Part III (Validating The Model)

This is the final part in a trilogy of articles that demonstrate how a UML like model may be used to derive an executable set of requirements for an automated shopping cart system in a typical retail store. Part 1 introduced the use case model. Part 2 introduces the logical model. How does the reader know that the use case model and the logical are consistent with each other?

Part 3 demonstrates how the logical model may be used to realize (execute paths through), the use case model, using interaction (or sequence) diagrams.

Each sequence diagram shows external actors sending and receiving messages between themselves and class instances (objects), derived from the logical model. A message represents an event. A message carries data that is passed with the event. An event triggers an operation within the receiving object.

Objects are labeled with the identifying attribute of the class instance, and with the class name. Messages do not show the data that is passed between instances, as this would clutter the diagram. (The reader may determine the data from the operations for the associated class and its attributes, in the logical model.)

1 Use Case Realizations

Interaction diagrams are used to check the consistency of the logical model. Ideally they are generated automatically as a result of executing the model within the modeling tool. Other times (as in this case), the analyst may model the use case scenarios manually using the components of the logical model to produce the following diagrams.

For each path in a use case, we map the events in that path to messages in a sequence diagram. Messages are passed between actors and instances of classes in the logical model. In some cases a single event may cause many messages to be passed between internal classes. Each use case event however, is initiated by a single message being passed between an actor and an instance of a class from the logical model.

When every use case path has been captured as a continuous sequence of messages in a sequence diagram, then the consistency between the use case model and the logical model has been proven.

If it is not possible to demonstrate the completion of any event through these sequence diagrams, then there is a gap between the use case model and the logical model.

If any operations in the logical model are not captured by this process then the analyst may question the validity of that operation.

The following examples contain an overview sequence diagram, showing the system to actor interactions only. These overview diagrams are expanded to show each path through the use case in terms of the interaction between the components of the logical model. In this manner it is possible to demonstrate that all paths through the use case have been captured by operations (which map to system requirements).

(With the addition of creation and deletion operations, it is possible to execute the logical model, but model execution is beyond the scope of this set of articles.)

1.1 Assigning Cart To Shopper Use Case

The following sequence diagrams describe how the Assigning Cart To Shopper use case is captured by the logical model.

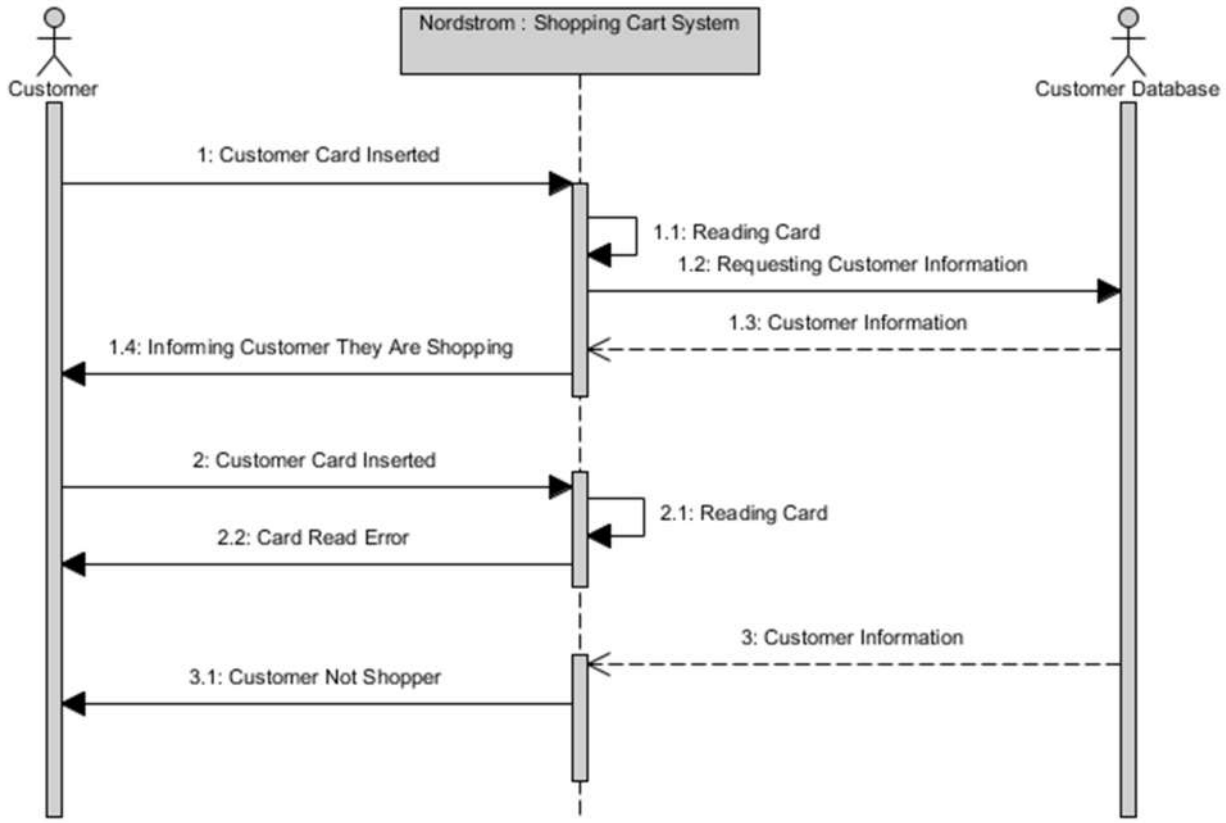

Figure 1: Assigning Cart To Shopper System Interaction Diagram

1.1.1 Main Path

This sequence diagram demonstrates how the main path through the Assigning Cart To Shopper use case is realized by the logical model.

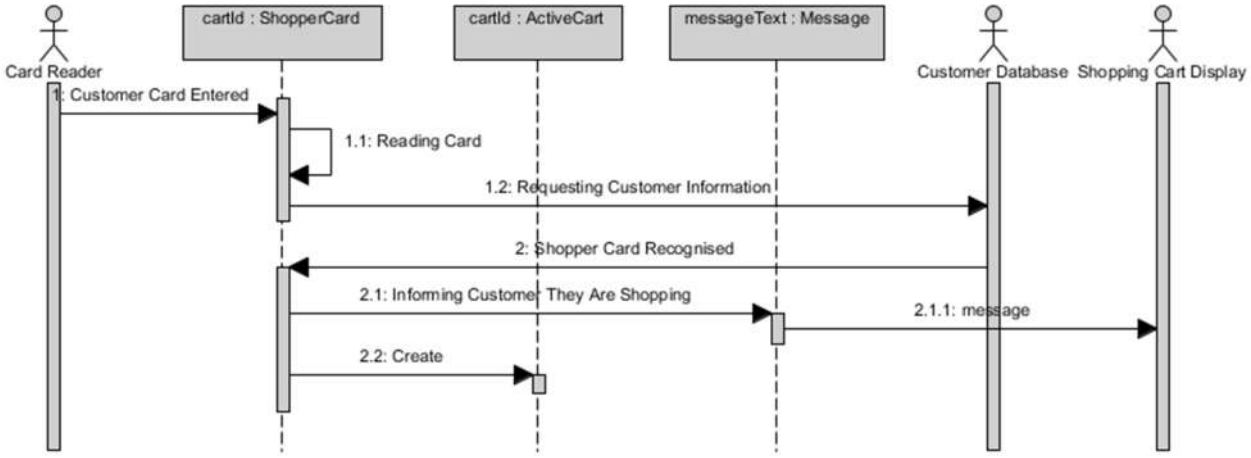

Figure 2: Assigning Cart To Shopper Logical Interaction Diagram

1. The use case starts when the Card Reader actor detects that a customer has entered a card into the shopping cart. This causes an externally generated event, named Customer Card Entered to be sent to an instance of the ShoppingCart.
2. The ShoppingCart reads the card information in order to identify the customer.
3. The ShoppingCart sends this information to the Customer Database actor as a request to identify the customer as a shopper.
4. The Customer Database returns the Shopper information requested by the ShoppingCart.
5. The ShoppingCart then sends a message to the Customer to inform them that they may continue shopping; this message is handled by an instance of the Message object. (In this manner, if anything changes about the way messages are displayed to the shopper, then there is no impact on the ShoppingCart.)
6. The Message instance sends an appropriate message to the Shopping Cart Display actor.
7. Finally an instance of the ActiveShoppingCart is created. (This is a purely internal interaction that has nothing to do with the requirements and is shown here for completeness only. The 'create' and 'delete' messages are used for execution of the model.)

1.1.2 Card Read Error Alternate Path

This next sequence diagram demonstrates how the Card Read Error alternate path through the Assigning CartTo Shopper use case is realized by the logical model.

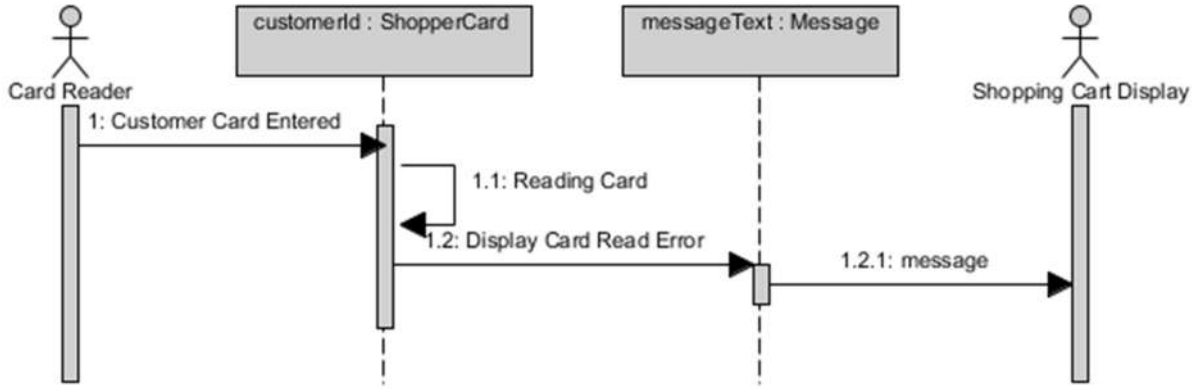

Figure 3: Card Read Error Logical Interaction Diagram

1. The alternate path starts after the customer has inserted their card, the ShoppingCart reads the card and determines that it is indecipherable.
2. The ShoppingCart sends a message to inform the customer that their card is invalid.
3. The Message instance sends a message to the Shopping Cart Display.

1.1.3 Customer Not Shopper

This next sequence diagram demonstrates how the Customer Not Shopper alternate path through the Assigning Cart To Shopper use case is realized by the logical model.

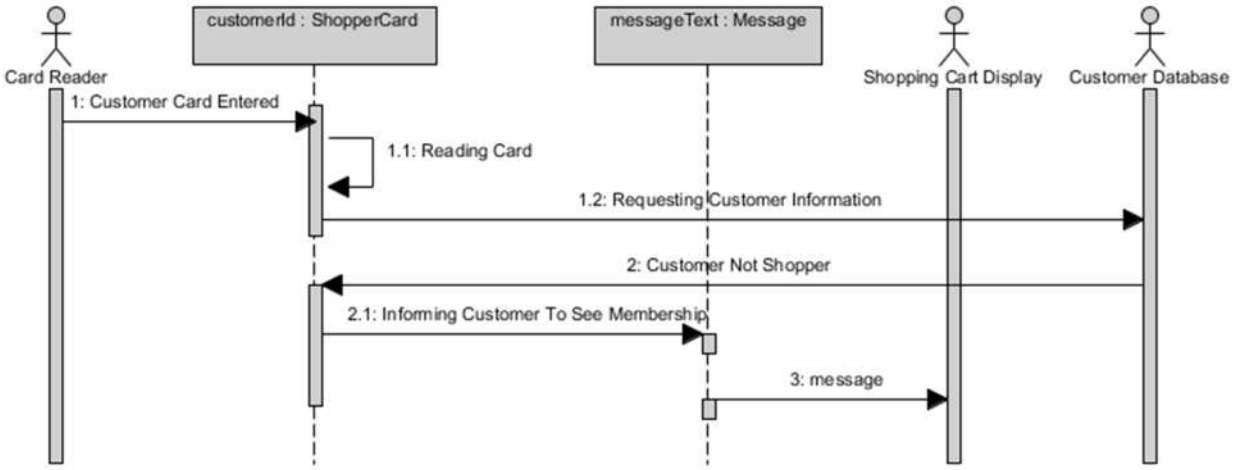

Figure 4: Customer Not Shopper Logical Interaction Diagram

1. This alternate path begins when a card is entered, the card is read and a request made of the Customer Database to identify the customer and the Customer Database actor informs the ShoppingCart instance that the customer is not a registered shopper.
2. The ShoppingCart sends a message informing the customer that they need to register to become a shopper.
3. The Message instance sends the appropriate message to the Shopping Cart Display.

1.2 Releasing Cart From Shopper

This sequence diagram demonstrates both the main path and the alternate path through the Releasing Cart From Shopper use case as realizations of the logical model.

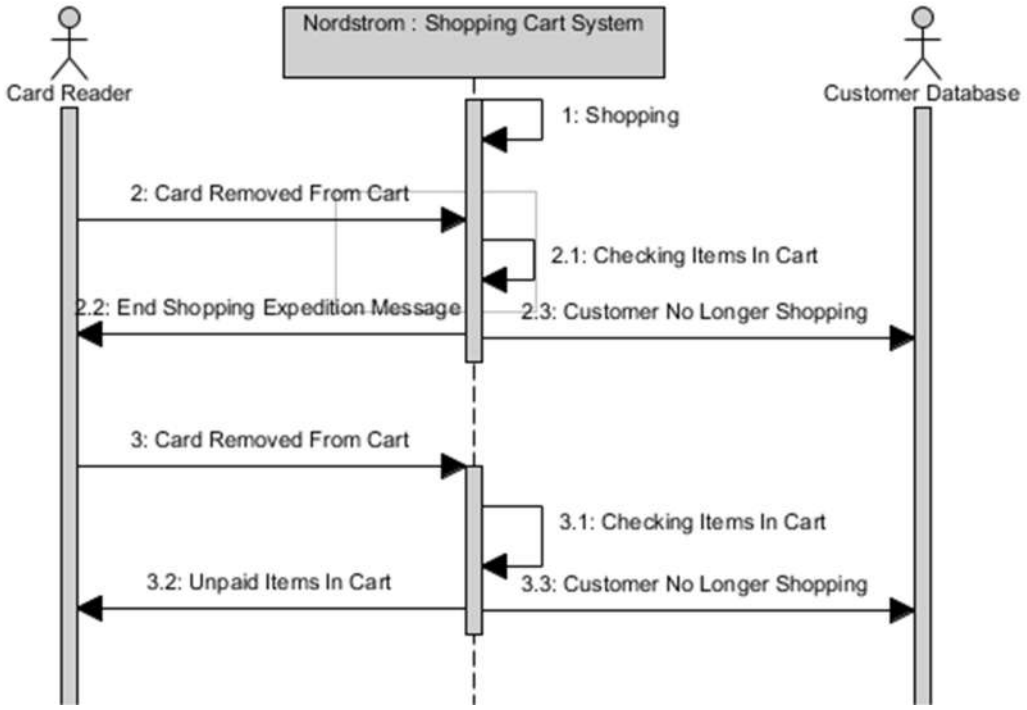

Figure 5: Releasing Cart From Shopper System Interaction Diagram

1.2.1 Main Path

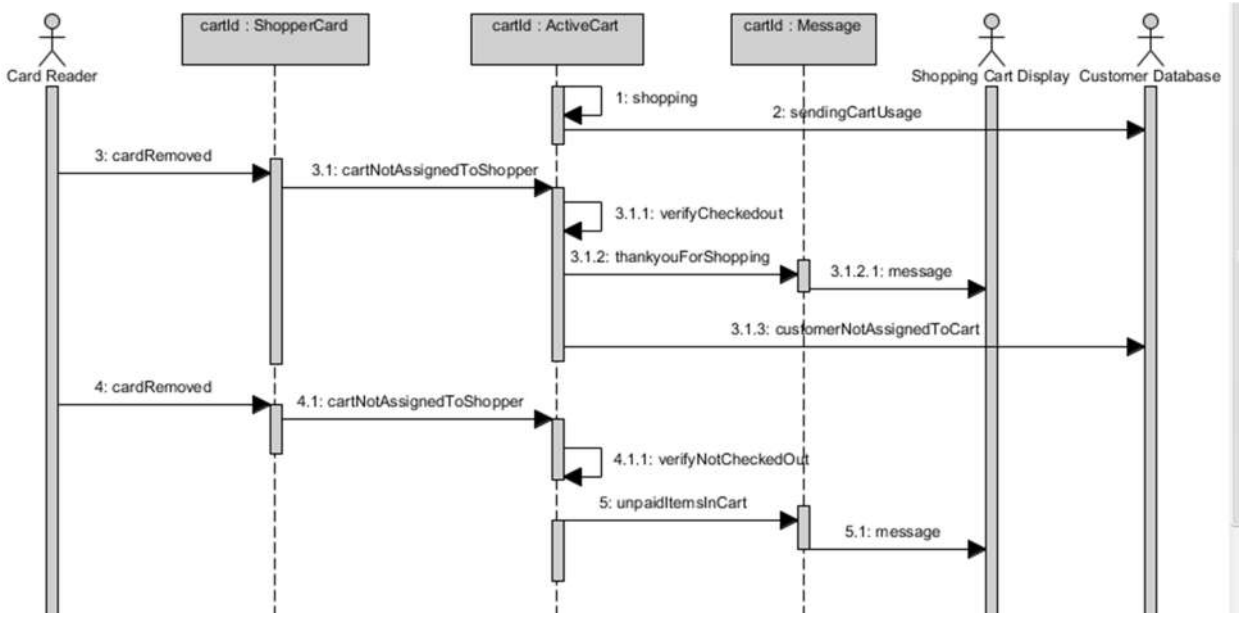

Figure 6: Releasing Cart From Shopper Logical Interaction Diagram

Prior to this scenario occurring, the shopper is assigned to a cart and the ActiveCart object is informing the customer database of the cart's activity.

1. The main path begins when the shopper removes their shopping card from the cart and the ShopperCard instance detects this.
2. The ActiveCart is informed that the cart is not assigned to the Shopper. The ActiveCart object knows that the shopper has checked out successfully and informs the customer that they have finished shopping. The ActiveCart also informs the CustomerDatabase that the customer is no longer assigned to the cart.

If the shopper has not checked out then when the shopping card is removed from the cart:

3. The ActiveCart object knows that the shopper has not checked out the cart and it sends a message to the customer that there are items in the cart that are not paid for.
4. The message is relayed to the CartDisplay via the Message object.

1.3 System Is Monitoring Shopper

The following sequence diagrams describe the System Is Monitoring Shopper use case.

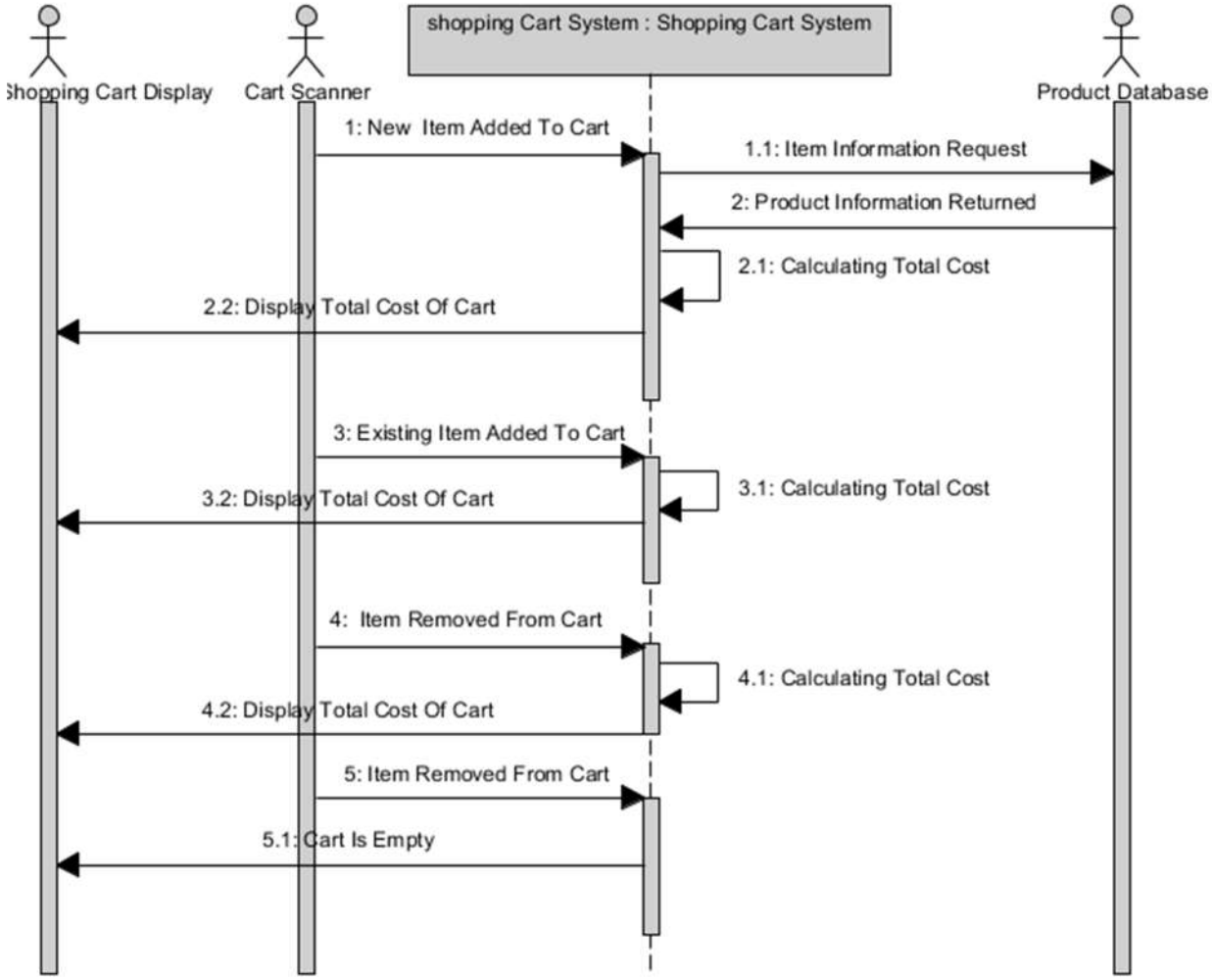

Figure 7: System Is Monitoring Shopper System Interaction Diagram

1.3.1 Main Path

This sequence diagram demonstrates the main path through the System Is Monitoring Shopper use case as a realization of the logical model.

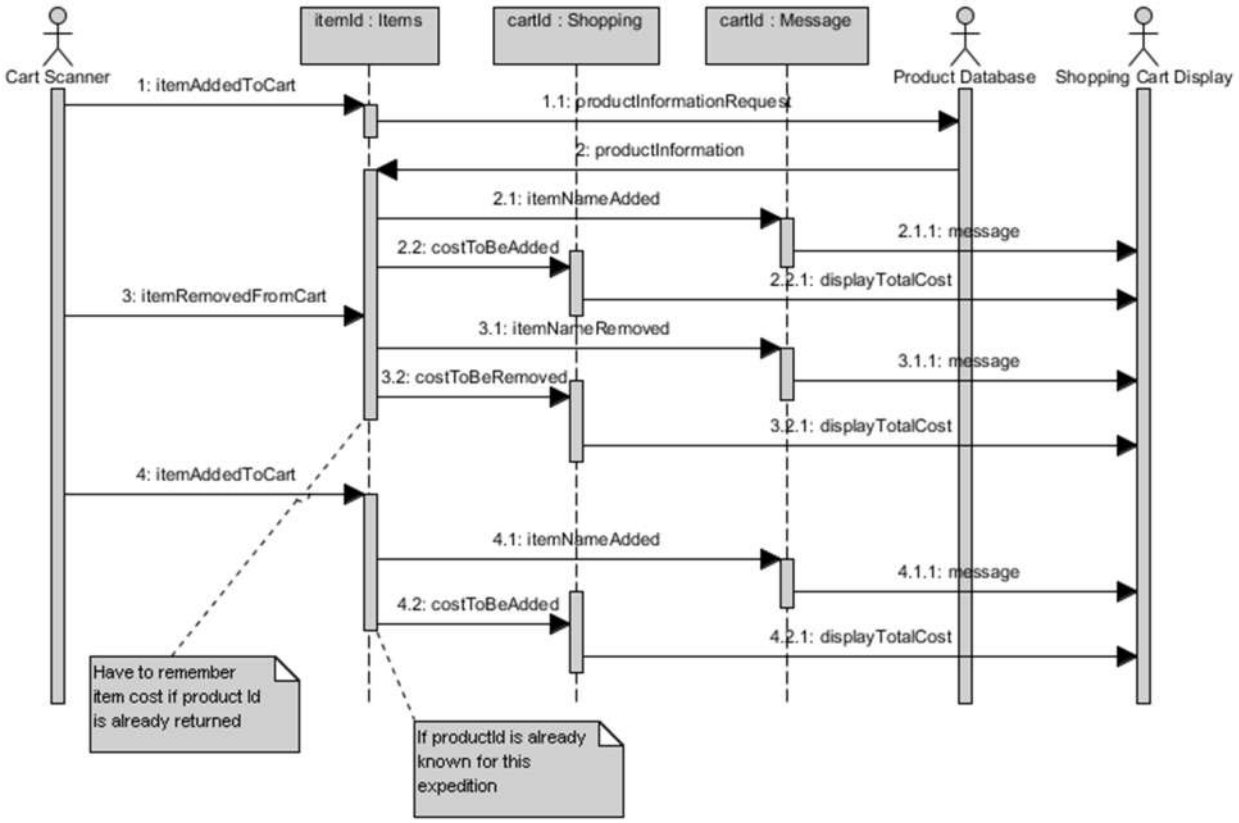

Figure 8: System Is Monitoring Shopper Logical Interaction Diagram

Add new item to cart.

1. The main path scenario starts when the Items object is informed that an item was added to the cart.
2. The Items object requests the Product Database return information about the item.
3. The Product Database returns the item information to the Items instance.
4. The Items object sends a message to the shopper informing them of the item added to the cart.
5. The Message object sends the message to the cart display.
6. The Items object sends the cost of the item to the Shopping object.
7. The Shopping object calculates the total cost of the shopping in the cart and displays this to the customer.
8. The Message object sends the total cost message to the cart display.

Remove item from cart.

1. The Items object is informed that an item was removed from the cart.
2. The Items object sends a message to the shopper informing them of the item removed from the cart.
3. The Message object sends the message to the cart display.

4. The Items object sends the cost of the item to the Shopping object.
5. The Message object sends the total cost message to the cart display.

Add Item already in cart.

1. The Items object is informed that an item was added to the cart.
2. The Items object recognizes that the item is already in the card and sends a message to the shopper informing them of the item added to the cart.
3. The Message object sends the message to the cart display.
4. The Items object sends the cost of the item to the Shopping object.
5. The Shopping object calculates the total cost of the shopping in the cart and displays this to the customer.
6. The Message object sends the total cost message to the cart display.

1.3.2 Item Not Recognized Alternate Path

This sequence diagram demonstrates the Item Not Recognized alternate path through the System Is Monitoring Shopper use case as a realization of the logical model.

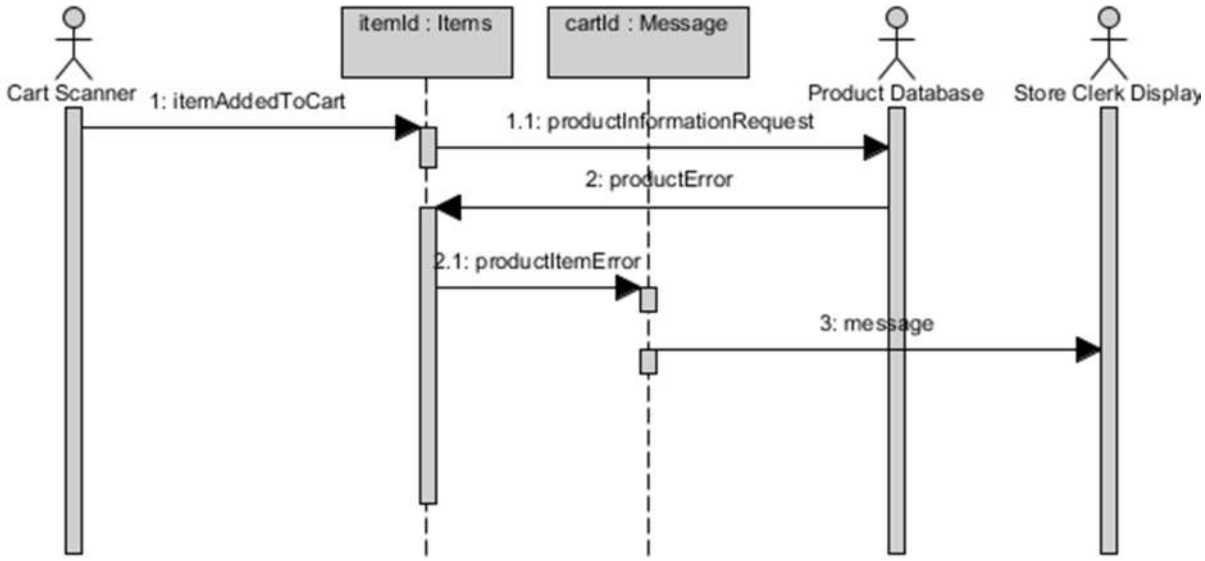

Figure 9: Item Not Recognized Logical Interaction Diagram

1. The Items object is informed that an item is added to the cart.
2. The Items object requests the Product Database return information about the item.
3. The Product Database returns a message that the item information is not available.
4. The Items object sends a message to the stre clerk that a product error has occurred.
5. The message object displays the error on the Store Clerk Display.

1.4 Shopper Requests Assistance

This sequence diagram demonstrates both the main path through the System Is Monitoring Shopper use case realizations by the logical model.

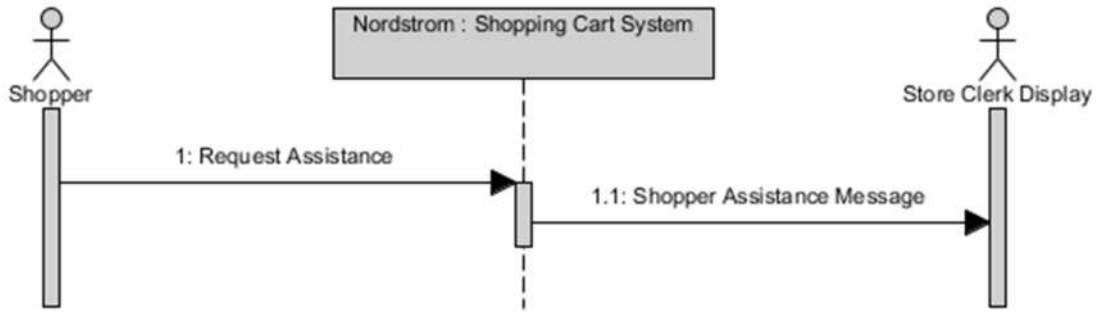

Figure 10: Shopper Requests Assistance System Interaction Diagram

1.4.1 Main Path

This sequence diagram demonstrates the main path through the Shopper Requests Assistance use case as a realization of the logical model.

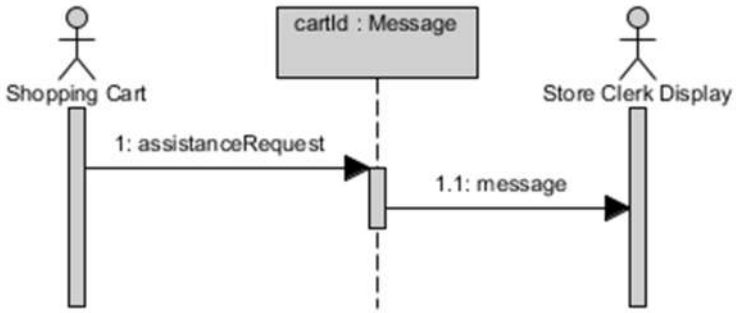

Figure 11: Shopper Requests Assistance Logical Interaction Diagram

1. The Message object receives a messge from the shopper requesting assistance.
2. The Message object sends this message to the Store Clerk Display.

1.5 Checking Out Shopper

The following sequence diagrams describe the Checking Out Shopper use case.

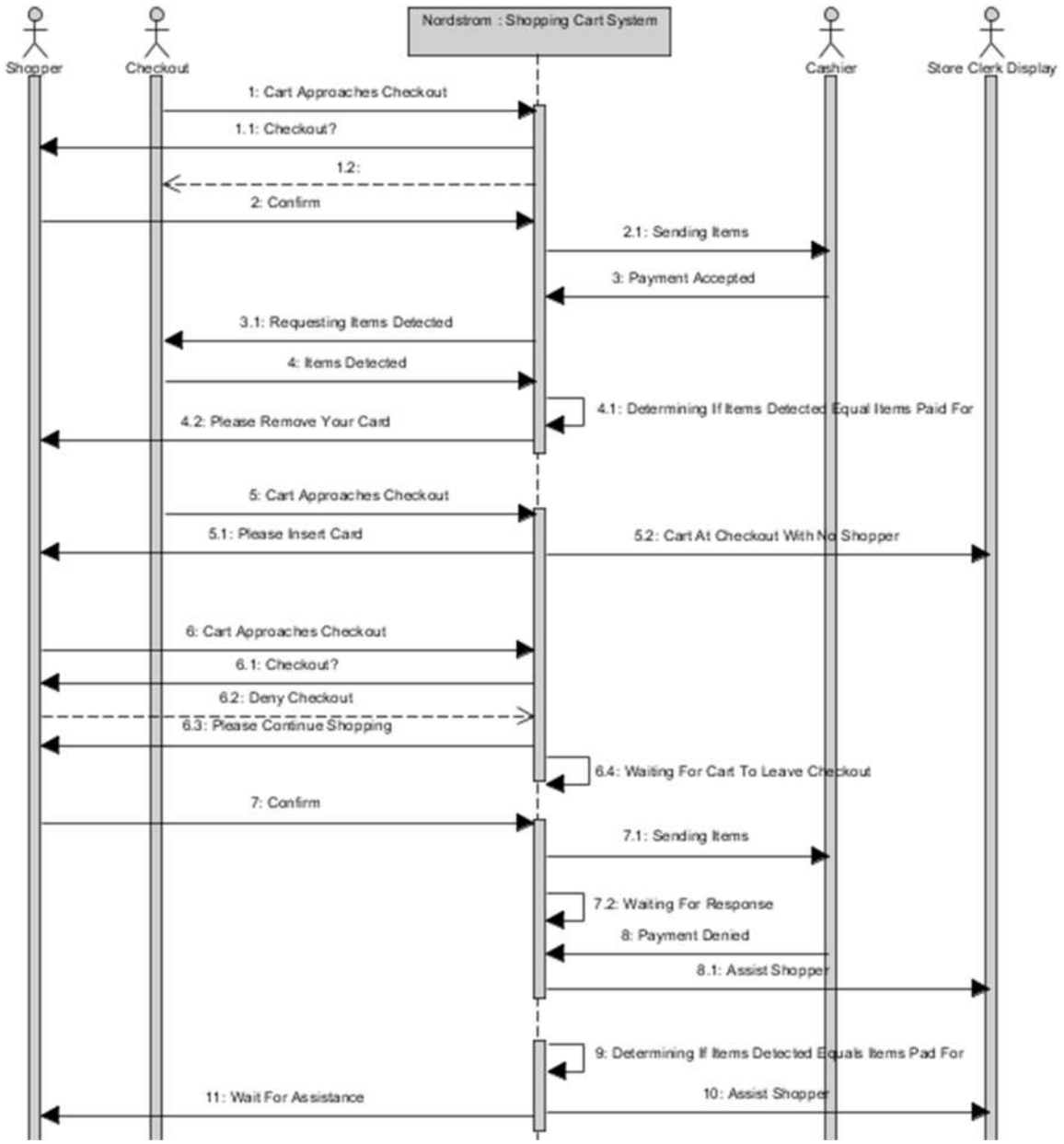

Figure 12: Checking Out Shopper System Interaction Diagram

1.5.1 Main Path

This sequence diagram demonstrates the main path through the Checking Out Shopper use case as a realization of the logical model.

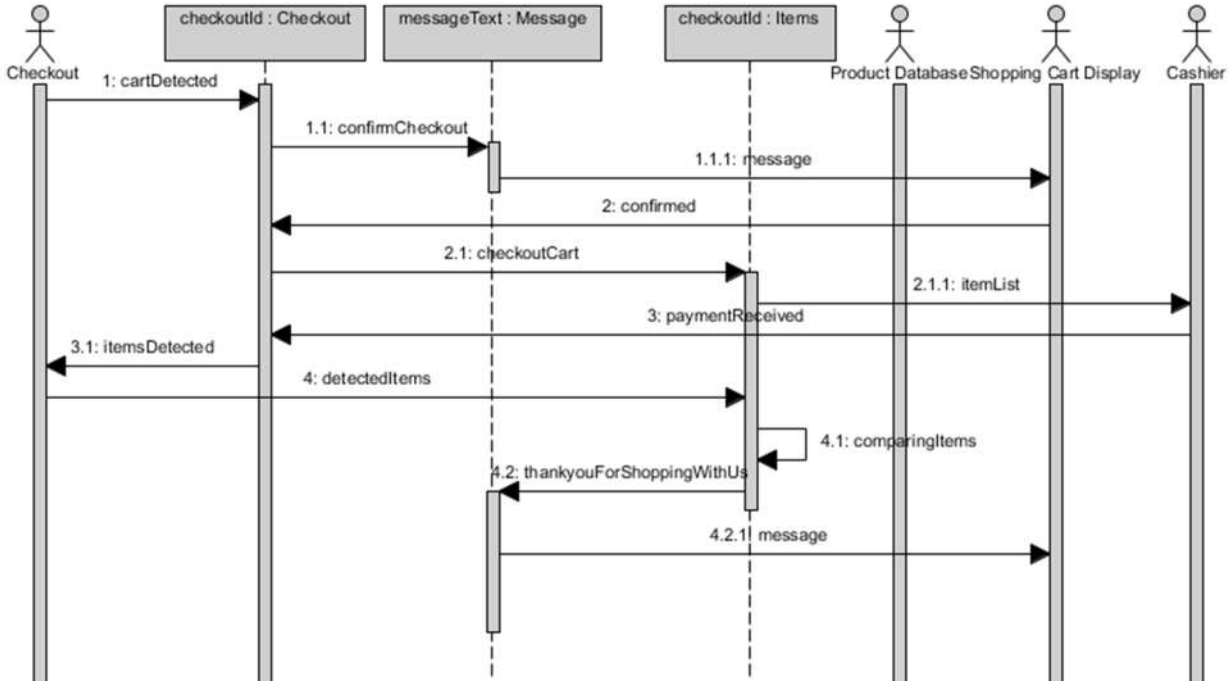

Figure 13: Checking Out Shopper Logical Interaction Diagram

1. The checking out shopper scenario starts when the Checkout object receives a message from a Checkout that the cart has been detected at the checkout area.
2. The Checkout object sends a message to te shopper requesting that they confirm that they wish to checkout.
3. The Message object sends the message to the Cart Display.
4. The Checkout object receives a message from the Cart Display confirming checkout of the cart.
5. The Checkout object sends a message to the Items object requesting checkout of items in the cart.
6. The Items object sends a list of items in the cart to the Cashier.
7. The Cashier confirms that payment was received for the items in the cart.
8. The Checkout object sends a request to the checkout for the number of items detected passing through the checkout. (The checkout includes an item detector that detects how many items are passing through the checkout. These are compared to the items paid for in the cart.)
9. The Items object receives the detected items with a request to compare them against the items in the cart.
10. The Items object confirms that the detected items match those that are in the cart.
11. The Items object sends a shopping complete message to the message object.
12. The Message object displays the ‘Thankyou For Shopping With Us’ message on the Cart Display.

1.5.2 Checkout Is Incomplete Alternate Path

This sequence diagram demonstrates the Checkout Is Incomplete alternate path through the Checking Out Shopper use case as a realization of the logical model.

There are three scenarios where the checkout process may fail to complete:

- The shopper decides that they do not wish to checkout at this time,
- The cashier denies payment for the items in the cart, or
- The checkout detects items that are not in the cart.

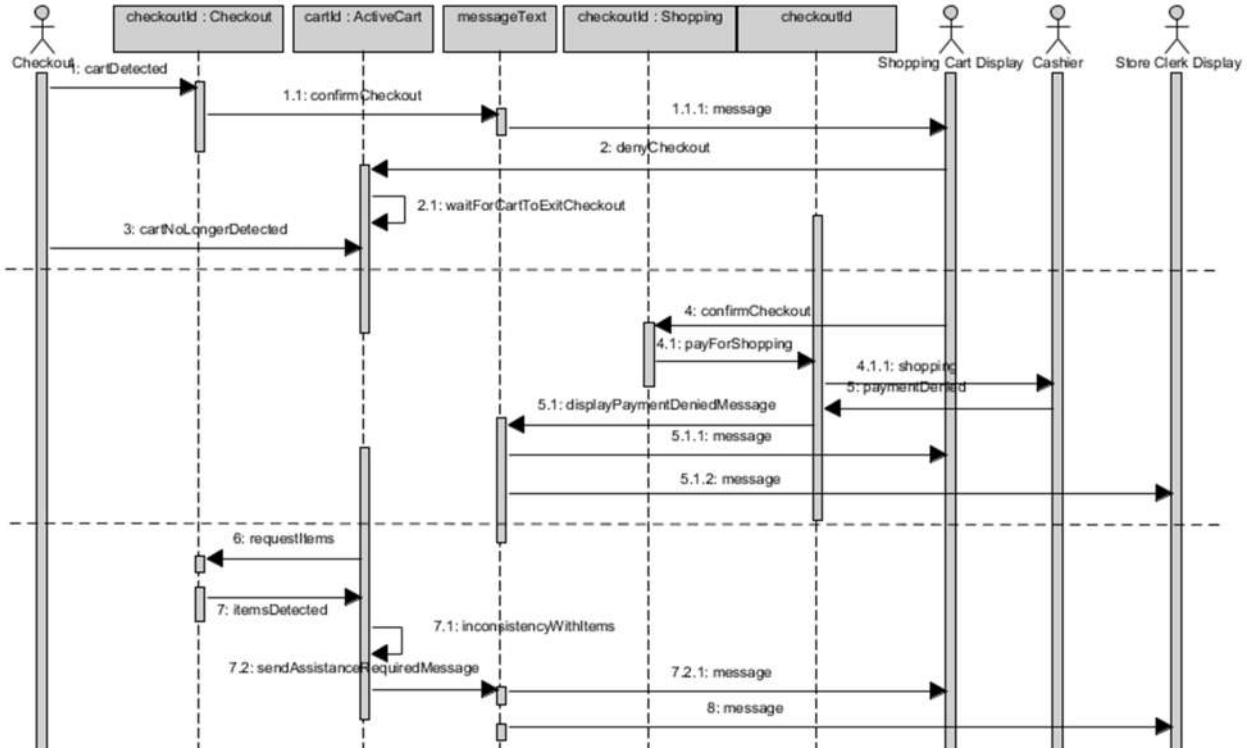

Figure 14: Checkout Is Incomplete Logical Interaction Diagram

Shopper denies checkout.

1. The Checkout object is receives a message that the cart has reached the checkout are.
2. The Checkout object sends a message to the shopper requesting that they confirm they wish to checkout.
3. The ActiveCart object receives a message from the Cart Display that the shopper does not wish to checkout.
4. The ActiveCart object waits for the cart to leave the checkout area.
5. The ActiveCart object receives a messae from the Checkout that the cart has left the checkout area.

Cashier denies payment for the items in the cart.

1. The Checkout object receives a message from the Cart Display confirming checkout of the cart.
2. The Checkout object sends a message to the Items object requesting checkout of items in the cart.
3. The Items object sends a list of items in the cart to the Cashier.
4. The Cashier denies that payment was received for the items in the cart.
5. The Items object sends a message to requesting checkout assistance.
6. The Message object sends a message to the Cart Display for the customer to wait for assistance.
7. The Message object sends a message to the StoreClerk Display requesting assistance for the shopper.

Items at checkout are different to items in cart.

1. The Checkout object sends a request to the checkout for the number of items detected passing through the checkout.
2. The Items object receives the detected items from the Checkout with a request to compare them against the items in the cart.
3. The Items object confirms the detected items do not match those that are in the cart.
4. The Items object sends a message to requesting checkout assistance.
5. The Message object sends a message to the Cart Display for the customer to wait for assistance.
6. The Message object sends a message to the StoreClerk Display requesting assistance for the shopper.

2 Summary

The purpose of this exercise is to prove that the logical model is complete. It demonstrates that all data and all functionality required by the use cases, has been captured by classes (Sufficiency).

What this exercise does not demonstrate is that all data and operations in the logical is required by the use cases (necessity). The reader may however, execute each sequence diagram in turn, and show which class data and operations are used in the use case realizations. Any unused data or operations unused may be considered as extraneous.

Note that any messages passing from 1 class instance to another are not externally visible, and hence do not form part of the requirements. They are used to demonstrate that and externally visible event causes the execution of an externally visible action.

The externally visible events and actions ARE the functional requirements.

This example concludes the analysis model. The information shown here may in some cases be more than sufficient for the average software development project. Especially if the team is proving the requirements by demonstration. For example, a typical Agile project may work directly from the use cases and consider formalizing of requirements unnecessary. Other, high quality projects for example, may consider this process to be not formal enough and want the requirements spelt out as single manageable lines of text with traceability between them.

The objective of these articles was to demonstrate the capabilities of using models to create a complete, consistent, sufficient and (possibly) necessary set of software requirements.

3 Credits

Visual Paradigm – used to draw diagrams, model the business and systems analysis.