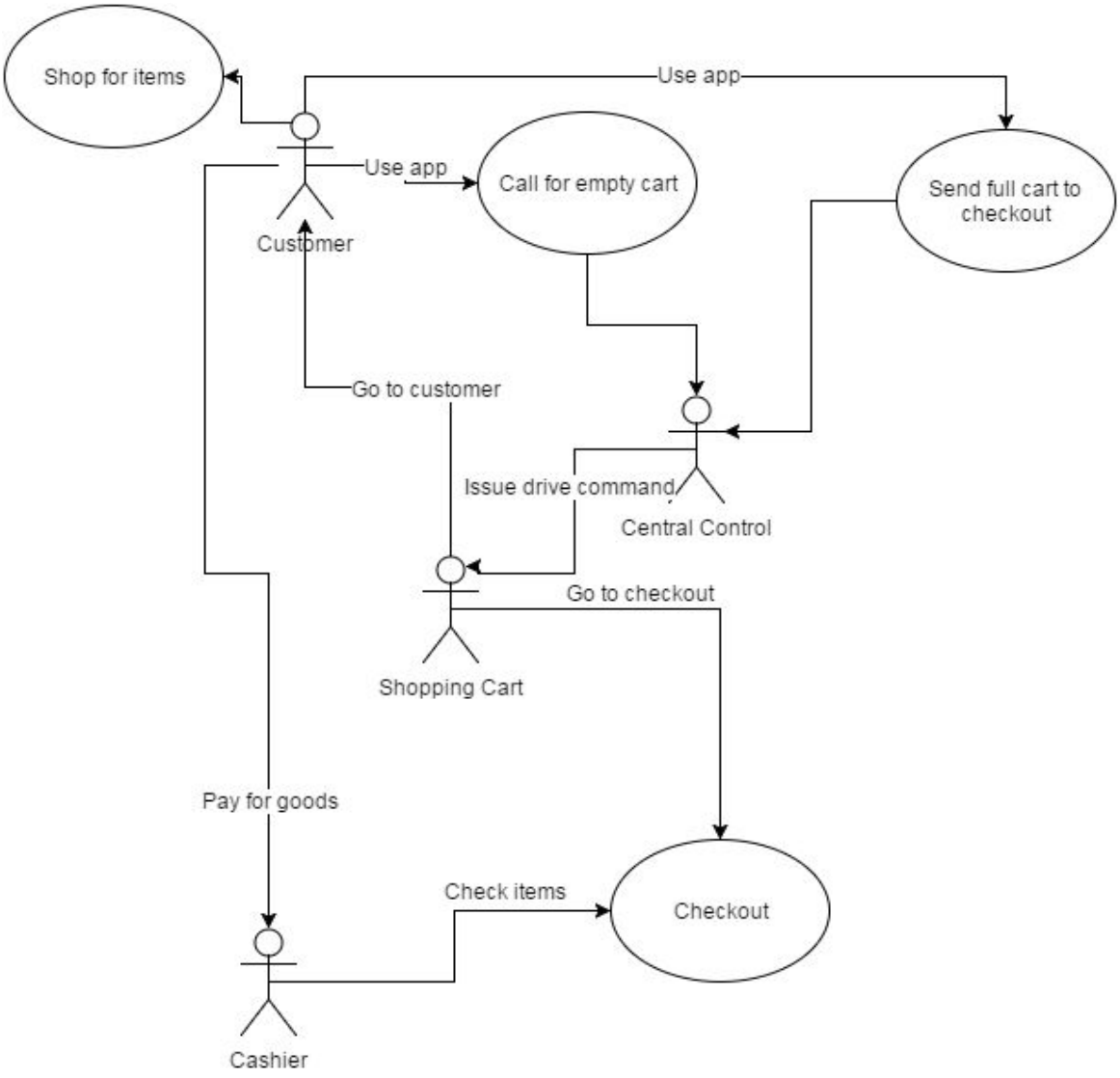
Stores will be required to have an army of robot shopping carts ready for usage. The robot carts will have driving motors and mechanisms capable of driving a predefined path through the store. Each robot must be equipped with rangefinder sensors and collision prevention software. Robots must be electric powered with a battery lasting for 30 minutes of usage. Stores require charging stations for the robots.

Beacons are positioned at regular intervals throughout the store for position information to the system, with at least 1 beacon per aisle.

The cart can not collide into other objects. The cart must arrive to the requesting customer within 1 minute of the issued request.

User Interaction

The application involves interacting with a mobile application and with the Servr shopping cart itself. To use the system, the customer must have the app downloaded.

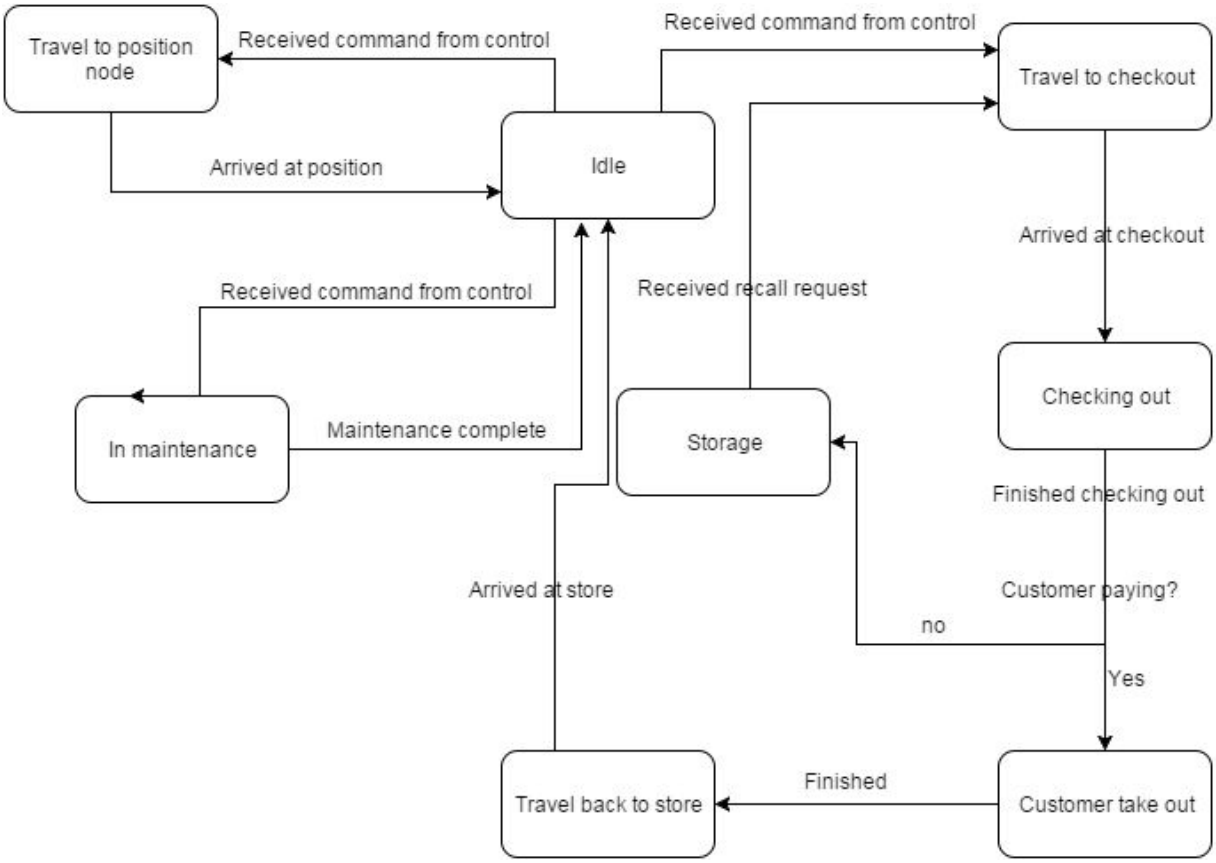


The customer uses the mobile application to request for a new empty cart to their location or to send their existing full cart to checkout. The system determines the location of the customer and an available self driving cart is dispatched to them. The user marks the existing full cart with unique identification and the cart leaves for the checkout station and waits till the customer is done with shopping. When the customer

is ready to checkout, they can use their unique identifier to bring up their checked total as well as recall the cart full of their goods.

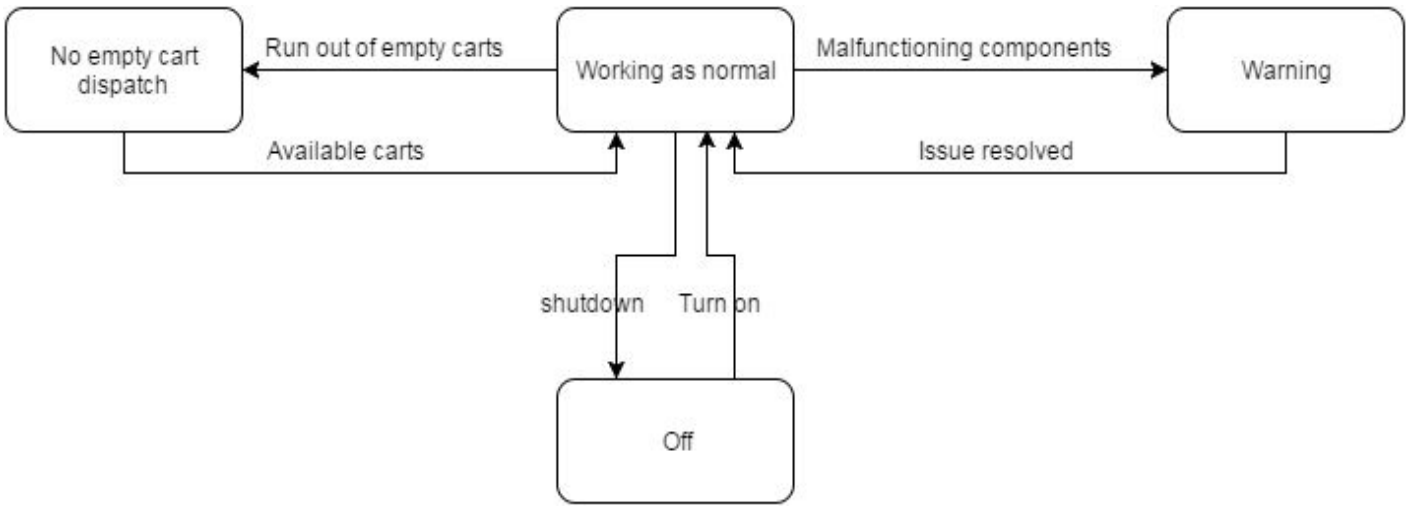
System Functions

Cart state Diagram



The cart can be in a number of different states. The states are changed in response to commands received and context such as current state. Idle is the state the cart is most often in. Idle state allows for actionable responses to commands received from the control center. A cart will experience all the states multiple times during its typical lifecycle.

Control State Diagram



The control system features three working states and one off state. The control system is typically in the “working as normal” state for most of the time. “Warning” and “no empty cart dispatch” are states of concern indicating a deficiency in the running system, these issues should be promptly resolved to avoid further inefficiencies or damages. The system will attempt to continue to work during states of distress.

Initial Configuration

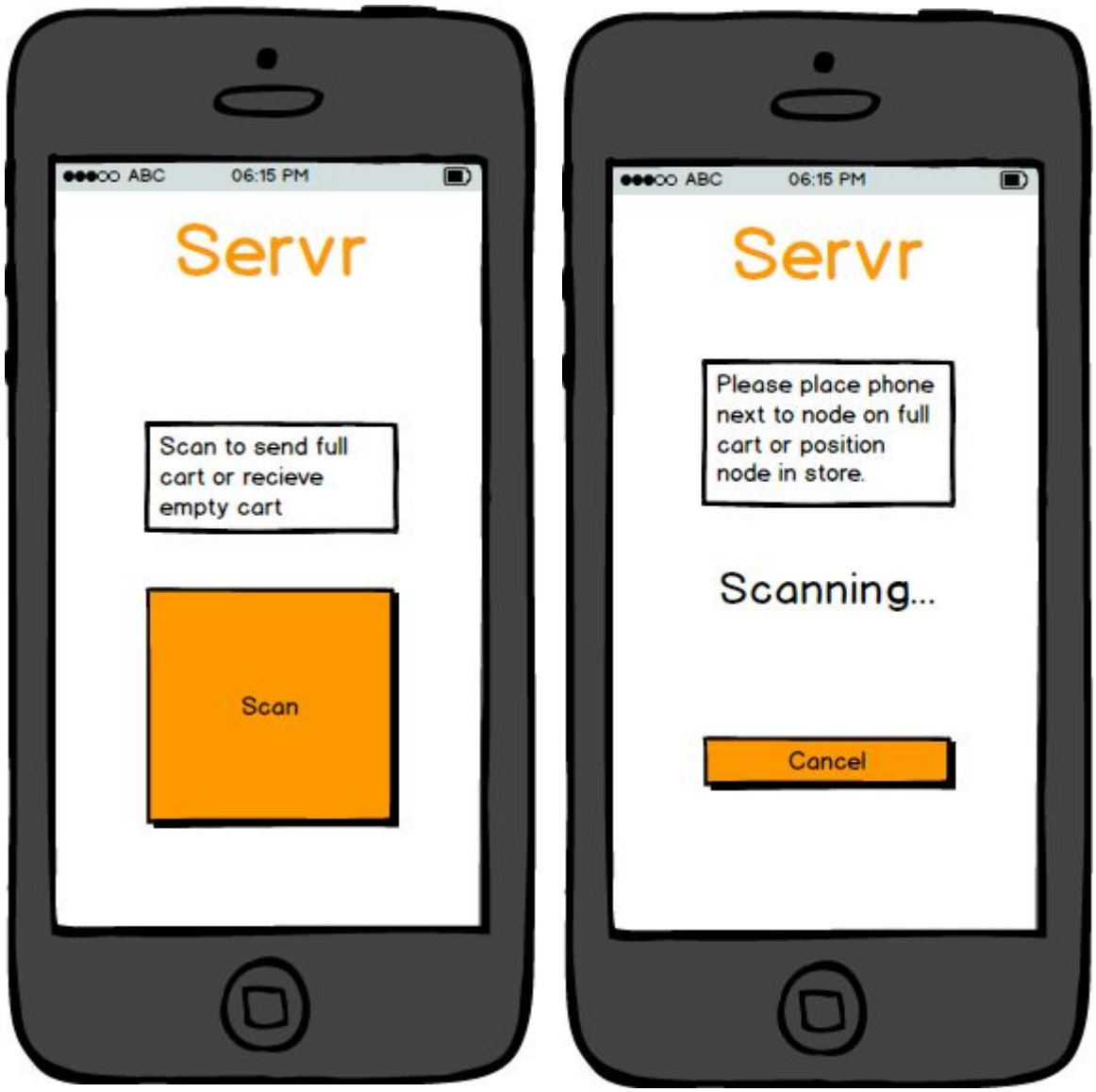
Initial configuration is performed by store management. Management sends a floor plan of the store including aisle placement and other major structures to Almost Games. Almost Games then sends back customized software, shopping cart robots, and position nodes with instructions on node placement. Management then follows the instructions to place nodes then activates the system using the central controller.

App

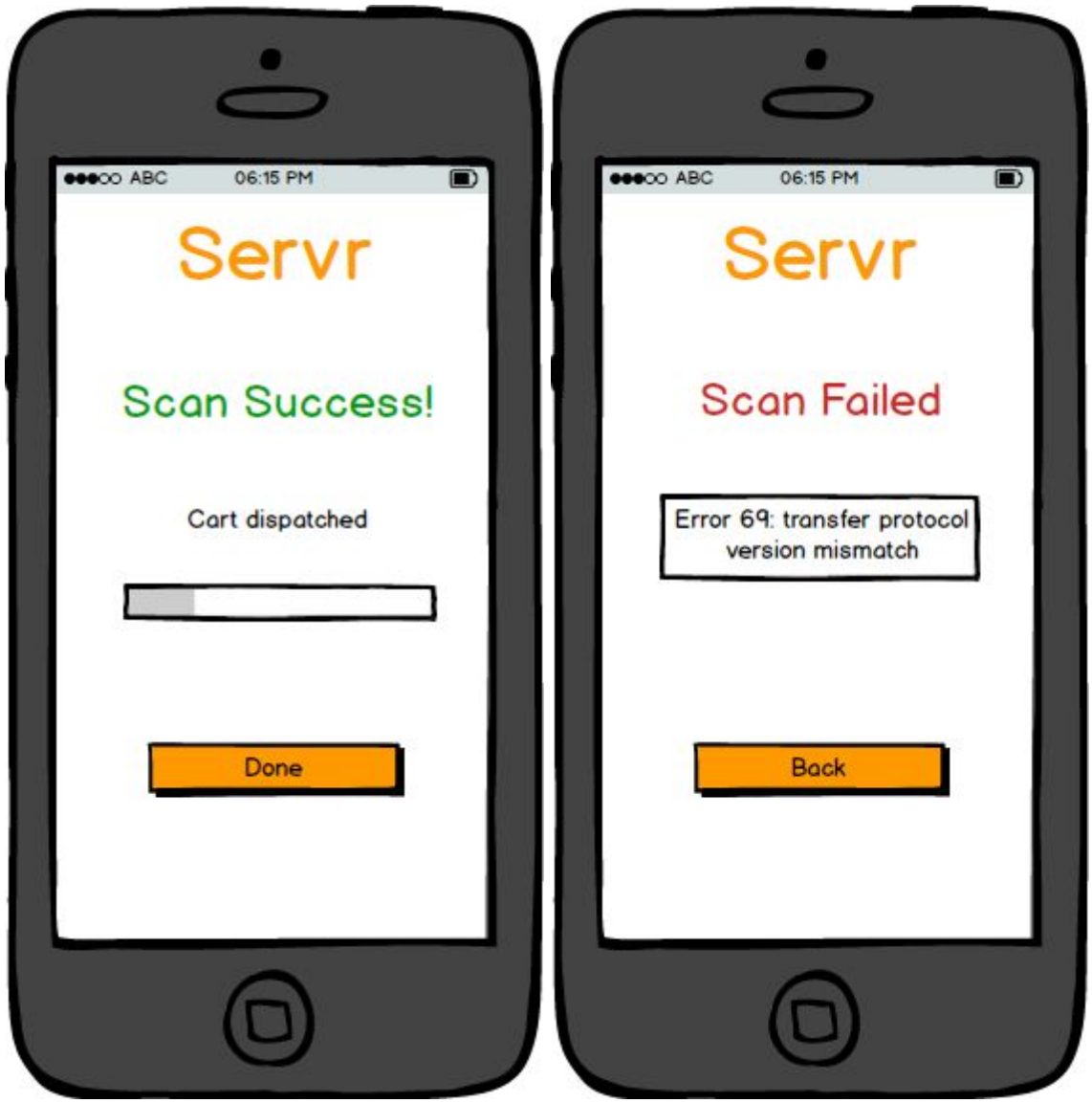
Customers intending to use the system must download the mobile app to their smartphones from the app store. Upon opening the app for the first time, some personal information is requested, including name and date of birth. The app has a single button named “activate scanner”. Pressing this button activates the module that interacts with the position nodes and shopping carts. Positioning the phone near a position node initiates a call for an empty cart to the customer's location. Positioning the phone near a shopping cart makes that shopping cart link to the users unique mobile token and sends the old cart to checkout. Both actions feature prompts indicating the action to be performed with an “are you sure?” message that the user can confirm on. When the

customer arrives at checkout, the scanner is once again activated and interacted with the checkout machine to bring their tagged carts to their location for item to be picked up.

Below is a mockup of the user interface on the mobile app used by customers. It demonstrates the scan functionality of the app.



As shown, a scan button is present in the middle of the screen. The user presses the button and a scan is initiated. This is when users can scan nodes on shopping carts to claim them, or scan nodes on positions to call for a cart to node position. When implementing the nodes with QR code, the scanning would be using the camera app for to read the QR code. If nodes are implemented with NFC, the scanning would shown a scanning screen as shown above and wait for the NFC chip to connect.



When the scan is successful, a screen like the one depicted on the left is shown on the user's phone. When the scan fails, the screen on the right is shown and the user must collect a cart manually.

Shopping Cart Robots

The shopping cart robots contain a scanner module and a status light near the handle. The scanner module is interacted with a mobile phone running the application. The status light is a small LED bulb. The light blinks a steady green for 5 seconds if the cart interaction is successful. The light blinks an intermittent red at a frequency of 15hz if an interaction was detected but unsuccessful. The Status light is off if no interaction is detected.

Central Control

The central control system runs as a desktop application on an in-store computer. Users can start and stop the system and see current status such as in-traffic carts and currently checked items. Analytics data is made available to the user, such as number of requests made, peak hours, popular areas for requests, etc.

Checkout

Checkout features a scanner module and a cashier. Customers interact with the scanner module using the mobile app function and placing the smartphone near it, this triggers the checkout to receive the customer's checked total and prompt for shopping cart delivery. The customer pays the checked total with usual methods, including either cash, debit or credit. The shopping cart delivery prompt contains a list of the customers checked carts, they can request to have all their carts delivered to their location or to have one at time delivered.

Context Awareness

The shopping cart robots can detect the collisions from all directions. The robot can detect any obstacles including the shoppers, the default isles, other robots nearby, and the speed bumps the stores put up in the the passageways occasionally. The system decides if the robot should go forward or not.

The central control system is aware of all in-traffic carts and all delivery and dispatch requests, the driving path of the robots is determined using this information.

Sample Interactions

An intuitive way to envision how the system operates is to think of the system as if it were another person whom the user was talking to. Under this assumption, the following are sample dialogs which explain the overall operation of the system:

User: Please send a cart. Here is my location.

System: Please wait. A cart has been dispatched to your location.

System: A cart has arrived at your location. Please mark the cart as ready for future checkout by swiping your phone.

User: I have swiped my phone.

System: Thank you. Your items will be ready for you at checkout.

If the system detects that no empty carts are available, the dialogue would instead go something like the following:

User: Please send a cart. Here is my location.

System: Please wait. A cart has been dispatched to your location

System: It seems that all of the available shopping carts are in use. Would you like to be notified when there is one available.

User: Yes

System: Okay, I will notify you when a shopping cart is available for usage.

At a point later in time, the dialogue would continue:

System: A shopping cart is now available. Would you still like an empty shopping cart?

User: Yes, here is my location..

System: Please wait. A cart has been dispatched to your location

System: A cart has arrived at your location. Please mark the cart as ready for future checkout by swiping your phone.

User: I have swiped my phone.

System: Thank you. Your items will be ready for you at checkout.

The user may change their mind at any point as well. For instance, if the cart arrives, and the user decides to not consent purchase, the dialogue may change into the following:

User: Please send a cart. Here is my location.

System: Please wait. A cart has been dispatched to your location

System: A cart has arrived at your location. Please mark the cart as ready for future checkout by swiping your phone.

User: I have swiped my phone.

System: Thank you. Your items will be ready for you at checkout.

When the user is done shopping, they will be asked by the cashier if they had any additional carts. If they have had previous carts, they will be added to the total cost, and returned to the customer.

Cashier: Hi, thank your purchase. Did you have any other carts during your shopping experience with us?

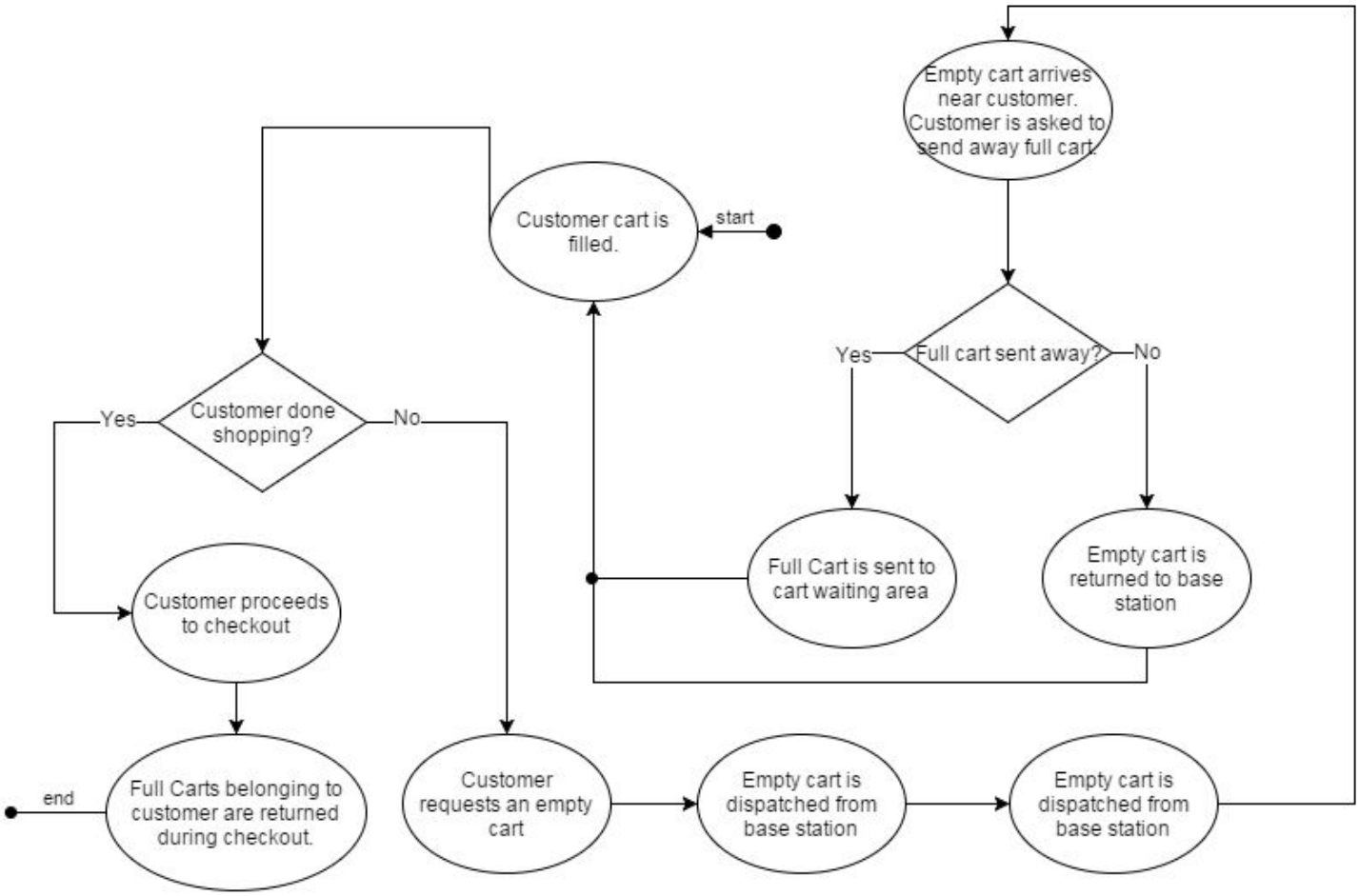
User: Yes, here is my phone

Cashier: Okay, I've let the system know that the carts should be returned to you, and added their total to your final price. How would you like to pay?

From that point on, the dialogue would remain the same as it is in grocery stores today, except that the full carts that the user sent away would be returned to the user.

Overview of the System

Activity Diagram



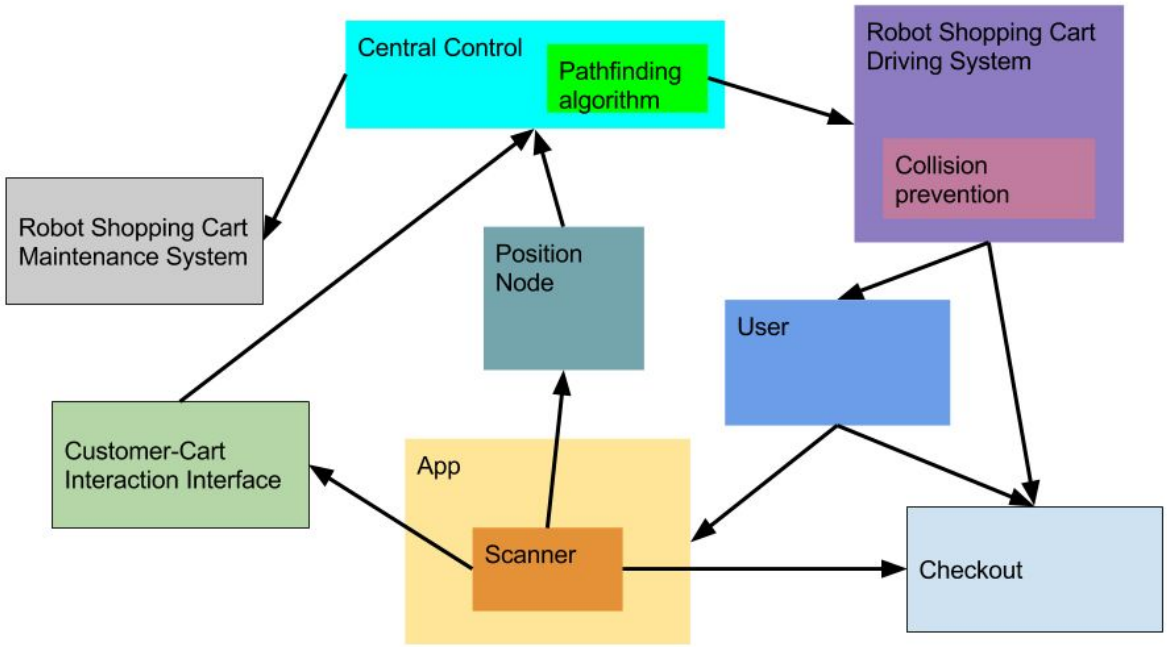
The activity diagram above shows the expected flow that customers will take during their interaction with the system. Because it is expected that all users will enter and exit the store at some point, there are entry and exit points marked "start" and "end" that designate when the user enters or exits the store. In addition, the pictorial representation of the expected flow for the customer does not take into account potential errors that the system occurs, and assumes the "happy path".

The diagram is read by following the arrows to each bubble or diamond. Bubbles represent events that are expected occur, with the arrow leading out of the bubble to be

followed after the event described by the bubble occurs. Diamonds represent different choices or possible events that may happen, and are read by following the arrow that exits the diamond that corresponds to the choice made.

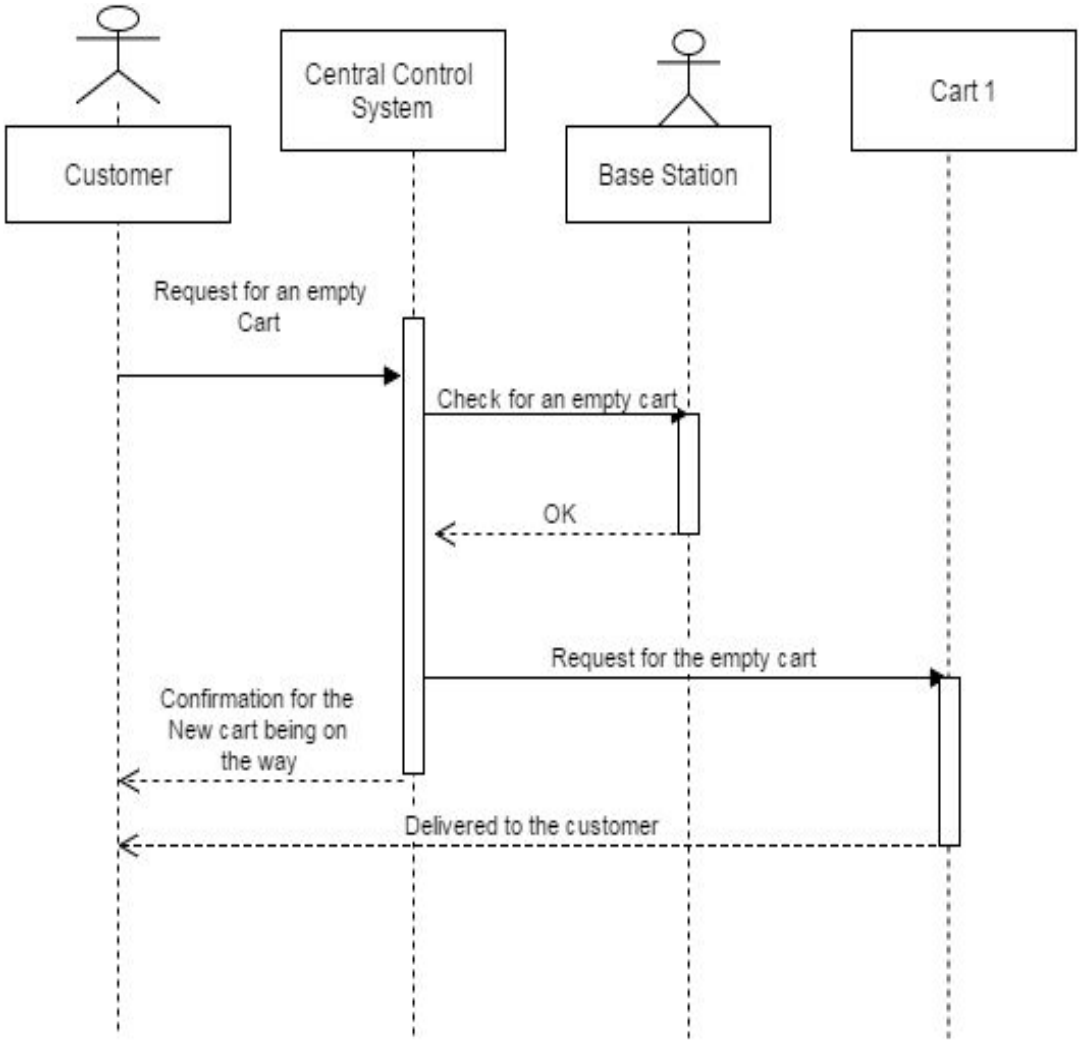
Management Plan

Function Classes and Relationships



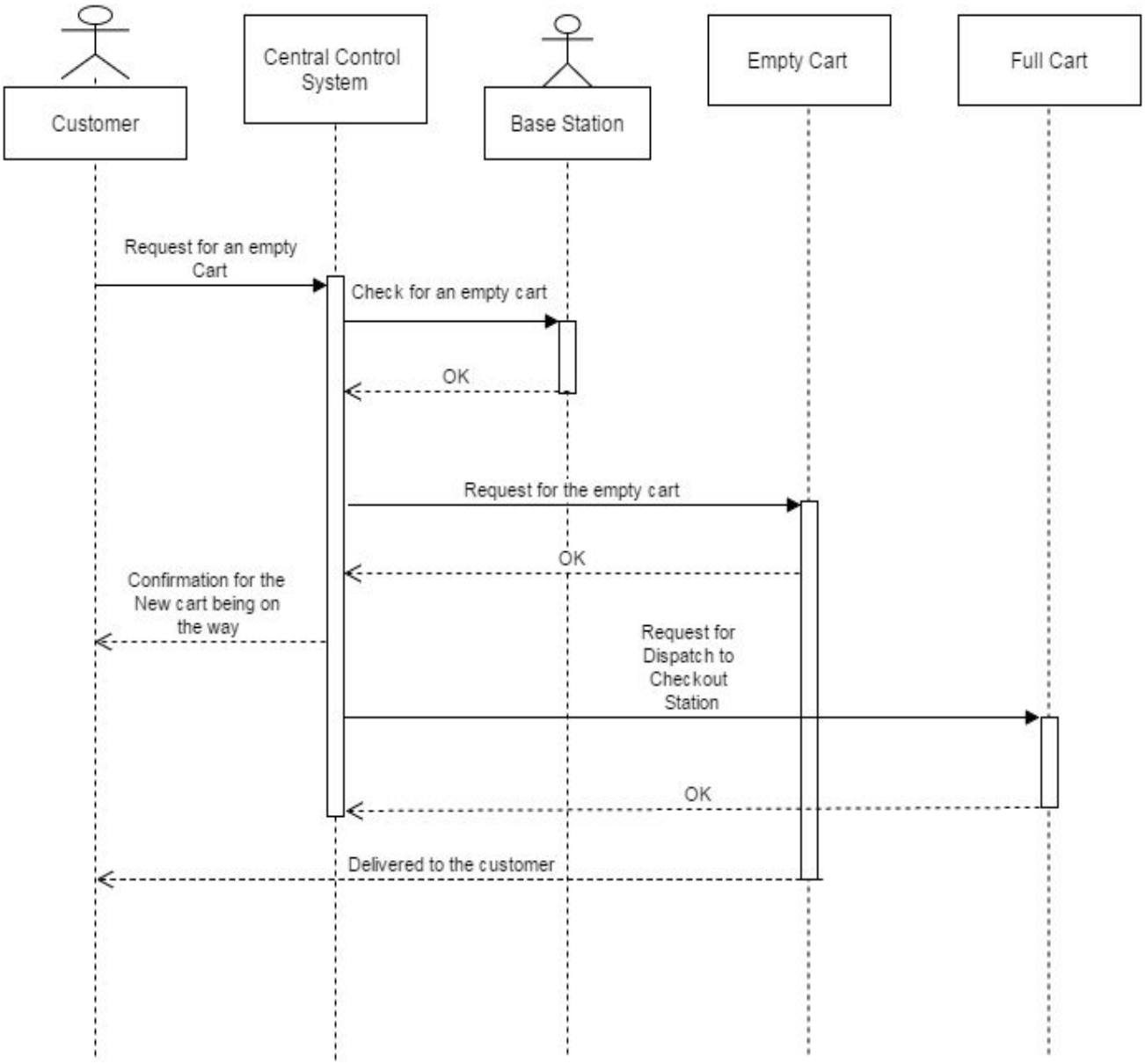
This is a rough function class relationship diagram. It describes how we would like to organize the system.

Sequence Diagram for an empty cart request



The customer requests for an empty cart from a particular location. The Central Control System gets the requests and checks if there are any empty cart in the base station. The base station returns an OK signal. The Central Control System requests an empty cart, and confirms the customer about the empty cart being on the way. In the meantime. the requested empty cart goes to the customer's location.

Sequence diagram for switching carts



The customer requests for an empty cart from a particular location. The Central Control System gets the requests and checks if there are any empty cart in the base station. The base station returns an OK signal. The Central Control System requests an empty cart, the empty cart sends an acknowledgement ; the Central Control System confirms the customer about the empty cart being on the way. The Central control requests the full cart to dispatch to the checkout station for the items to be checked out. The full cart sends a confirmation and heads to the checkout station. The empty cart arrives to the customer.

App

The mobile app is the primary interface for the user. The app receives input from the user to interact with the system.

Scanner

Within the mobile environment resides a scanner module. The scanner is activated by the user and physically placed near a receiving scanner node to trigger one of the three following events:

- Customer-cart interaction interface: this receiving node is on every shopping cart robot, when activated it initiates the command for the cart to go to checkout.
- Position node: this receiving node is placed at regular intervals in the store, when activated it initiates the command for an empty cart to go to the triggered node location.
- Checkout: this receiving node is present at checkout locations, when activated the customer's checked total is loaded and their full carts are prompted to arrive at the checkout station.

User

The user is interacting with the system using the mobile app. The users make decisions regarding when to dispatch and call for carts. Users have the option to shop without using the system.

Customer-Cart Interaction Interface

A module on each shopping cart receives scanner activation from a nearby smartphone running the system mobile app. Upon receiving the activation, the module receives a unique identifier from the customer and detects its location in the store. It sends the data it has acquired to the central control system and awaits direction for a path leading to checkout.

Position Node

Nodes located at regular intervals throughout the store, are activated with the scanner from a nearby smartphone. Upon receiving the activation a data package containing the nodes location in the store is sent to central control.

Robot Shopping Cart Maintenance System

The maintenance system is a storage facility that charges robot's batteries and performs other maintenance operations. Statuses, such as charge level, about robots currently in the facility is regularly sent to the central control system. The central control system directs robots in and out of the facility.

Central Control

Central control is a software system running on a computer in the store, it is initiated and configured by a store employee. The system coordinates events and processes within itself using received input and state information. Multiple modules directly interact with central control:

- Robot shopping cart maintenance system: send data to central control about the status of robots in maintenance. Central control decides when robots are sent and retrieved from maintenance based on contextual data about the store environment and system status.
- Customer Cart Interaction Interface: send data to central control when activated.
- Position Node: send data to central control when activated.

Pathfinding Algorithm

The pathfinding algorithm is a module inside central control, initiated when central control has received an activation signal from either a position node or a shopping cart robot. The algorithm determines a path that the requesting cart can take to safely reach its destination using contextual store and system status data. When the path is rendered it is downloaded to the requesting cart (if a position node is activated and an empty cart has been requested). After downloading the path data, the cart's driving system is activated.

Robot Shopping Cart Driving System

The driving system downloads directions and activation from central control. Once activated the driving system controls its motors and steering mechanism to drive the directions it has received.

Collision Prevention

Within the robot shopping cart driving system is a collision prevention module, it scan the surroundings of the shopping cart in real time detecting obstacles. If an obstacle is detected, the collision prevention system issues an emergency stop and requests new routes for the cart(s) to be generated to resolve the collision.

Checkout

The checkout is located near the exit of the store. Users must be physically present at the checkout station to interact with it. A scanner receiving node is present at each checkout station that the user can activate with the mobile app causing the checkout to load their checked total and owned carts data from central control. Data about the checkout event is sent to central control. Users pay for their goods using traditional methods.

Possible Implementations

Customer Features:

Customer-Cart Interaction Interface

Function 1: Interface needs to have a button customers can use to call carts to their location. There are two possible implementations. First, design a software button on the mobile app and the system will triangulate the user's location using a NFC communication system and send a cart to that location. The second method, we can place physical buttons around appointed locations in the store, and those locations would act as cart pick up locations.

Function 2: The carts can be claimed by customers and once claimed, the carts will remember their claimant's identity until discharged. Two possible implementations of this function: First, we can place QR code on the shopping cart. To claim the cart, the shopper scans the QR code with the camera on their smartphones, which syncs the cart to the mobile app. Second method is to build nfc chip on the handles of the shopping cart and the user can tap connect their phone to the cart.

Function 3: The customer can send the cart to the cashier for checkout. This feature can be implemented with a physical button on the cart, or a software button on the app.

Function 4: The user can un-claim the cart. This feature can be implemented with an unclaim button, a timeout system, or a proximity sensor, so if the customer is too far away from the store, the cart will become unclaimed. Once unclaimed, if the cart is empty, it will be set into a free roaming state, if there are items in the cart, it will be sent to an area to unload.

Robot shopping Cart Locating System

Function 1: Mobile app must have a compass that points the user to the location of the cart. This can be implemented with a graphical pointer on the app.

Function 2: When a customer owns multiple carts, each cart would have its own uniquely colored pointer.

Management Features:

Robot Shopping Cart Locating System

Function 1: Management system needs to determine the location of all carts in real time. One possible implementation is to triangulate each cart's location using NFC, and that information would be relayed to the central control management system.

Function 2: Carts need to be able to triangulate their own location. We can use NFC here as well.

Robot Shopping Cart Central Control

Function 1: Manage roaming free cart. One possible way to differentiate free cart and cart in use is with a owner tag in the program, so when a user claims a cart the ownership tag changes to "in use." When the cart is in storage, the owner tag changes to "waiting." If the cart does not have the "in use" or "waiting" tags, the cart is in free roam mode.

Function 2: Free roam carts must default to not in use or "waiting" when no users call for it. To implement this function, we can program all carts in free roam move toward the cart storage area.

Function 3: Carts in the "not in use" state will be deployed when there is no available "free roam" carts.

Function 4: When "in use," the customer should have full control of the cart. To implement this, we can build the interface app to communicate with Central Control, and Central control will relay the command to the cart. Or we can have a mini control center built into the app, and each mobile phone app would act as a temporary control independent of Central Control.

Function 5: The cart is not allowed to leave the perimeter of the shop. We can have a lock system where if a cart is too far away from the shop, the wheels would lock up and no longer able to move.

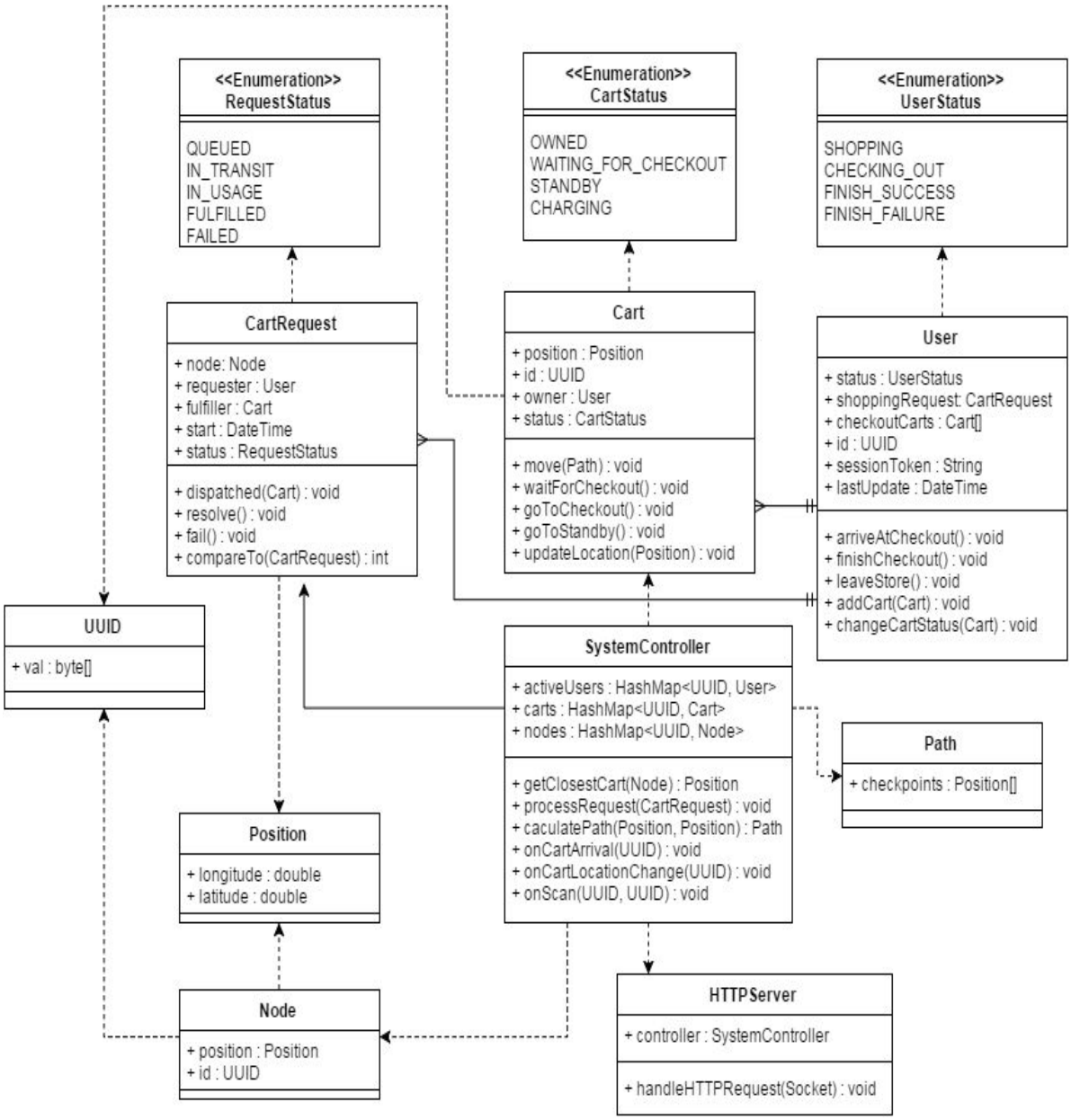
Robot Shopping Cart Pathfinding

Function 1: Application must be able to set lane maps for the cart to move on. One possible implementation is to have magnetic strips built into the floor, so the carts would snap to the magnetic grid when moving through the store. Another way is to have a digital map, and using NFC to locate and orient the cart into the direction they need to go based on the digital lane map.

Function 2: Shopping carts must be able to detect obstacles and avoid them. There are multiple ways to implement this features as well. One way is to have an ultrasound sensor for detecting obstacles, the other way is we can use a kinect type of webcam sensor. For obstacle avoidance, if the carts are on magnetic strips, we can simply stop the cart and restart the cart once obstacle is cleared. For free map, we can orient the cart and move around the obstacle.

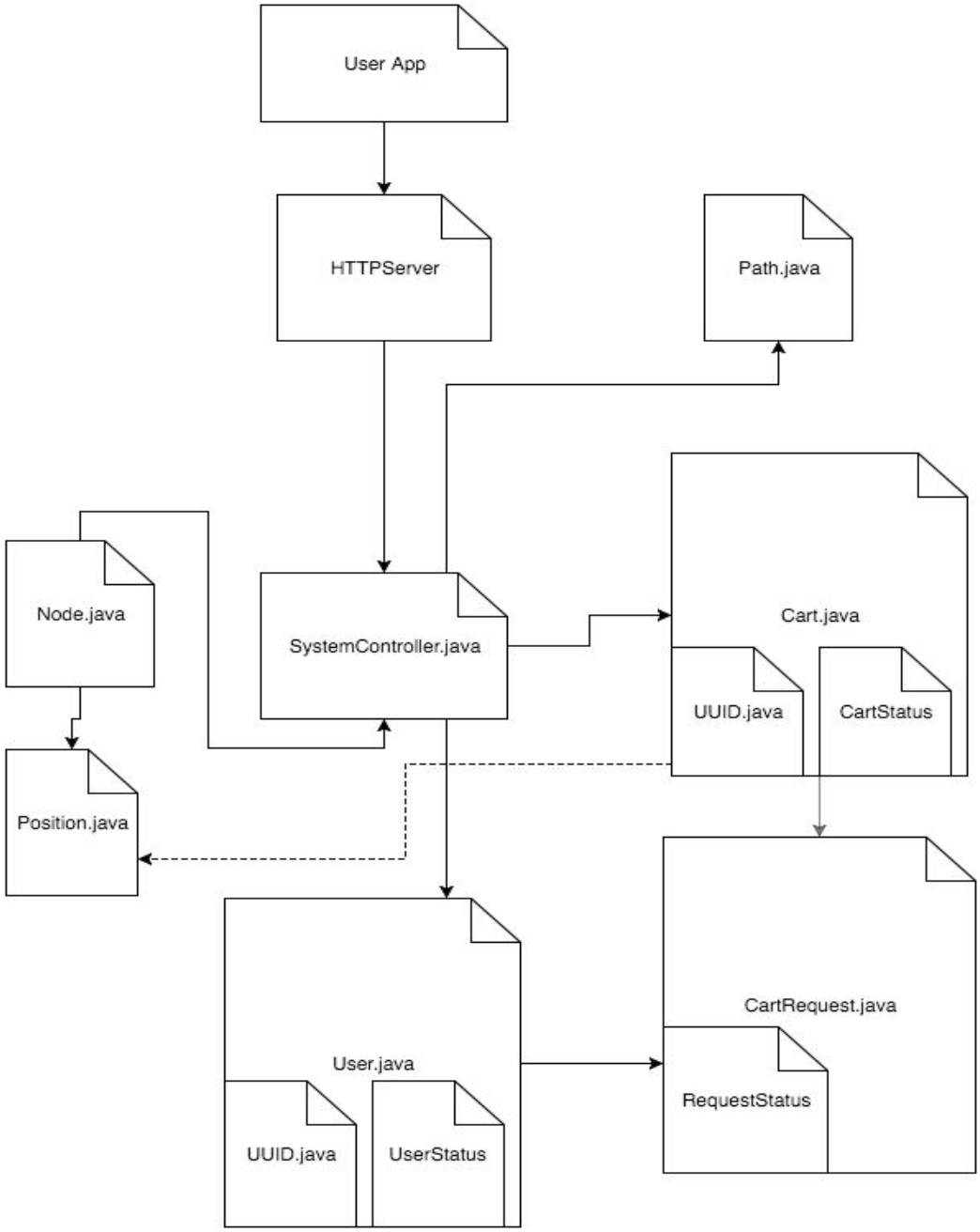
Function 3: Carts need to orient themselves. With magnetic strips built beneath the floor, the carts will automatically snap into place, so we don't need algorithm for cart orientation. We can also add two sensors on the the cart, one on the front one on the tail, then we can determine the orientation based on readings from both sensors.

Class Diagram



This is the class diagram. It outlines one possible way to organize our code. SystemController is the control Server for our application, and the user application communicates with the SystemController through the HTTPServer, where it sends requests and cart position information. All other objects such as User, CartRequest, and Cart are handled through the SystemController. Position and Node determines location of carts, and Path is calculated by the server and sent to the Carts.

Component Diagram



The above diagram describes the expected source files to be created for the final system. File displayed within other files correlate to the class diagram to show classes included on the class diagram that will not be separate source files in the final program. In addition, the phone will have a scanner which has a single function of making an HTTP request describing its scan to the store's base station. The cart, which is not in the scope of this project, is expected to make HTTP requests to the base station which will be handled by SystemController.java.

Minimal System

This section highlights the most barebone necessary functions needed to make our product useful. There are two main ideas. One is that the carts must move on their own, and the second is that the customer needs to be able to call a cart to his or her location. The following functions are absolutely necessary to realize the two ideas.

Customer-Cart Interaction Interface

Function 1: There must be button the users can use to call for a cart.

Robot Shopping Cart Locating System

Function 2: Carts need to be able to triangulate their own location for pathfinding and obstacle avoidance.

Robot Shopping Cart Path Finding

Function 1: Lane maps are needed for navigating the carts.
Function 2: Carts need to be able to detect obstacles.
Function 3: Carts need to be able to orient themselves for navigation and obstacle avoidance.

Central Control

Function 1: Manage roaming free cart.
Function 2: Free roam carts must default to not in use or "waiting" when no users call for it.
Function 3: Carts in the "not in use" state will be deployed when there is no available "free roam" carts.
Function 4: When "in use," the customer should have full control of the cart.
Function 5: The cart is not allowed to leave the perimeter of the shop.

Enhancements

The possible future implementations include giving the store management the ability to customize the store layout in the application. The store management will be able to change the locations of the speed bumps, any temporary obstacles, restricted area etc on the system easily from the 'custom store layout option'. Only the authorized users will be able to access the functionality. Moreover, user authorized for accessing and managing the layout for a particular store will will not be able to access other stores' information unless they have been exclusively given the authority to do so.

Additional feature in the long term is having the carts deliver the products bought at the store to the user's home, provided the user leaves within a certain radius from the store. Here, instead of sending the carts to the cashier, the user will be able to checkout the items in the cart itself. The cart will have a machine to authorize debit and credit card transactions. If the user is paying with cash, he/she has to go through the cashier to checkout the items. If the user lives outside the radius covered by the shopping cart delivery system, the cart will alert the user of the situation. The user can either choose to deliver the products to the car or to the nearest bus station.

Future implementation for finding the carts after they have been sent to the checkout station is under consideration. The customer will be able to track down the cart(s) that has the user's items for at the checkout station from a crowd of similar looking carts full of items

Possible enhancements for checkout stations in the future include implementing an interface to checkout and store the total price for a particular user so that when the customer comes back for the items payment, it can be done immediately without having to go through the checkout process at the station.

Technical Design

Data Flow

There is a single source from which data originates in the system. This source is the store's customer, and it happens when they swipe their phone or the leave the store to indicate a contextual action. The gesture of swiping the phone near pre-designated spots, namely nodes, signals to the system that an action, on the part of the system, is desired. This action will then be determined by the context in which the swipe occurred. For instance, if the swipe occurred in an aisle inside the store, the system will determine that the customer wishes to have an empty cart delivered to them.

This is completed by the phone contacting the central control system with the context of the swipe. That is, the person who swiped the phone, and where and what was the phone swiped against will allow the central control system to contextually determine the correct course of action to take. The following are the actions that the system may determine the user wishes to take.

1. The user has swiped their phone in an aisle, and so wished for an empty cart to be delivered to them. In this case, the flow of information will change based on whether or not that user currently has a cart in his or her possession.

   If the user does have a cart, the central control will send a signal to the user's cell phone prompting them to swipe their phone on their current cart to send it away, and store that the user has made the initial swipe to a nearby node. When the user swipes their phone on their current cart, the newest swipe is sent to the central control system. The central control system will then notice that the user has swiped both their cart and a location within the store, and will in turn send out two signal. The first signal will be sent to the cart currently in the user's possession, and tell the chart to move to a location arranged by store, and wait for checkout. The second signal will be to an empty cart, which will be instructed to move to the location of the user's initial swipe. Finally, the empty cart will be recognized as being the customer's current cart.

   If the user does not have cart, the central control system will detect this, and instead of sending a signal to the user's cell phone to request swiping of their full cart, it will instead immediately send a signal to an empty cart. The signal will request the cart to move to the customer's location. In addition, the empty cart will be recognized as being the customer's current cart.

2. The user has swiped their phone on their current cart. In this case, the central control system will receive the message from the user's cell phone, and send back a signal to the user's cell phone that prompts the user to swipe a nearby node. When the user completes this action, the flow of information will continue in the same way as

outlined above in point 1.

3. The user has swiped their phone at a checkout station. In this case, the central control system will receive the message from the user's phone, as in the other cases, and proceed to detect that the user wishes to begin the checkout process. At this point in time, the central control system will calculate a path for all the the user's carts that are waiting for checkout to the checkout station that the cart is at, and send a signal to the relevant carts to move to the checkout station. At this point in time, a store employee will take over and begin scanning items in the carts for the customer, and charge the customer based on the products in the carts.

4. The user has not performed any action that sends a signal to the central control system for 2 hours. In this case, the system will send a signal to the user to prompt them to scan anything. If the user fails to scan something in the allotted time, the central control system will automatically send a signal to the carts sending them to a checkout station where they will be unloaded, and their product returned to the shelves. Once they are unloaded, the carts are prompted by the central control system to return to the cart charging area for further instructions by the central control system.

   If the user is able to swipe something in the allotted time, sending a signal to the central control system, they will be determined to still be shopping. In this case, the internal timer in the central control system will be restarted, and no further action will be taken.

Pseudocode

HTTP Server

The HTTPServer class describes the object that handles communication between the user phone app and the central control. It handles functionalities such as listening for requests, and handling NFC or QR scans from customer mobile devices.

```
Class HTTPServer:
    Algorithm handleHTTPRequest:
          Input: Socket - The socket connection going to the
central control station coming from a customer or cart
          Output: None

          loop forever
               request <- Socket.readRequest()

               if request.type is a phone scan request
```

```
                    if request.scanned_uuid is a uuid belonging
to a Node
                         if request.customer already has a cart
                              if request.customer approves
dismissing their current cart

request.customer.current_cart.waitForCheckout()

SystemController.processRequest( request.cart_request() )
                         else
                              SystemController.processRequest(
request.cart_request() )
                    else if request.scanned_uuid is a uuid
belonging to a Cart

request.customer.current_cart.waitForCheckout()
                    else if request.scanned_uuid is a uuid
belonging to Checkout
                         for each cart in request.customer.carts
                              cart.goToCheckout()

               else if request.type is a cart update request
                    cart <- cart with uuid request.scanned_uuid
                    update cart with request
                    if cart path is blocked?
                         cart.move(
SystemController.calcualtePath(request.start, request.stop) )
```

SystemController

The SystemController class is an object that describes Central Control in our system. It handles functionalities such as sending cart's to a designated location and processing requests sent in by customers. For example the function getClosestCart locates the cart closest to the user who sent a request for carts. The algorithm processRequest handles requests from users and sends cart to the requester; it does this by first obtaining the closest cart with the getClosestCart function and then it calculates a path with the calculatePath function and then it sends the pathing to the closest cart so the cart has directions to move toward the customer. The following is a complete list of functions in the SystemController class.

```
class SystemController:
     Algorithm getClosestCart:
          Input: Node - The node to get the closest cart to.
          Output: Cart - The closest cart to the node
```

```
            position <- Node.position
            min_distance <- Infinity
            min_cart <- null

            for each cart in the system
                 if distance between Node and cart < min_distance
                      min_distance <- distance between Node and
cart
                      min_cart <- cart

            Cart <- min_cart

      Algorithm processRequest:
            Input: CartRequest - The cart request to processRequest
            Output: None

            cart <-
SystemController.getClosestCart(CartRequest.node)
            path <- SystemController.calculatePath(cart.position,
CartRequest.node.position)

            cart.move(path)
            cart.status <- CartStatus.OWNED
            CartRequest.dispatched(cart)

      Algorithm calcualtePath:
            Input: Position start - The start position for the path
                   Position end - The end position for the path
            Output: Position[] path - The path to follow to get from
start to end.

            vertices <- vertices in the graph describing store
layout
            path <- empty array

            for every vertex in vertices that is in the smallest bfs
tree that connects the start position to the end position
                 path.append(vertex)

      Algorithm onCartArrival:
          Input: UUID - The uuid of the cart that arrived
          Output: None

          cart <- cart which owns the uuid UUID
          User.addCart(cart)
```

```
       Algorithm onScan:
          Input: UUID scannerUUID - The uuid of the customer.
                 UUID scannedUUID - The uuid of the scanned uuid
          Output: None

          scanner <- The customer which owns the scannerUUID uuid.
          scanned <- The node or cart which owns the scannedUUID
uuid.

          if scanned is a Node not at a checkout station
              CartRequest <- new CartRequest
              scanned.shoppingRequest <- CartRequest
          else if scanned is a Cart
              scanned.waitForCheckout()
          else if scanned is a Node at a checkout station
              scanner.arriveAtCheckout()
```

User

The user class is an object that describes customers in our system. It handles functionalities such as describing user status. For example, addCart is perhaps the most important function because it allows carts to be attached to a customer in the store. AddCart appends a Cart object to the user's list of carts owned. ArriveAtCheckout function marks user status as CHECKING_OUT when user arrive at the checkout station and orders all user carts to go there as well. FinishCheckout function marks a transaction successful if the user finishes checkout and pays for the items in the cart. LeaveStore marks a user as left the store and puts all the carts owned by the user on standby. Below is a complete list of all the functions contained in the User class.

```
class User:
     Algorithm arriveAtCheckout:
          Input: User - The user which arrived at checkout.
          Output: None

          for each cart in User.checkoutCarts
               cart.goToCheckout()

          User.status <- UserStatus.CHECKING_OUT
          User.lastUpdate <- now

     Algorithm finishCheckout:
          Input: User - The user who finished checkout.
          Output: None
```

```
            User.stats <- UserStatus.FINSIH_SUCCESS

      Algorithm leaveStore:
            Input: User - The user who left the store.
            Output: None

            for cart in User.carts
                  cart.goToStandby()

      Algorithm addCart:
          Input: User - The user to add the cart to
                 Cart - The cart to add to the user
                 Node - The node which the user has scanned
         Output: None

         User.current_cart.waitForCheckout()
         User.carts.append(User.current_cart)

         path <- SystemController.calcualtePath(Cart.position,
Node.position)
         Cart.move(path)
         changeCartStatus(User, Cart)

     Algorithm changeCartStatus:
         Input: User - The user which now owns the cart
                Cart - The cart to change the status of
         Output: None

         Cart.status = CartStatus.OWNED

CartRequest is a class that identifies
```

Cart Request

The cart request class is an object that describes requests from customers. It handles functionalities such as describing request type and request status. For example, the resolve function changes the cart status based on the situation. This class also has a compareTo function which prioritizes cart requests. This is so that requests are sortable and we can make a priority queue to order the list of requests to be handled by the SystemController class. Below is a complete list of functions in the CartRequest class.

```
Class CartRequest:
      Variable node Node
      Variable requester User
```

```
Variable fufiller Cart
Variable start DateTime
Variable status RequestStatus

Algorithm dispatched:
      Input: Cart - the cart that will be dispatched
      Output: None

      status = IN_TRANSIT
      fufiller = Input: Cart
      start = current_time


Algorithm resolve
      Input: None
      Output: None

      if fufiller.owner not equal to null
            status = IN_USE

      else if cartlist in SystemCotroller is empty
            status = IN_QUEUE

      else if node not equal to null
            status = IN_TRANSIT
      else if user unclaim cart
            status = FULFILLED
      else
            status = FAILED


Algorithm fail
      Input: None
      Output: None

      if current time - start > 30 minutes
            status = FAILED


*compareTo compares two cartRequests and gives them priority
Algorithm compareTo
      Input CartRequest
      Output int

      if this.start < other.start
            return -1
```

```
            else
                  Return 1
```

Cart

The cart class is the code that runs on the microcontroller located inside the shopping cart robot. It handles receiving controls from central control and initiates those actions. The code runs the collision detection system on a separate thread so that possible collisions can be constantly monitored while still executing useful commands. The following is pseudocode for the cart class.

```
Class cart:
      enum IDLE 0
      enum DRIVE 1

      static Class drive:
            void driveTo(int position)
                  controlDrive(parsePosition(position));
            void stop()
                  controlDrive(0);

      int recieveControlData()
            return controlcode;

      int[] recieveDriveData()
            return drivepath;

      void sendPositionData()
            sendData(nearestnode);

      void detectCollision()
            while (true)
                  if (collisionDetected())
                        throw new collisionDetectedException();

      void sendCollisionData(Exception e)
            sendData(e.state);

      void drive(int[] directions)
            for (nextNode in directions)
                  drive.driveTo(nextNode);
                  sendPositionData();

      int main()
            while(true)
```

```
                try
                      Thread.new(detectCollision())
                      while (true)
                            int control = recieveControlData();
                            if (control == DRIVE)
                                  int[] directions =
recieveDriveData();
                                  drive(directions);
                catch collisionDetectedException e
                      drive.stop();
                      sendCollisionData(e);
```

App

The app class is the code that runs as a mobile application on a smartphone. It receives user input and communicates to the central server. It also draws the user interface according to current state and inputs. Status screens and progress bars are outputted to the user. The following is pseudocode for the App class.

```
Class app:
      Frame homescreen = homescreen()
      Frame scanner = scanner()
      Frame success = success()
      Frame failure = failure()
      Frame currentframe
      Stack<Frame> framestack

      Class homescreen extends Frame:
            bool connectionStatus
            Button scan

            onActivate()
                  self.paintFrame()
                  connectionStatus = linkToServer()

            onScanButtonSelect()
                  if connectionStatus
                        framestack.push(self)
                        currentframe = scanner
                  else
                        displayErrorMessage()

      Class scanner extends Frame:

            onActivate()
```

```
            self.paintFrame()
            for (30 seconds)
                  Data scanned = scanForNode()
                  if scanned
                        break

            sendData(scanned)
            if not scanned
                  currentframe = failure
            else
                  currentframe = success

Class success()

      onActivate()
            self.paintFrame()
            bool complete = false
            while not complete
                  status = getStatusData()
                  displayProgressBar(status)
                  complete = status.isComplete()

Class failure()

      onActivate()
            self.paintFrame()
            displayErrorMessage()


onCreate()
      currentframe = homescreen
      framestack.push(homescreen)
      while (true)
            activate(currentframe)

onBackButton()
      terminate(currentframe)
      currentframe = framestack.pop()
      if not currentframe
            closeApp()
```

Test Plan

In this section, we describe how we plan to test our product once a prototype is finished. The Test Plan Summary gives an overall summary of this entire section. Proposed Date for Submission provides a rough timeline of our plan for completion as well as responsibilities of each team member. And Tests describes each item in a list of tests that we think are necessary.

Test Plan Summary

The Servr will have most of its prototype feature tested for validation and verification. The set of tests include the correctness/ shortest path test, cart request reliability test, cart drive performance test, System functionality test, Acceptance test, End-to-end cart request test, unit test for user credentials input, unit test for user confirmation, unit test for putting user on a queue, and the integration test for sending the cart to the user.

Test scripts are written by Brandon Mabey, Tania Akter, Tal Melamed and Haodong Tao for each of the test cases. The reports are exported into a document and later analysed to provide the rate of failure/pass. The 99.8% has to pass. The report generated will include the methods they were carried out with, the test scripts used for each of the test cases, and any anomalies encountered. Each members of the team are assigned at least two test features, and the test report will be submitted before the end of the school semester. The test names , assignee and tentative date for report submission has been summarized below.

A bug tracking software has been used for assigning the task to each of the members. The software is synced with the development and the testing machines for each of the members and notifies the user about the remaining task and the deadline for each of the stages. Once the task is done, the user can cross it off the assigned test list and submit the report generated from the test.

Proposed dates for submission:

The following is a table of tests that are necessary for our project. In the table are the test names, the team member responsible for that test, and a rough timeline for completion.

Table: Tests and Completion Date

Test name	**Assigned to**	**Tentative date for report submission**
Correctness/ shortest path test	Haodong Tao	March 24th, 2016
Cart request reliability test	Tal Melamed	March 24th, 2016
Cart drive performance test	Brandon Mabey	March 26th, 2016
System functionality test	Tania Akter	March 26th, 2016
Acceptance test	Brandon Mabey	April 1st, 2016
End-to-end cart request test	Tal Melamed	March 27th, 2016
Unit test for user credentials input	Haodong Tao	March 28th, 2016
Unit test for user confirmation	Tal Melamed	March 28th, 2016
Unit test for putting user on a queue	Brandon Mabey	March 28th, 2016
Integration test for sending the cart to the user	Tania Akter	April 1st, 2016

Test Plan Details

Correctness/ Shortest Path Test

The objective of this test is to ensure the central control system produces the desired (shortest path for a particular node position) path for any position nodes within a store layout. The test also ensures that the time taken for generating each of the path is within a certain limit to make the system look instantaneous.

Test for the shortest measured path provided by the central control system can be tested by requesting cart from each positions. The action will be stimulated by sending a request to the server with different position values. The expected shortest route can be found by computing an optimal right angle paths. The generated path is then compared to the expected shortest route, and returned pass or fail depending on if the generated shortest path by the system is less than or equal to the expected path for each of them.

Moreover, the generated path is run through an algorithm to to check if it ever deviates from the target, provided there are no obstacles on the way. The output generates a pass or fail depending on if the route generated path moves towards the target efficiently.

The unit test is done multiple times and at least 99.8% of the test cases should pass in order or the unit tests to be successful.

For the integration test, the time taken for generating the right path is recorded for each of the position nodes. The time taken for each cases should be less than or equal to 0.1s for the test to pass. The system should not fail in any way during the process. At least 99.8% of the test cases has to pass for the system to be considered usable.

The test has been assigned to Haodon Tao. The report submission for the test is due by March 24th, 2016.

Cart request reliability Test

The objective of the test is to ensure the cart does not function outside the system specification. The test ensures that the cart itself is executing according to the instructions it downloads from the central control system. The test also test if the central control is producing the path depending on the correct position of request.

Test for the cart request reliability has one successful outcome - the cart reaches the customer successfully. The cart request will be stimulated by sending sending a request to the server with different position nodes. An empty cart should be then dispatched from the system, and has to go to the requested position successfully.

The test is passed if the the cart reaches the requested node position without any anomalies, fails otherwise. At least 100% of the unit test has to pass.

A performance test and integration test can be carried out by sending 1000 requests simultaneously and recording the number of test the carts have passed. This test is done in order to check if the excess load on the system makes the central control system produce wrong paths or wrong destination point. After stimulation, check if all the carts reaches its designated position of request.

The tests are done multiple times at least 99.8% of the test has to pass in order for the feature to be accepted.

The test has been assigned to Tal Melamed. The report submission for the test is due by March 24th. 2016.

Cart Drive Performance Test

The objective of the test is to ensure the system reaches the customer at a reasonable time without the customer having to wait for a long time. The test ensures that the carts do not slow down while executing the commands. It also ensures that the carts are moving at 1m/s to ensure public safety as well as fast delivery of the cart.

Request for empty carts are stimulated by sending a request to the server with different position nodes. From the shortest distance found, the estimated time taken is calculated using the formula estimated time taken = total distance in meters/ speed(1m/s) . The time taken for the empty cart to successfully reach the destination is recorded. The recorded time should be within a standards deviation of 30 seconds. The test passes if the recorded time is within the range for a set distance and fails otherwise.

The test is carried out 100 times from the same position node , and repeated with all different position nodes within the store. All the test has to pass in order for the feature to be acceptable.

The test has been assigned to Brandon Mabey. The report on the test is due by March 26th, 2016 for submission.

User Interface System functionality test

The objective of the test is to ensure the system meets user expectation for reliability and robustness. The System should be functional at all times when interacting with the users.

The user app must not crash from any input or action from the user within the domain of the system. All expected domain interactions must be accounted for and tested working for every customer build of the system. Non-domain interactions (outside of store, Android crashes, hacking, etc) are not covered by the test.

Specific unit tests will run for each type of user input and functionality. The inputs are tested blackbox. All tests must pass.

The test has been assigned to Tania Akter. The report submission is due by March 26th, 2016.

End-to-end data integrity test:

The objective of the test is to ensure data transfer is successful and correct when transported through data transfer infrastructure. All system data endpoints are tested, app to central control, node to app, cart to central control, node to cart, central control to app, central control to cart. Data transfer is done through either a HTTP server protocol or a near field/local data transfer scheme.

Test data will be emitted at a source endpoint and read at a receiving endpoint. Data reception success and correctness is recorded. The test data will vary from commonly transferred data to edge cases and stress conditions. The endpoint protocols are expected to attempt to transfer the data robustly, reattempting the transfer and running error correction until success or eventual timeout.

The data transfer must be successful (not timing out) for all data transmission in the system during a healthy state of infrastructure.

The responsibility for carrying out the acceptance test has been assigned to Tal Melamed. The report on the test results is due by March 27th, 2016.

Unit test for user the credentials input

The objective of the test is to ensure that the proper security stemming from maintaining a user authentication system is being sustained. The system should ensure that users are unable to misidentify themselves in any way, as to not compromise any information the system may have recorded on any particular user.

The test will comprise of running on a system which contains a few pre-registered users in the system's data.

Testing will comprise of attempting to access the login restricted areas of the system using invalid credentials, and expecting that the system denies access. In addition valid crediations will be inputted into the system, and the system must respond by allowing the user to access the login protected areas, with data that is relevant to their account. Allowing access with data from other user's accounts will be considered a failure of this test.

The responsibility for carrying out the acceptance test has been assigned to Hadodong Tao. The report on the test results is due by March 28th, 2016.

Unit test for user confirmation

The objective of this test is to ensure the system gives the appropriate response in between interactions. The test also confirms that the user gets notifications if there is any anomalies.

A request for an empty cart is stimulated by sending request to the server (the central control system). If there is an the system should check if there is any empty cart found from the base station. The system confirms has to confirm the user that a cart has been found. If not, the system must notify the user that there is no available cart at the base station, and that the user has been put in a queue for an empty cart. When the cart dispatches from the base station to the user's location, the system notifies the user that an empty cart is on the way.

A test with normal load of 25 requests per minute. A stress test is done by requesting 400 carts at the same time.

The test passes if the system notifies about the availability of the cart, or about being put in a queue if there is no empty cart found at the moment, and notifies the user when an empty cart is on the way. 100 of the test has to pass in order for the feature to be acceptable. The test does not consider the case when user's phone is not operable or dead.

The responsibility for carrying out the acceptance test has been assigned to Tal Melamed. The report on the test results is due by March 28th, 2016.

Out of carts system state Test

The objective of the test is to ensure the system behaves appropriately when entering the out of carts state. The system enters that state when no more empty carts are available to be issued to customers.

If carts run out, the system should immediately notice and enter the out of carts state. In this state all cart requests are put into a queue and fulfilled in order of reception when an available cart appears. Users are notified in the app if their cart request is pending and their position in the queue.

The test checks if the out of cart state actions are performed correctly and in a timely manner. The system must seamlessly enter and exit the out of cart state.

The responsibility for carrying out the acceptance test has been assigned to Brandon Mabey. The report on the test results is due by March 28th, 2016.

Integration test for the cart being assigned to a particular user

The objective of this test is to ensure that a user is properly given temporary "ownership" of a particular cart during the execution of the system while the user is shopping within the confines of the store.

The test will consist of a user who has already been granted access to the system by the login system swiping a cart at various nodes connected to the system network under various conditions. These conditions include the cases when the user has no cart, the user has a single cart, and the user has multiple carts that are waiting for checkout, in addition to having a cart they are pushing around. In addition, the situation caused by the user completing a checkout, both successful and unsuccessful will be test.

The expected result is that the system in the non-checkout cases is that a new cart is assigned to the user, which is checked by checking the stored information inside the database, and that all previous carts that the user may own are assigned the used, albeit not as the primary cart. In the case of a checkout, either successful or unsuccessful, the carts that were owned by user will be marked as being unowned.

The responsibility for carrying out the acceptance test has been assigned to Tania Akter. The report on the test results is due by March 28th, 2016.

Acceptance test

The project prototype is presented to the customer along with the requirements document. The customer evaluates the prototype and decides on the correctness based on their vision for the product.

General customer satisfaction is necessary to pass the test. The test is a milestone that stops further progress if failed. If failed, the prototype and requirements are updated using customer input and the test is repeated. Repeat until success.

The responsibility for carrying out the acceptance test has been assigned to Brandon Mabey. The report on the test results is due by April 1st, 2016.

Conclusion

The proposed project, Servr, focuses on benefiting both the departmental stores and their customers. The project enhances the customer experience in shopping by providing them the option to shop for more items which in turn increases the revenue of the store. The system consists of a mobile app, a central control system, and shopping cart robots. The control system will track the location of all the robots in the system in real time. The customer can request an empty cart to his or her location. The control system will determine a path with avoiding collision with the other carts and the stationary objects, then dispatch an available empty shopping cart to the requested location. The user can tag a full cart with their identity and send it to the cashier for checkout.

The possible implementations include the customer-cart interaction interface, robot shopping cart locating system. The future implementations include providing the store management the ability to modify the store layout at any time in the system. The process requires authorization for accessing the feature for each store. Another feature includes the shopping cart delivery system to the customer's home location within a certain radius. Future implementation of the locating the cart at the checkout station is under consideration.

Appendix A

Ethics

Due to the system having access to potentially personal information, we want to document that Almost Games is committed to developing the system in a way that complies with APEGBC Code of Ethics. The system will never store information beyond what is required for the system to operate, and in addition, will never know the location of any user or track the location of any users. All actions that require a specific location will be initiated by the user, and used for the sole purpose of improving the shopping experience in both efficiency and convenience for said user.

Modifications

1.1

Changed checkout system from simple storage to check items as they arrive to checkout. Added to enhancements section.

Added to “needs” scan location identifiers within store.

Fixed typos.

Specified payment methods.

Consistency added throughout to reflect changes.

Added new route calculation as a solution to blockages.

Added diagrams

1.2

Added user sign in to mobile application

Glossary

Almost Games	The company that is researching, creating ,and developing the Servr system.
NFC	Stands for Near Field Communication. It is a communication protocol which allows for electronic devices to communicate wirelessly with each other while in close proximity.
Nodes	Devices which passively transmit/receive a signal.

Team

Brandon Mabey	The lead web developer for Almost Games. He is in his third year in his Software Engineering degree at the University of Victoria, and a member of the 1st place team in division II of the 2015 ACM-ICPC Pacific Northwest regionals.
Tania Akter	The analyst, editor and the lead software collaborator for Almost games. She manages the team conflicts at times. She is in third year Software Engineering degree at University of Victoria. She has worked with IBM for developing analytical software for providing visual statistics that are used for vital business decisions.
Haodong Tao	The lead product analyst and design architect. He is in his third/fourth year in the combined major of Computer Science and Health Information Science. He as worked on many projects before in classes like CSC375, which is systems analysis, and his experience will make this team awesome.
Tal Melamed	The lead solutions and design architect. He is in his third year(sort of) of a software engineering degree at the University of Victoria. He has worked at Intel developing industry leading solid state drives.