Part II (The Logical Model)

Part I demonstrated how to create a use case model from problem statements describing an everyday business activity. This document demonstrates how the use case model is converted to system requirements and a potential high-level design.

The use cases are analyzed in order to determine the data being impacted by the use case model. Data is modeled using classes and the relationships between them. Classes are assigned functionality that is described in the use case model. The functionality is captured as class operations, which in turn can be used to derive functional system requirements.

The completed class diagram forms a potential Platform Independent Model (PIM) of the system which may be used as a high-level system design.

This article discusses the requirements for the Shopping Cart System that was described in Part I.

1 Architecture and Use Cases Summary

Part I included a system architecture and a set of use cases for the new Shopping Cart System. These are summarized below.

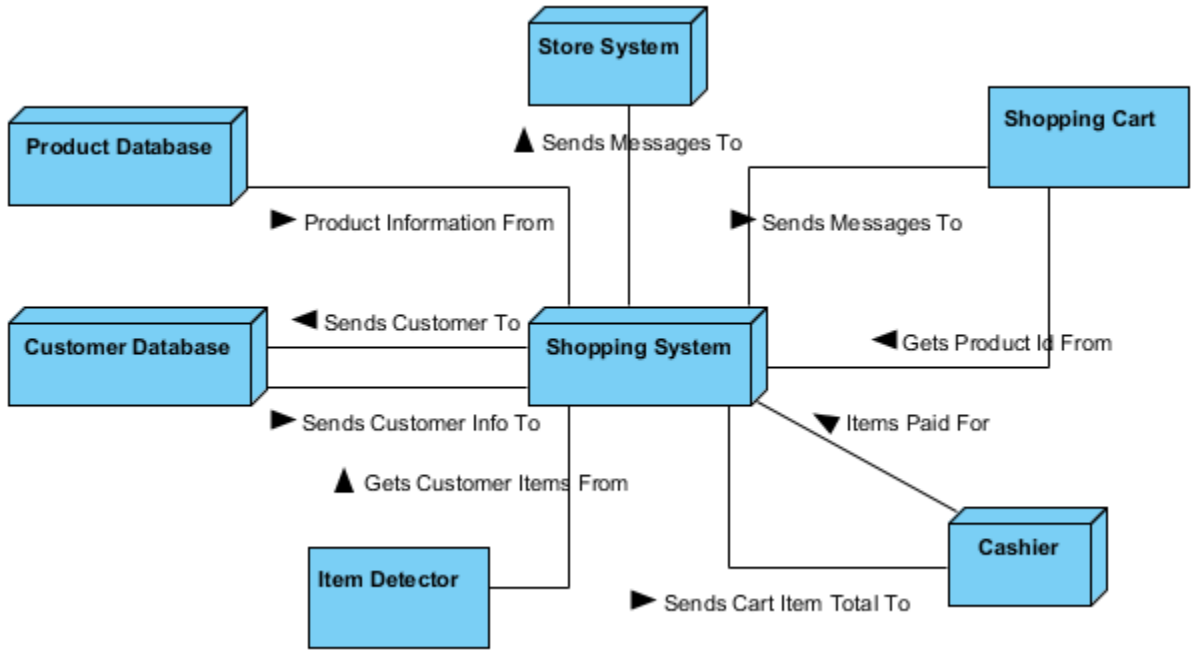

Figure 1: Shopping System Architecture Diagram

In Figure 1: the shopping system communicates with the following systems:

- Store System – to send messages to the store clerk.
- Product Database – to obtain information about store products.
- Shopping Cart – to send messages to the shopper.
- Shopping Cart – to receive item information.
- Customer Database – to communicate information about the shopper.
- Item Detector – to receive information about items leaving the store.
- Cashier – to send information about items in the shopping cart.
- Cashier – to receive information about items that have been paid for.

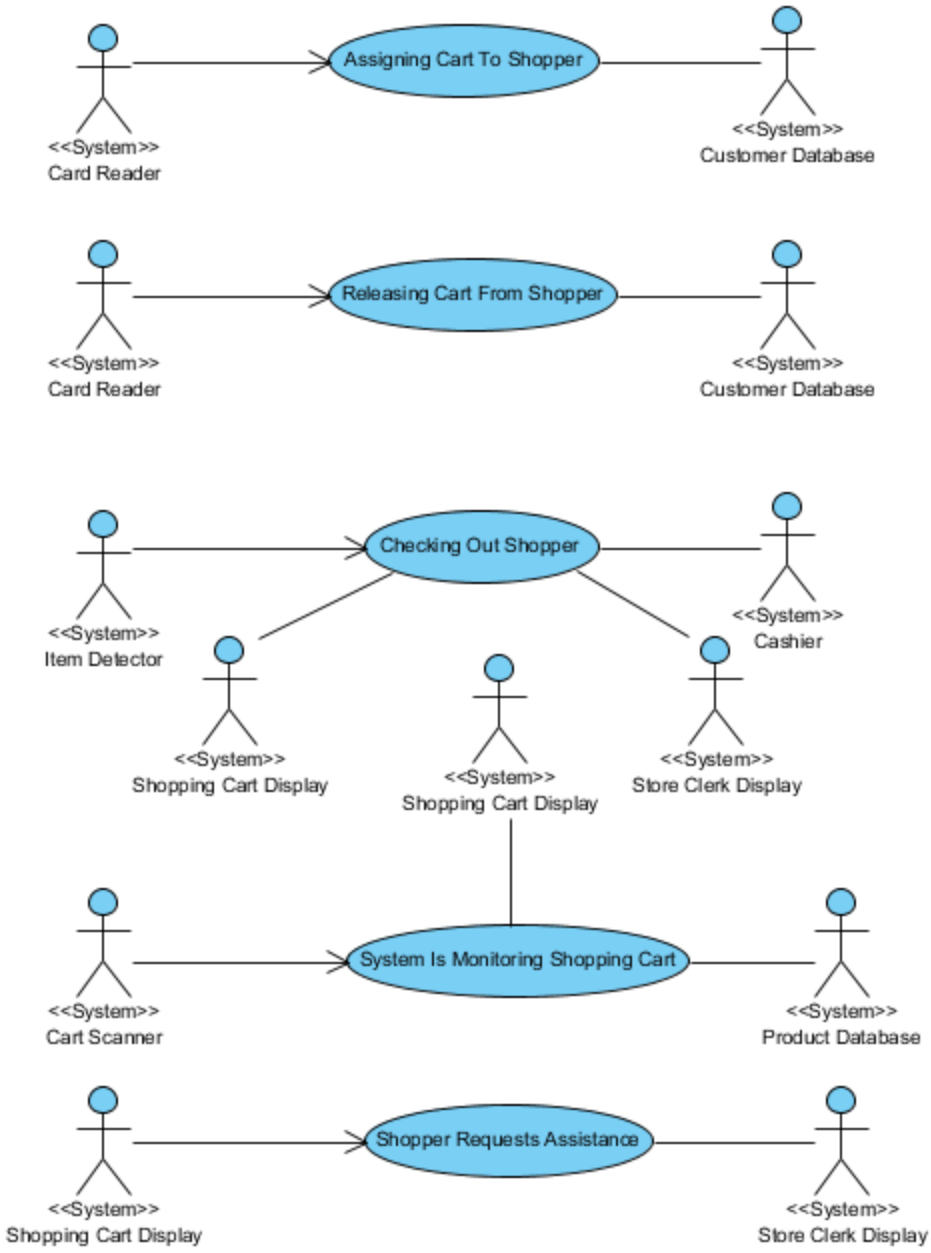

Figure 2: Shopping Cart System Use Cases

In Figure 2: the use cases for the Shopping System are concerned with:

- Assigning Cart To Shopper – allows a customer to start their shopping experience.
- Releasing Cart From Shopper – allows the customer to complete their shopping experience.
- Checking Out Shopper – allows the customer to remove items from the store.
- Monitoring Shopping Cart – ensures that the shopping experience goes as planned.
- Shopper Requests Assistance – allows a customer to to get help from the store.

2 The Logical Model

A business analyst may think that the analysis has been completed in Part I (Use Case Model). But systems analyst should ask the question, 'How do you know that it is going to work?' Up to this point we have speculated on what appears to be an acceptable architectural solution to the problem, but we have not yet researched the amount of actual work, required resources or external restrictions (regulatory, technology, etc.[1]), that may impact the solution.

The use case model shows the functions that are to be implemented by systems, but it does not discuss data and storage requirements, nor timings and communication between systems to which the solution is deployed.

The next step in the process is to build a logical (or implementation independent) model of the use cases. This will ensure that we have captured all the alternative flows and prove that there are no inconsistencies, gaps or contradictions in the use cases. It will also allow us to derive a complete set of consistent and feasible requirements that include storage and processing estimates.

2.1 Identify Objects

The logical model can be derived from the use case model by adding potential[2] objects (instances of classes), to actions in order to determine the information that is handled by each use case.

[This example only considers the (new) Shopping System. Similar models may be created for enhancements to the other systems.]

2.1.1 Shopping System Use Cases

The following objects are assigned to the use case actions for the Shopping System.

2.1.1.1 Assigning Cart To Customer

[1] Neither does this article discuss these.

[2] Potential – because as the logical model is built and formalized, the final classes may look very different.

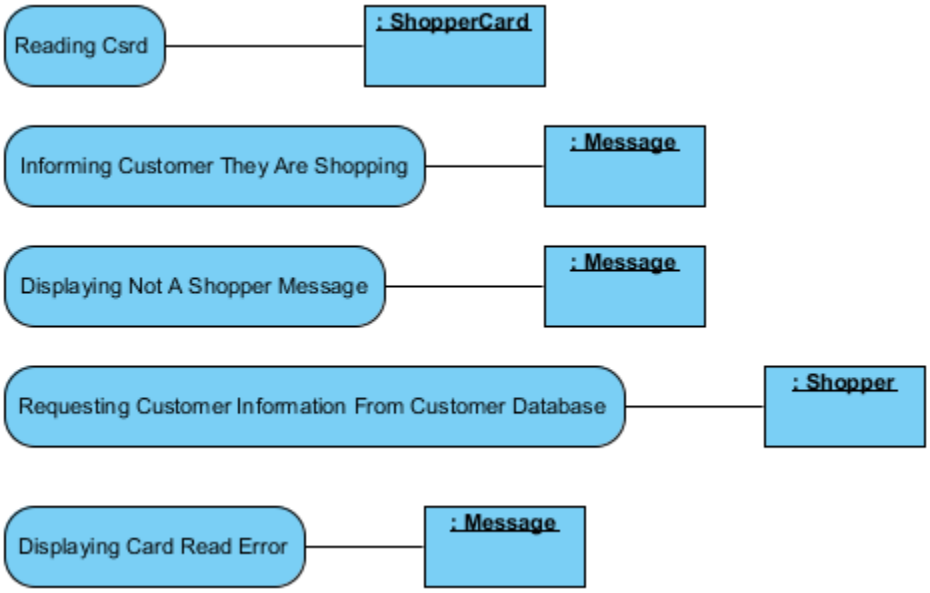

Figure 3: Assigning Cart To Customer Object Mapping

Figure 3: shows:

Reading Card – Creates a system object named ShopperCard.

Informing Customer They Are Shopping – Creates a Message to the customer.

Displaying Not A Shopper Message - Creates a Message to the customer.

Requesting Customer Information From Customer Database – Creates a Shopper object.

Displaying Card Read Error - Creates a Message to the customer.

2.1.1.2 Checking Out Shopper

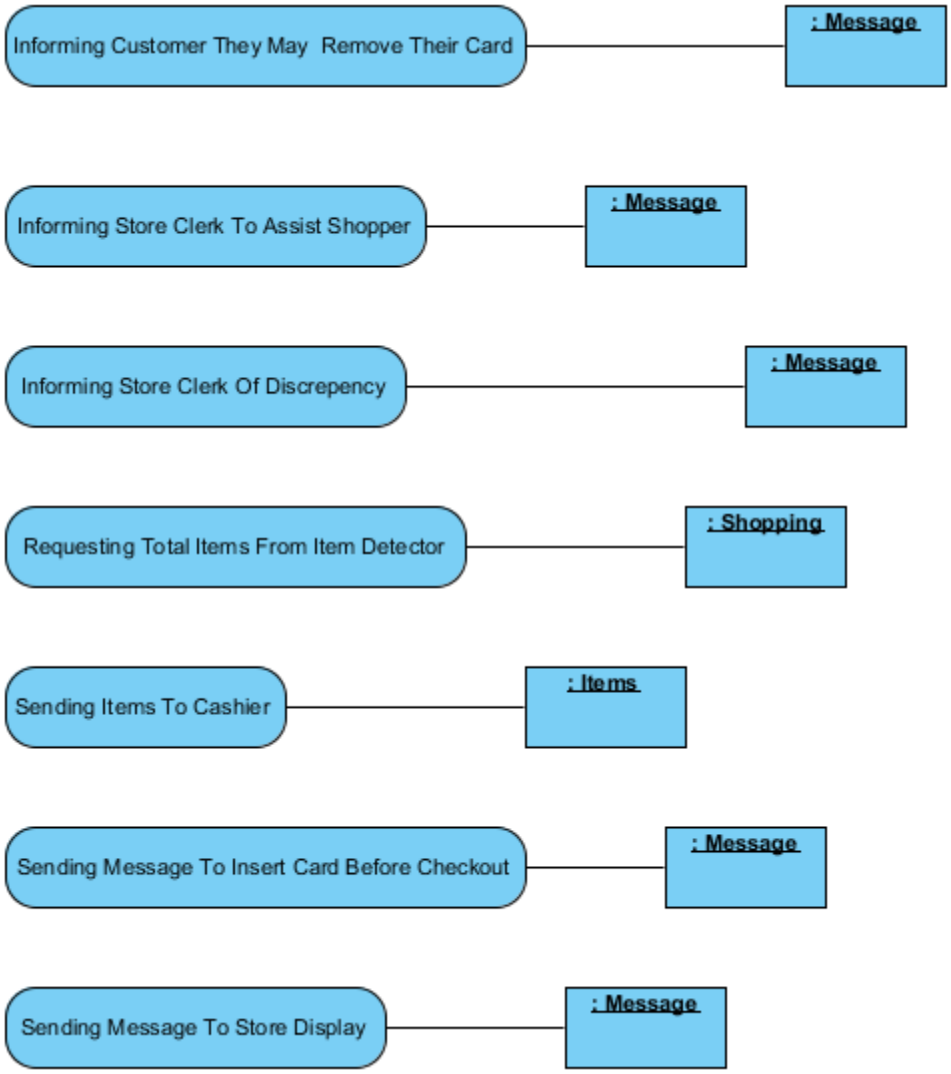

Figure 4: Checking Out Shopper Object Mapping

Figure 4: shows:

Informing Customer They May Remove Their Card – Creates a Message to the customer.

Informing Store Clerk To Assist Shopper – Creates a Message to the store clerk.

Informing Store Clerk Of Discrepancy – Creates a Message to the store clerk.

Requesting Total Items From Item Detector – Updates the Shopping object to verify the cart contents.

Sending Items To Cashier – Creates a list of Items object.

Sending Message To Insert Card – Creates a Message to the customer.

Sending Message To Store Display - Creates a Message to the store clerk.

2.1.1.3 Releasing Cart From Shopper

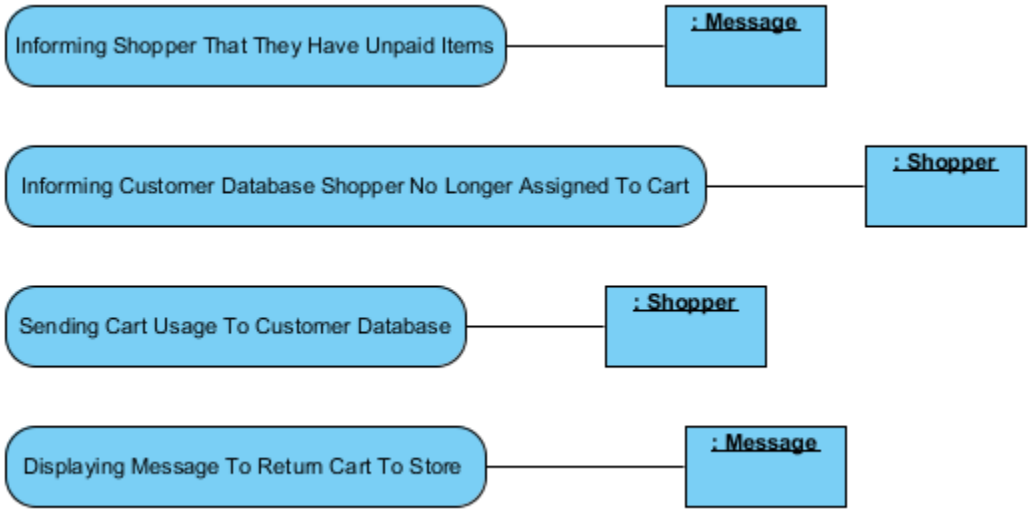

Figure 5: Releasing Cart From Shopper Object Mapping

Figure 5: shows:

Informing Shopper That They Have Unpaid Items – Creates a Message to the customer.

Informing Customer Database Shopper No Longer Assigned To Cart – Deletes the Shopper object.

Sending Cart Usage To Customer Database – Updates the Shopper object.

Displaying Message To Return Cart To Store - Creates a Message to the customer.

2.1.1.4 Shopper Requests Assistance

Figure 6: Shopper Requests Assistance Objects

Figure 6: shows:

Sending Message To Store Clerk - Creates a Message to the store clerk.

2.1.1.5 System Is Monitoring Shopping Cart

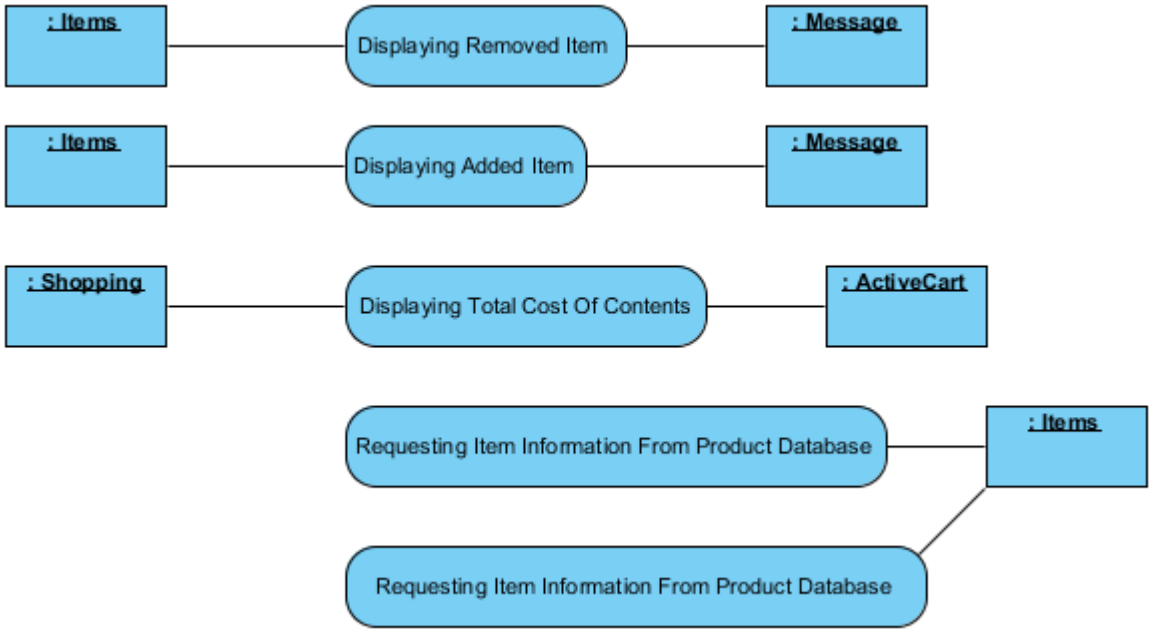

Figure 7: System Is Monitoring Shopping Cart Object Mapping

Figure 7: shows:

Displaying Removed Item – Uses the Items object to display a Message to the shopper.

Displaying Added Item - Uses the Items object to display a Message to the shopper.

Displaying Total Cost Of Contents – Uses the Shopping object to display a Message to the shopper.

Requesting Item Information From Product Database – Updates the Items object.

2.1.2 Other System Object Mappings

Other Systems are the Cashier, Item Detector, Product Database and Customer Database systems. The analysis of these systems is out of scope for this article.

Mapping objects to use cases is not an exact science. The purpose is to give a set of classes for an initial logical model, such that every action is represented, not to get a complete set of classes that match the use cases.

2.2 Build A Class Diagram

From the object diagrams, an initial class diagram is built. The class diagram captures objects as instances of classes in diagram and shows the relationships between those classes. It also demonstrates that every ‘system’ actor is accounted for in the model.

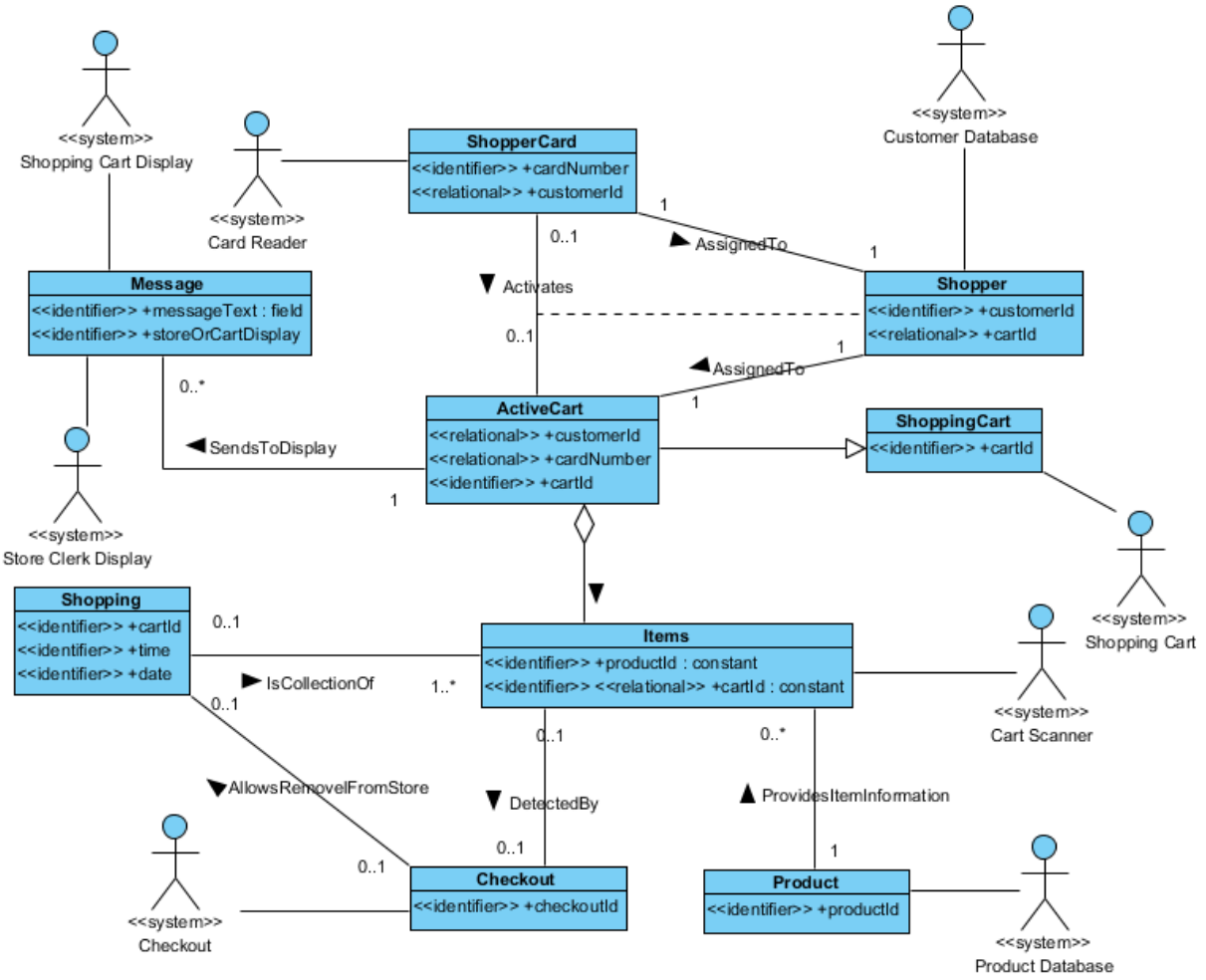

Figure 8: Initial Class Diagram

In Figure 8: each object has been assigned one or more identifying attributes. These attributes are propagated across relationships where appropriate. The classes are connected by relationships that show how they communicate. Cardinality is added to these relationships to show how many instances take part in the communication.

Finally, system actors are added to demonstrate external interfaces (these are not part of the logical model, but added for consistency checking).

2.2.1 Identifying Attribute

This attribute(s) identify a unique instance of the class. They are assigned a value when the instance is created; this value never changes.

2.2.2 Relational Attribute

For each relationship in the class diagram, one or more identifying attributes are passed from one class to another. These attributes specify exactly which class instances are taking part in the relationship (i.e. who is communication with who).

2.2.3 Cardinality

For each relationship, the number of instances of each class participating in the relationship is identified, and identifiers and foreign keys for each class are determined in order to cement those relationships.

(Note that UML allows the use of the '*' character to represent 'any number' of instances. Personally I want to specify an absolute maximum and minimum for the cardinality at each end of a relationship, since 'any number' is not implementable without infinite hardware resources. I use the '*' character to represent an 'unknown' number of instances.)

2.3 Build State Models

Each class is assigned functionality from the use case model. The associated attributes for those functions are added to the class as attributes. The functionality of a class is modeled with a State Transition Diagram (STD). Each STD begins with a Start state indicating that the instance does not yet exist. It is created upon some triggering even occurring. It exists for a period of time and is finally deleted, ending in a Stop state.

2.3.1 ShopperCard

The customer membership card has a card identifier. This can be used to identify the shopper. A card instance is created when it inserted into the card reader. It is deleted when it is no longer (or cannot be) associated with the cart.

The purpose of the ShopperCard is to monitor when the card is inserted into the cart, identify the customer that the card belongs to and recognize when the card is no longer inserted in the cart.

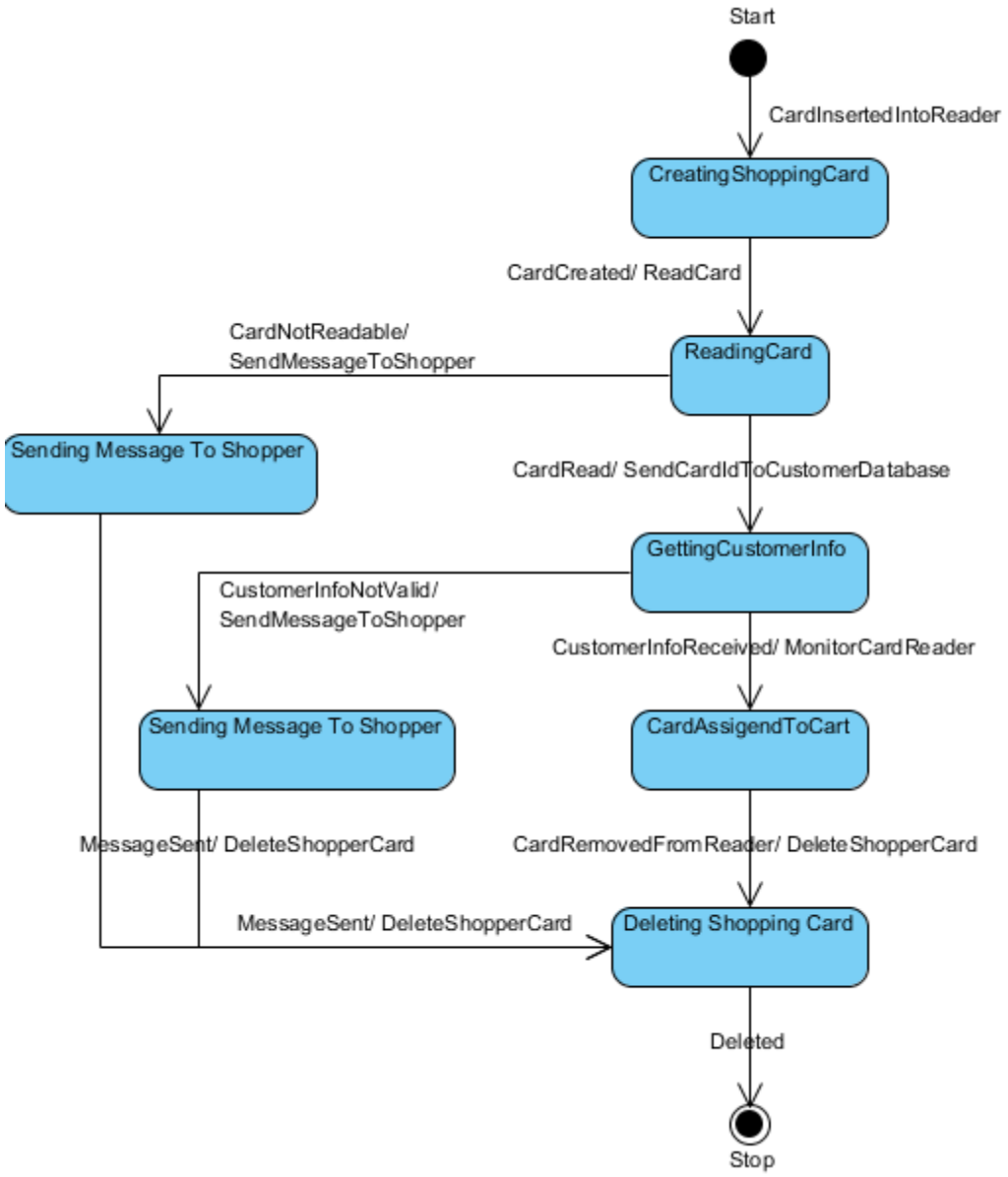

Figure 9: ShopperCard State Transition Diagram

When a card is inserted, an instance of the ShopperCard is created. The customer information associated with the card is retrieved from the customer database; this identifies the shopper. When the card is removed; or if the card cannot be read, or if the card is invalid, the ShopperCard instance is deleted.

2.3.2 ShoppingCart

The ShoppingCart class is used to maintain an inventory of available shopping carts in the store. A cart is added by the system administrator. ShoppingCarts are removed by the administrator, or if the system can no longer detect the cart. The shopping cart is identified by an arbitrary identification number.

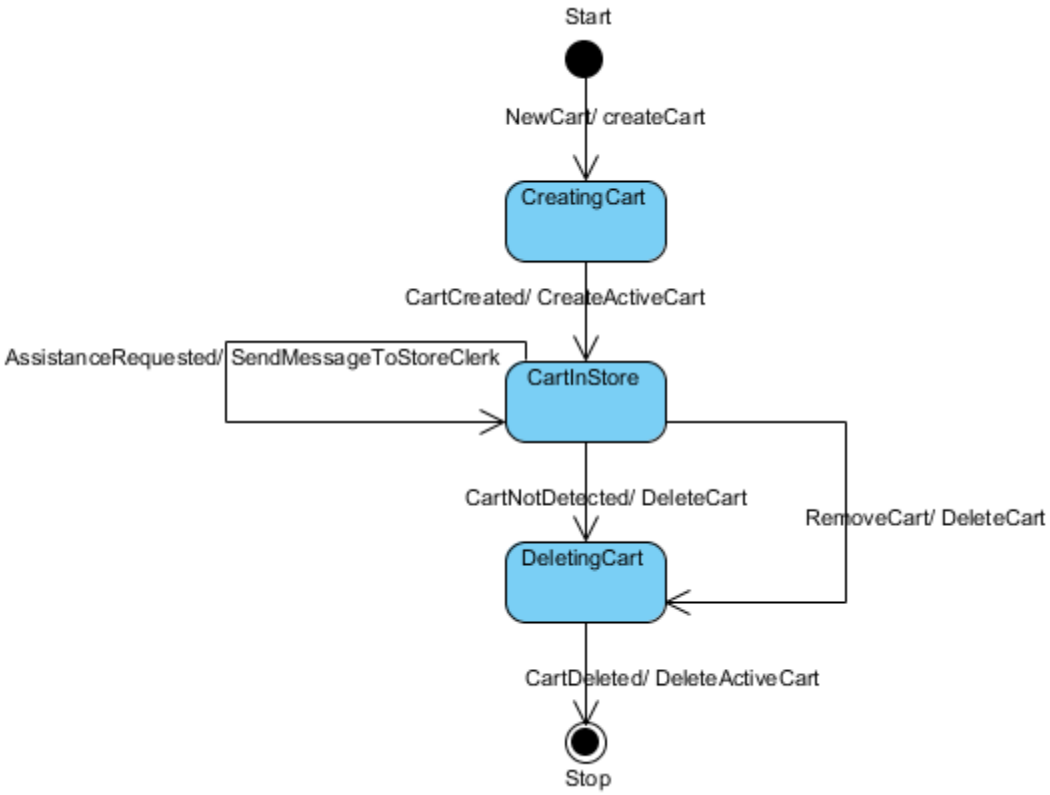

Figure 10: ShoppingCart State Transition Diagram

A ShoppingCart is created upon request from a list of available carts (not described here). The cart instance is deleted upon request, or if the system is unable to detect the cart. At any time a cart may be used to request assistance from a store clerk.

2.3.3 ActiveCart

Each shopper is recognized by a unique customer identifier in the customer database. The shopper is either shopping or they are not. This information can be derived from the card. Therefore the shopper is associated with an ActiveCart.

The primary purpose of the ActiveCart class is to identify when a cart is being used for shopping and keep the shopper (and store clerk) informed about their experience. The ActiveCart monitors the cart through to the shopper being unassigned from the cart.

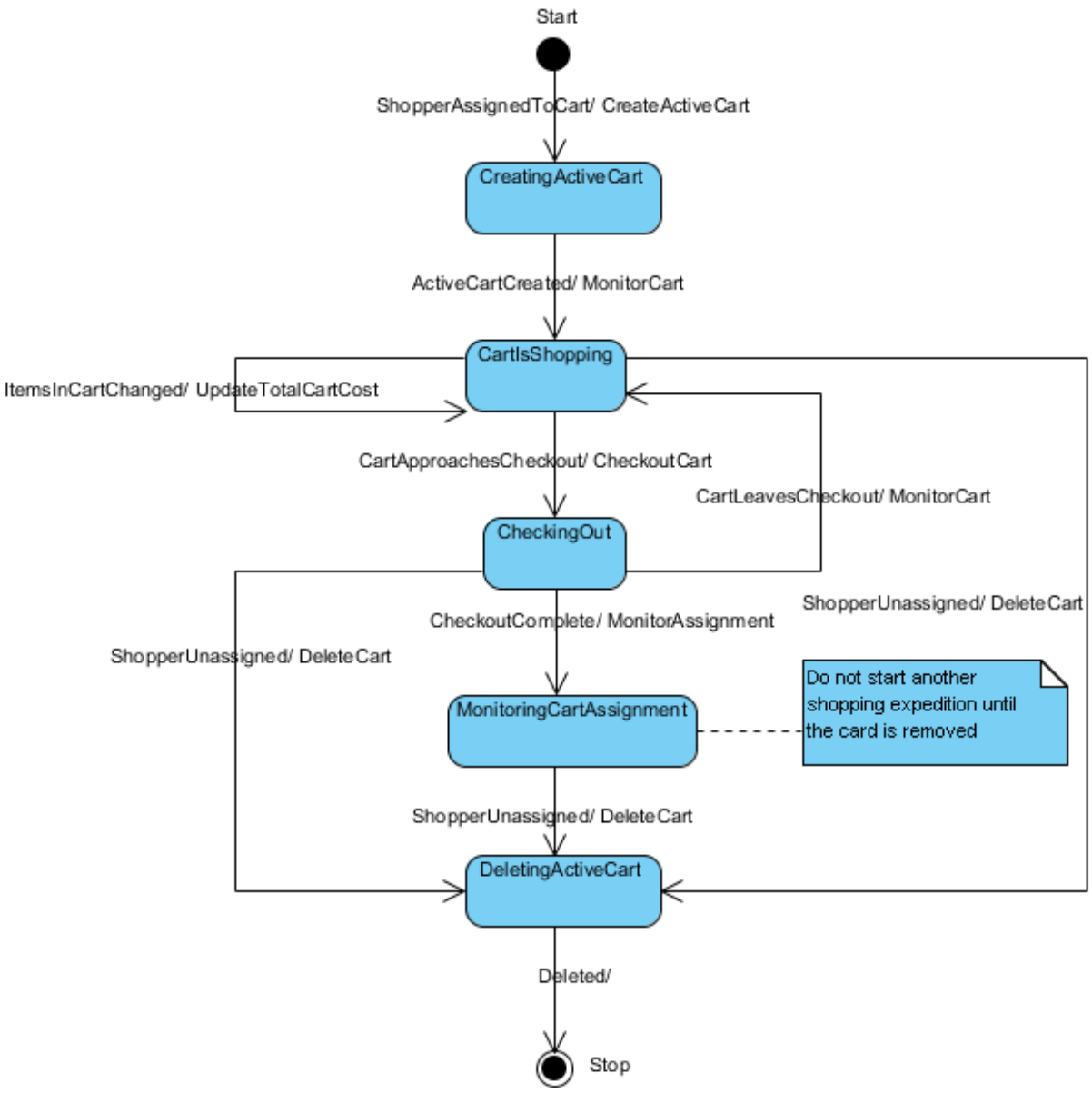

Figure 11: ActiveCart State Transition Diagram

The ActiveCart instance is created when a shopper is assigned to the cart. (Note that the cart does not know why the shopper is assigned or unassigned to the cart, only that it happens. The ShopperCard creates events assigning the customer to the cart or unassigning the shopper.)

Once assigned to a shopper, the cart is now shopping. The cart monitors items being added to or removed from the cart. (The items class informs the ActiveCart when these events occur.) When the cart approaches a checkout (signal received from the checkout system), the ActiveCart requests that the shopper checkout their cart. Once checkout is complete, the ActiveCart is idle and waiting for the shopper to become unassigned.

[Note that the ActiveCart instance is deleted if the corresponding ShoppingCart instance is deleted. This is implicit in the class diagram, and not shown in the state transition diagram.]

2.3.4 Items

The items class is used to monitor the scanning of items as they are added to or removed from an active shopping cart. A set of items are identified by their productId. This class keeps track of how many of each type of product are in the shopping cart.

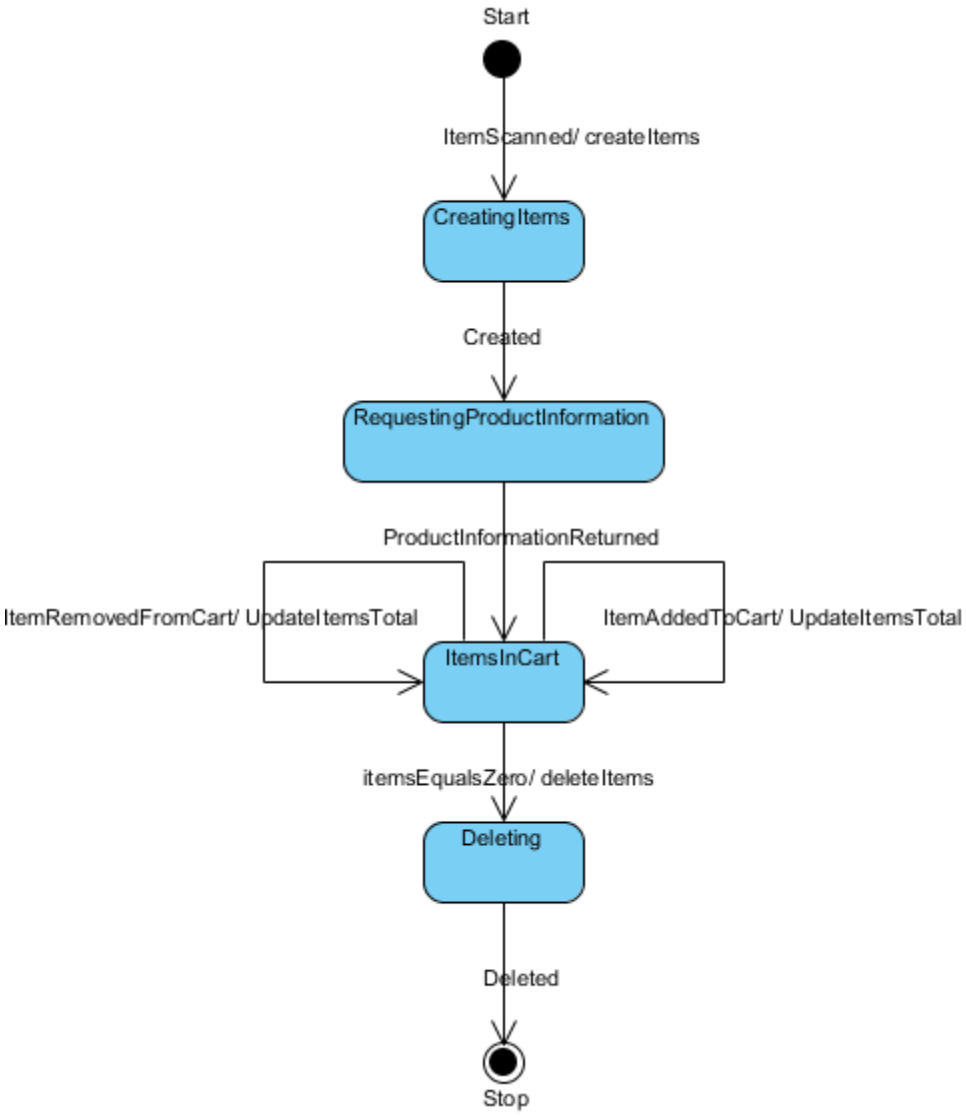

Figure 12: Items State Transition Diagram

The shopping cart scanner recognizes when an item is added to the cart. If no item with that product id is already in the cart, then a new item is created. As items of that product type are added to or removed from the cart, the totals for that product are updated. [This method allows for discounts on bulk items. It also allows for the total of a particular product to be displayed to the shopper, instead of displaying each item of the same type.]

2.3.5 Shopping

The shopping class keeps track of items in the shopping cart. Its primary purpose is to inform the shopper of the status of their shopping, by updating the cart display. Shopping is created when a cart is assigned to a store, and deleted when the cart is no longer assigned to the store.[3]

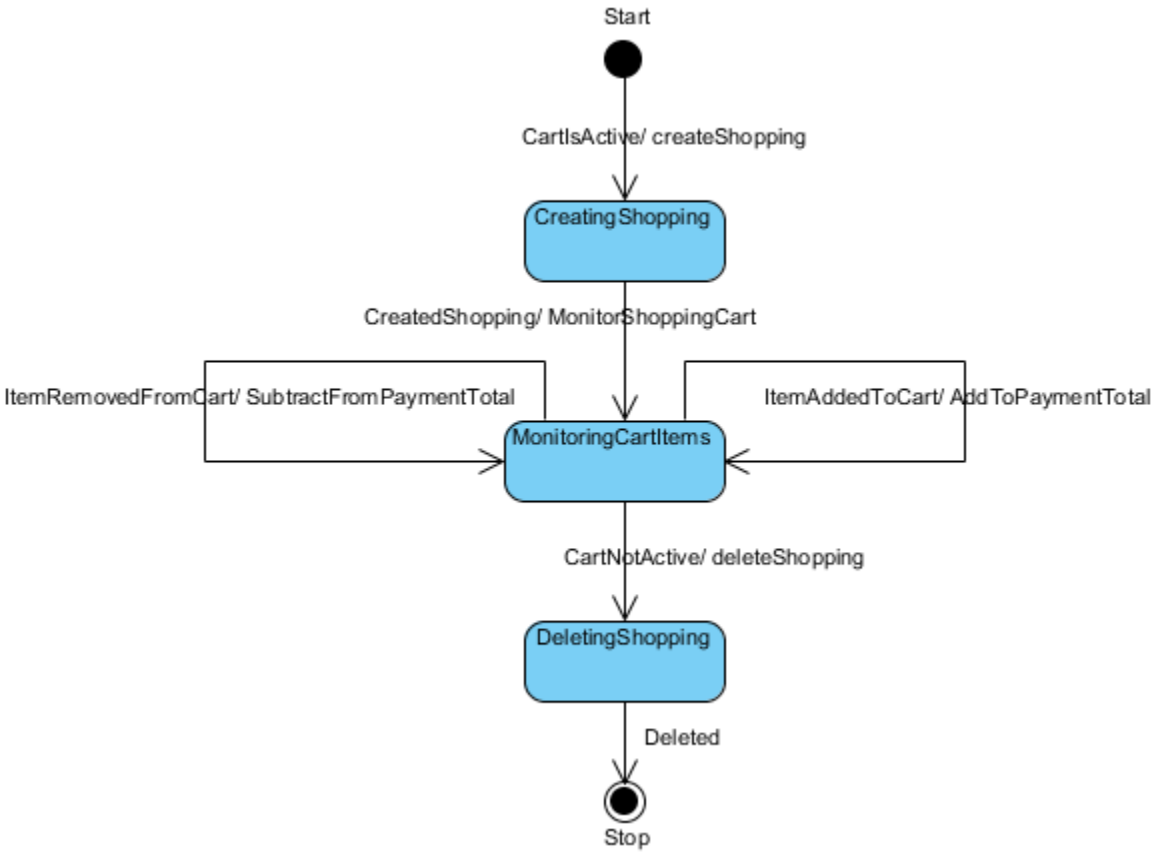

Figure 13: Shopping State Transition Diagram

A Shopping instance is created when a cart is added to the shopping system. The Shopping instance displays the value of items in the cart and maintains this total as items are added to or removed from the cart. When the cart is removed from the Shopping System the Shopping instance deleted.

2.3.6 Checkout

The Checkout class monitors the payment for items in an ActiveCart. A checkout instance is created when a cart is detected by the checkout counter and deleted when the cart is no longer within the checkout area. Each checkout instance is identified by a unique ID for the checkout counter, by the cart that arrives at the checkout and the date and time of arrival. [The same cart may arrive at the checkout several times in the same day, and we need to identify each time this occurs. The same cart cannot arrive at 2 checkouts at the same time.]

[3] Note that Shopping and Items are different classes and that 1 shopping instance may include may instances of Items.

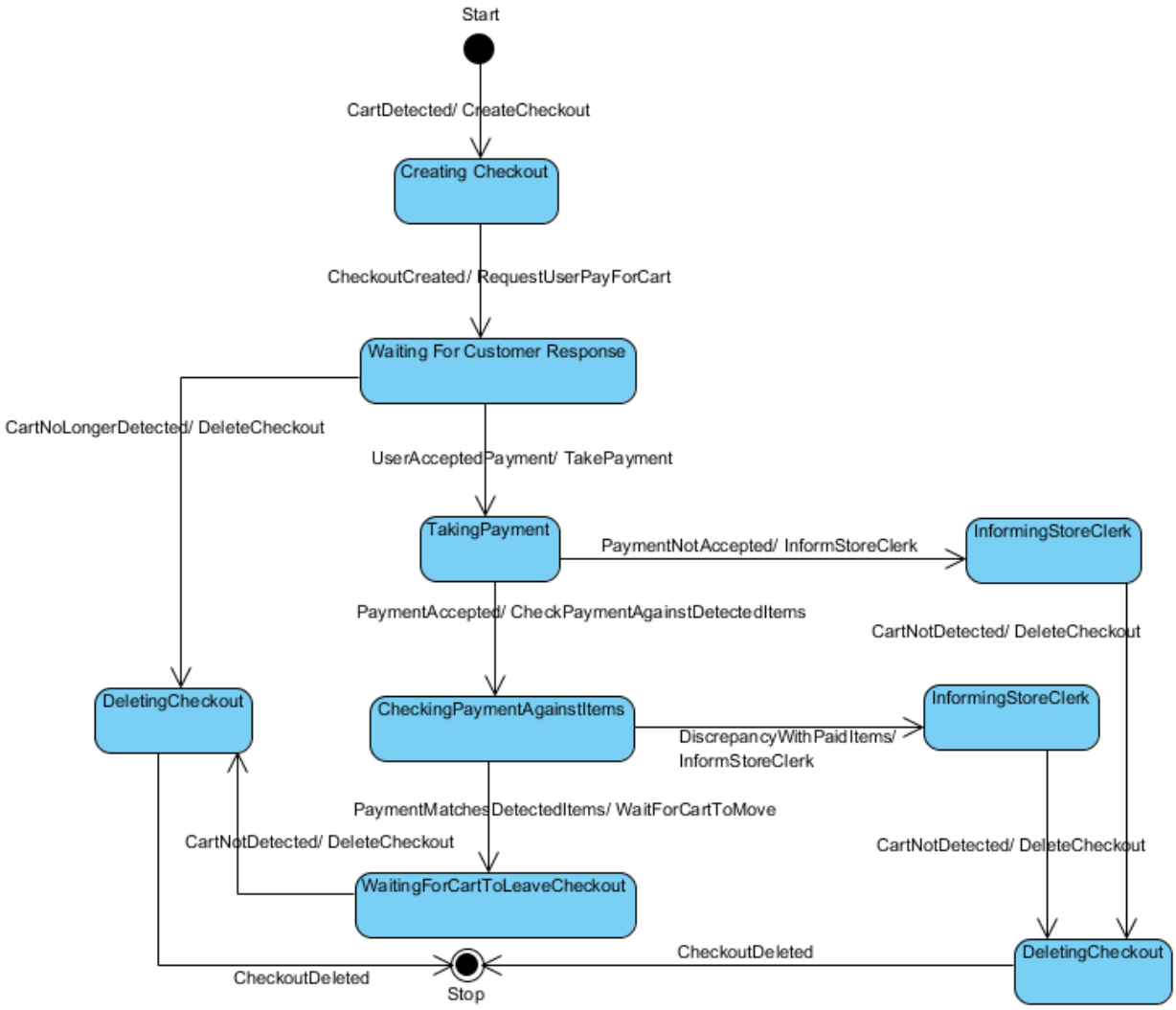

Figure 14: Checkout State Transition Diagram

A Checkout instance is created each time an ActiveCart is detected by the checkout counter. The user is requested to make payment. When payment is accepted from the cashier, the Checkout instance asks the item detector for the items detected in the checkout counter area. The Checkout instance verifies that what is being paid for corresponds with the items leaving the checkout area. Once the ActiveCart leaves the checkout area, the Checkout instance is deleted.

If anything goes wrong with the checkout process, a store clerk is sent a message to assist the shopper and the reason for the assistance request. [The InformStoreClerk actions generate an event to create a message for the store clerk display.]

2.3.7 Message

Messages are identified by the text that they convey to a display. A message is for a specific cart or for a store clerk display or maybe for both. If the message is for a shopper, the CartId identified which cart it is sent to. If the message is for the store clerk, the cartid identifies which cart the message came from.

[This is not a well-formed analysis class, because it introduces unnecessary design into the model. The message text to the store clerk could include the cartid, or maybe no cartid is even required. A better model is shown below.

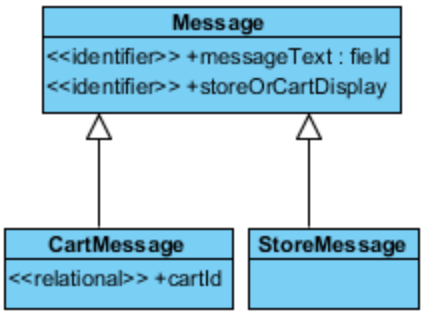

Figure 15: Well-Formed Message]

A message is created upon request from another class and is deleted when it has been sent to the receiving system.

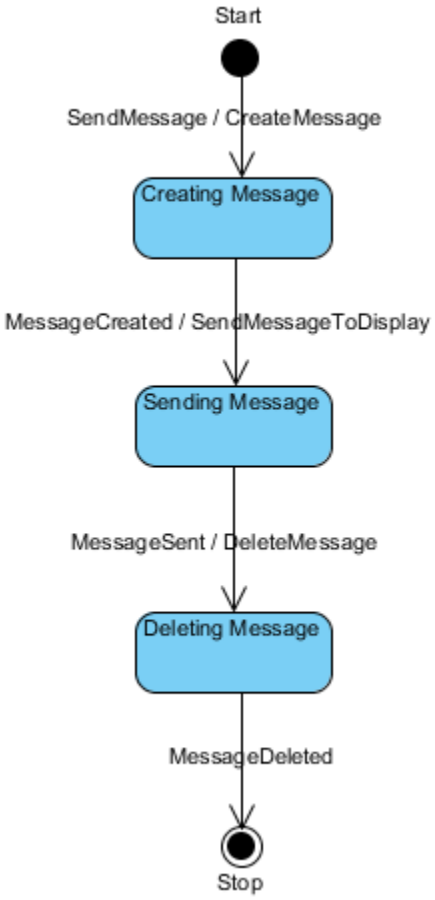

Figure 16: Message State Transition Diagram

When a message is required, the Message instance is created. The message is sent to the appropriate display. Once the message is acknowledged, it is removed from the display and deleted.

2.4 Operations and Events

Each transition is triggered by an event. Events come from external systems, from other class instances within the system, or they may be generated by the class instance as a result of some action completing within the class.

Each transition triggers an action. All actions are internal to the class instance and are captured by the class specification. The detailed class diagram is updated with class operations resulting from the state transition diagrams and the parameters used by those actions, as shown in Figure 17:

The Create and Delete actions are omitted from the diagram.

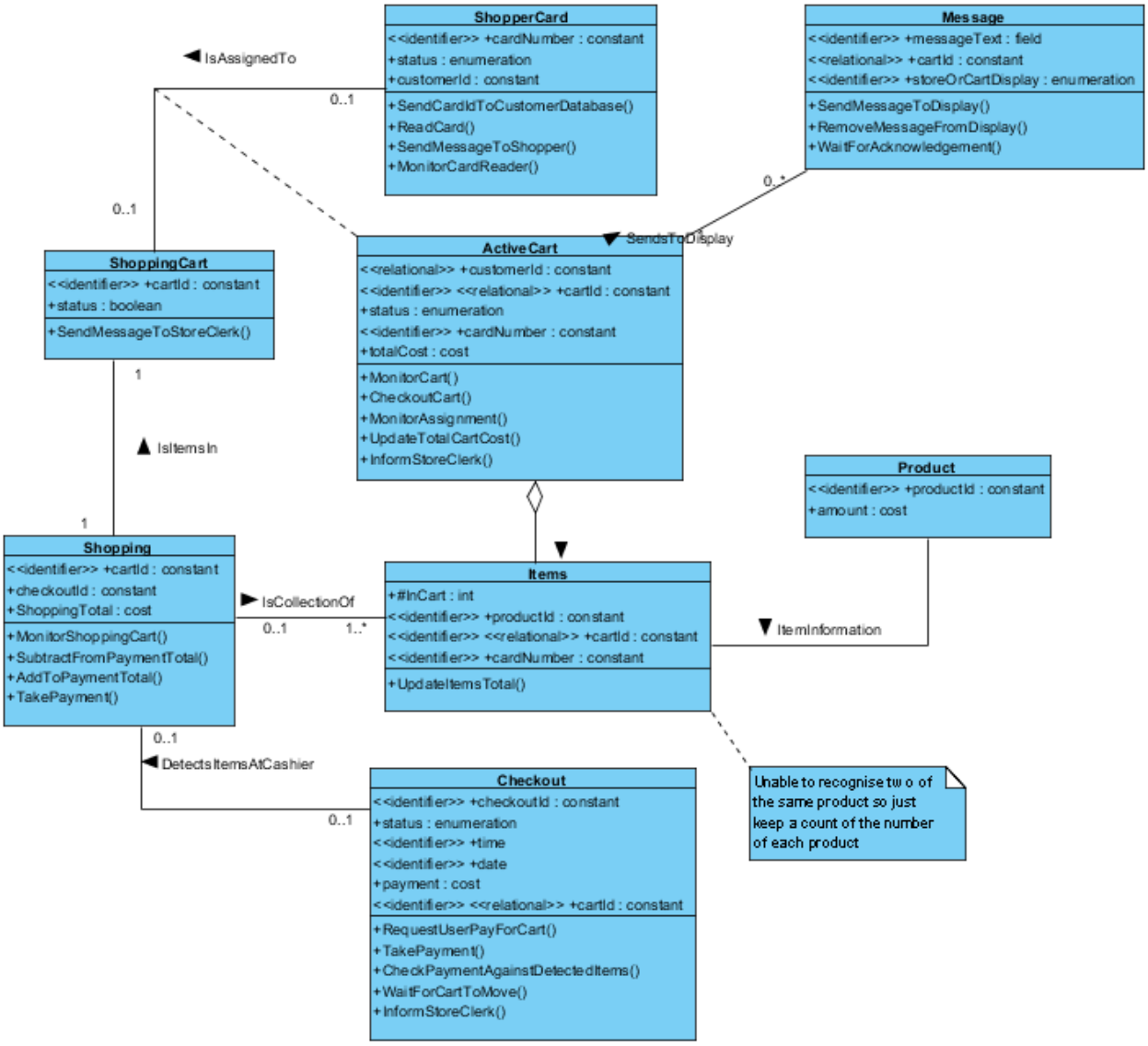

Figure 17: Detailed Class Diagram

Each operation in Figure 17: may be written out as a set of functional requirement for the Shopping system as follows:[4]

2.4.1 Message

Operations performed by the Message class:

- SendMessageToDisplay – When (certain events occur) the Shopping System will send (one of the following messages) to the (shopping cart of store clerk) display.[5]
- WaitForAcknowledgement – When a message has been displayed to a shopping cart it will be displayed until an acknowledgement is received and then removed.
- RemoveMessageFromDisplay – When (after a certain amount of time or if a user acknowledges the message) the Shopping System will display TBD (some blank of logo type display).

2.4.2 ShoppingCart

Operations performed by the ShoppingCart class:

- SendMessageToStoreClerk – When shopper requests assistance the Shopping System sends an assistance request message to the store clerk with shopping cart information (cart id and may include location in the store if known).

2.4.3 ShopperCard

Operations performed by the ShopperCard class:

- SendCardIdToCustomerDatabase – When a customer shopping card is inserted into the card reader and read by the Shopping System it will send customer information to the customer database with a request for shopper verification.
- ReadCard – When the customer database informs the Shopping System that a card inserted into the card reader is for a valid shopper it will send a message to the shopping cart display.
- SendMessageToShopper – When a card inserted into the shopping cart is unable to be read, the Shopping System will send a message to the shopping cart display that the card is not understood. When a card inserted into the shopping cart is not recognized as a valid shopper card, the Shopping System will send a message to the shopping cart display that the card is not valid for this system.
- MonitorCardReader – When a valid shopper card is inserted into the shopping cart and that card is removed, the Shopping System will display a message to the shopping cart that the cart is free.

2.4.4 ActiveCart

Operations performed by the ActiveCart class:

[4] The mapping between requirement and operation may not be exactly 1-1, but verifying that every operation is covered by 1 or more requirements will ensure that at least every system function is covered by the requirements.

[5] Each event creates a functional requirement – all captured by this single operation.

- MonitorCart – When shopper is assigned to a cart and the active cart is no detected by a checkout the Shopping System will update the cart display with items and costs of those items.
- CheckoutCart – When the active cart is detected by a checkout, the Shopping System will send the items and their costs to the checkout with a request to checkout the active cart.
- MonitorAssignment – When a cart that has been checked out is no longer assigned to a shopper is is made available for assignment to a shopper.[6]
- UpdateTotalCartCost – When an item is added to or removed from the active cart, the total cost of items is updated on the cart display.
- InformStoreClerk – When cart is checking out and the total cost of items in the cart changes, the store clerk is informed to assist the shopper.

2.4.5 Shopping[7]

Operations performed by the ActiveCart class:

- MonitorShoppingCart – When a cart is assigned to the store the Shopping System continuously keeps track of items in the cart until it is no longer assigned to the store.
- SubtractFromPaymentTotal – When an item is removed from the cart, the display is updated to reflect the cost of items in the cart.
- AddToPaymentTotal - When an item is added to the cart, the display is updated to reflect the cost of items in the cart.

2.4.6 Items

Operations performed by the Item class:

- UpdateItemsTotal – When product type is added to the cart the number of items of that product type is sent to the shopping cart display.

2.4.7 Checkout

Operations performed by the Checkout class:

- RequestUserPayForCart – When the active cart is detected by a checkout, the shopper is sent a message requesting payment for the items in the cart.
- TakePayment – When the shopper confirms that they want to pay for the items, the checkout is sent information about the items in the cart.
- CheckPaymentAgainstDetectedItems – When payment has been accepted by the checkout the Shopping System compares the items paid for against the items at the Item Detector and informs the store clerk if there is an inconsistency.
- WaitForCartToMove – When the cart leaves the Checkout area the customer is informed that they may remove their card from the cart.

---

[6] You get the idea that is is the Shopping System that does this by now.

[7] When a relationship has 1-1 cardinality, as in the Shopping to ShoppingCart relationship, these two classes may be easily combined (as they probably should in this example).

- InformStoreClerk – If the Cashier informs the Shopping System that payment was not accepted, or if the items detected are not consistent with the items paid for, the store clerk is informed.

3 Summary

The logical model takes the use case model and assigns it to the architecture which will be used to host the use cases. The architecture includes several systems over which the use cases are distributed. An independent logical model is built for each system.

Potential objects are identified for each action described by the use case. These objects are combined into an initial class model for a specific system (the Shopping System, for example).

Classes in the model are assigned identifying attributes. Relationships are defined between classes in terms of cardinality, a direction and foreign key attributes that are passed between the classes. The class model is normalized to 3NF[8] (not discussed here).

Each class is modeled by a state transition diagram. The state transition diagram adds dynamic behavior to the class. A state transition diagram describes the externally visible properties that an instance of the class may take. Each instance begins by not existing (or in the 'start' state). When an instance is required, it is created and the instance initializes into its first observable state. Once the instance has completed its work, it is deleted, ending in a non-existent ('stop') state.

Lastly, the actions arising from the state transition diagrams are added to the respective classes as operations, and the class attributes are updated, as necessary, to handle the data required by those operations.

The logical model is complete, but is it consistent? Part III will demonstrate how is to verify that the model is internally consistent.

As a final note, I leave it up to the reader to determine what happens to the Shopping System if the customer unexpectedly removes their card from the shopping cart reader. This is a typical scenario that will be explored in part III.

[8] Wiki or multiple web sites clarify 3NF .. http://en.wikipedia.org/wiki/Third_normal_form