Institute of Systems Science

Design Model Report for vmcs

Issue 4.0

Revision History

Date	Issue	Description	Author
15 September 2000	1.0	First Issue	Howard Russon
13 September 2001	2.0	Revised for MTech Unit 3 in 2001	Howard Russon
14 August 2003	3.0	Revised for MTech Unit 3 in 2003	Howard Russon
23 July 2004	4.0	Revised for MTech Unit 3 in 2004	Howard Russon

Table of Contents

Design Model Report for vmcs

1. Introduction

The Vimto Soft Drinks Company operates and maintains a large number of automatic soft drinks dispensers (vending machines) at various sites throughout Singapore. Vimto requires a computerised control system for their new range of vending machines. For this part of the project, the system is to be designed and implemented as a computer simulation of a vending machine. The main requirements for the system are defined in "User Requirements Specification for the Vending Machine Control System (VMCS) Computer Simulation", reference ISS/VMCS/TR.1/1. The system is to be developed using object-oriented techniques and the Java programming language.

The main users of the vending machine are the:

1. The **Customer** who enters Coins, selects a Drinks Brand, and collects a drinks-can from the machine.
2. The **Maintainer** who monitors the stock levels of cans of each Drinks Brand and Change, and replenishes or removes stock when necessary.

In the VMCS computer simulation, which this project principally addresses, it is not possible to interface with the actual vending machine and its sensors. Hence the user panels of the vending machine, and the physical actions of the Maintainer have to be simulated. The main device chosen to achieve this is through an actor, referred to as the Controller. The sequence of the use cases which are typically performed in the system are described in the following subsections.

1.1 System Set-Up

The controller has the responsibility to set-up the user interface so that the system can be used by the other actors as follows:

1. The use case System Start-up & Close-Down provides the Simulator Control Panel from which the system can formally be switched on or off, and from which each of the other user panels can be activated and displayed.
2. When the system simulation is started, the use case Set-Up panels is used to set-up the panels. This use case determines the drinks and cash stocks in the machine and displays them appropriately.

1.2 Customer Operations

The functions that the Customer can perform with the vending machine user interface (Customer Panel) are undertaken through the use case Coin Input and Select Brand. This uses separate abstract use cases to handle its main interactions with the hardware elements of the vending machine. That is, it uses the following use cases:

1. Dispense Drink;
2. Give Change; and
3. Refund Money.

1.3 Maintenance Operations

The functions required to maintain stocks of drinks cans and change in the vending machine are carried out by two actors: the Maintainer, who performs user interface operations which will be provided in the *real* vending machine; the Controller who performs simulated functions which will not be computerised in the *real* machine.

The Maintainer's functions are provided through the use case System Maintenance. The Maintainer works in collaboration with the Controller and use case Change State as follows:

1. Using the Maintenance Panel, the Maintainer can review stock levels.
2. This also uses use case Transfer All Cash to enable the Maintainer to collect all the cash inside the vending machine.
3. Using the Machinery Simulator Panel, the Controller can now change (add/remove) stocks.
4. When the Maintainer/Controller have completed their tasks, the Customer Panel displays are update.

2. Transition from Analysis to Design

2.1 Introduction

The design of the Vending Machine Control System (VMCS) is based upon the robustness of the system defined in the "Use Case Realization Report (Analysis)" documents for each use case (nine documents filed at ISS/VMCS/TR.3). The use case structure is defined in the Use-Case-Model Survey (reference ISS/VMCS/TR.2/1), and the requirements for each use case are defined in the "Use Case Realization Report (Requirements)" documents for each use (nine documents filed at ISS/VMCS/TR.2).

The changes that have been introduced to transition from analysis to design are discussed in the following subsections in terms of:

- the new objects (and object structures) that have been introduced);
- changes to the dynamic behaviour;
- object creation; and
- other design issues.

2.2 New Design Objects

New objects have been introduced into the system during the design phase to facilitate the generalising of the storage objects (Cash Store and Drinks Store) and the implementation of the user interface. These are described in the following subsections.

2.2.1 Store Initialization Objects

The Cash Store and the Drinks Store are initialized at system start-up by reading-in their initial values from property files (one file for each type of store). The data are structured as key and value pairs. Examples of property files for the Drinks Store and Cash Store shown below:

Drinks Store Property File	**Cash Store Property File**

`NumOfItems=5`
 	
`Name5=Coca-Cola`
`Name4=soya Bean`
`Name3=Sarsi`
`Name2=Fanta`
`Name1=Coca-Cola`
 	
`Price5=80`
`Price4=80`
`Price3=60`
`Price2=10`
`Price1=80`
 	
`Quantity5=0`
`Quantity4=20`
`Quantity3=26`
`Quantity2=40`
`Quantity1=41`
	`NumOfItems=5`
	
	`Name5=$1`
	`Name4=50C`
	`Name3=20C`
	`Name2=10C`
	`Name1=5C`
	
	`Value5=100`
	`Value4=50`
	`Value3=20`
	`Value2=10`
	`Value1=5`
	
	`Quantity5=0`
	`Quantity4=6`
	`Quantity3=0`
	`Quantity2=0`
	`Quantity1=0`
	
	`Weight5=100.0`
	`Weight4=50.0`
	`Weight3=20.0`
	`Weight2=10.0`
	`Weight1=5.0`

To read-in these files and initialize the store objects, two new control objects, CashPropertyLoader and DrinkPropertyLoader, have been introduced, each inheriting from a new object, PropertyLoader, as shown in section 4.4.3.

2.2.2 Storage Objects

The Cash Store and the Drinks Store have been given the same structure by introducing the Cash Store Item object as shown in section 4.4.1.

As result, three generic objects have been introduced: Store, Store Item and Store Object. The objects at each level of the Cash Store and Drinks Store structures can then inherit from these objects as shown in section 4.4 2.

By creating this new structure, Cash Store Item and Drink Store Item have identical attributes and behaviour, and can be implemented as instances of Store Item.

2.2.3 Machinery Controller

The packaging structure determined in the design phase (see Section 4.1 for details) uncovered some structural issues in the analysis madel thaty had to be resolved. We needed to package the Store functionality as a separate package (to facilitate both testing and also deployment of the system on the target hardware), but we found that the store functionality was too tightly coupled with the machinery simulation functionality. In particular, the Store controller managed both the Cash Store and Drinks Store objects, as well as the Machinery Simulator Panel, thus merging the two subsystem.

In order to decouple the two packages, we split the Store Controller into two separate control objects:

- The MachineryController class, which manages the machinery simulation functionality. It controls the Machinery Simulation Panel and its component, as well as the Door object.
- The StoreController class, which manages the stores separately. It controls the Cash Store and Drinks Store.

Both controllers are created and managed by the Main Controller (see section 2.5.2).

There were some methods (namely storeCoin(), dispenseDrink() and giveChange()) which involved both controllers: they execute a store operation, which is then displayed by the Machinery Simulation Panel. In these cases, it was decided that both controllers should provide the method; clients objects would invoke the MachineryController object's implementation of the method, which would invoke the Store Controller implementation and subsequently refresh the Machinery Simulation Panel.

An additional issue that emerged was the use of Property File loaders (see section 2.2.1). If the store functionality had to be packaged separately from the machinery simulation, it should also have no knowledge of the file storage mechanisms. As a result, it could later on be deployed in a different type of environment. This issue was resolved by creating an abstract interface PropertyLoader with the basic methods of property loaders: initialize(), saveProperty(), getNumOfItems(), setNumOfItems(), getItem(), setItem(). This is included in the store package, and is the only type the StoreController class is aware of. File-based implementations (FilePropertyLoader, CashPropertyLoader, DrinksPropertyLoader) are then included in the system functionality package, so they are instantiated by the MainController, which passes them to the StoreController at object creation time. A side benefit of this structure is that the StoreManager no longer needs a reference to the MainController, thus guaranteeing the independence of the store package.

2.2.4 User Interface Objects

Java Class Library objects from the AWT package are used to provide the majority of the behavior of the user interface panel. In particular, Frame, Panel and Dialog are used to provide the basic *windowing* functionality through inheritance. Each panel is constructed from button and text field components, by using the AWT objects, Button and TextField. In addition, three generic objects have been introduced (that make use of Button and TextField) to provide common user interface functionality. These are: LabelledDisplay, Warning Display, ButtonItemDisplay and ButtonItem. See sections 4.3.1, 4.3.2 and 4.6.1 for further details.

To control the main behaviour of the Customer Panel, two new objects have been introduced as components of the panel: Coin Input box and Drink Selection Box. The Drink Selection Box is further decomposed into an array of Drink Selection Item objects. The remainder of the components of the Customer Panel are instances of Button, Warning Display and LabelledDisplay. See sections 4.5.1 and 4.5.2 for further details.

To control the main behaviour of the Maintenance Panel, two new objects have been introduced as components of the panel: Coin Display and Drink Display, each of which make use of ButtonItemDisplay. The remainder of the components of the Maintenance Panel are instances of Button, ButtonItem, Warning Display and LabelledDisplay. See sections 4.6.1 and 4.6.2 for further details.

To control the main behaviour of the Machinery Simulator Panel, a new object, StoreViewer has been introduced. Two instances of this object are used as the main components of the panel: the Cash Store Display and the Drink Store Display. In addition, an AWT object, Checkbox, is used to present and change the Door Status. See section 4.7.1 for further details.

The Simulator Control Panel is constructs from five instances of the Button object. See section 4.7.1 for further details.

2.2.5 User Interface

To implement commands being issued from the user interface (ie: buttons being pressed or data being entered into text fields) it is necessary to tie these commands to call-back operations. This can be achieved in Java by using the ActionListener library object (from EVENT package).

Hence new listener objects have been introduced for each button etc (which inherit the ActionListener object). The interface object will trigger the listener on every user input. In this way, the interface is decoupled from the application dependent control object which carries out the user request - all the interface needs to know is the identity of its listener.

Hence the following listener objects have been introduced:

- Activate Customer Panel Button Listener
- Activate Machinery Simulator Panel Button Listener
- Activate Maintainer Panel Button Listener
- Begin Simulation Button Listener
- Coin Display Listener
- Coin Input Listener
- Door Listener
- Drink Display Listener
- Drink Selection Listener
- End Simulation Button Listener
- Exit Button Listener
- Password Listener
- Price Display Listener
- StoreViewer Lister
- Terminate Button Listener
- Total Cash Button Listener
- Transfer Cash Button Listener

2.2.6 VMCSException

As several classes needed to throw exceptions, a special VMCSException was added to the util package to support this need.

2.3 Changes to Dynamic Behaviour

Various changes to the dynamic behavior of the system have been made, principally as a result of introducing the new objects described above and changing the responsibilities of certain objects. These changes are summarised in the following subsections. The revised dynamic behaviour of the system can be seen in the Sequence Diagrams.

2.3.1 New Objects

The changes that have been introduced as a result of introducing each type of new object are described below.

2.3.1.1 Store Objects

With the introduction of the 3-layer structure for the Cash Store and the Drinks Store, a set of generic operations are being being used to read and update the stores, and to access specific Coins and Drinks Brands. These functions rely heavily on the following operations of Store Item:

- Increment : BOOLEAN
- Decrement : BOOLEAN
- GetContent : STORE OBJECT
- GetQuantity : INTEGER
- SetQuantity
- SetContent

The greatest effect of this change is with respect to the Cash Store, which now is required to access Cash Store Items to locate specific Coins. Hence some of the responsibilities assigned to Coin in analysis and been moved to Cash Store Item.

2.3.1.2 Initialization and Saving of Store Object Attributes

To facilitate the initialization of the Cash Store and Drinks Store, use case System Start-Up and Close-Down has been changed. In particular, the Store Controller has new operations to interact with the CashPropertyLoader and DrinkPropertyLoader to read-in the data and then initialize the attributes Cash Store, Drinks Store and their contained objects.

2.3.1.3 Listeners

As a result of introducing the listener objects, user interface objects used for user input (ie: buttons and input text fields) now send the user input to their listener object, and communicate with no other objects. The listener objects then send the input to the appropriate application control object for processing.

Control is then assumed by the application control object, and the listener plays no further part in controlling the transaction.

2.3.2 Changes to Responsibilities

Various changes have been made to the responsibilities of objects, and hence to their dynamic behaviour. These are described below.

2.3.2.1 Store Controller

In the analysis, the Store Controller has the role of managing the Change State, but also participates in several other use cases to update the Machinery Simulator Panel as a result of changes to the Drinks Store and Cash Store. Given the Store Controller was involved on every occasion that store updating takes place, it was decided to avoid repetition of operation calls by also giving the Store Controller the responsibility for making any changes to the Drinks Store and Cash Store. This approach also has the advantage of protecting the stores from unauthorised change by only allowing one object to make changes. The use cases effected by this change are :

- Coin Input and Select Brand
- Dispense Drink
- Give Change

2.3.2.2 Change Giver

In the analysis, the Change Giver coordinated the giving of change, but the intelligence to determine how to give the change (ie: which Coins to issue) resided in the Cash Store.

It was decided that it is inappropriate to have this division of responsibilities as this activity is essentially control behaviour, and hence belongs in the Change Giver. In addition, change giving would appear to make the Cash Store application dependent.

Hence the Give Change use case has been modified to move the decision making to the Change Giver.

2.3.2.3 Cash Store

In the analysis, there was an implicit assumption that the Cash Store (and it component object, Coin) would contain operations and attributes to store total amounts of cash, change availability etc. In the design it has been decided to remove these responsibilities from the Cash Store, and require the application control objects (that require these information) to call appropriate operations of Cash Store, Cash Store Item and Coin to calculate the values when required. In this way, the Cash Store is less application dependent. This change has a minor effect on the Customer transaction use cases.

2.3.2.4 User Interface Update

The updating of the user interface panels has been rationalized to ensure that it is controlled through the *responsible* control objects, as follows:

1. The Simulation Controller manages interactions with the Simulator Control Panel.
2. The MaintenanceController manages interactions with the Maintenance Panel.
3. The Store Controller manages interactions with the Machinery Simulator Panel.
4. The Transaction Controller manages interactions with the Customer Panel at the highest level, but passes on control to the Dispense Controller, Coin Receiver and Change Giver to facilitate updates to the parts of the panel that these controllers have specific responsibility for.

In some cases we have found that the analysis model could be simplified. In the StoreViewer, for example, the individual store items were updatable, which meant that the invoking code had to pass indices. In reality, the overhead of updating all of the StoreViewer items (rather than a single one) is negligeable, so a simplified update() method that does that was implemented instead.

2.3.2.5 Transfer Cash Controller

In the analysis, the Transfer Cash Controller was used to control the Transfer All Cash use case. However, this object had very limited functionality. Hence, its operations have been merged into the Maintenance Controller, and Transfer Cash Controller has been removed.

2.3.3 Other Changes

Changes have been made to the operations of the user interface objects as follows:

- Each user interface object in analysis had operation Activate and Deactivate. In design, these pairs of operations have replaced in each case by one operation - SetActive - which has a boolean parameter.
- The interface objects that receive user input (buttons and input text fields) no longer have an operation executing the user request. This is transferred to the listener objects.
- The operations of interface objects used for display purposes have been rationalised to avoid a multiplicity of operations each with only minimal functionality.

2.4 Changes to the Static Structure

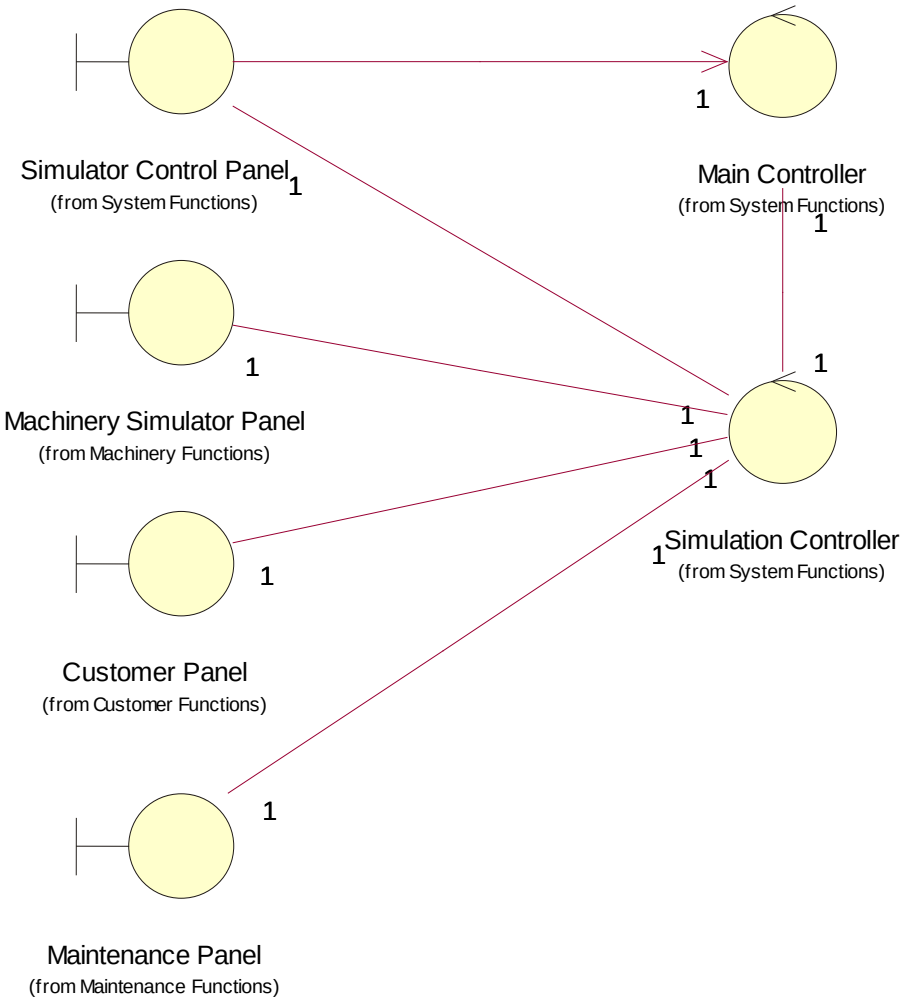
The analysis model included several diagrams showing how objects reference each other statically (i.e. showing references that are permanent, rather than those that exist only for the purpose of sending a message for a particular use case). These static models show the organization of the classes, and it is an objective of good design to make these structure as "clean" as possible. A class diagram that has many overlapping associations often reveals poor

structure, which results in complex and error-prone implementation. When transitioning to design, it was observed that the static structure could be improved in certain areas, described in the following subsections.

2.4.1 Start-up and Shut-down of Panels

In the analysis model, the Customer Panel, Machinery Panel and Maintenance Panel were created and displayed by the Simulation Controller, but closed down by the Main Controller. This was probably an oversight due to different analysts being in charge of different sub-use cases. In the design phase, we restructured the static relationships, making Simulation Controller the sole controller responsible for the panels. The new structure is shown in section 2.4.1.1.

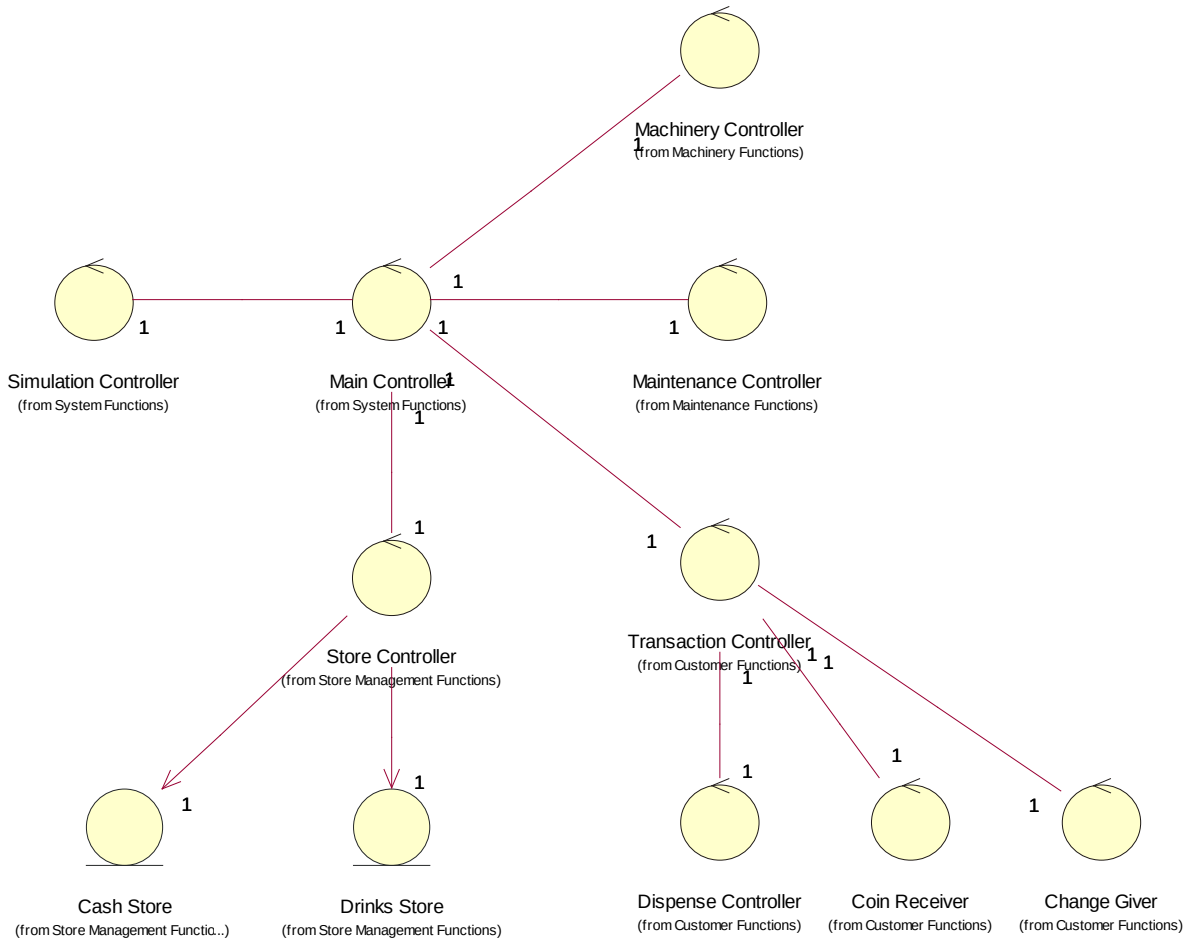
2.4.1.1 Class Diagram: User Interface Control Structure



2.4.2 Controllers

In the analysis model, the Main Controller acted as a coordinator, and interacted with all the use case controllers, as is common in RUP-style object oriented analysis and design. It was noted however that, for the sizeable use case "Coin Input and Select Brand", the main coordinator should be Transaction Controller, which manages the interactions with ChangeGiver, Dispense Controller and Coin Receiver. The associations between the controllers were therefore restructured as shown in section 2.4.2.1.

2.4.2.1 Class Diagram: Controllers Structure



2.5 Object Creation

The scheme for creating the objects of the system and for ensuring that all objects have references to the objects to which require to communicate is described in the following subsections.

2.5.1 Object Constructors

Each object is created by a constructor operation. This operation has the same name as the object in Java language. These operations have been added to each object.

2.5.2 Control Objects

The constructor of the Main Controller is called is called when the application is launched (in use case System Start-Up and Close-Down). The Main Controller is then instructed to constructs all the control objects of the system. The Transaction Controller, as the main controller for customer transactions, creates the other controllers involved in customer transactions (Dispense Controller, Coin Receiver and Change Giver) when its constructor is called. Similarly, the Maintenance Controller creates the Access Manager when its constructor is called.

The Main Controller, Transaction Controller and the Maintenance Controller pass a reference to themselves to each object that they create. Hence, using these references, all control objects can reference each other if required.

2.5.3 Entity Objects

The constructor of the Store Controller creates the Cash Store, Drinks Store and Door. As all control objects have a reference to the Main Controller or Transaction Controller, they can access the Store Controller, and hence can gain access to the Cash Store, Drinks Store and Door when required.

The constructor of the Access Manager creates the Password object.

2.5.4 Interface Objects

The main panels (ie: Customer Panel, Machinery Simulator Panel, Maintenance Panel and Simulator Control Panel) are created by the control object that displays them (ie: in use cases System Start-Up & Close-Down and Set-Up Panels).

When a panel is created, it is passed a reference to the control object that is responsible for updating the panel. The constructor of each panel then creates its sub-objects.

The constructor of each user interface object that accepts user input (buttons and input text fields) then creates a listener object and passes the reference to its controller, and other relevant information. This is used in the implementation of the *actionPerformed* operation.

2.6 Other Design Issues

When the application opens (ie: in use case System Start-Up and Close-Down) the VMCS object creates the Main Controller, which then performs all the initialization tasks. As part of these tasks, an *environment properties file* is read to determine the names/locations of the properties files containing the Cash Store and Drinks Store initialization information.

3. Sequence Diagrams

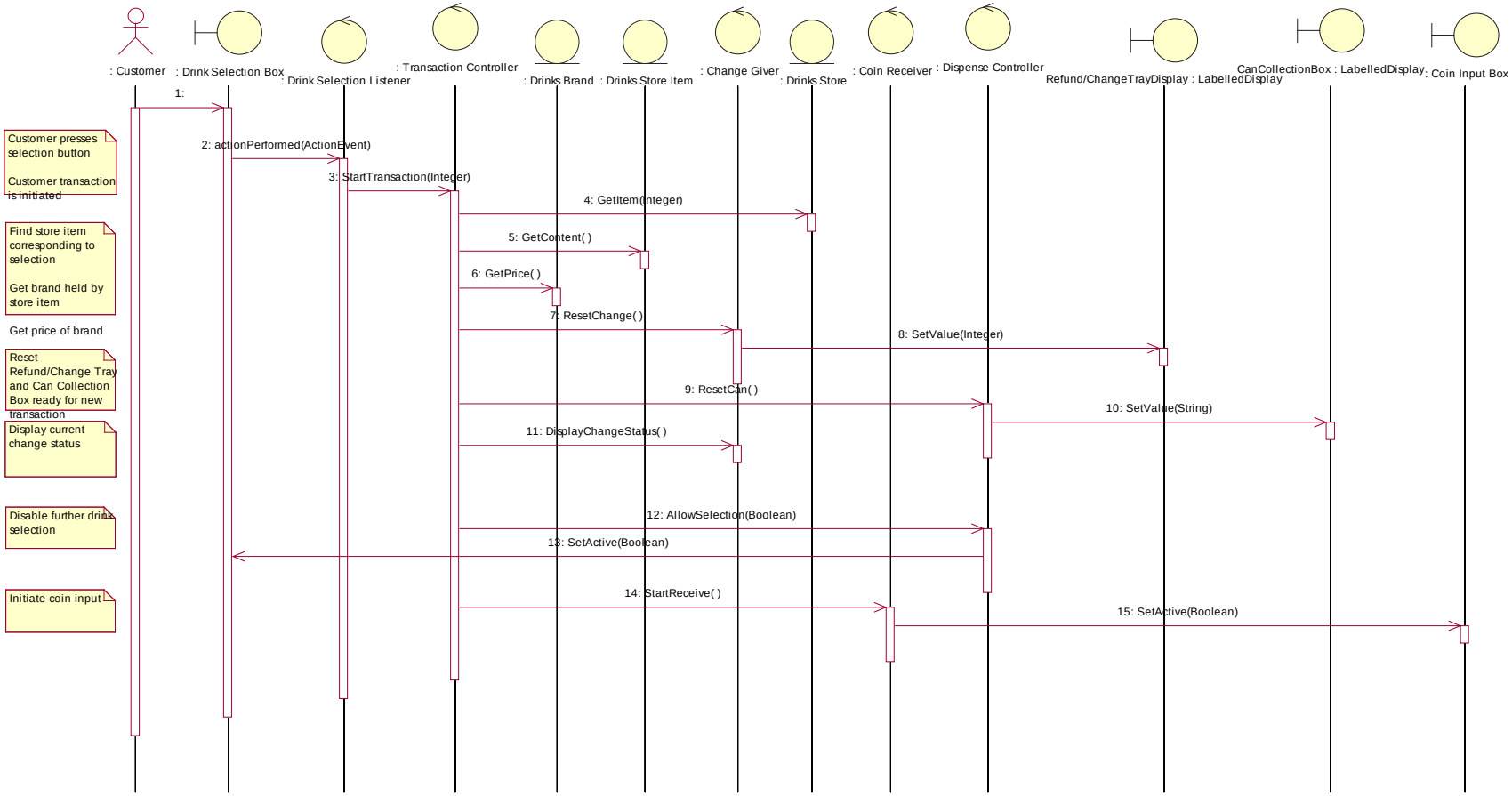
3.1 Introduction

Sequence diagrams representing the dynamic behaviour of the objects of the system are presented in the following subsections. They are organised according to the nine use cases of the system:

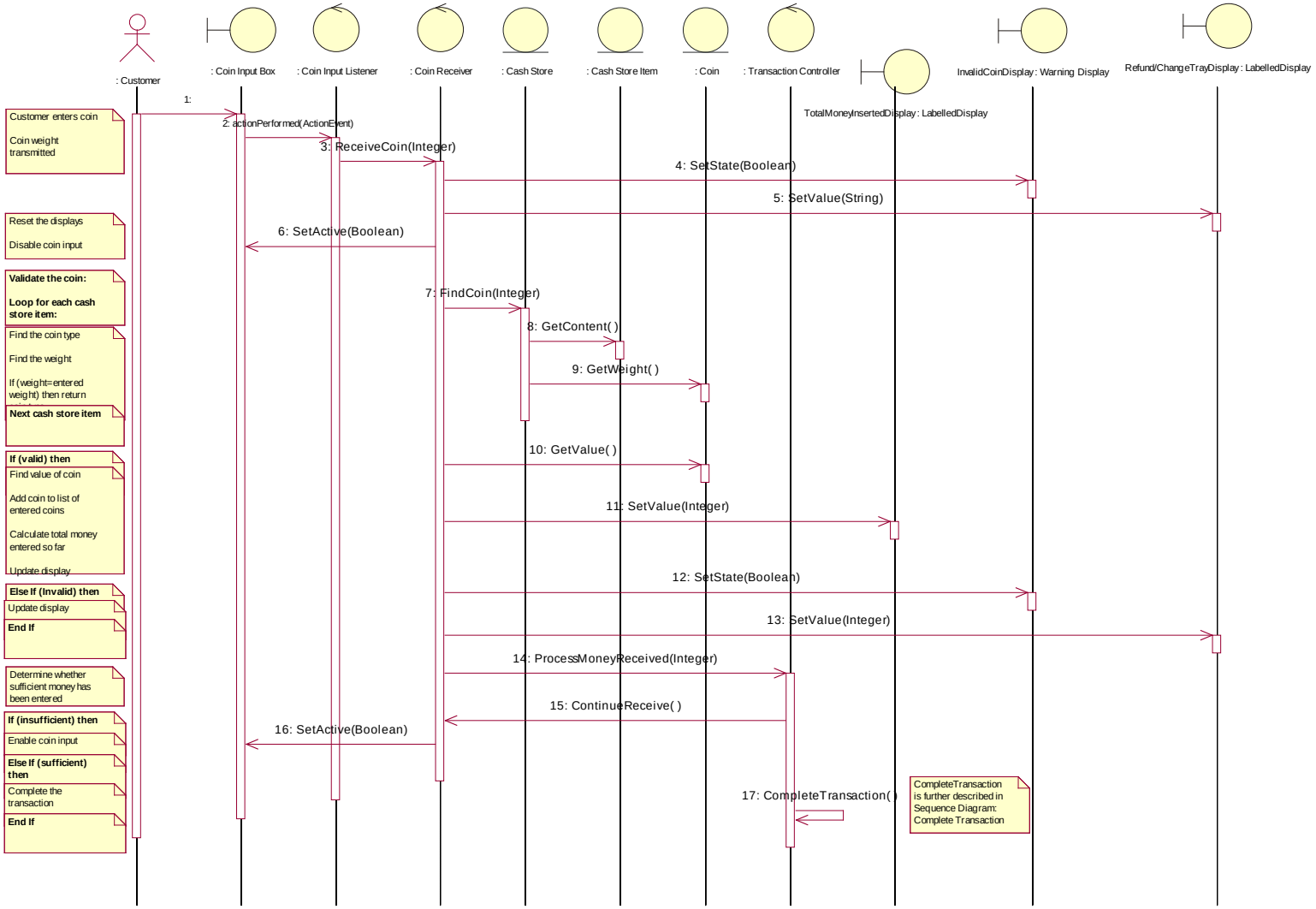
- Coin Input and Select Brand.
- Dispense Drink.
- Give Change.
- Refund Money.
- System Maintenance.
- Transfer All Cash.
- System Start-Up & Close-Down.
- Set-Up Panels.
- Change State.

3.2 Coin Input and Select Brand

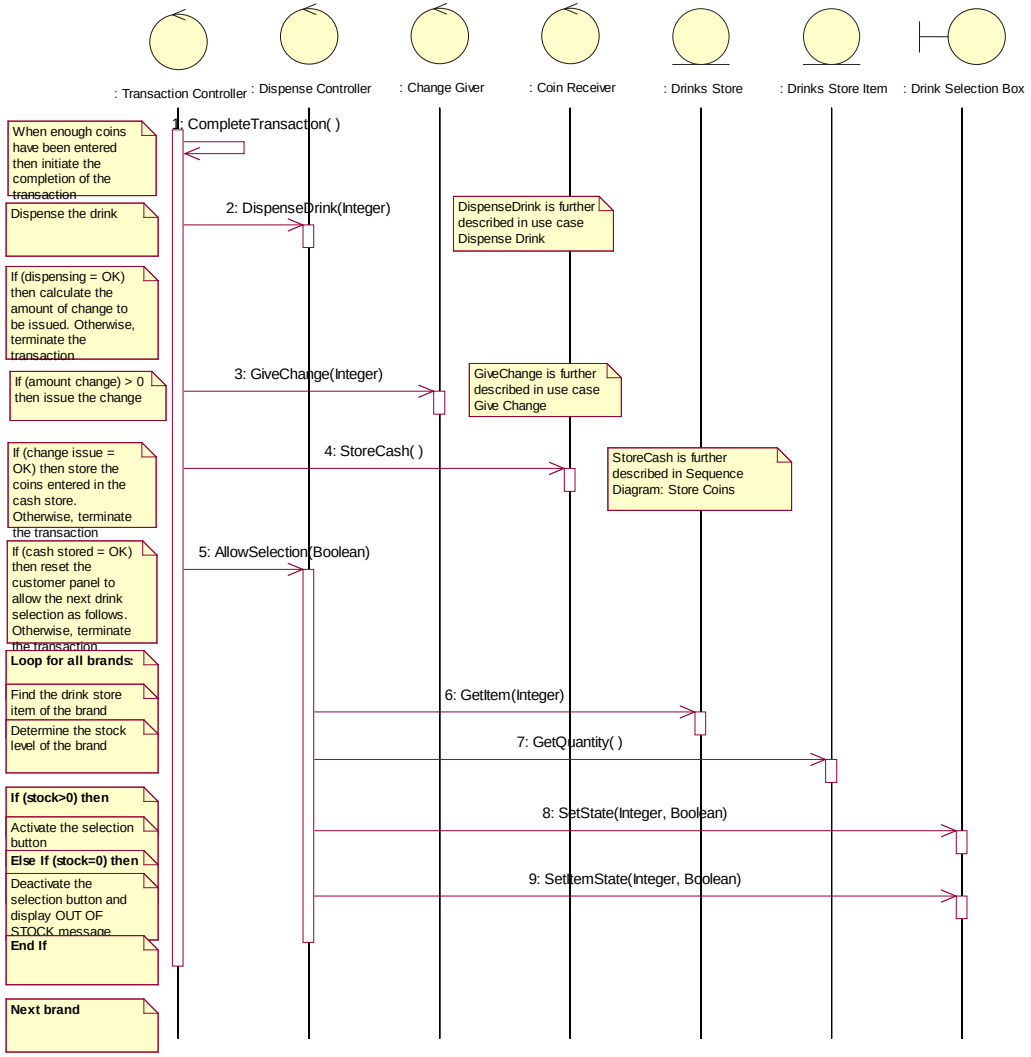
3.2.1 Sequence Diagram: Sequence Diagram: Select Drinks Brand



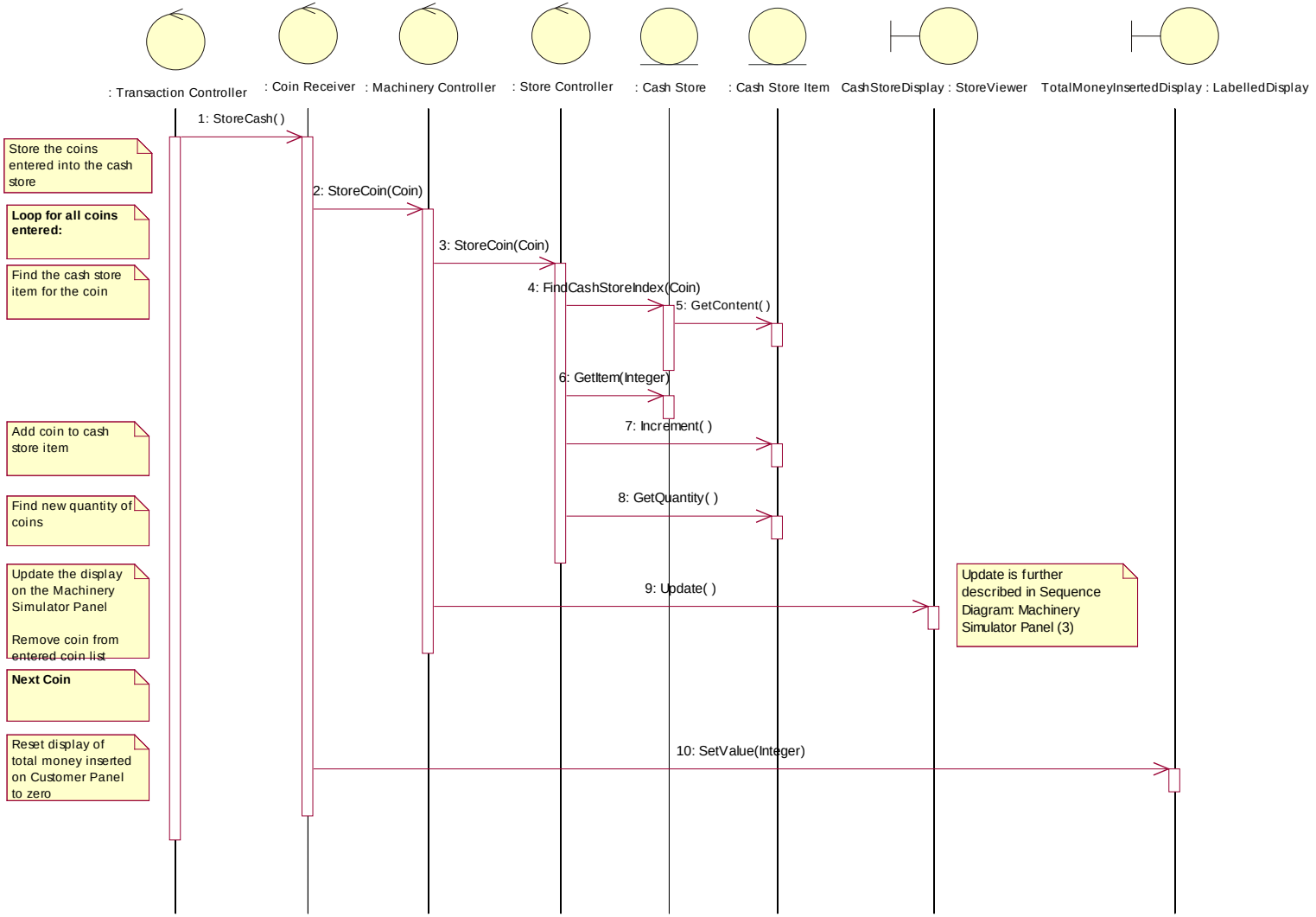
3.2.2 Sequence Diagram: Sequence Diagram: Enter Coins



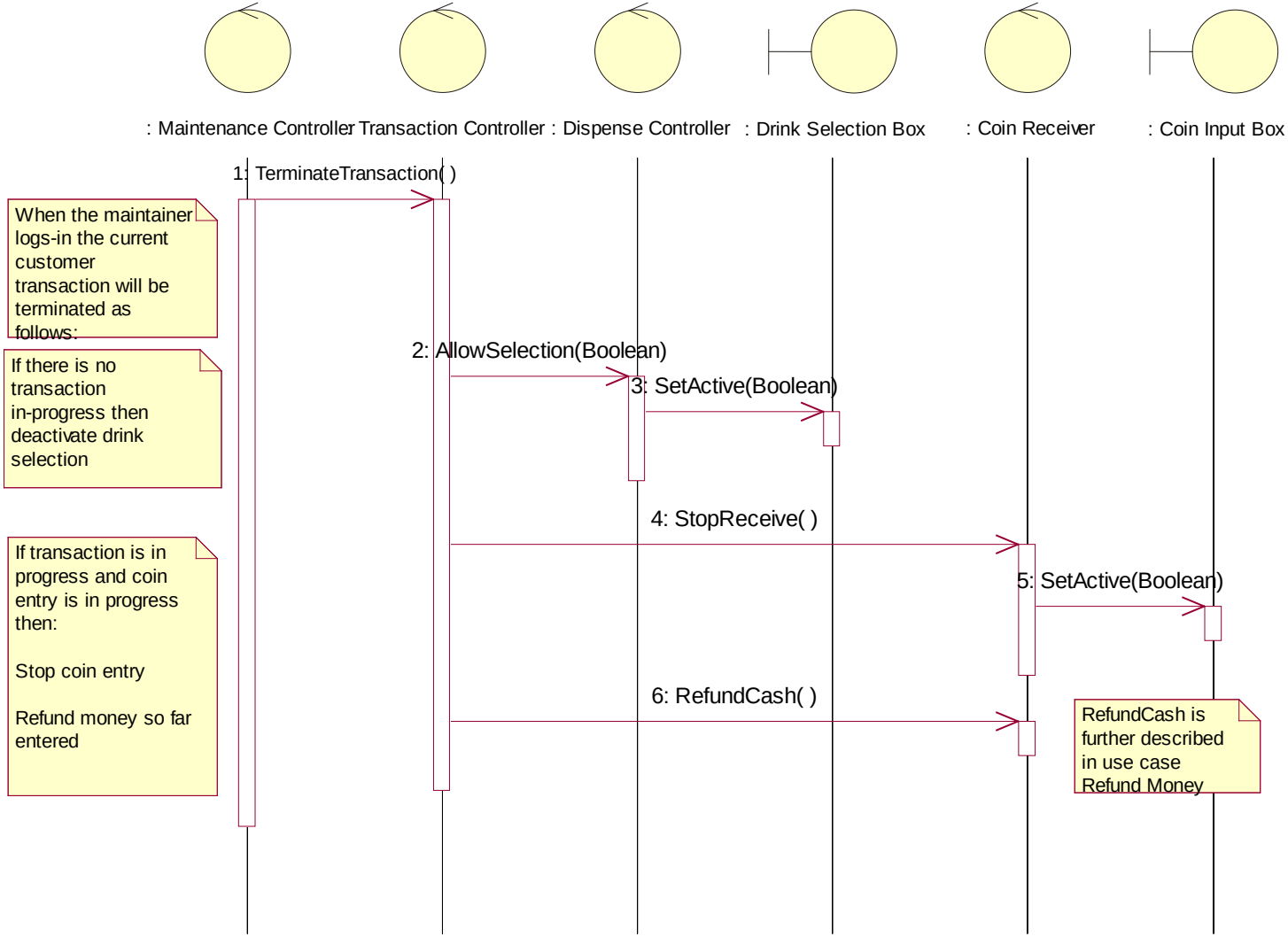
3.2.3 *Sequence Diagram: Sequence Diagram: Complete Transaction*



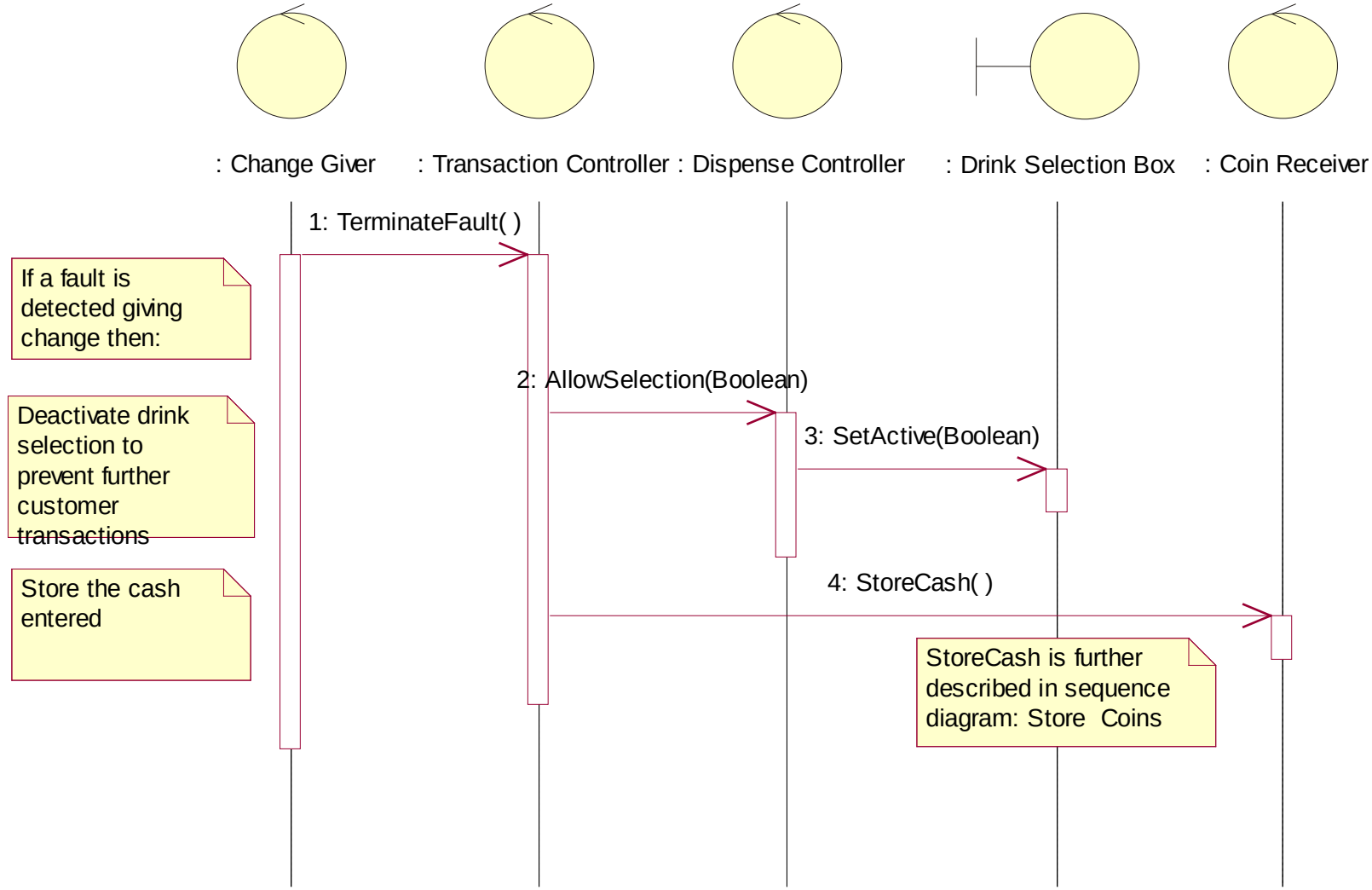
3.2.4 Sequence Diagram: Sequence Diagram: Store Coins



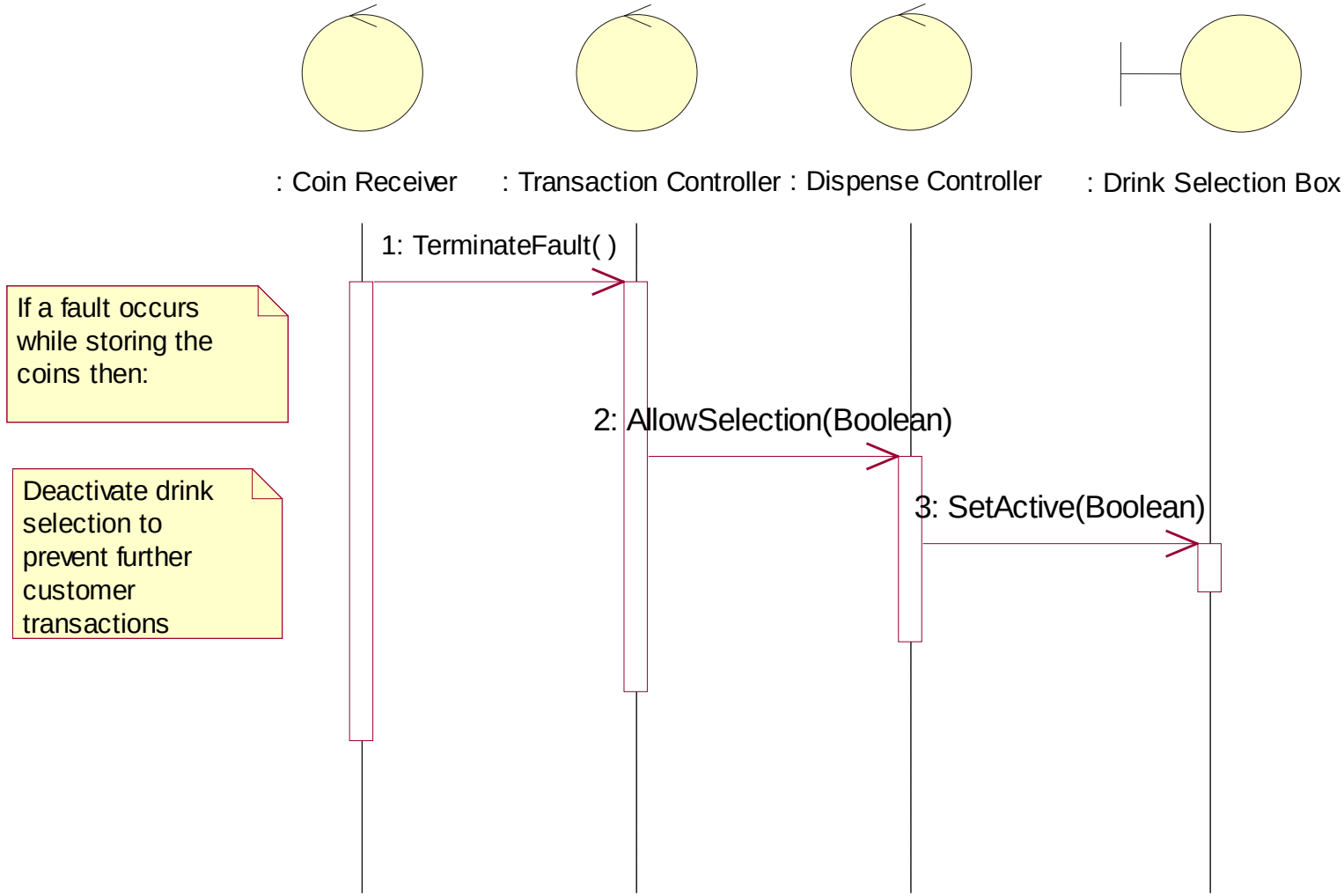
3.2.5 Sequence Diagram: Sequence Diagram: Terminate (Maintainer Logs-in)



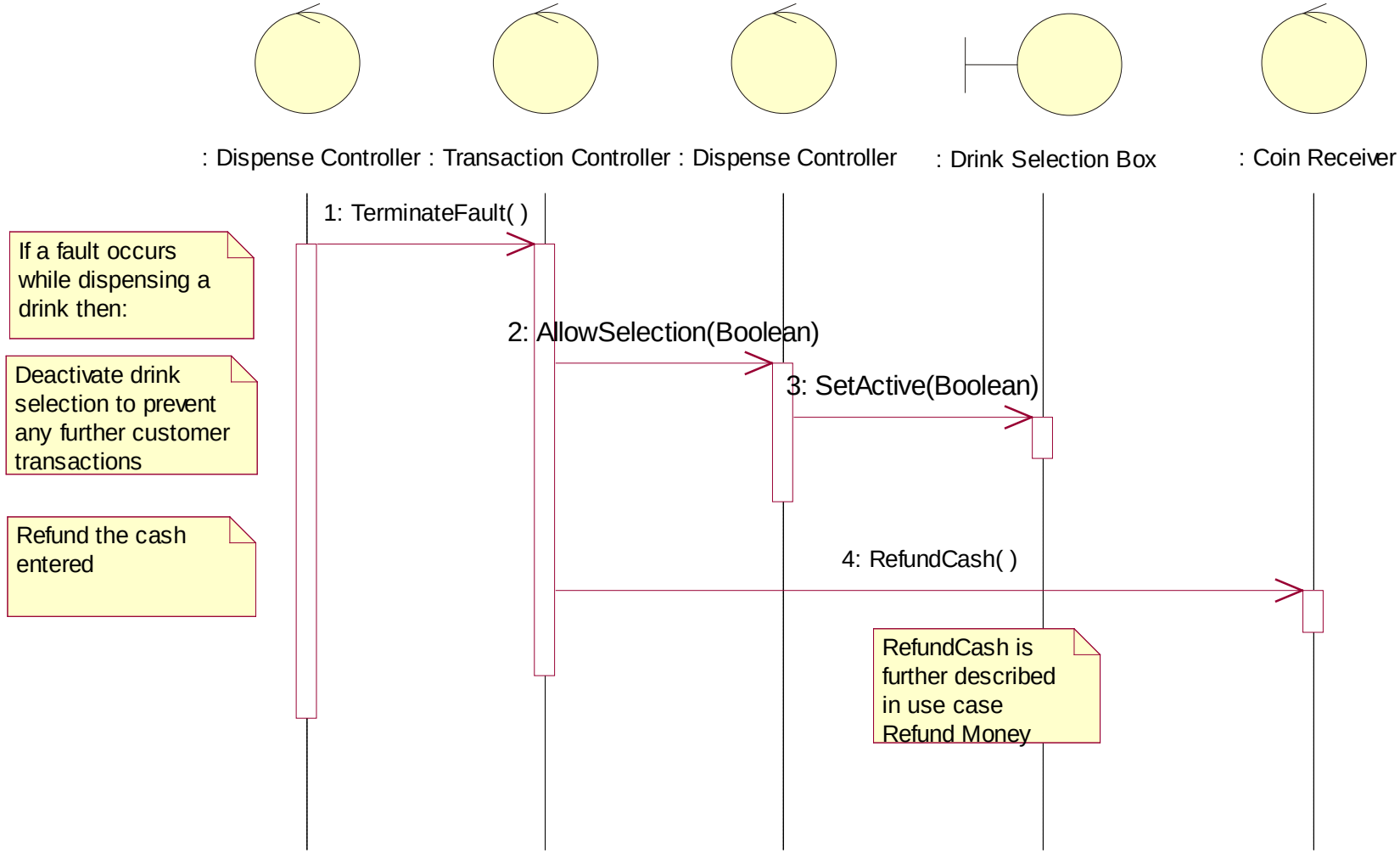
3.2.6 Sequence Diagram: Sequence Diagram: Terminate (Fault is Detected - Change)



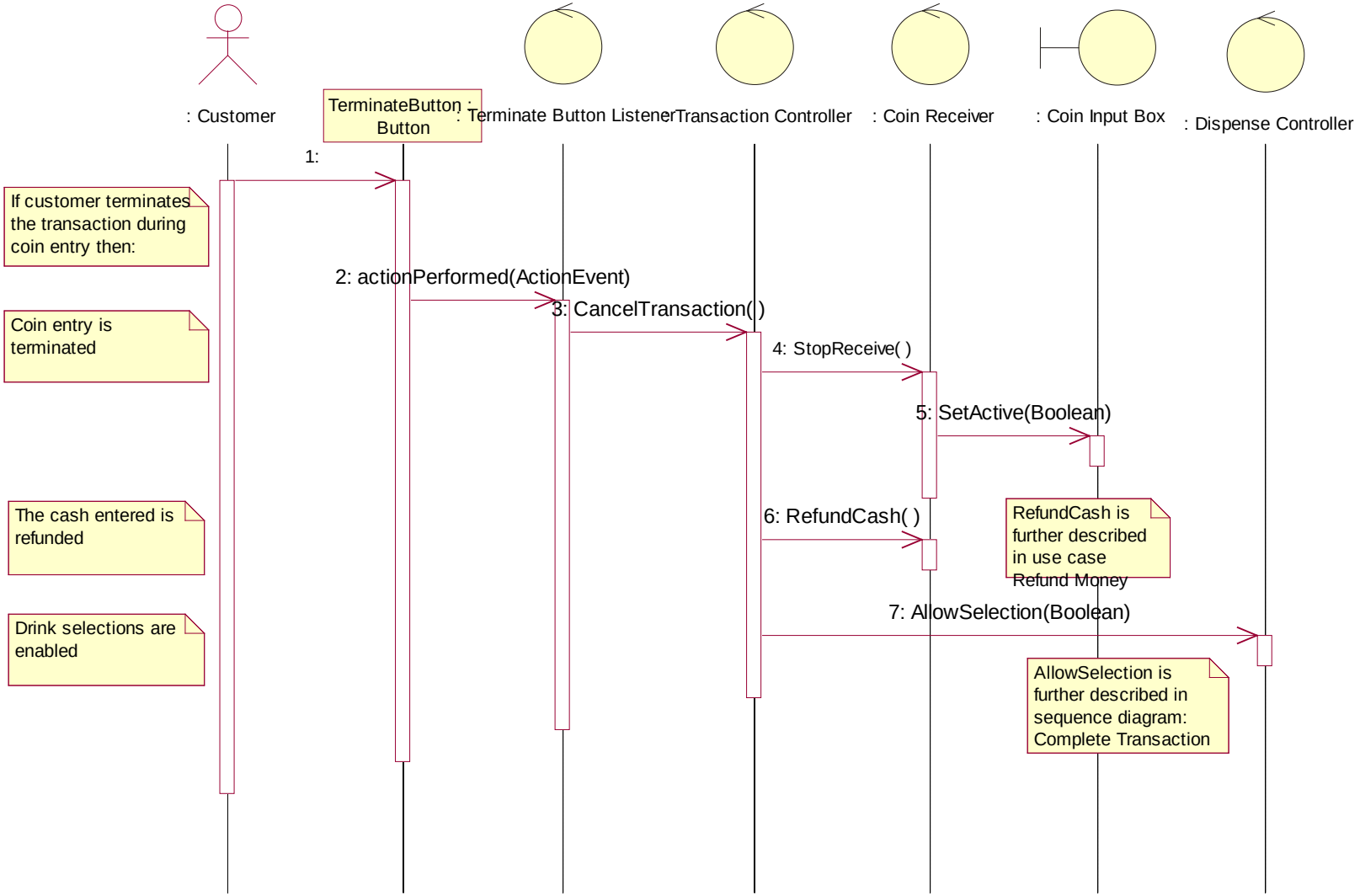
3.2.7 Sequence Diagram: Sequence Diagram: Terminate (Fault is Detected - Storing Coins)



3.2.8 Sequence Diagram: Sequence Diagram: Terminate (Fault is Detected - Dispense)

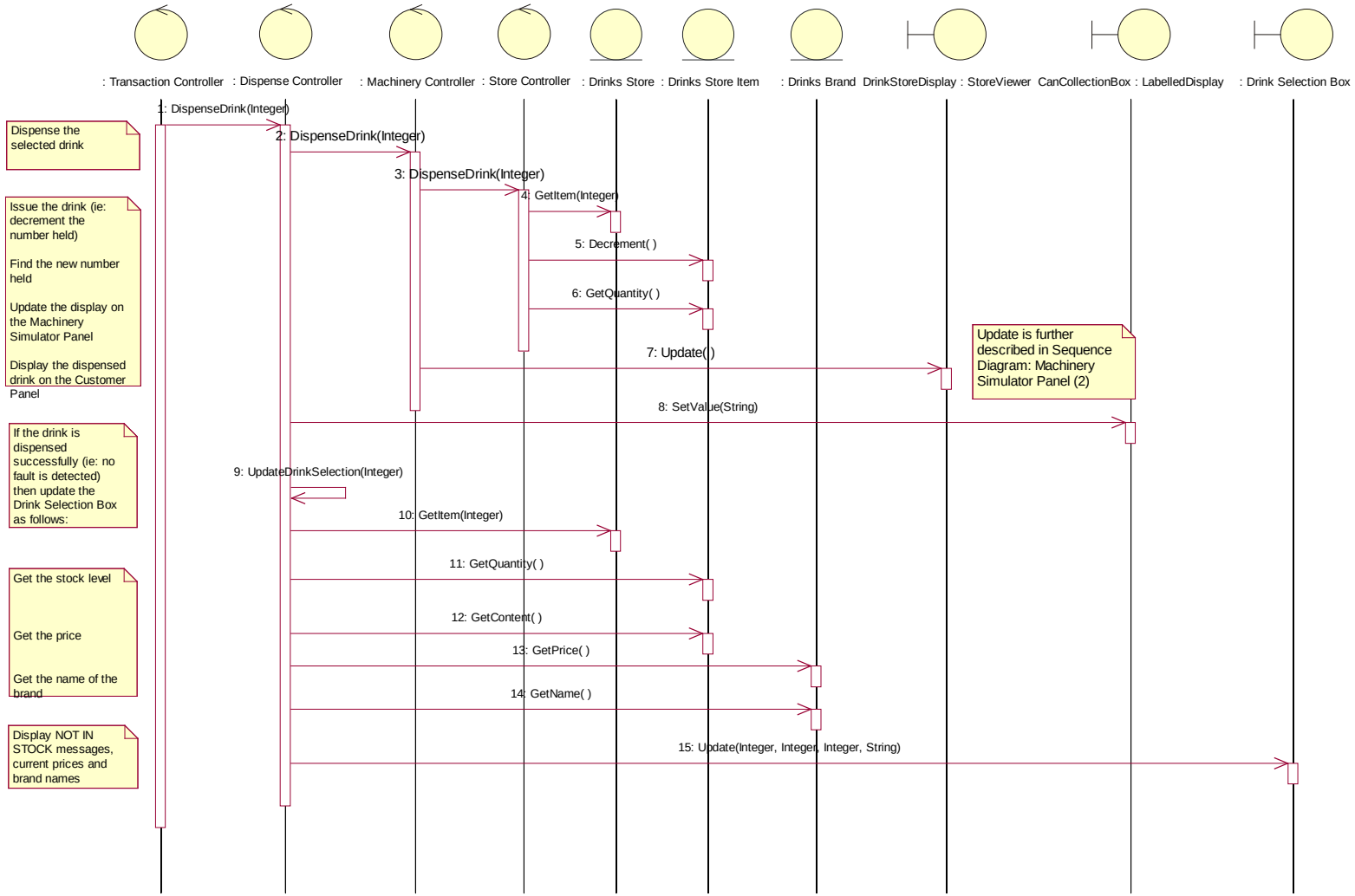


3.2.9 Sequence Diagram: Sequence Diagram: Terminate (Customer Terminates Transaction)



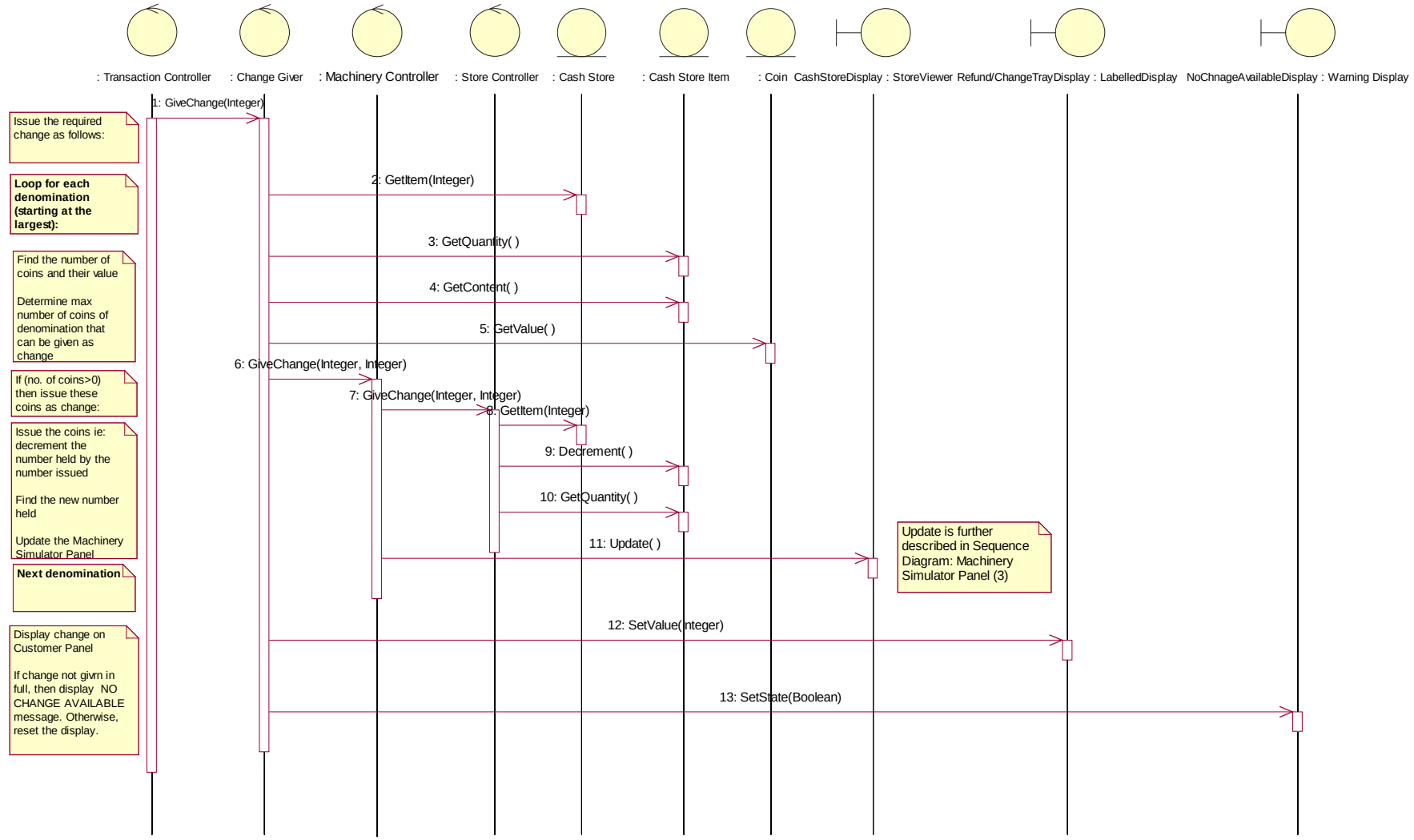
3.3 Dispense Drink

3.3.1 Sequence Diagram: Sequence Diagram: Dispense Drink



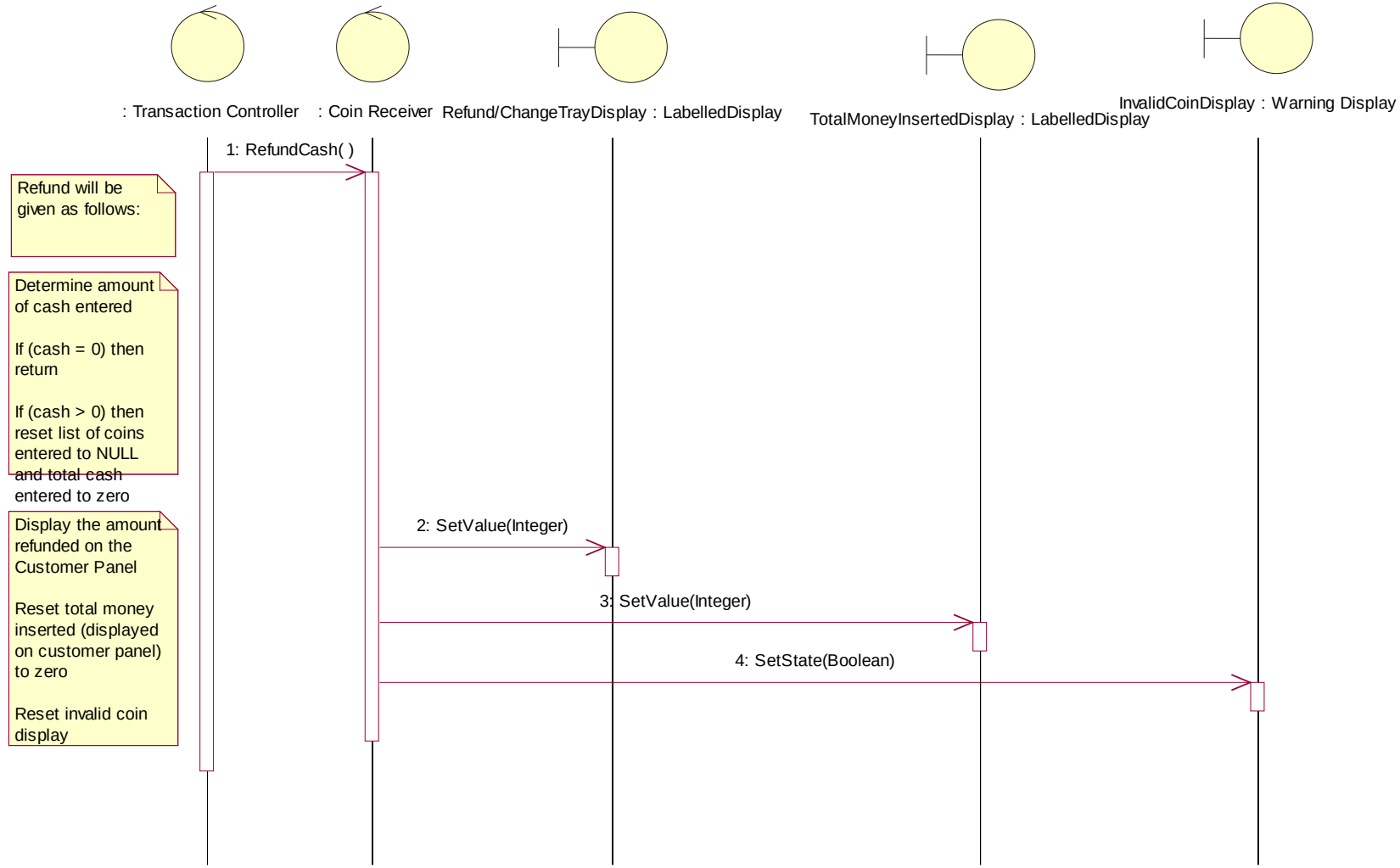
3.4 Give Change

3.4.1 Sequence Diagram: Sequence Diagram: Give Change



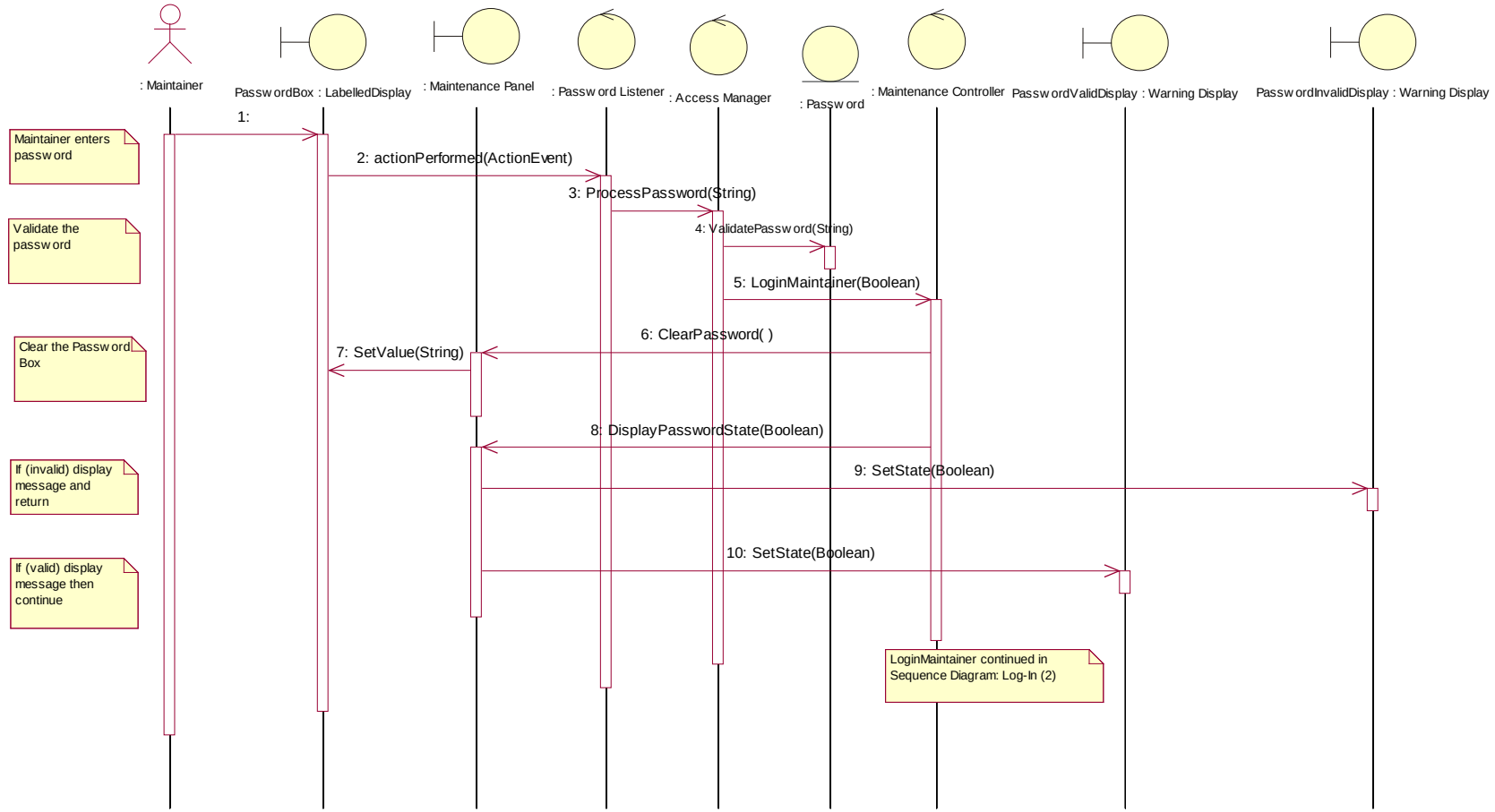
3.5 Refund Money

3.5.1 Sequence Diagram: Sequence Diagram: Refund Money

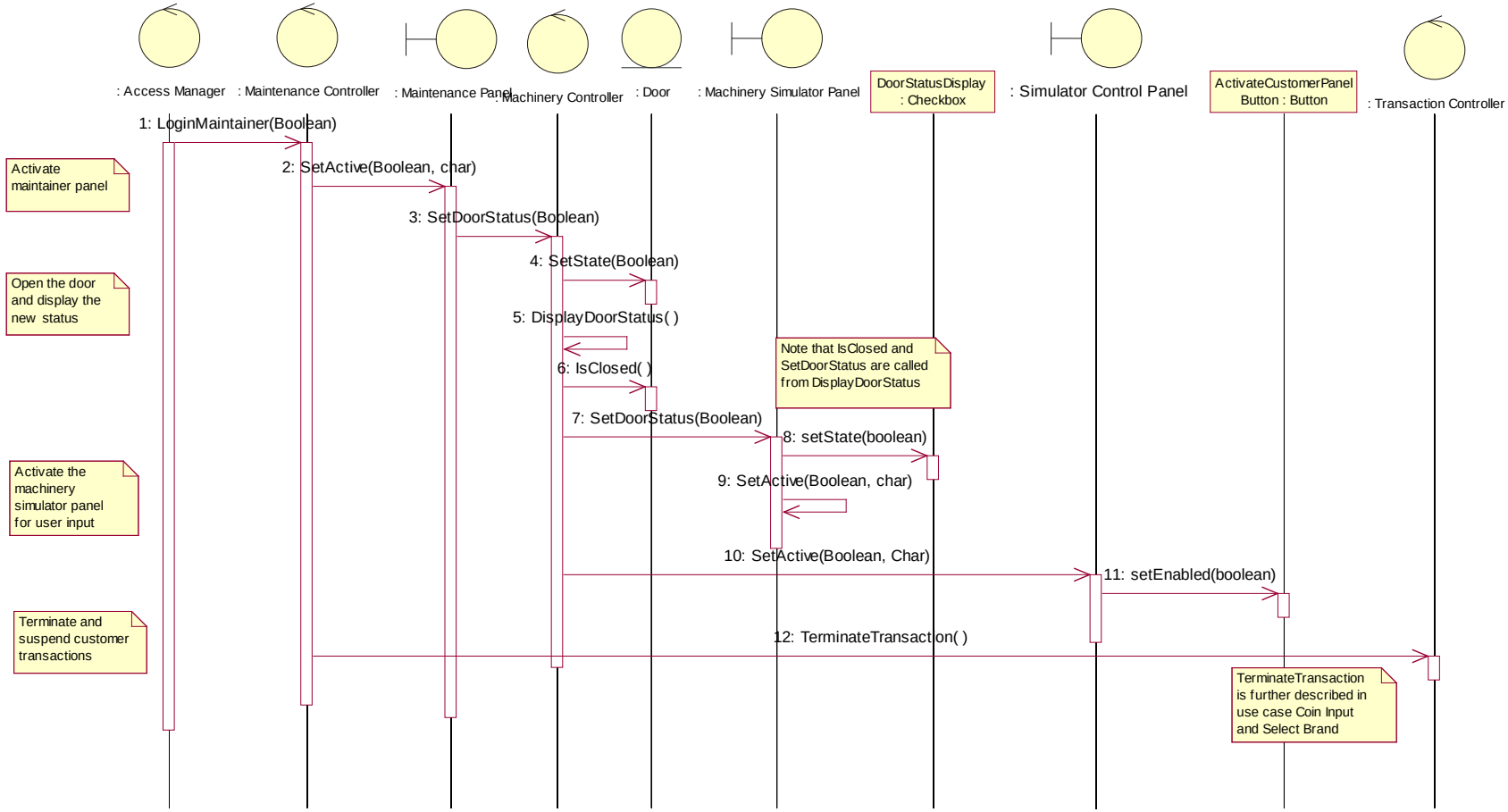


3.6 System Maintenance

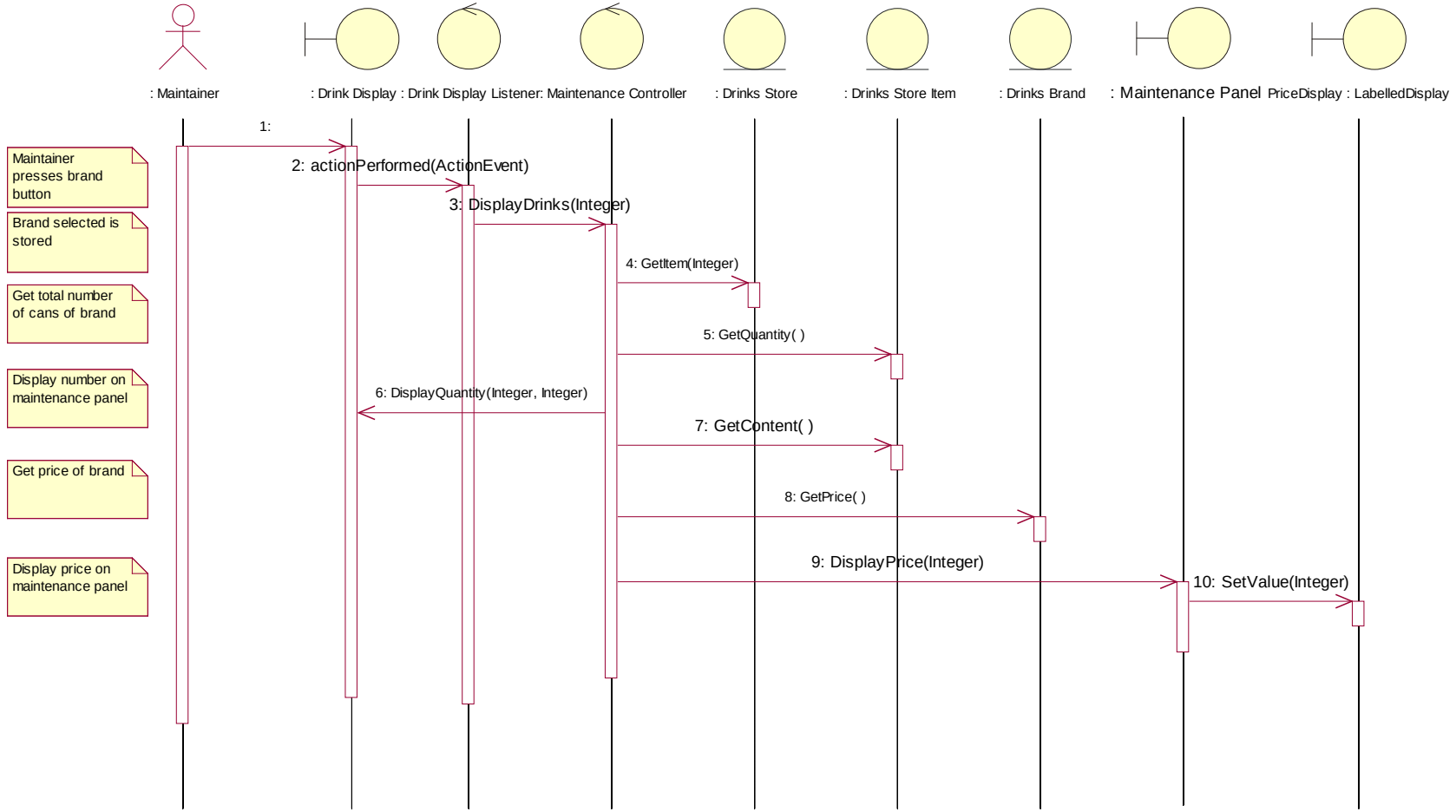
3.6.1 Sequence Diagram: Sequence Diagram: Log-In (1)



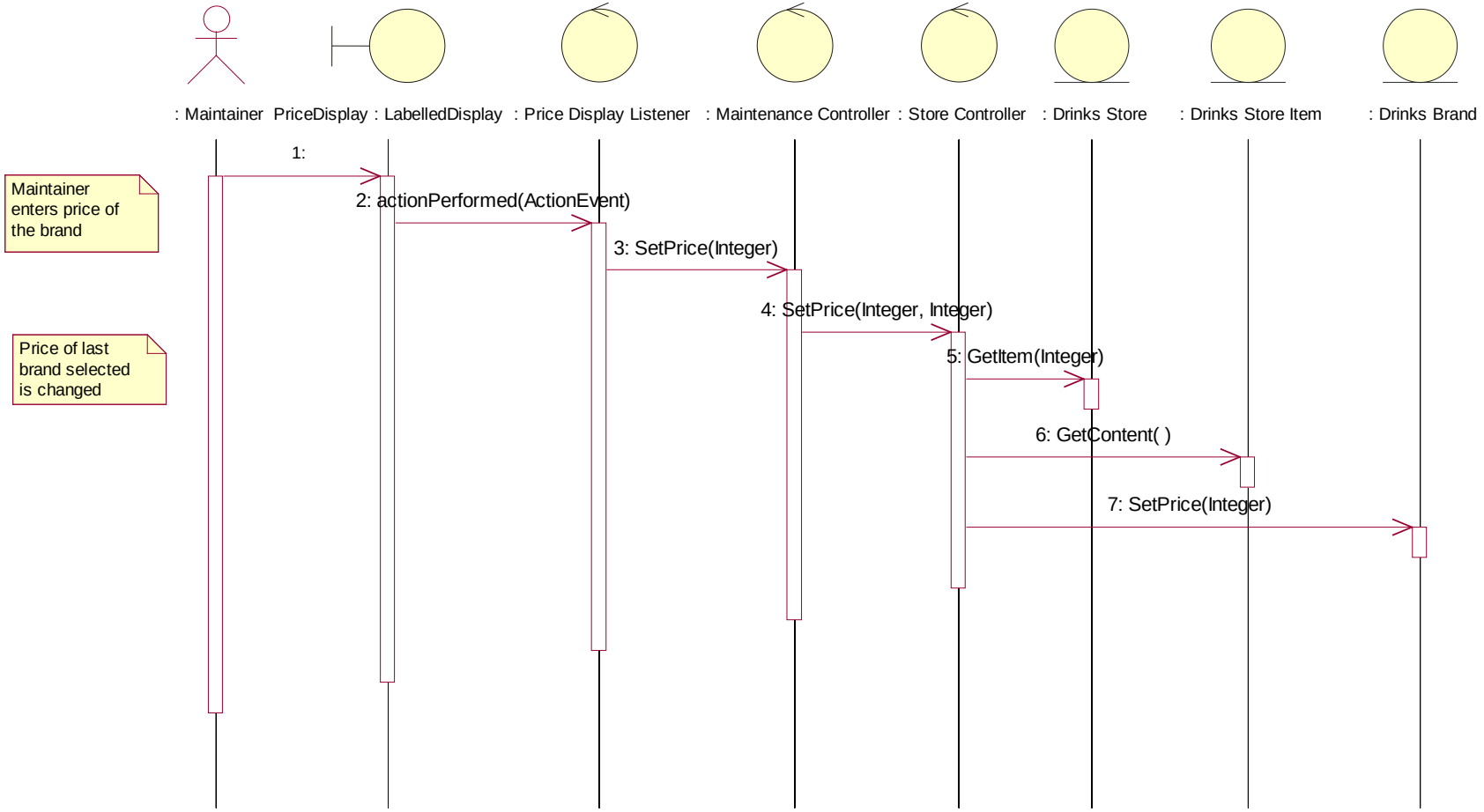
3.6.2 Sequence Diagram: Sequence Diagram: Log-In (2)



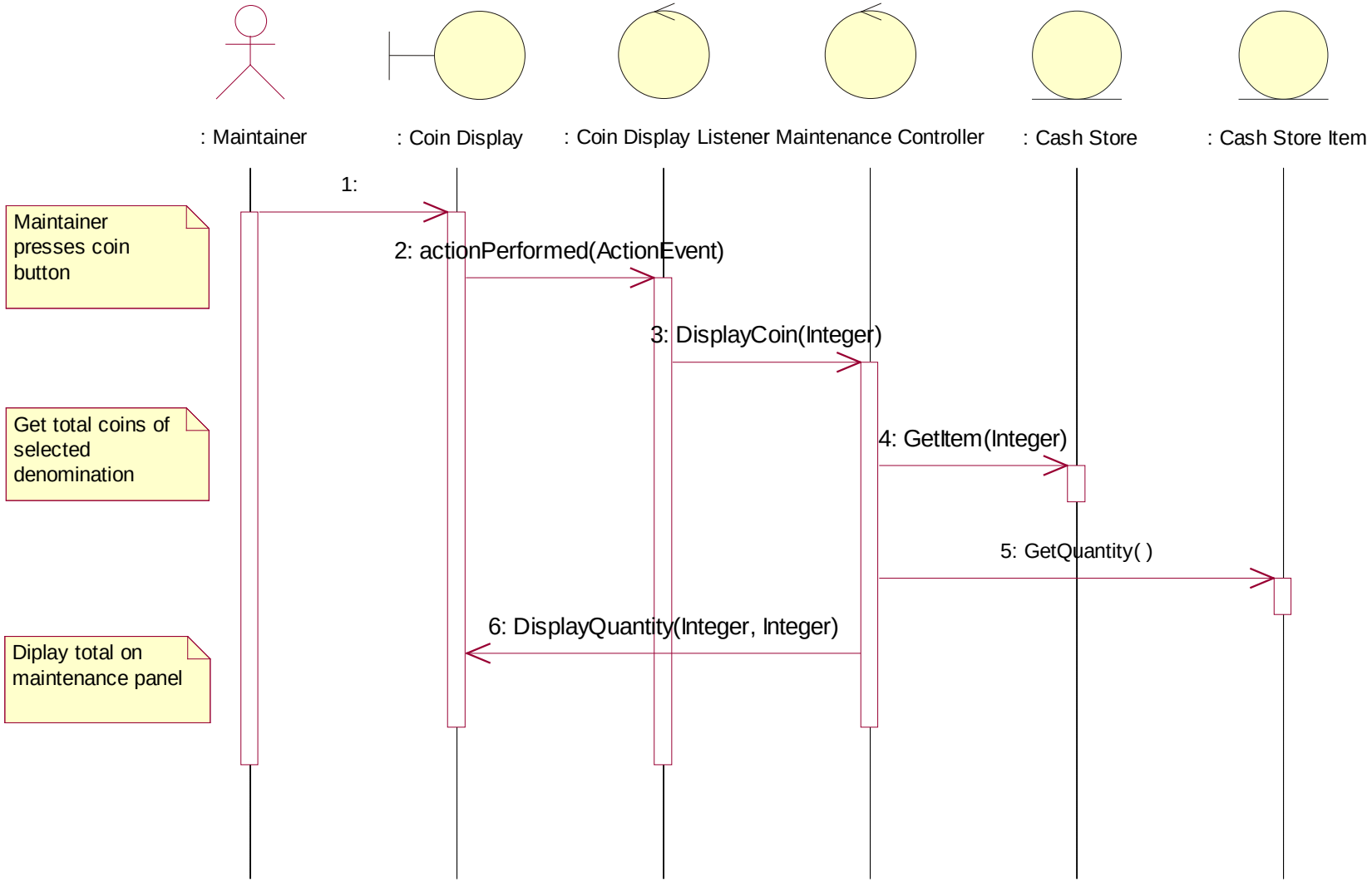
3.6.3 Sequence Diagram: Sequence Diagram: Display Drink Stocks and Prices



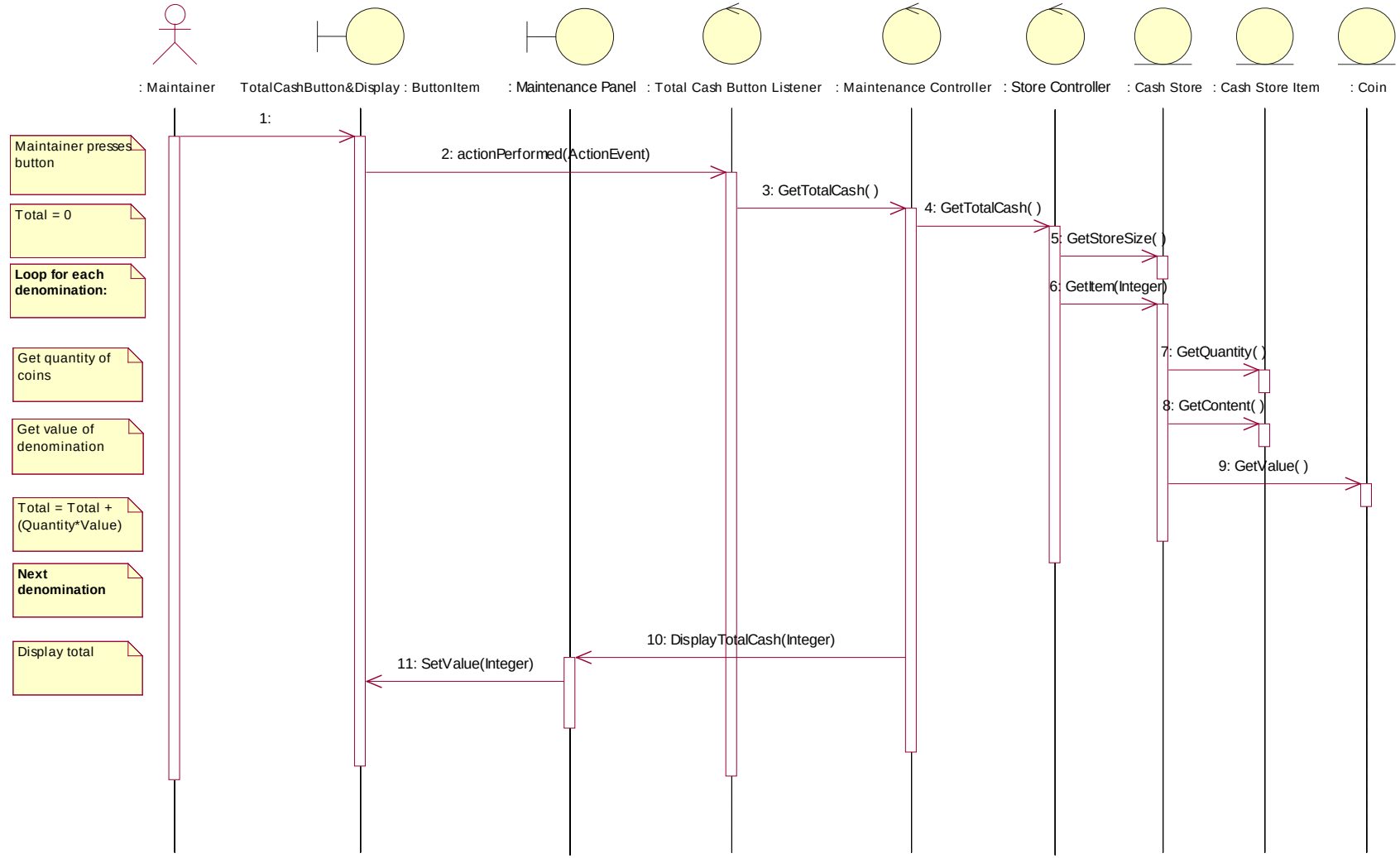
3.6.4 Sequence Diagram: Sequence Diagram: Change Prices



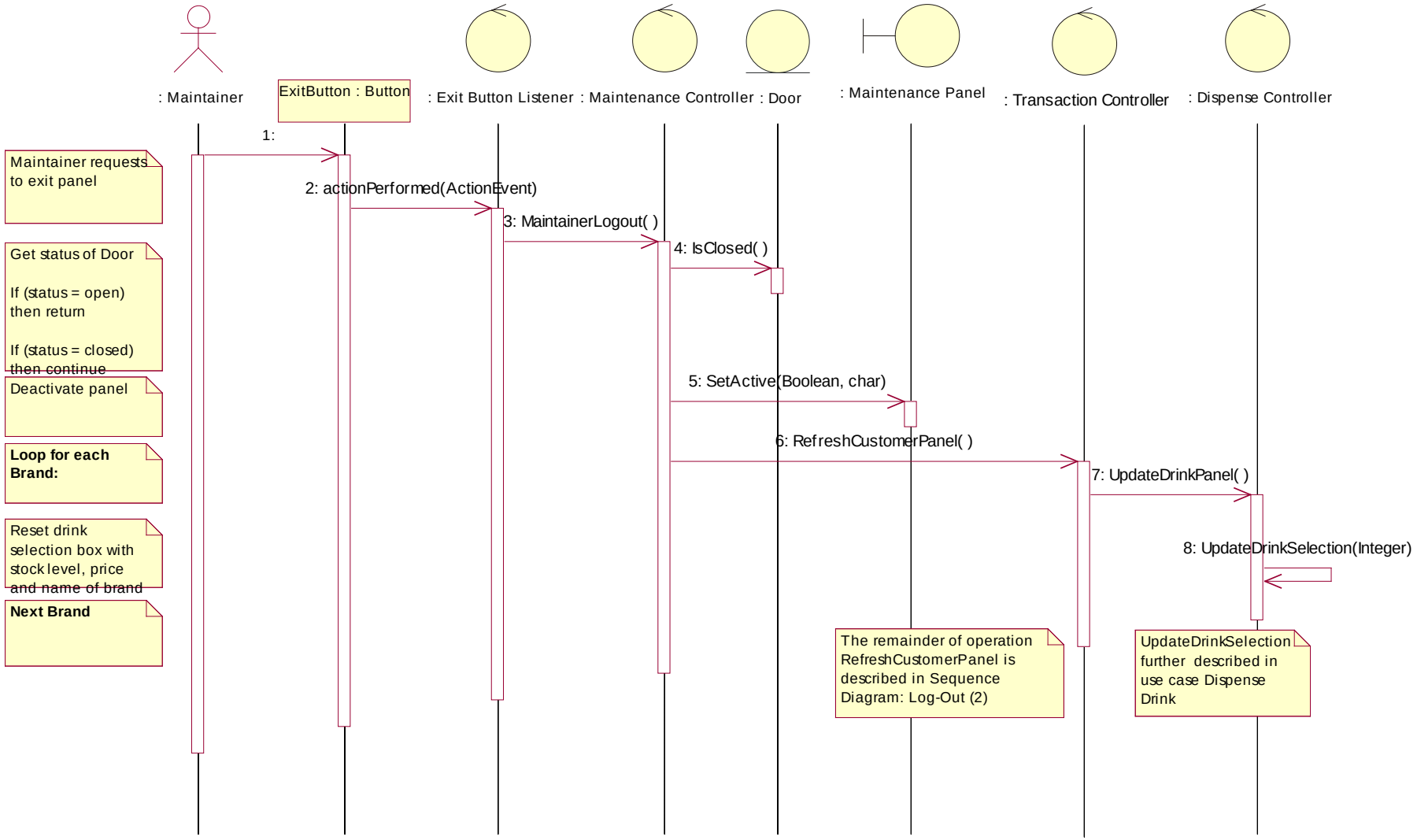
3.6.5 Sequence Diagram: Sequence Diagram: Display Cash Stocks



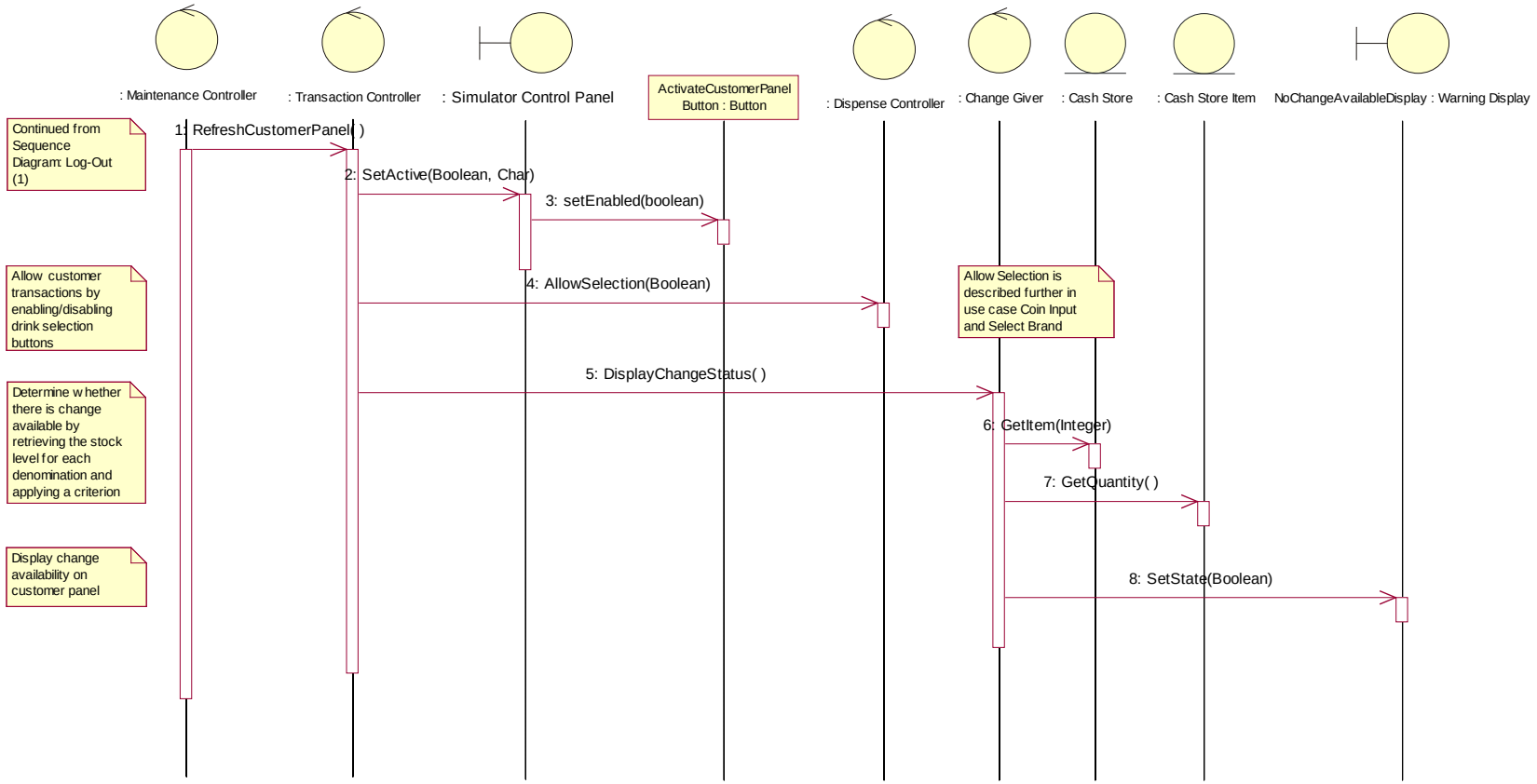
3.6.6 Sequence Diagram: Sequence Diagram: Display Total Cash



3.6.7 Sequence Diagram: Sequence Diagram: Log-Out (1)

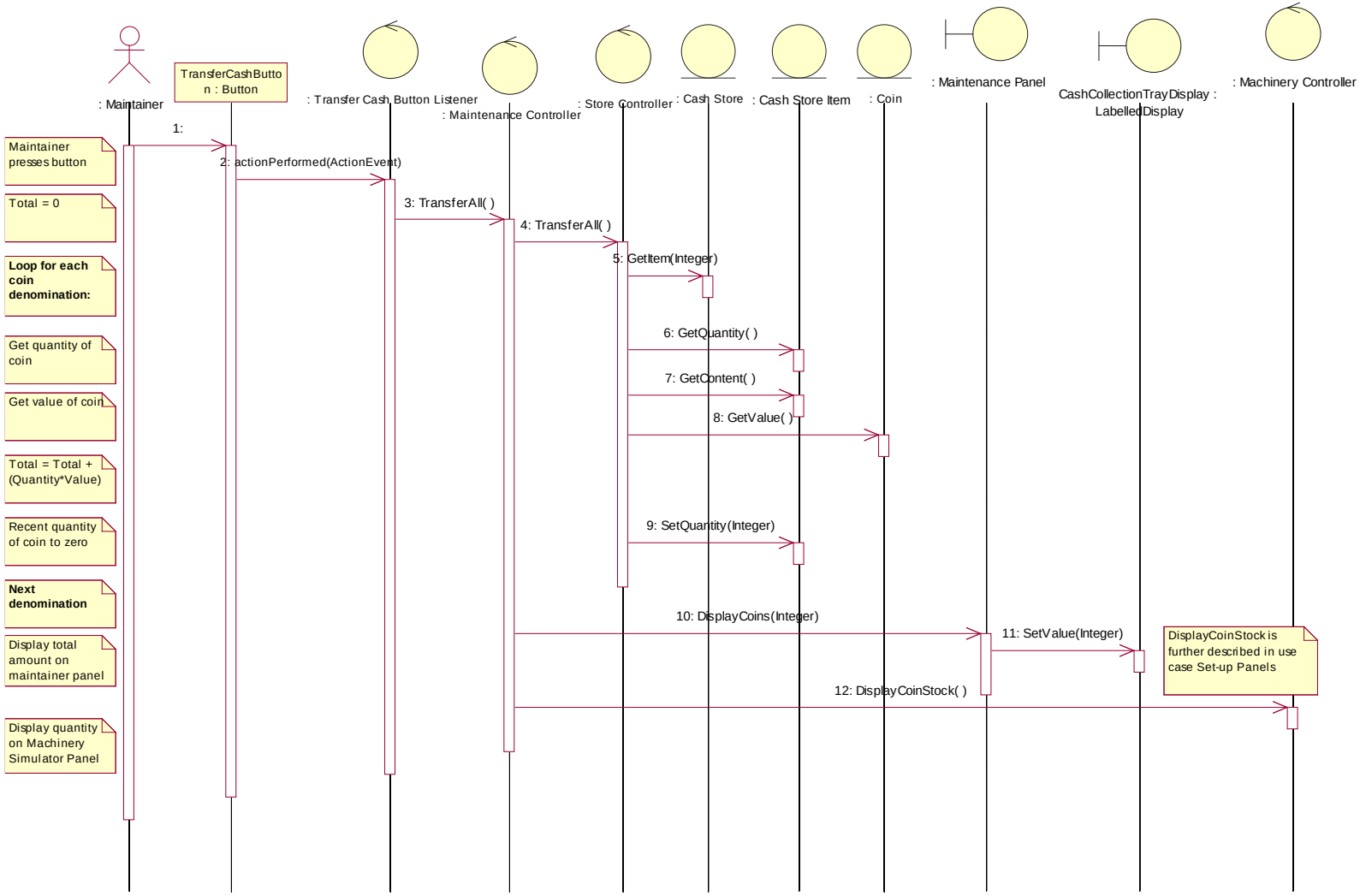


3.6.8 Sequence Diagram: Sequence Diagram: Log-Out (2)



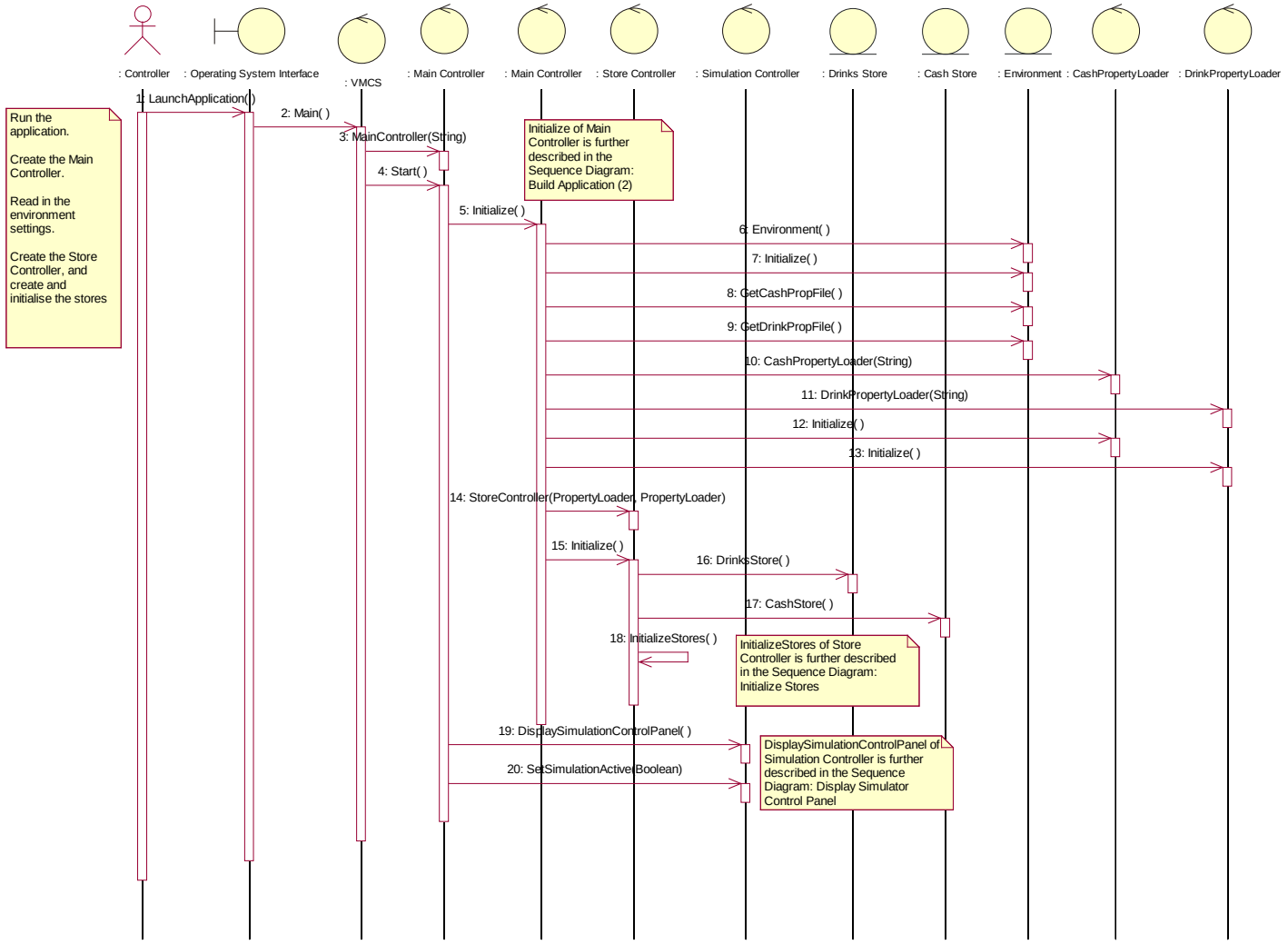
3.7 Transfer All Cash

3.7.1 Sequence Diagram: Sequence Diagram: Transfer All Cash

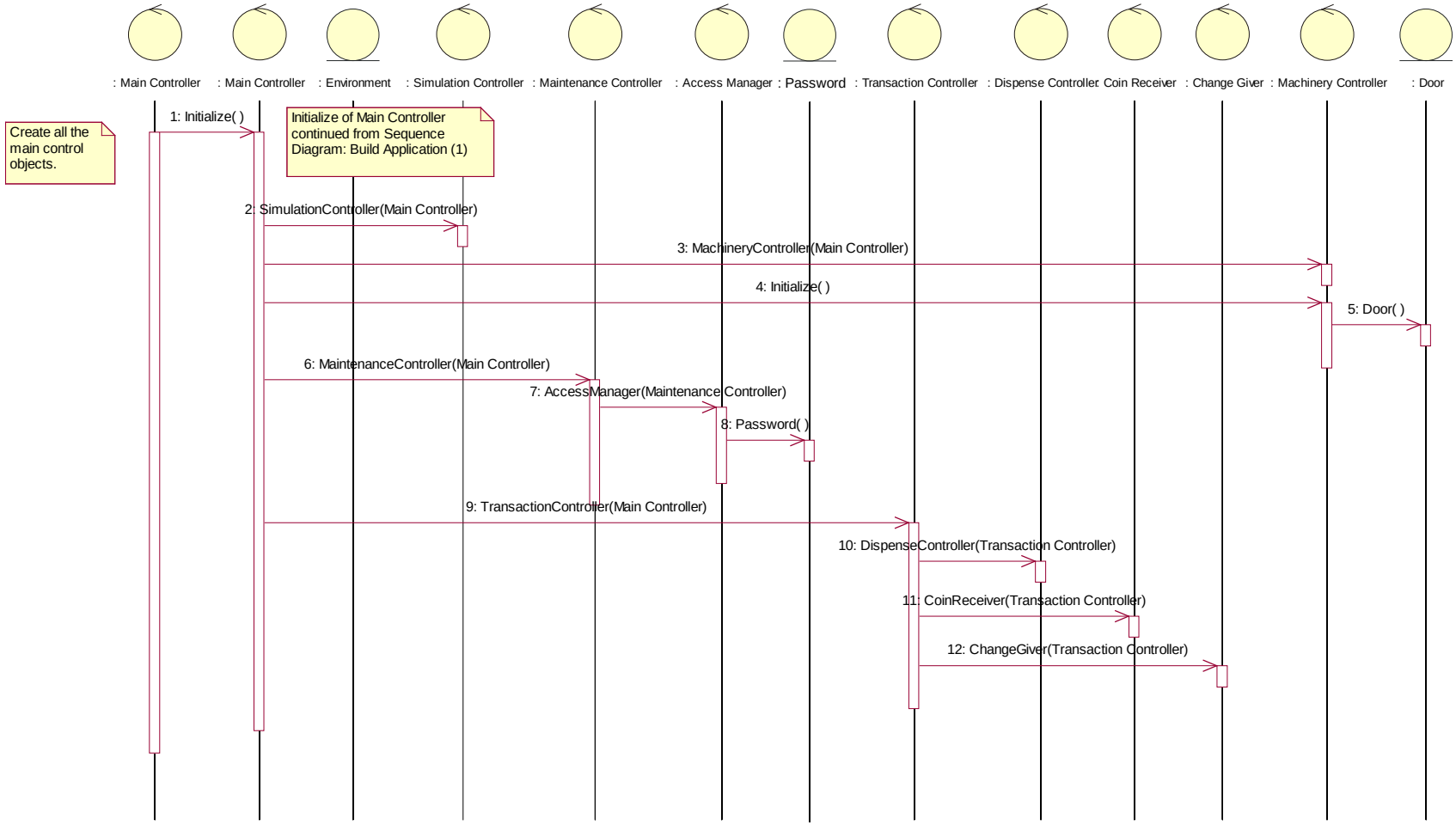


3.8 System Start-Up & Close-Down

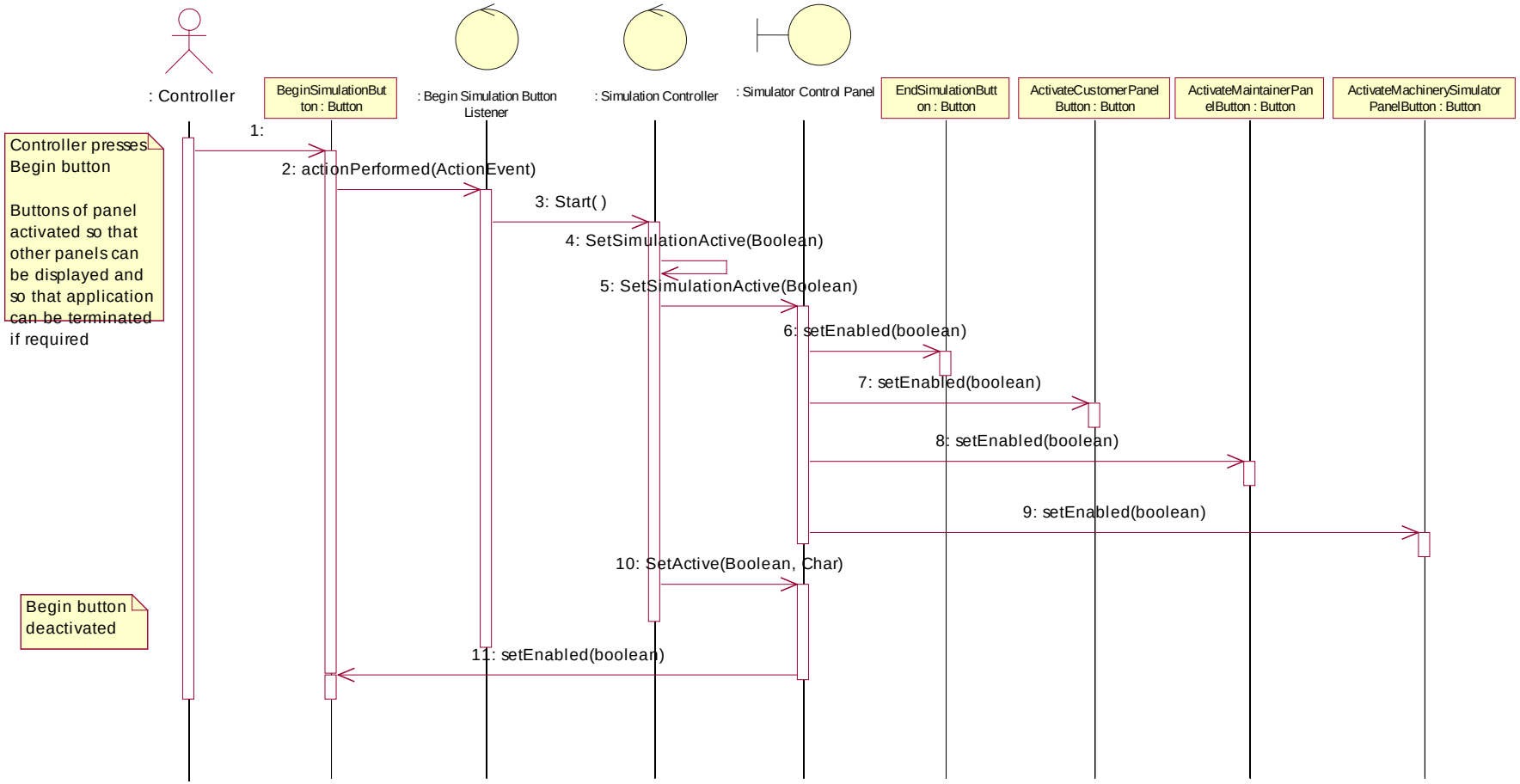
3.8.1 Sequence Diagram: Sequence Diagram: Build Application (1)



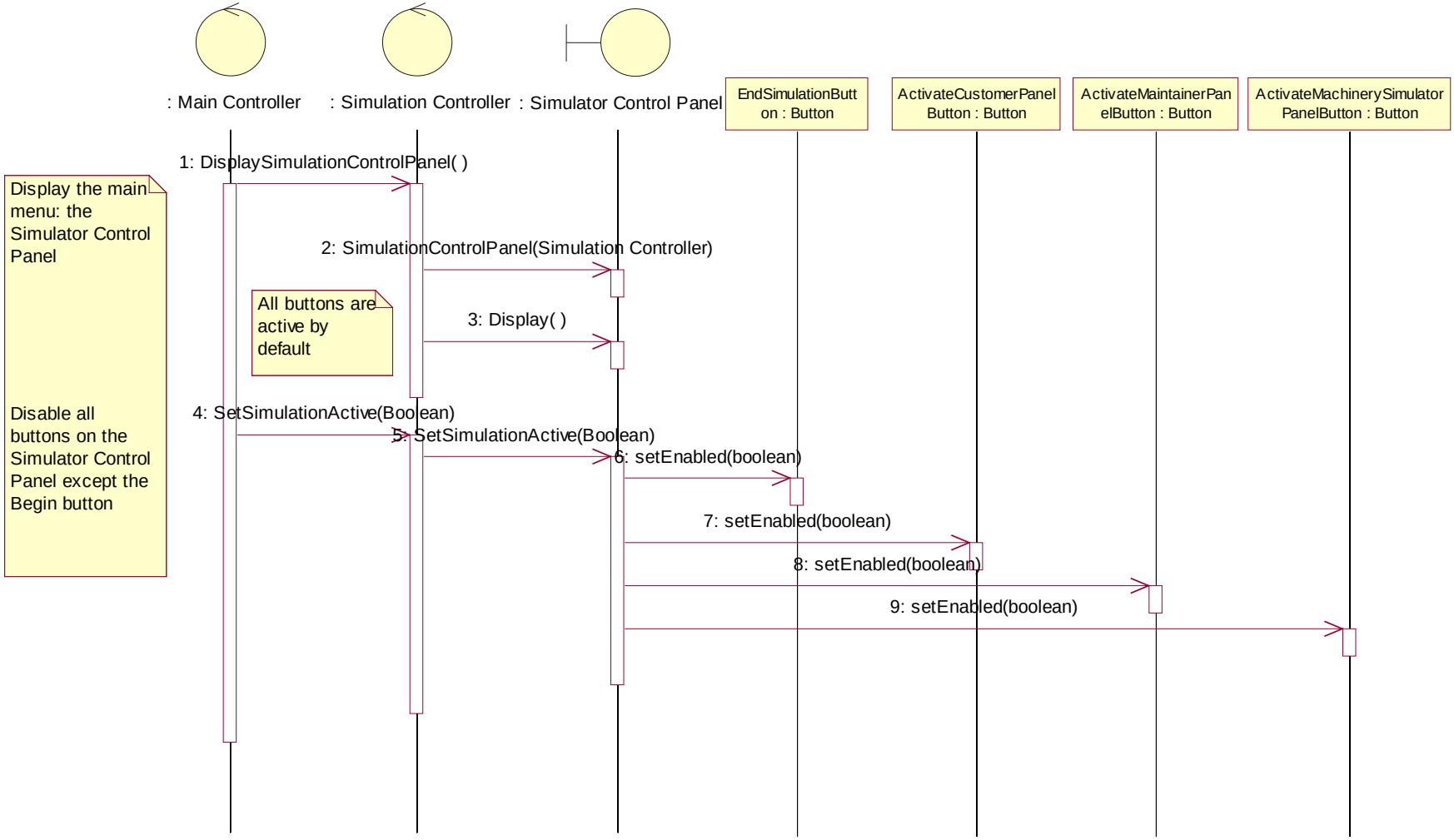
3.8.2 Sequence Diagram: Sequence Diagram: Build Application (2)



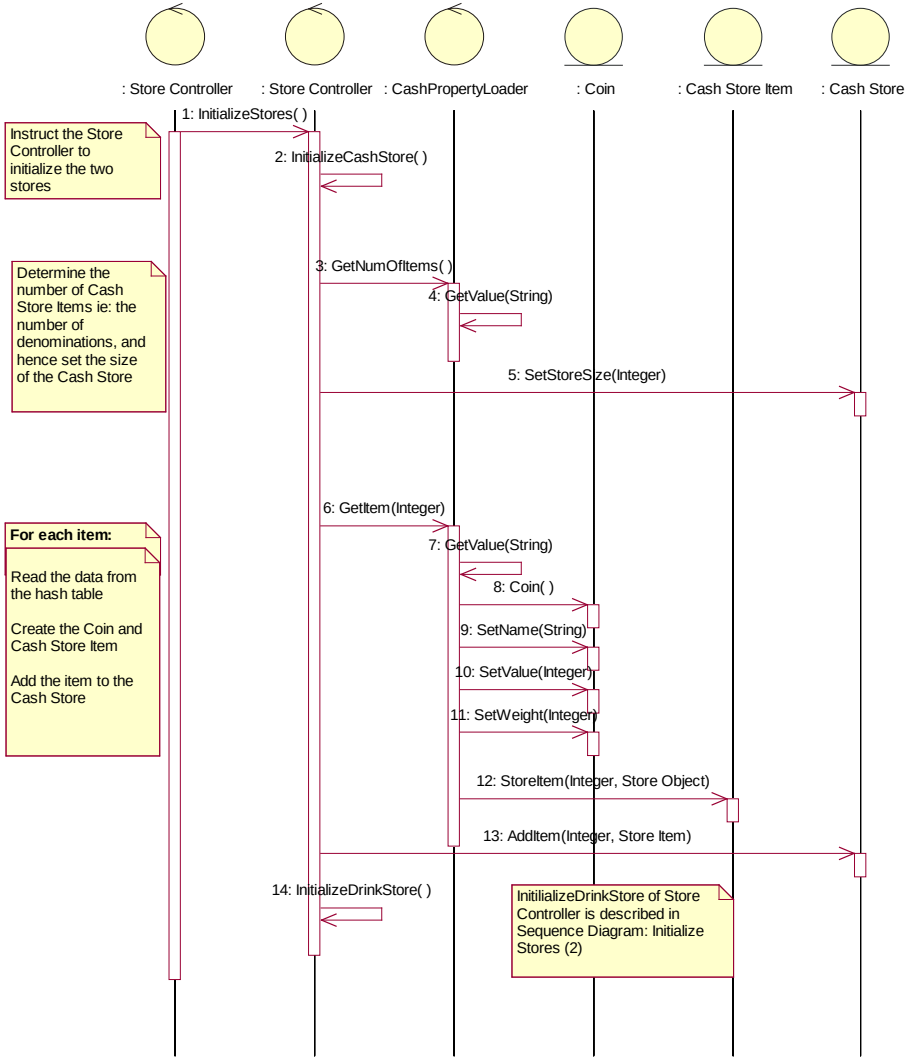
3.8.3 Sequence Diagram: Sequence Diagram: System Start-Up



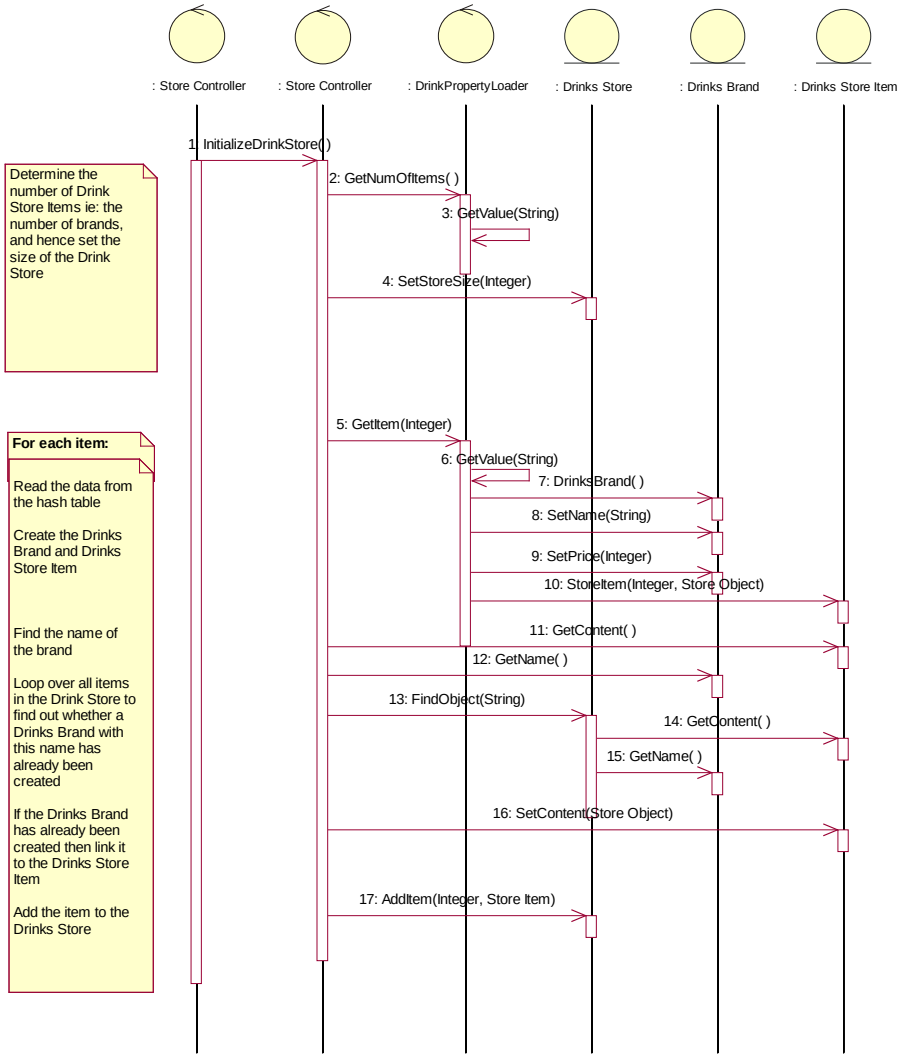
3.8.4 Sequence Diagram: Sequence Diagram: Display Simulator Control Panel



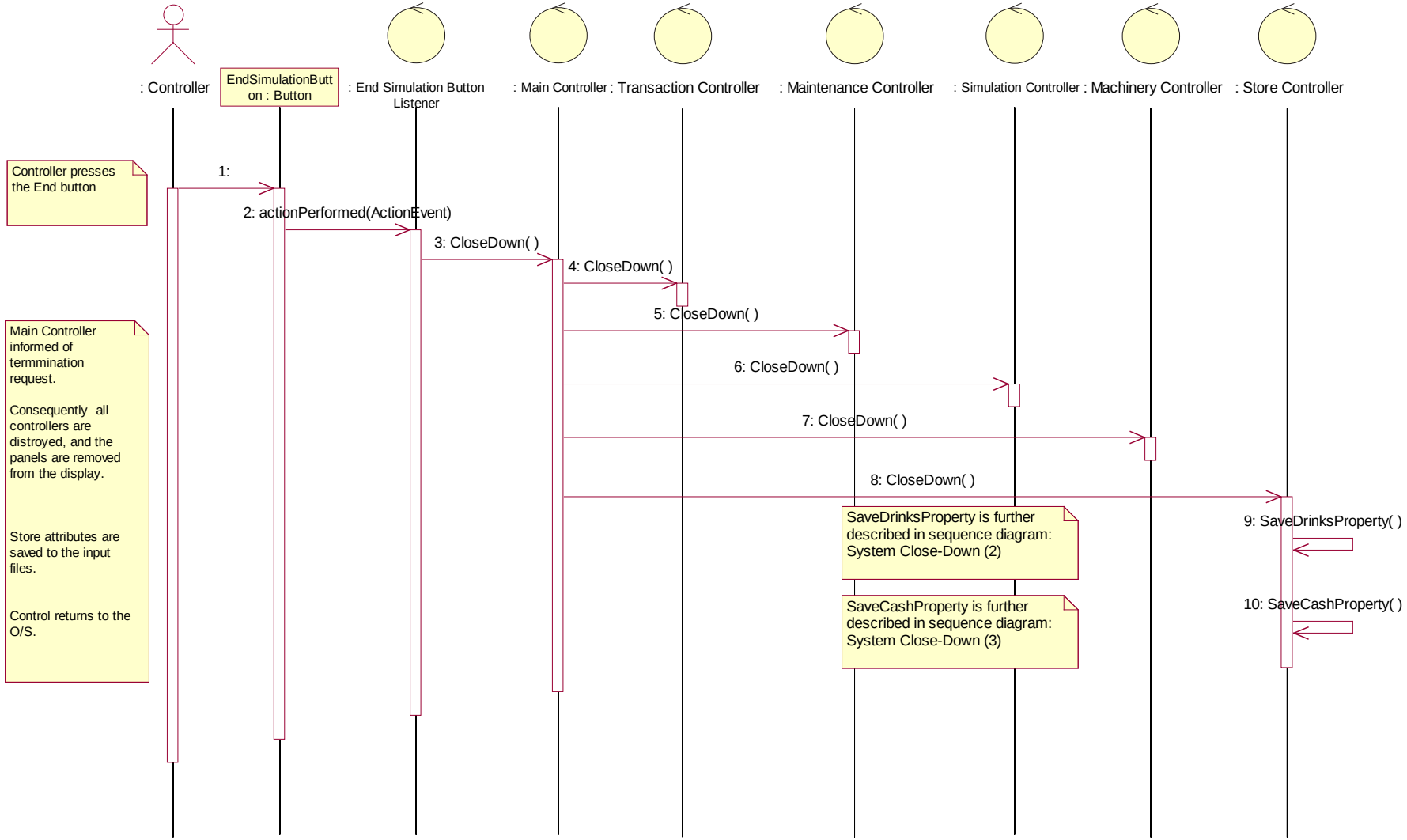
3.8.5 *Sequence Diagram: Sequence Diagram: Initialize Stores (1)*



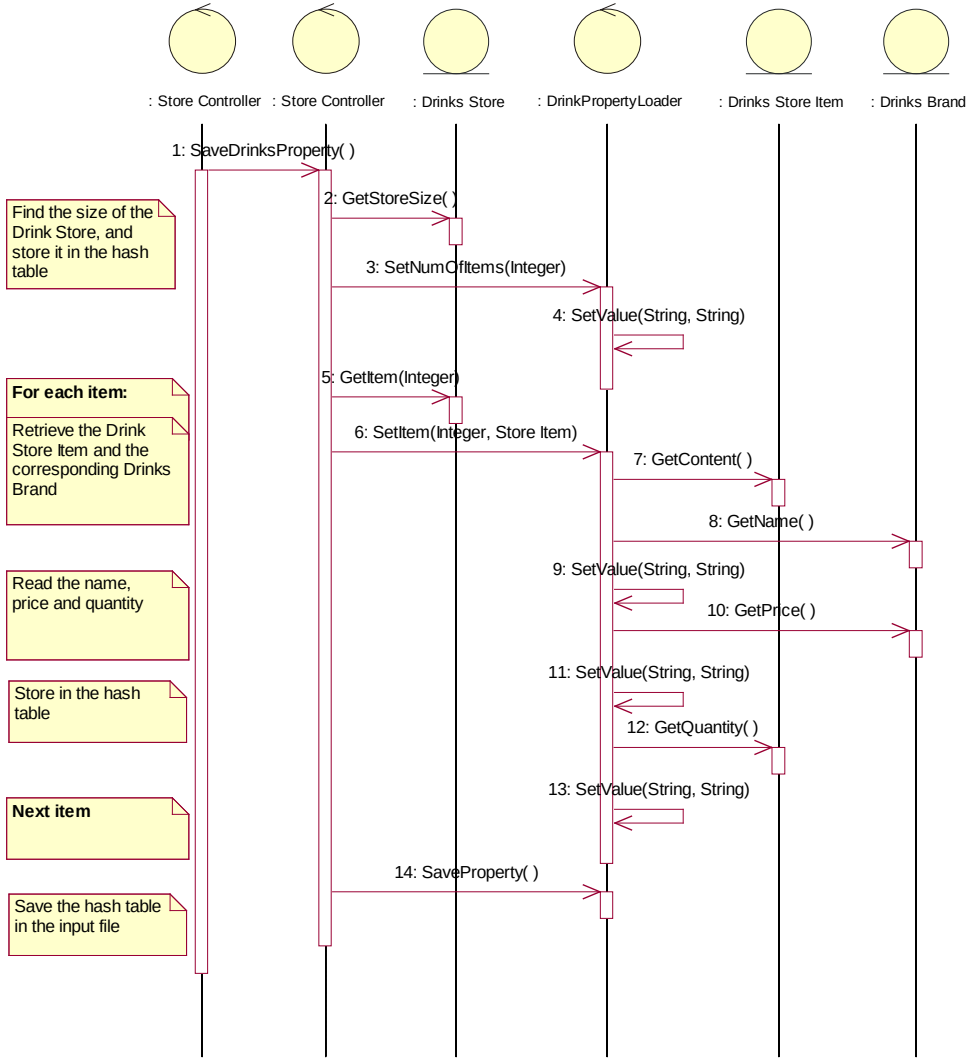
3.8.6 *Sequence Diagram: Sequence Diagram: Initialize Stores (2)*



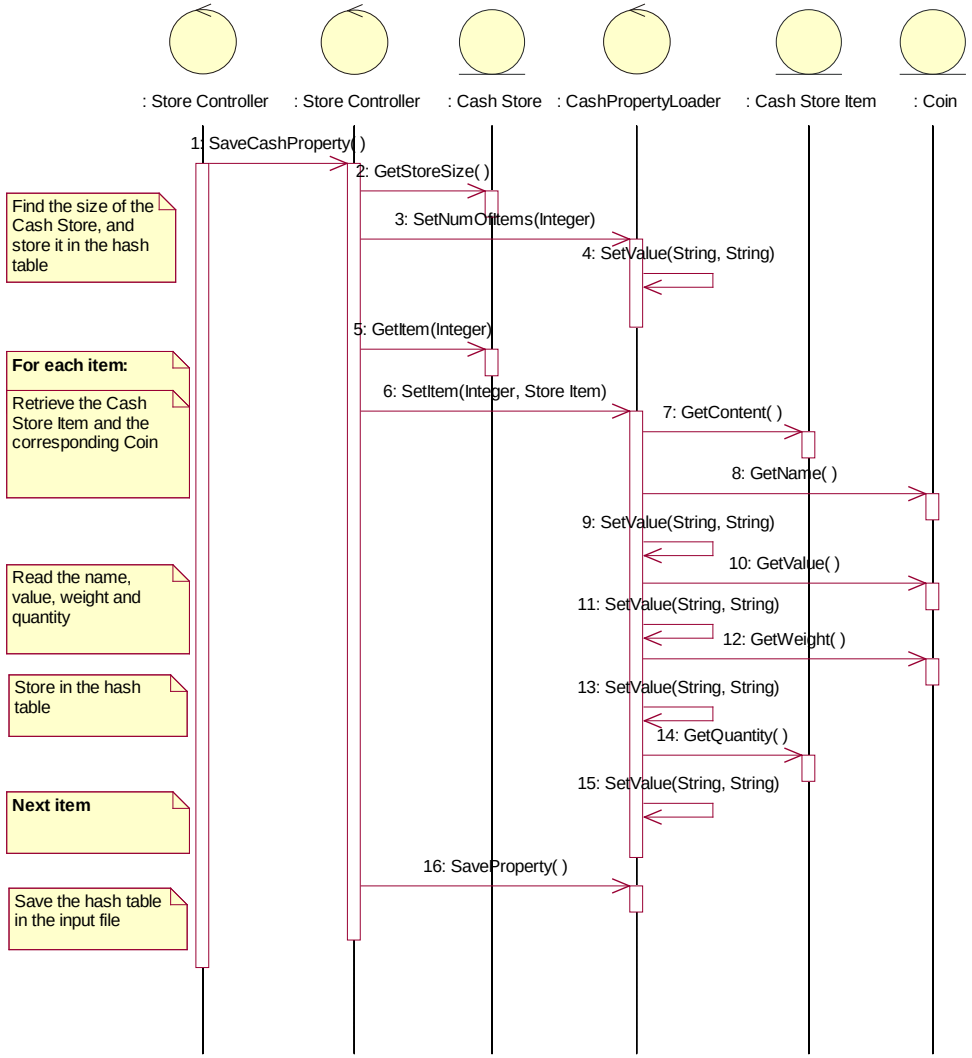
3.8.7 Sequence Diagram: Sequence Diagram: System Close-Down (1)



3.8.8 *Sequence Diagram: Sequence Diagram: System Close-Down (2)*

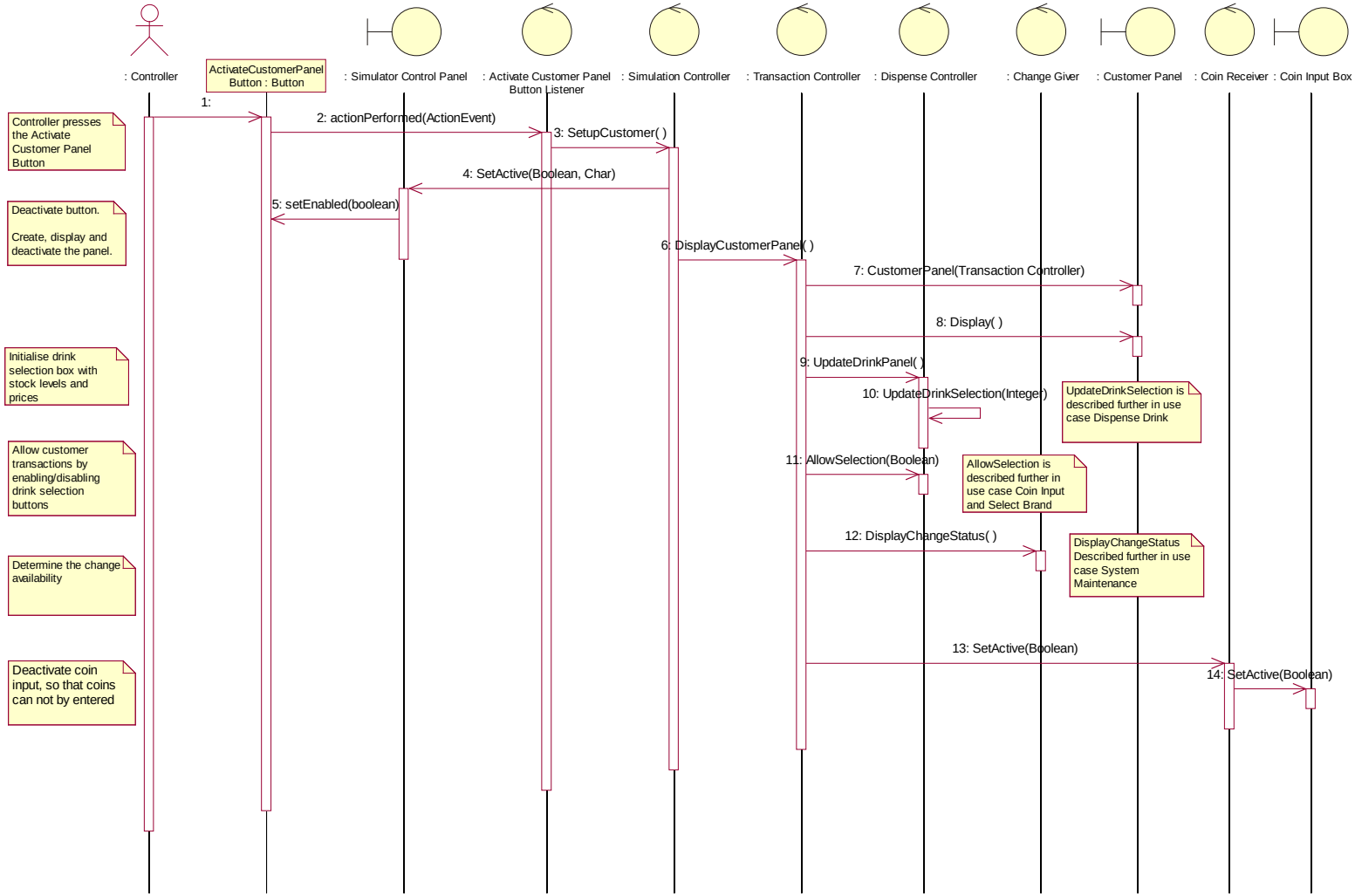


3.8.9 *Sequence Diagram: Sequence Diagram: System Close-Down (3)*

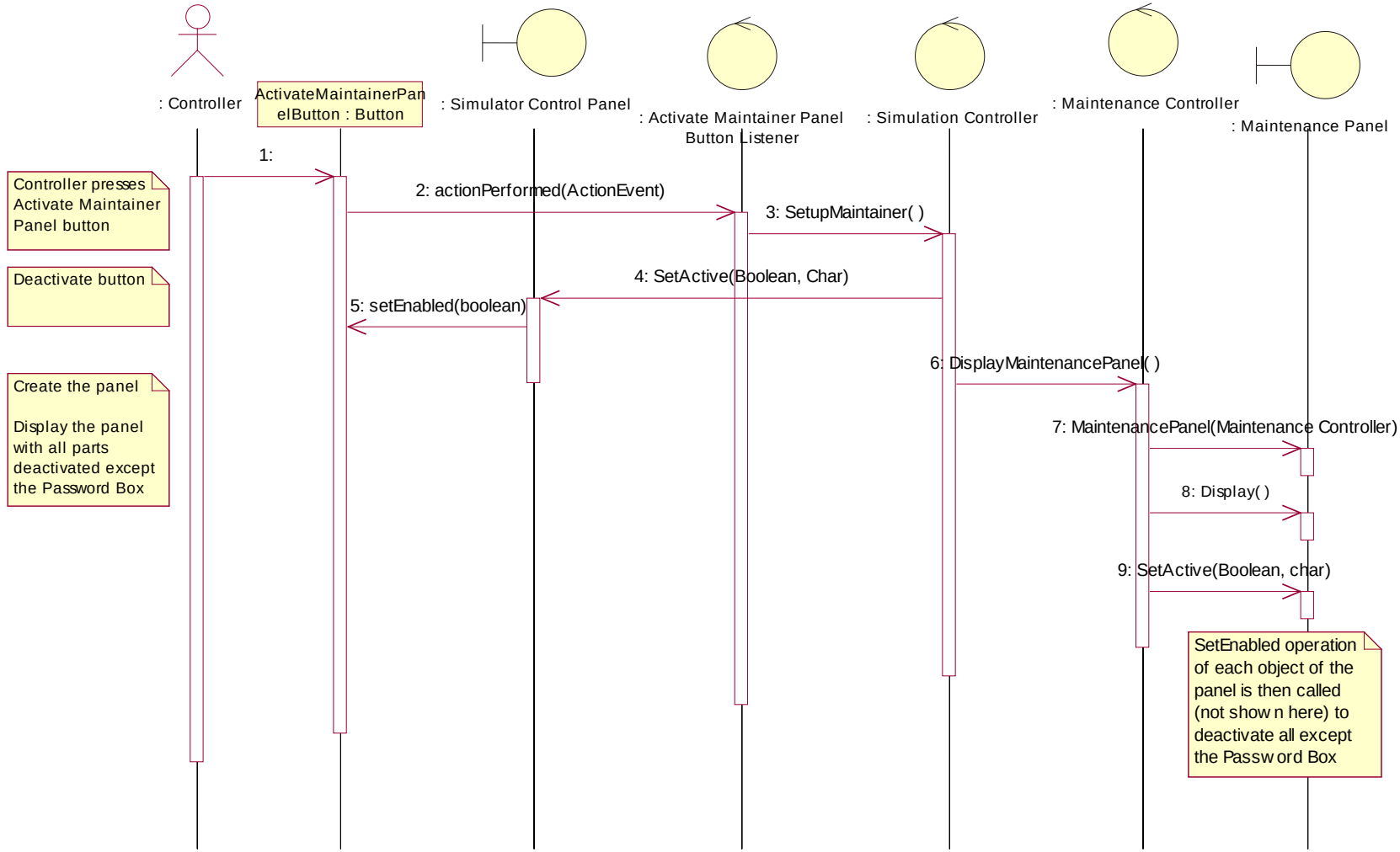


3.9 Set-Up Panels

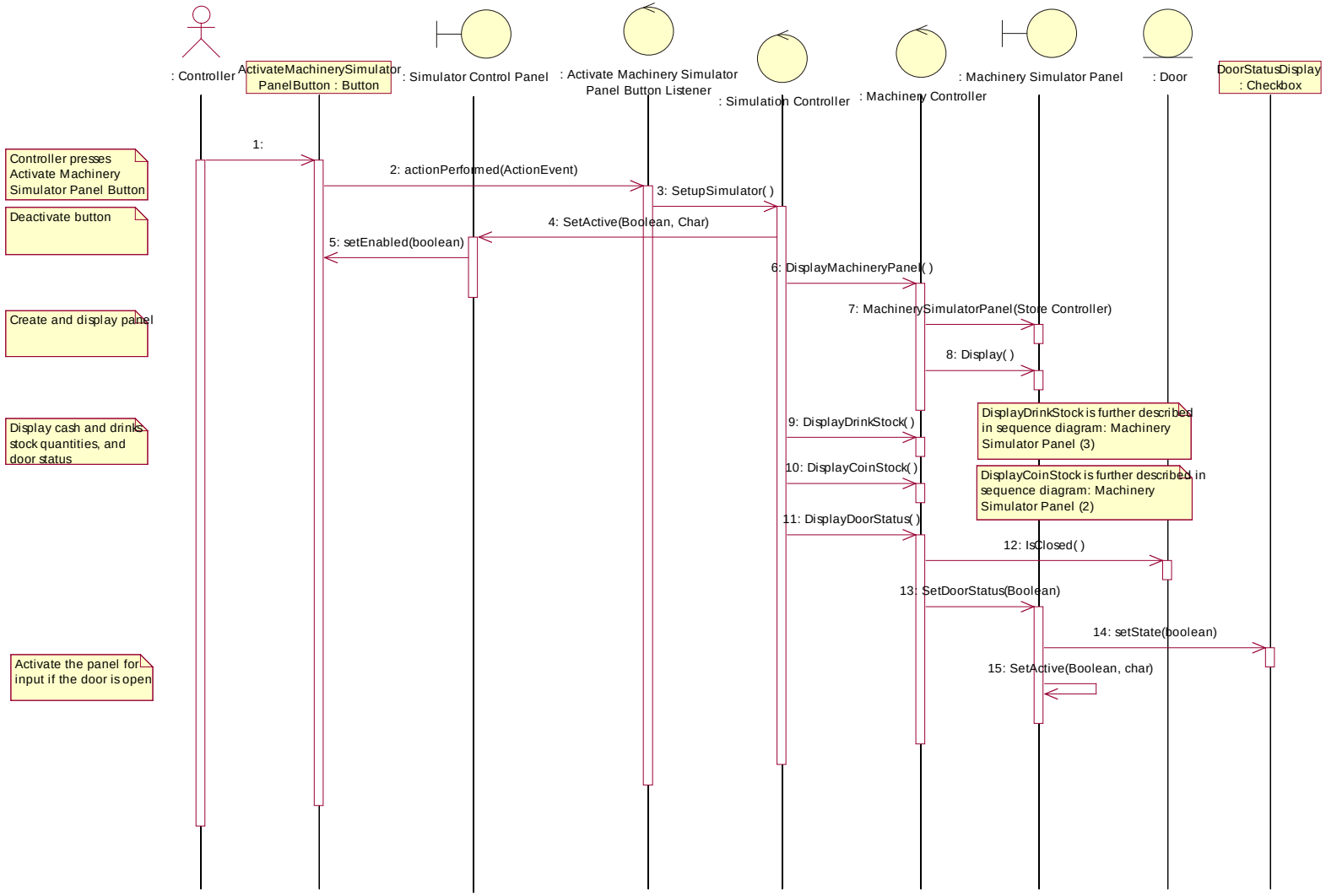
3.9.1 Sequence Diagram: Sequence Diagram: Customer Panel



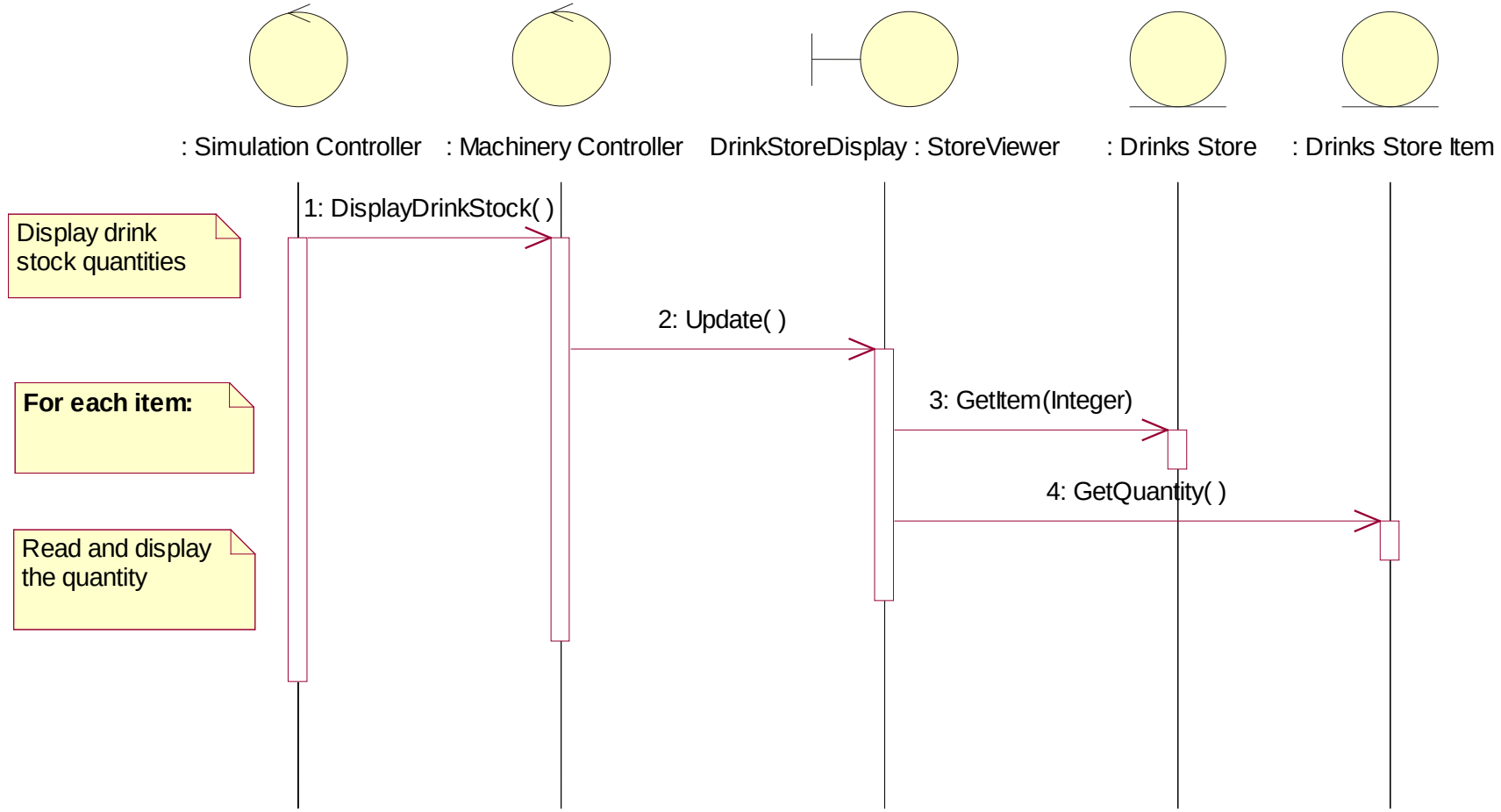
3.9.2 Sequence Diagram: Sequence Diagram: Maintenance Panel



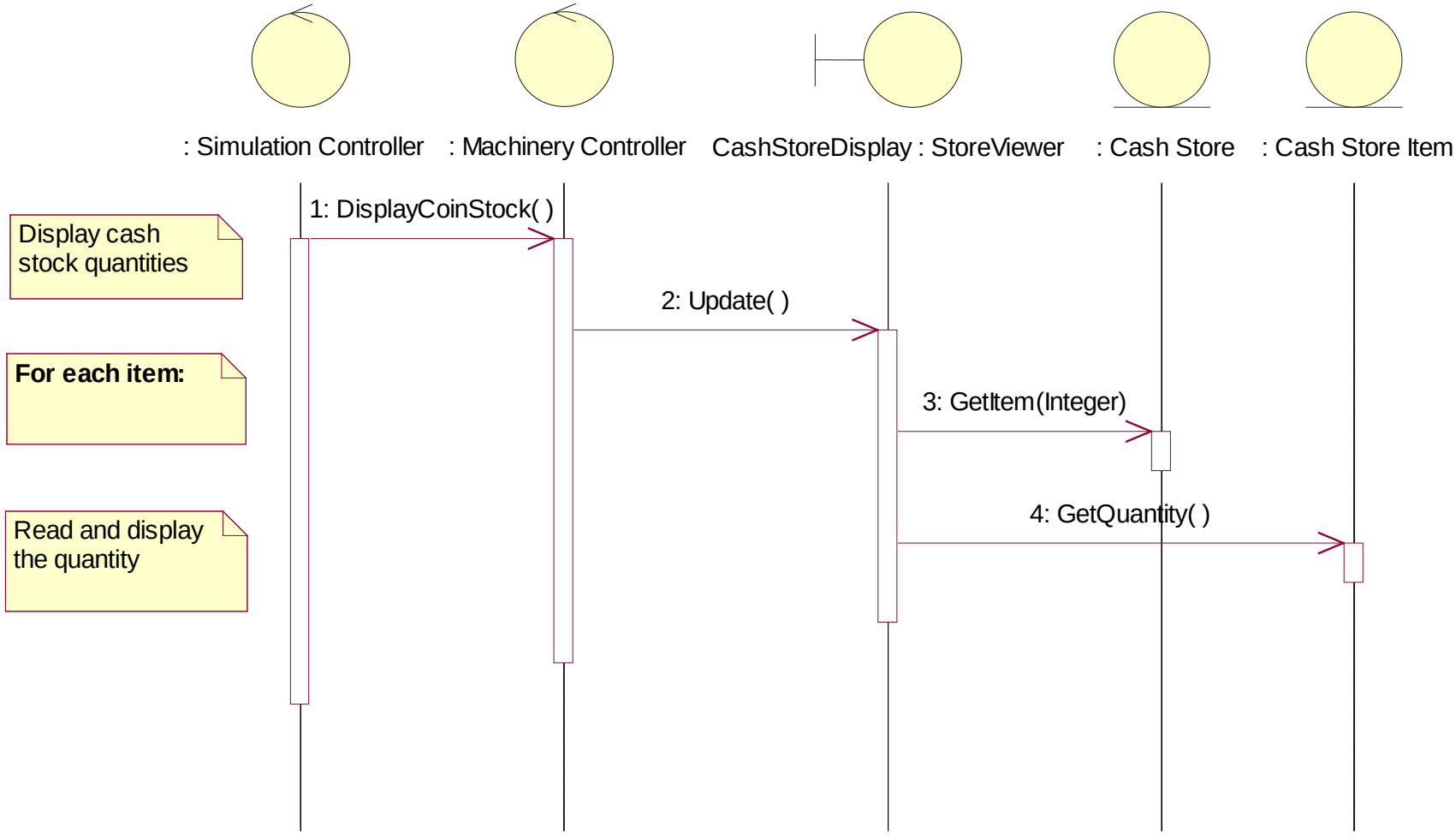
3.9.3 *Sequence Diagram: Sequence Diagram: Machinery Simulator Panel (1)*



3.9.4 Sequence Diagram: Sequence Diagram: Machinery Simulator Panel (2)

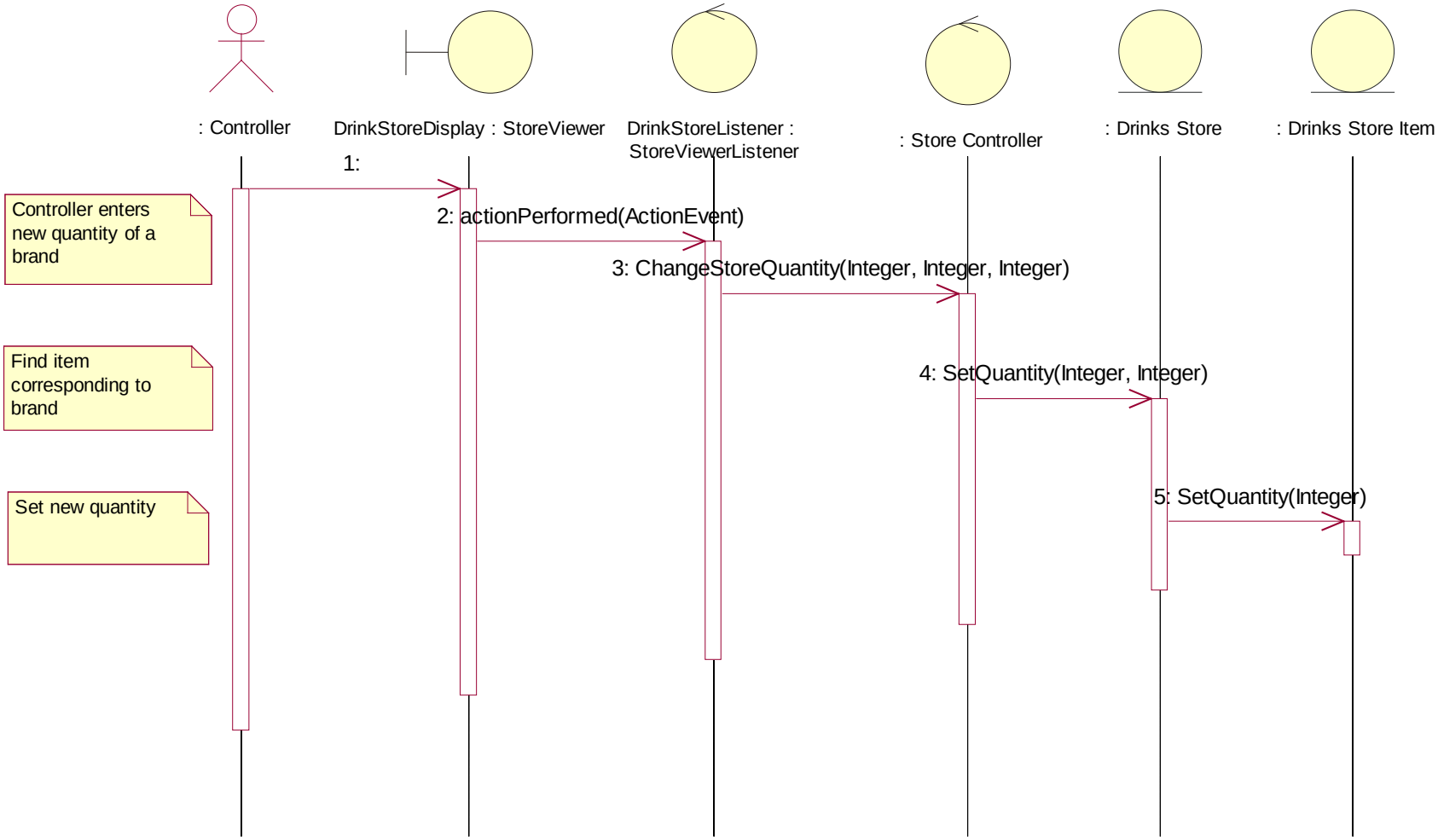


3.9.5 Sequence Diagram: Sequence Diagram: Machinery Simulator Panel (3)

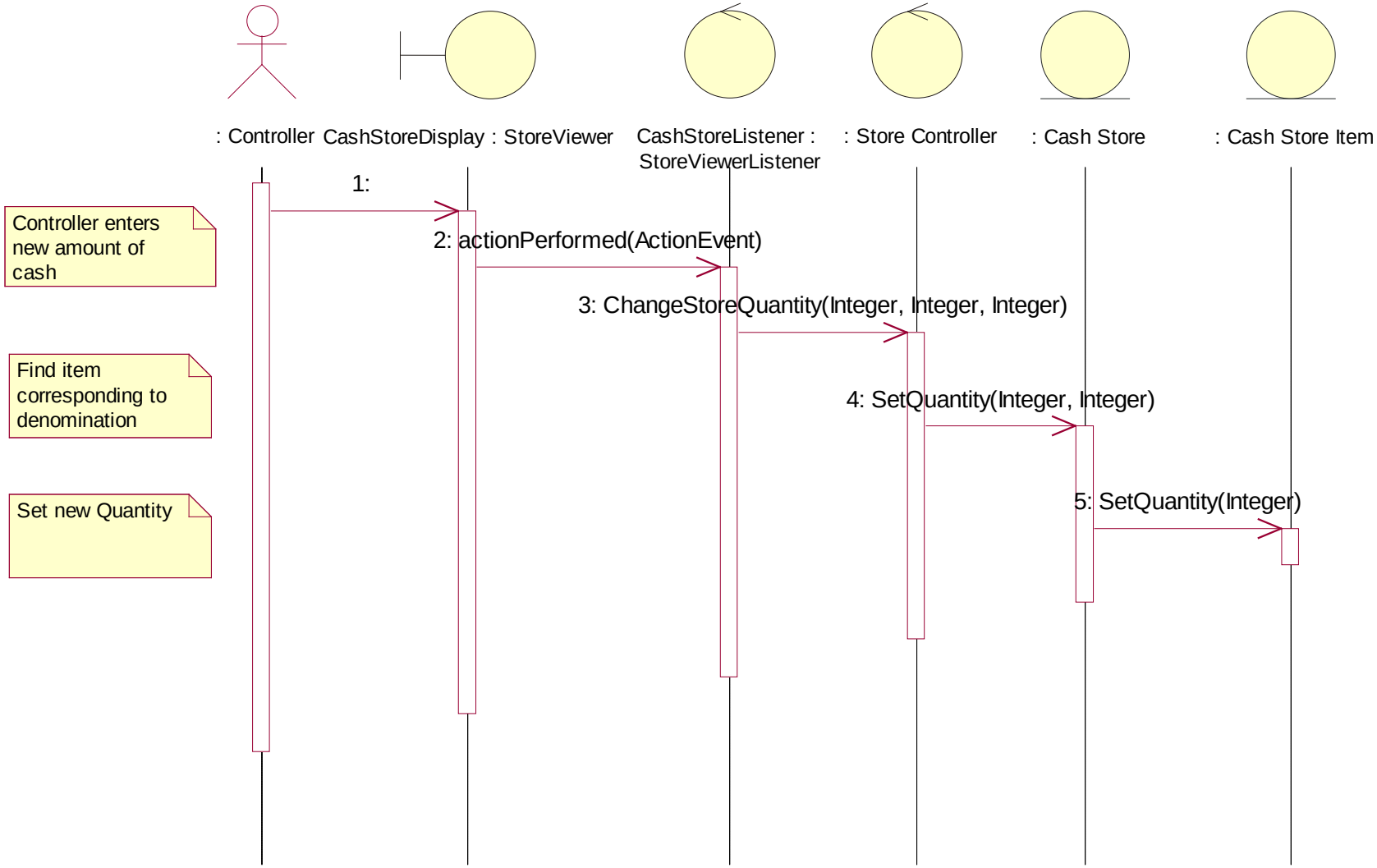


3.10 Change State

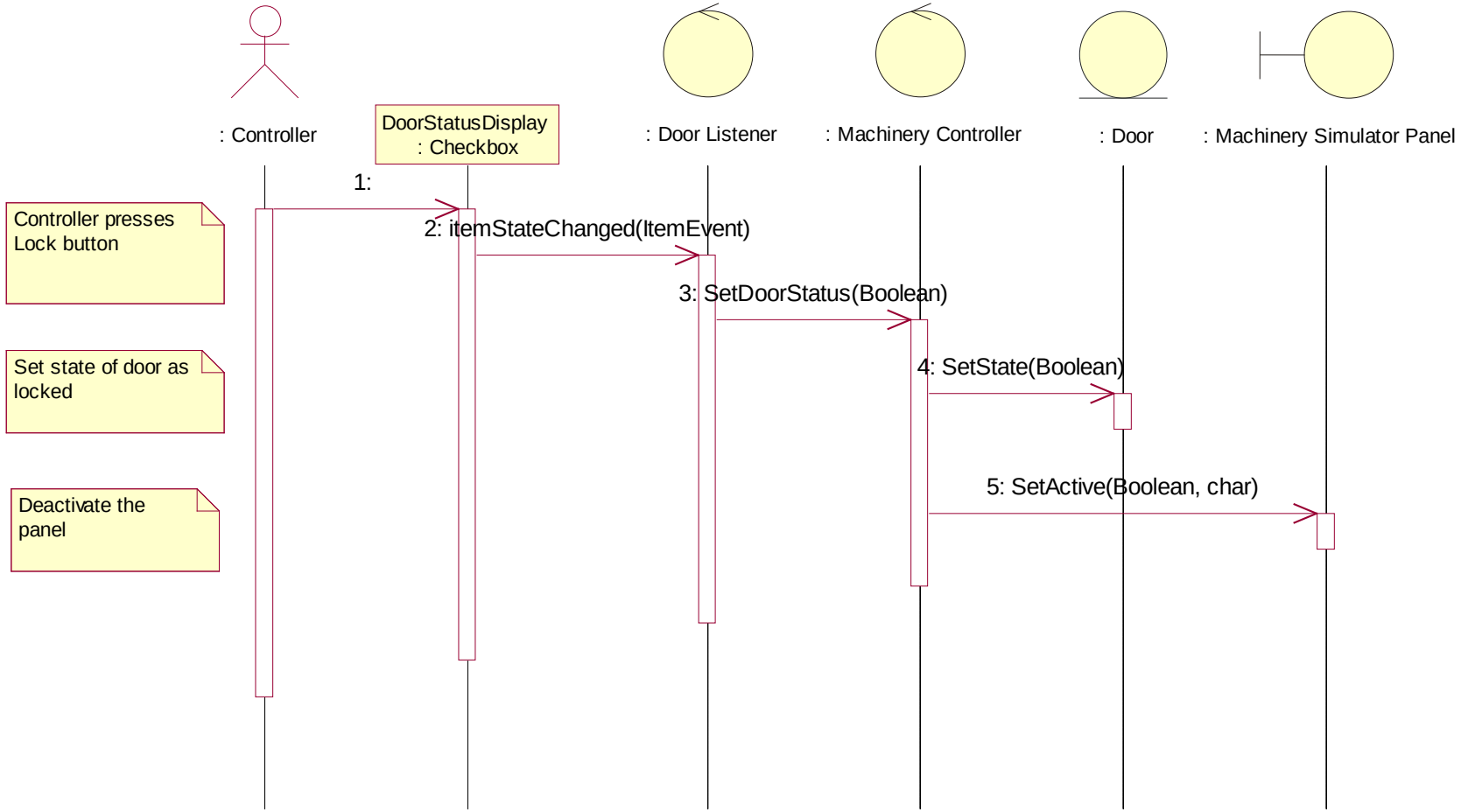
3.10.1 Sequence Diagram: Sequence Diagram: Change Drink Store



3.10.2 Sequence Diagram: Sequence Diagram: Change Cash Store



3.10.3 Sequence Diagram: Sequence Diagram: Lock the Door

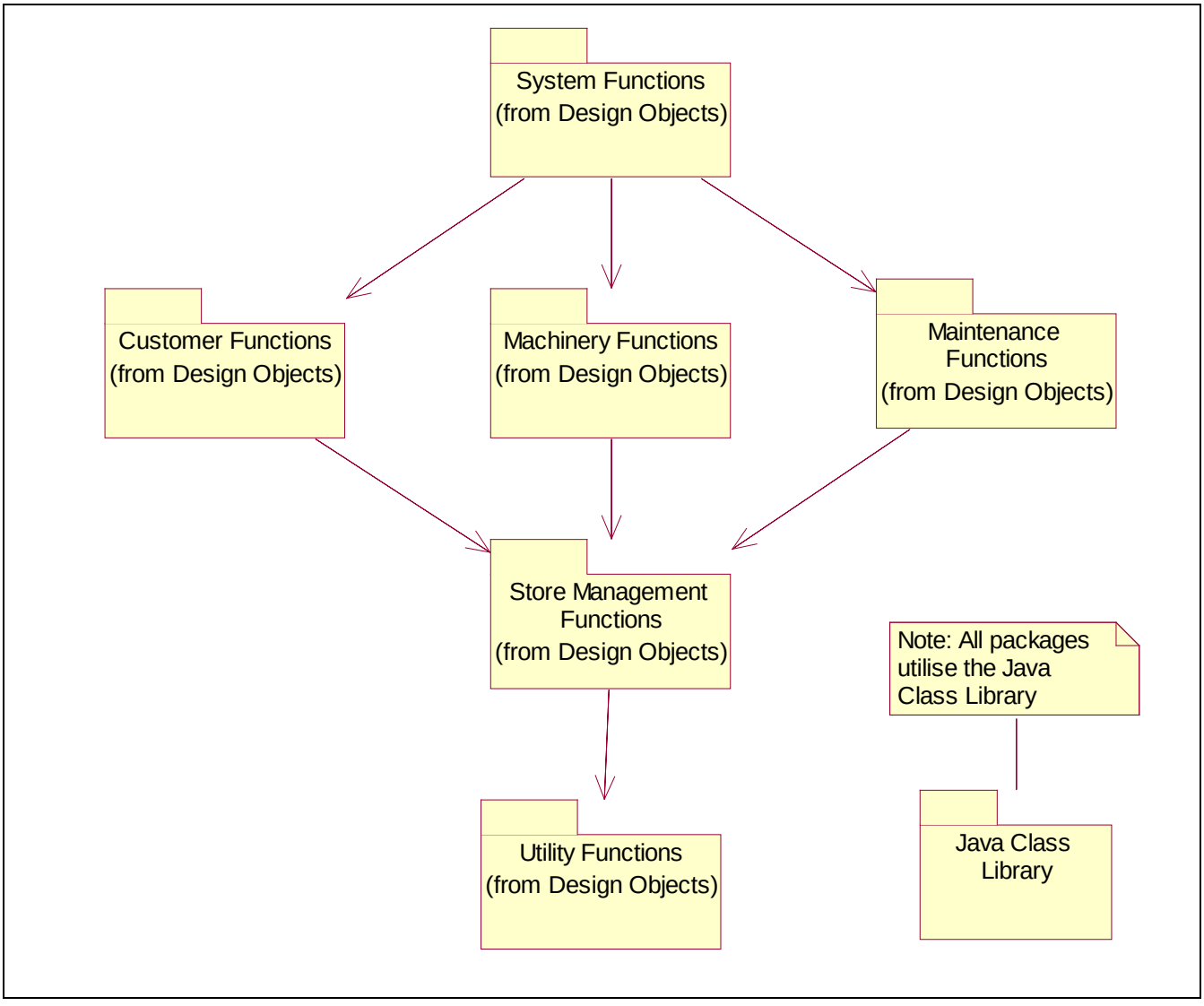


4. Object Specifications

4.1 Introduction

This section describes each object of the system in terms of attributes, operations and associations. The objects are structured into five packages, supported by the Java Class Library, as shown in section 4.1.1.

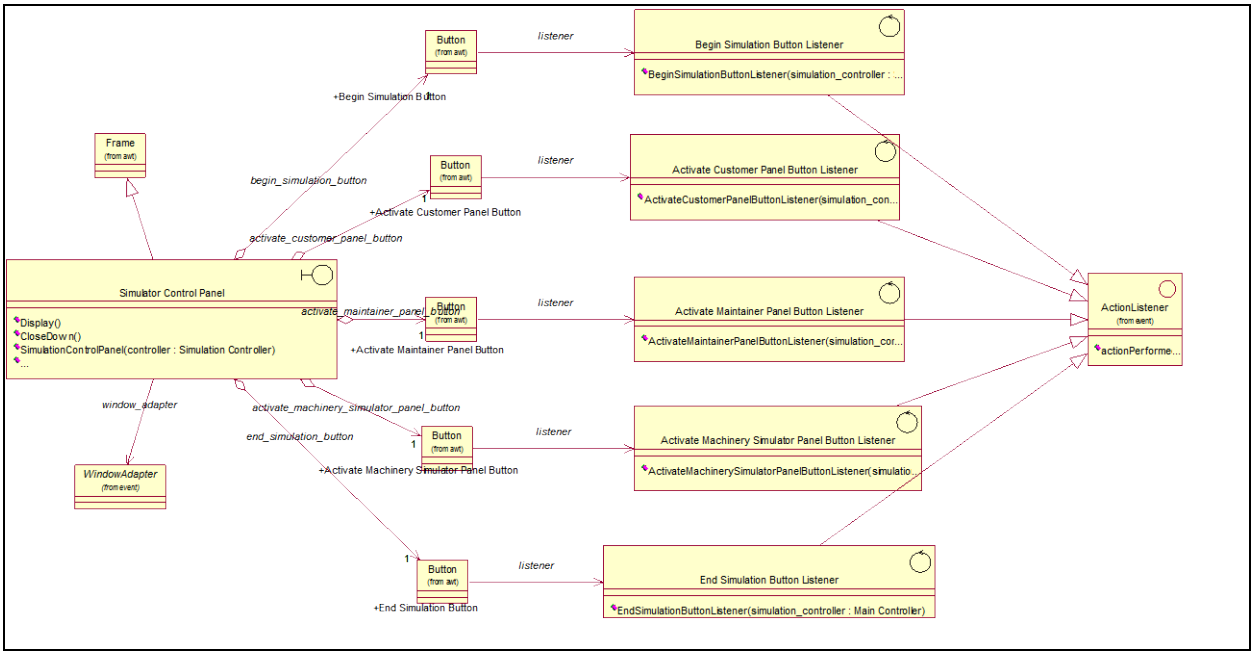
4.1.1 Package Structure



4.2 System Functions

The System Functions package contains the objects that facilitate the start-up and close-down of the system, and provide the main menu (Simulator Control Panel). A static view of the main objects is presented below followed by specifications of each object in the package.

4.2.1 *Static Structure of Simulator Control Panel*



4.2.2 ***Activate Customer Panel Button Listener***

This control object monitors the Activate Customer Panel Button and performs actions in response to the button being pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Button*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: ActivateCustomerPanelButtonListener
Documentation: This constructor creates an instance of the Activate Customer Panel Button Listener.

Parameters:

Name: *simulation_controller*
Type: *Simulation Controller*

4.2.3 *Activate Machinery Simulator Panel Button Listener*

This control object monitors the Activate Machinery Simulator Panel Button and performs actions in response to the button being pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Button*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: ActivateMachinerySimulatorPanelButtonListener
Documentation: This constructor creates an instance of the Activate Machinery Simulator Panel Button Listener.

Parameters:

Name: *simulation_controller*
Type: *Simulation Controller*

4.2.4 *Activate Maintainer Panel Button Listener*

This control object monitors the Activate Maintainer Panel Button and performs actions in response to the button being pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Button*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: ActivateMaintainerPanelButtonListener
Documentation: This constructor creates an instance of the Activate Maintainer Panel Button Listener.

Parameters:

Name: *simulation_controller*
Type: *Simulation Controller*

4.2.5 *Begin Simulation Button Listener*

This control object monitors the Begin Simulation Button and performs actions in response to the button being pressed.

Relationships

Object Name: *ActionListener*

Kind of Relationship: *InheritsRelationship*

Object Name: *Button*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: BeginSimulationButtonListener
Documentation: This constructor creates an instance of the Begin Simulation Button Listener.

Parameters:

Name: *simulation_controller*
Type: *Simulation Controller*

4.2.6 CashPropertyLoader

This control object manages the initialization of the Cash Store.

Relationships

Object Name: *File Property Loader*
Kind of Relationship: *InheritsRelationship*

Superclasses

File Property Loader
PropertyLoader

Operations

Name: GetItem
Documentation: This operation reads data from the hash table and creates a CashStoreItem.
Return Class: *Store Item*

Parameters:

Name: *identifier*
Type: *Integer*

Name: SetItem
Documentation: This operation updates the hash table with data from the CashStoreItem.

Parameters:

Name: *identifier*
Type: *Integer*

Name: *cash item*
Type: *Store Item*

Name: CashPropertyLoader
Documentation: This operation constructs the CashPropertyLoader.

Parameters:

Name: *filename*
Type: *String*

4.2.7 DrinkPropertyLoader

This control object manages the initialization of the Drinks Store.

Relationships

Object Name: *File Property Loader*
Kind of Relationship: *InheritsRelationship*

Superclasses

File Property Loader
PropertyLoader

Operations

Name: GetItem
Documentation: This operation reads data from the hash table and creates a DrinkStoreItem.
Return Class: *Store Item*

Parameters:

Name: *identifier*
Type: *Integer*

Name: SetItem
Documentation: This operation updates the hash table with data from the DrinkStoreItem.

Parameters:

Name: *identifier*
Type: *Integer*

Name: *drinks item*
Type: *Store Item*

Name: DrinkPropertyLoader
Documentation: This operation constructs the DrinkPropertyLoader.

Parameters:

Name: *filename*
Type: *String*

4.2.8 *End Simulation Button Listener*

This control object monitors the End Simulation Button and performs actions in response to the button being pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Button*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: EndSimulationButtonListener
Documentation: This constructor creates an instance of the End Simulation Button Listener.

Parameters:

Name: *simulation_controller*

Type: *Main Controller*

4.2.9 *Environment*

This entity object holds the environment settings used to configure the system.

Operations

Name: Environment
Documentation: This constructor creates an instance of the object.

Name: Initialize
Documentation: This operations reads-in the environment setting from a properties file and stores them in a hash table.

Name: GetCashPropFile
Documentation: This operation returns the name/location of the Cash Store properties file.
Return Class: *String*

Name: GetDrinkPropFile
Documentation: This operation returns the name/location of the Drinks Store properties file.
Return Class: *String*

4.2.10 *File Property Loader*

This control object implements the interface, Property Loader, to provide the generic functionality required to initialise the stores.

Relationships

Object Name: *PropertyLoader*
Kind of Relationship: *InheritsRelationship*

Object Name: *Properties*
Kind of Relationship: *AssociationRole*

Superclasses

PropertyLoader

Subclasses

DrinkPropertyLoader
CashPropertyLoader

Attributes

Name: property
Type: *Properties*
Documentation: Hash table into which the properties will be loaded from file.

Name: fileName
Type: *String*
Documentation: Name of the file from which the properties will be loaded.

Operations

Name: SetValue
Documentation: This operation sets a value in the hash table.

Parameters:

Name: *key*
Type: *String*

Name: *value*
Type: *String*

Name: GetValue
Documentation: This operation gets a value from the hash table.

Parameters:

Name: *key*
Type: *String*

Name: Initialize
Documentation: This operation reads the properties file into a hash table.

Name: Save Property
Documentation: This operation writes the properties from the hash table to the file.

Name: GetNumOfItems
Documentation: This operation returns the number of items (either Cash Store Items or Drink Store Items) stored in the hash table.
Return Class: *Integer*

Name: SetNumOfItems
Documentation: This operation sets the number of items (either Cash Store Items or Drink Store Items) stored in the hash table.

Parameters:

Name: *items*
Type: *Integer*

Name: GetItem
Documentation: This operation reads data from the hash table and creates a StoreItem.
Return Class: *Store Item*

Parameters:

Name: *index*
Type: *Integer*

Name: SetItem
Documentation: This operation updates the hash table with data from the StoreItem.

Parameters:

Name: *index*
Type: *Integer*

Name: *item*
Type: *Store Item*

Name: FilePropertyLoader
Documentation: This operation constructs the FilePropertyLoader.

Parameters:

Name: *filename*
Type: *String*

4.2.11 *Main Controller*

This object is the main controller of the vending machine. It coordinates the main operations of the VMCS.

Operations

Name: MainController
Documentation: This constructor will create the Main Controller.

Parameters:

Name: *Property_File*
Type: *String*

Name: CloseDown
Documentation: This operation destroys all the object instances created for operating the vending machine. It will instruct the Simulation Controller to close down (by closing down all the vending machine panels) and the Transaction Controller to close down. It will also close down other control objects and the entity objects created for simulating the vending machine.

Name: Start
Documentation: This operation will initiate the creation of all the control objects necessary for simulation of the vending machine, and initiate the display of the Simulator Control Panel (the main menu).

Name: Initialize
Documentation: This operation creates all the control objects.

4.2.12 *Operating System Interface*

This object represents the operating system user interface at which the VMCS application can be launched.

Operations

Name: LaunchApplication
Documentation: This operation represented the operating system command that launches the VMCS application.

4.2.13 *Simulation Controller*

This control object is used to set up the 3 panels (Customer Panel, Maintenance Panel and the Machinery Simulator Panel) of the vending machine.

Operations

Name: Start
Documentation: This operation is triggered when the Begin Simulation Button is pressed on the Simulator Control Panel. It will start the simulation of vending machine by activating the three panel buttons namely; Activate Customer Panel Button, Activate Maintainer Panel Button and Activate Machinery Simulator Button.

Name: SetupMaintainer
Documentation: Set-up the Maintenance Panel. This will be acheived as follows:
1 Send a message to deactivate the Activate Maintainer Panel Button.
2 Send messages to instruct all its constituent objects to display themselves, and also instruct them all, except for the Password Box to become deactivated.

Name: SetupCustomer
Documentation: Set-up the Customer Panel. This will be acheived as follows:
1 Send a message to deactivate the Activate Customer Panel Button.
2 Send a message to the Customer Panel instructing it to display all its constituent interface objects.
3 Display the current values for out of stock brands and prices.
4 Activate the No Change Available Display (if necessary).

Name: SetupSimulator
Documentation: Set-up the Machinery Simulator Panel. This will be acheived as follows:
1 Deactivate the Activate Machinery Simulator Button.
2 Display Machinery Simulator Panel with intial values set.
3 Deactivate the panel.

Name: SimulationController
Documentation: This constructor will create a new instance of Simulation Controller.

Parameters:

Name: *main_controller*
Type: *Main Controller*

Name: DisplaySimulationControlPanel
Documentation: Set-up the Simulator Control Panel. This will be acheived as follows:
1 Create and display the panel.
2 Activate the Begin Simulation Button.
3 Deactivate the rest of the buttons on the panel.

Name: SetSimulationActive
Documentation: If FALSE is passed the Begin Simulation Button is enabled and the other buttons of the Simulator Control Panel are disabled. If TRUE is passed the Begin Simulation Button is disabled and the other buttons of the Simulator Control Panel are enabled.

Parameters:

Name: *active*
Type: *Boolean*

Name: CloseDown
Documentation: This operation will instruct all four panels of the vending machine to close down when instructed by the Main Controller.

4.2.14 *Simulator Control Panel*

This panel is displayed when the system is started up. It enables the user (the Controller) to :
1 start (switch on);
2 stop (switch off);
3 activate the customer panel;
4 activiate the maintenance panel;
5 activate the machinery simulator panel.

Relationships

Object Name: *Frame*
Kind of Relationship: *InheritsRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Object Name: *Button*

Kind of Relationship: *AggregateRelationship*

Object Name: *WindowAdapter*
Kind of Relationship: *AssociationRole*

Superclasses

Frame
Window
Container
Component

Operations

Name: Display
Documentation: Display the Simulator Control Panel and its constituent objects.

Name: CloseDown
Documentation: Remove the panel objects and the Simulation Control Panel from the display.

Name: SimulationControlPanel
Documentation: This constructor creates an instance of the Simulator Control Panel. It further creates Activate Customer Panel Button, Activate Machinery Simulator Panel Button, Activate Maintainer Panel Button, Begin Simulation Button, End Simulation Button.

Parameters:

Name: *controller*
Type: *Simulation Controller*

Name: SetActive
Documentation: This operation activates or deactivates the panel and its component objects.

Parameters:

Name: *active*
Type: *Boolean*

Name: *component*
Type: *Char*

Name: SetSimulationActive
Documentation: If FALSE is passed then all buttons of the Simulator Control Panel except the Begin Simulation Button are disabled. If TRUE is passed then all buttons of the Simulator Control Panel except the Begin Simulation Button are enabled.

Parameters:

Name: *active*
Type: *Boolean*

4.2.15 *VMCS*

This object acts as the main program of the application.

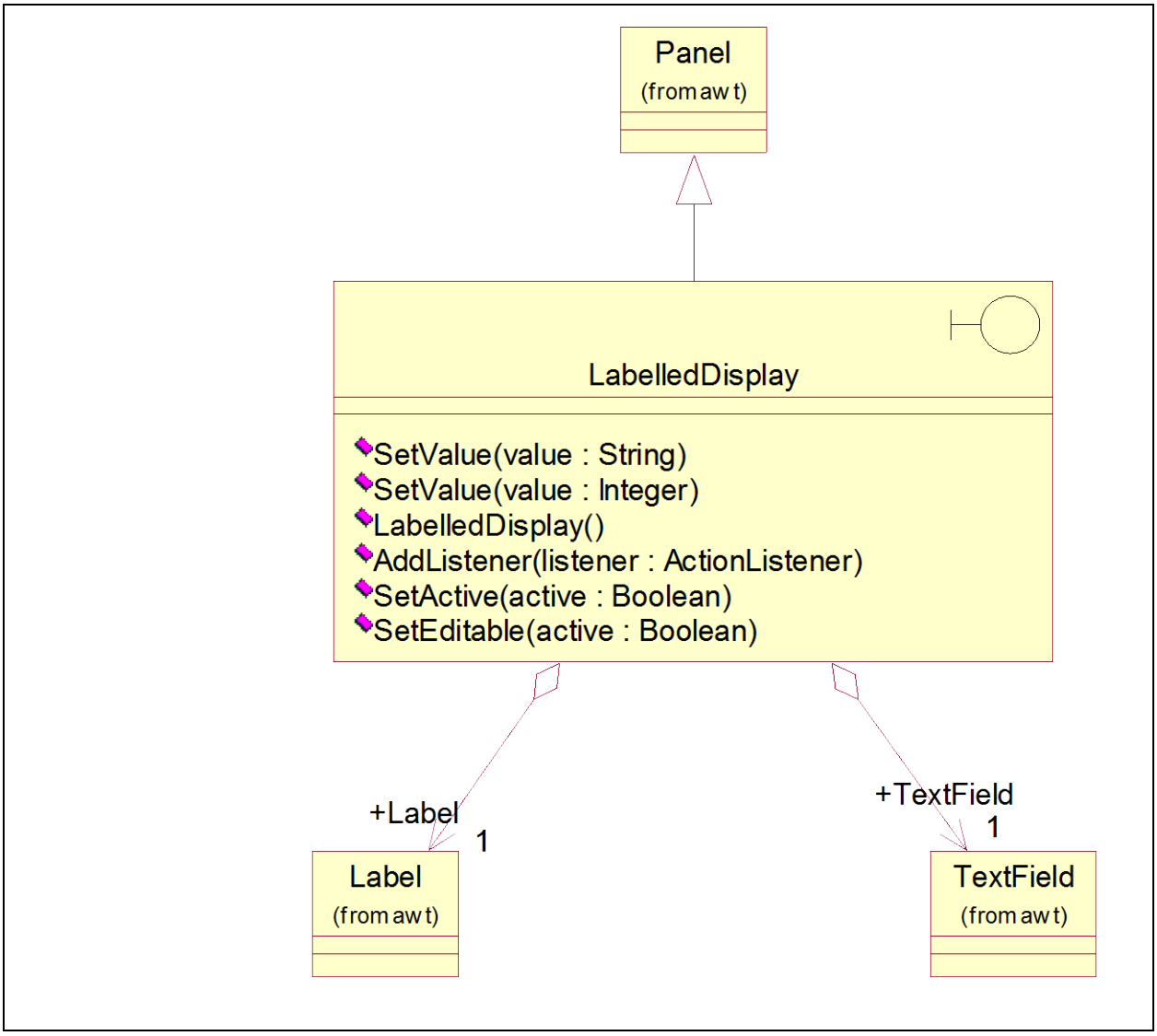
Operations

Name: Main
Documentation: The operation builds/runs the application.

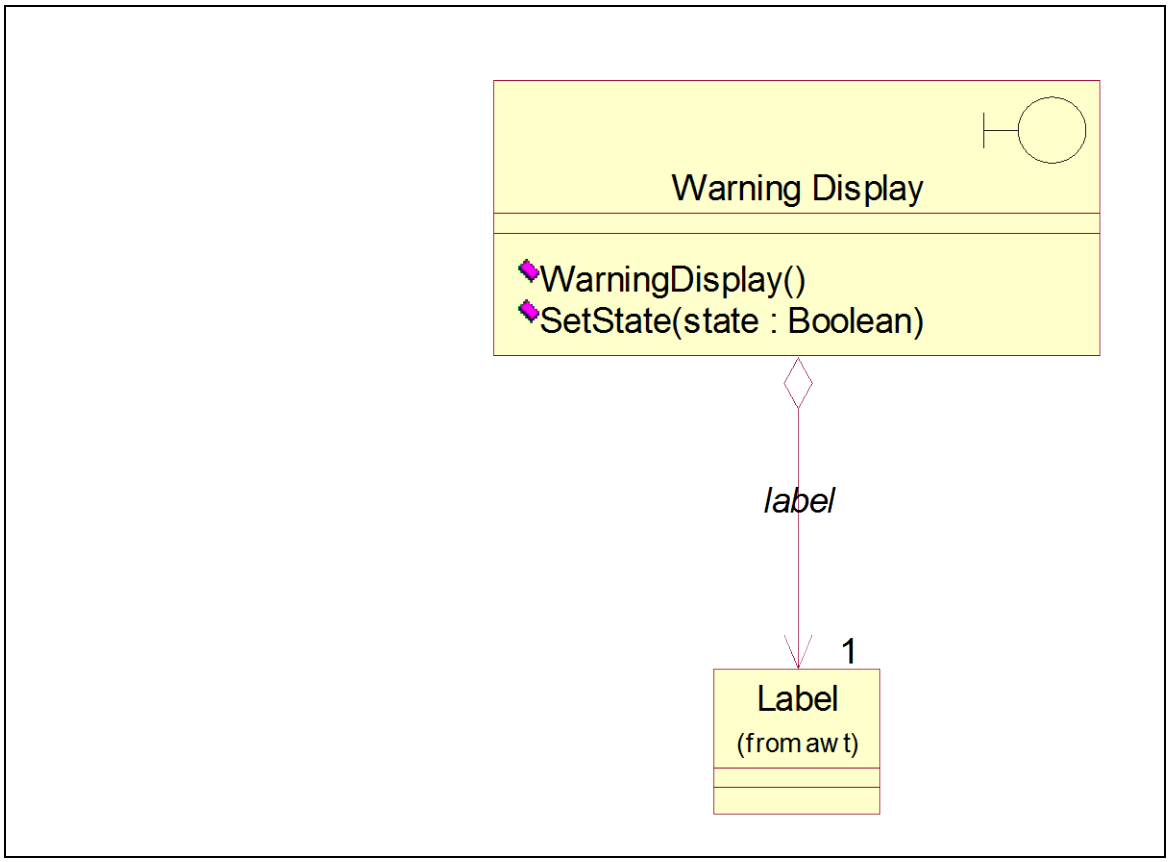
4.3 Utility Functions

The Utility Functions package contains objects that provide common functionality utilised across the system – specifically objects that provide common user interface functionality. Static views of the main objects are presented below followed by specifications of each object in the package.

4.3.1 Structure of LabelledDisplay



4.3.2 *Structure of Warning Display*



4.3.3 *LabelledDisplay*

This bounbary object displays an updatable text fiels next to a fixed label.

Relationships

Object Name: *Panel*
Kind of Relationship: *InheritsRelationship*

Object Name: *Label*
Kind of Relationship: *AggregateRelationship*

Object Name: *TextField*
Kind of Relationship: *AggregateRelationship*

Object Name: *Price Display Listener*
Kind of Relationship: *AssociationRole*

Object Name: *Password Listener*
Kind of Relationship: *AssociationRole*

Superclasses

Panel
Container
Component

Operations

Name: SetValue

Documentation: This operation sets the text to be displayed.

Parameters:

Name: *value*
Type: *String*

Name: SetValue
Documentation: This operation sets the integer value to be displayed.

Parameters:

Name: *value*
Type: *Integer*

Name: LabelledDisplay
Documentation: This constructor will create a new instance of the LabelledDisplay.

Name: AddListener
Documentation: This operation attaches a listener to the LabelledDisplay.

Parameters:

Name: *listener*
Type: *ActionListener*

Name: SetActive
Documentation: This operation activates the LabelledDisplay if the parameter is TRUE. Otherwise, the LabelledDisplay is deactivated.

Parameters:

Name: *active*
Type: *Boolean*

Name: SetEditable
Documentation: This operations sets whether the TextField in the LabelledDisplay is editable.

Parameters:

Name: *active*
Type: *Boolean*

4.3.4 *Warning Display*

This interface object class implements a generic display label. It provides for setting the background and foreground colors for display labels, and to set their state to on or off.

Relationships

Object Name: *Label*
Kind of Relationship: *AggregateRelationship*

Operations

Name: WarningDisplay
Documentation: This constructor creates an instance of the Warning Display.

Name: SetState
Documentation: This operation will set the background and foreground colors for the displays depending on whether the display needs to be on or off.

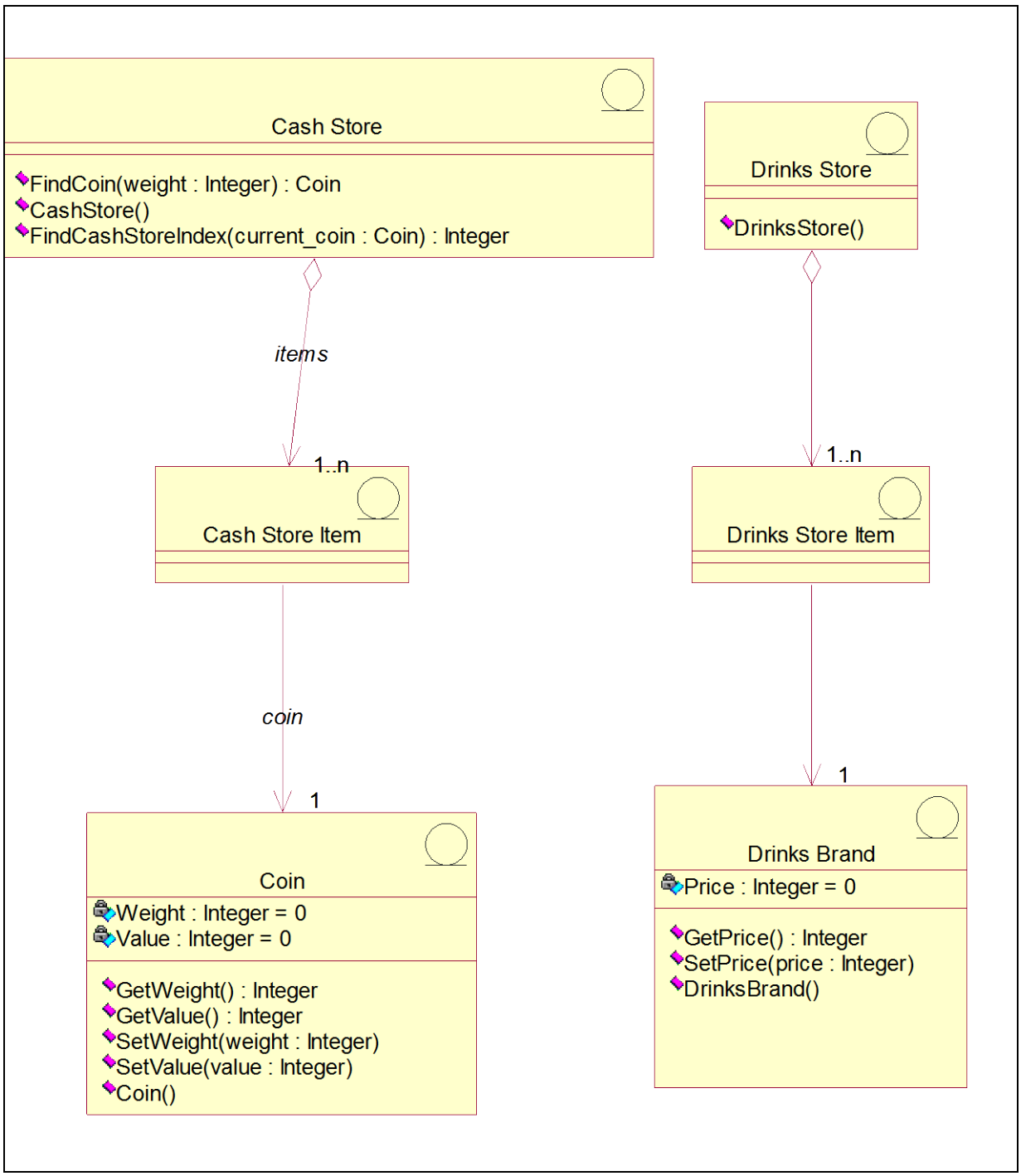
Parameters:

Name: *state*
Type: *Boolean*

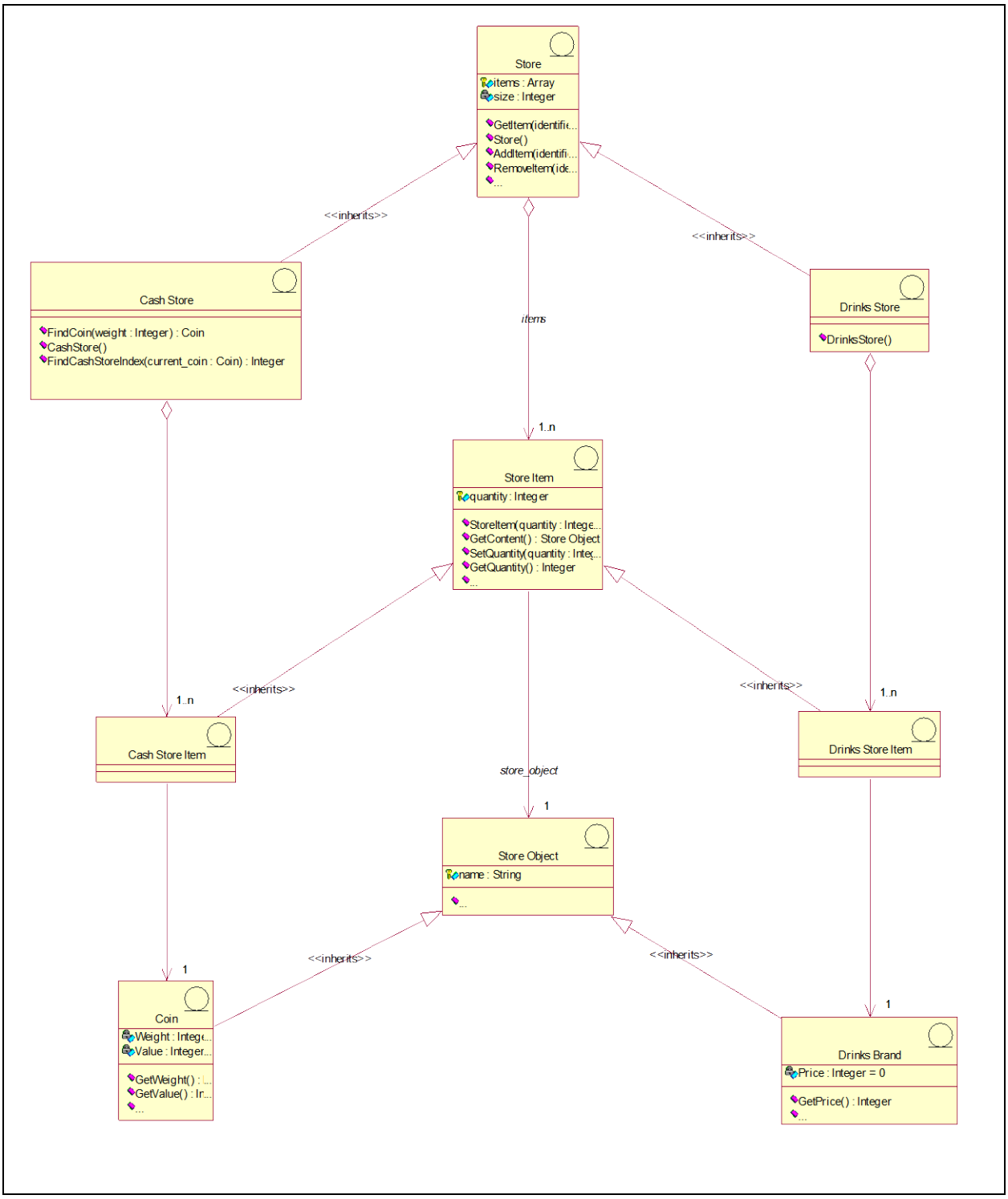
4.4 Store Management Functions

The Store Management Functions package contains the objects that facilitate the access to the Drink Store, Cash Store and Door of the vending machine. Static views of the main objects are presented below followed by specifications of each object in the package.

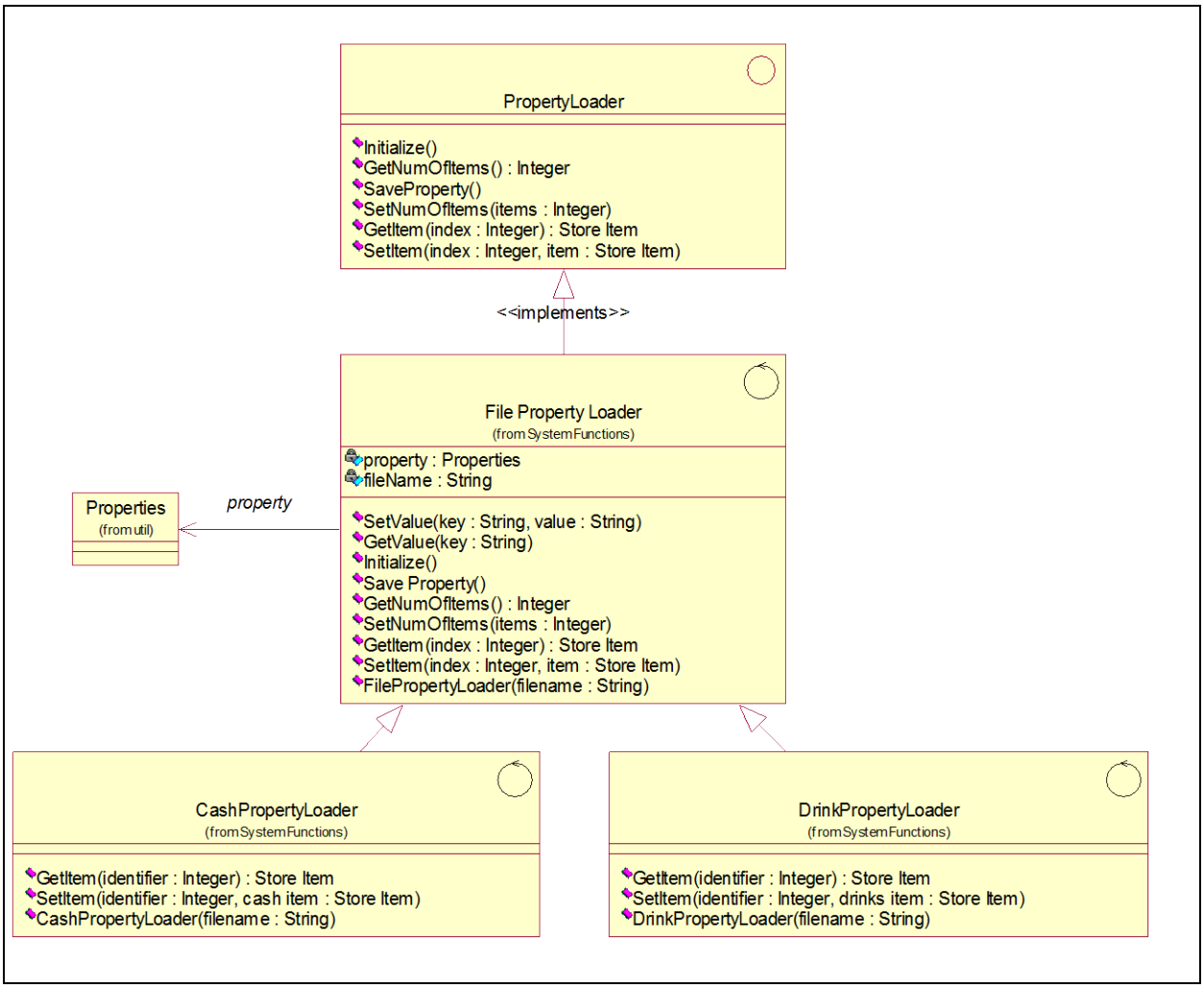
4.4.1 Containment Structure of Stores



4.4.2 Inheritance Structure of Stores



4.4.3 Structure of the Property Loaders



4.4.4 *Cash Store*

This object represents the store of cash in the vending machine.

Relationships

Object Name: *Store*
Kind of Relationship: *InheritsRelationship*

Object Name: *Cash Store Item*
Kind of Relationship: *AggregateRelationship*

Superclasses

Store

Operations

Name: FindCoin
Documentation: This operation will locate a coin denomination held, with the input data (coin weight). If found, it returns an existence identifier (reference). Otherwise, it informs the requestor that the coin is invalid.
Return Class: *Coin*

Parameters:

Name: *weight*
Type: *Integer*

Name: CashStore
Documentation: This constructor creates a new instance of the Cash Store. It creates the required number of instances of Cash Store Item and Coin objects

Name: FindCashStoreIndex
Documentation: This operation is used to find the index to a Cash Store Item for a specific Coin.
Return Class: *Integer*

Parameters:

Name: *current_coin*
Type: *Coin*

4.4.5 Cash Store Item

This entity object represents a column of coins in the vending machine. It can store a coin of any particular type. There can be as many Cash Store Items as we desire in the vending machine. There can be more than one Cash Store Item storing the same or different types of coin. This number will be configurable.
This will be implemented as an instance of Store Item.

Relationships

Object Name: *Store Item*
Kind of Relationship: *InheritsRelationship*

Object Name: *Coin*
Kind of Relationship: *AssociationRole*

Superclasses

Store Item

4.4.6 Coin

This object stores the weight and value of each type of Coin, and hence enables the machine to recognise each Coin entered.

Relationships

Object Name: *Store Object*
Kind of Relationship: *InheritsRelationship*

Object Name: *Cash Store Item*
Kind of Relationship: *AssociationRole*

Superclasses

Store Object

Attributes

Name: Weight
Type: *Integer*
Initial Value: *0*
Documentation: The weight of the coin in grammes.

Name: Value
Type: *Integer*
Initial Value: *0*

Documentation: The value of coin in cents.

Operations

Name: GetWeight
Documentation: Returns the weight of the coin.
Return Class: *Integer*

Name: GetValue
Documentation: Returns the value of the coin.
Return Class: *Integer*

Name: SetWeight
Documentation: Sets the weight of the coin.

Parameters:

Name: *weight*
Type: *Integer*

Name: SetValue
Documentation: Sets the value of the coin.

Parameters:

Name: *value*
Type: *Integer*

Name: Coin
Documentation: This constructor creates an instance of the object.

4.4.7 *Drinks Brand*

This entity object stores the name of the drink brand and it's price. There can be as many drink brands as we desire, for a particular configuration.

Relationships

Object Name: *Store Object*
Kind of Relationship: *InheritsRelationship*

Object Name: *Drinks Store Item*
Kind of Relationship: *AssociationRole*

Superclasses

Store Object

Attributes

Name: Price
Type: *Integer*
Initial Value: *0*
Documentation: The price of this brand in cents.

Operations

Name: GetPrice
Documentation: This operation sends the price of the requested drinks brand to the requestor.
Return Class: *Integer*

Name: SetPrice
Documentation: This operation sets the price of the brand.

Parameters:

Name: *price*
Type: *Integer*

Name: DrinksBrand
Documentation: This constructor creates an instance of the Drinks Brand.

4.4.8 *Drinks Store*

The object is storage, in the vending machine's memory, for the total number of cans of each Drinks Brand held by the vending machine.

Relationships

Object Name: *Store*
Kind of Relationship: *InheritsRelationship*

Object Name: *Drinks Store Item*
Kind of Relationship: *AggregateRelationship*

Superclasses

Store

Operations

Name: DrinksStore
Documentation: This constructor creates a new instance of the Drinks Store. It creates the required number of instances of Drink Store Items.

4.4.9 *Drinks Store Item*

This represents a column of drinks in the vending machine. It can store a drink of any particular brand type.There can be as many drink store items as we desire in the vending machine. This number will be a configurable item for the vending machine. This will be implemented as an instance of Store Item.

Relationships

Object Name: *Store Item*
Kind of Relationship: *InheritsRelationship*

Object Name: *Drinks Brand*
Kind of Relationship: *AssociationRole*

Superclasses

Store Item

4.4.10 *PropertyLoader*

This interface provides the generic functionality required to initialise the stores.

Subclasses

File Property Loader
DrinkPropertyLoader
CashPropertyLoader

Operations

Name: Initialize
Documentation: This operation reads the properties file into a hash table.

Name: GetNumOfItems
Documentation: This operation returns the number of items (either Cash Store Items or Drink Store Items) stored in the hash table.
Return Class: *Integer*

Name: SaveProperty
Documentation: This operation writes the properties from the hash table to the file.

Name: SetNumOfItems
Documentation: This operation sets the number of items (either Cash Store Items or Drink Store Items) stored in the hash table.

Parameters:

Name: *items*
Type: *Integer*

Name: GetItem
Documentation: This operation reads data from the hash table and creates a StoreItem.
Return Class: *Store Item*

Parameters:

Name: *index*
Type: *Integer*

Name: SetItem
Documentation: This operation updates the hash table with data from the StoreItem.

Parameters:

Name: *index*
Type: *Integer*

Name: *item*
Type: *Store Item*

4.4.11 *Store*

This entity object implements a generic Store. It has operations to load (add) Store Items into the Store and release Store Items from the Store.

Relationships

Object Name: *Store Item*
Kind of Relationship: *AggregateRelationship*

Object Name: *StoreViewer*
Kind of Relationship: *AssociationRole*

Object Name: *ButtonItemDisplay*
Kind of Relationship: *AssociationRole*

Subclasses

Cash Store
Drinks Store

Attributes

Name: items
Type: *Array*
Documentation: List containing the Store Items contained in the store.

Name: size
Type: *Integer*
Documentation: The size of the Store (ie: the size of the items array).

Operations

Name: GetItem
Documentation: This operation returns the Store Item corresponding to the index entered.
Return Class: *Store Item*

Parameters:

Name: *identifier*
Type: *Integer*

Name: Store
Documentation: This constructor creates a new instance of Store.

Name: AddItem
Documentation: This operation adds store items into the store.

Parameters:

Name: *identifier*
Type: *Integer*

Name: *store_item*
Type: *Store Item*

Name: RemoveItem
Documentation: This operation removes a store item from the store.

Parameters:

Name: *identifier*
Type: *Integer*

Name: FindObject
Documentation: This operation finds a Store Object in the store with a specified name.
Return Class: *Store Object*

Parameters:

Name: *name*
Type: *String*

Name: SetStoreSize
Documentation: This operation sets the size of the items array in the Store.

Parameters:

Name: *size*
Type: *Integer*

Name: GetStoreSize
Documentation: This operation gets the size of the items array in the Store.
Return Class: *Integer*

Name: SetQuantity
Documentation: This operation sets the total number of a store item held.

Parameters:

Name: *index*
Type: *Integer*

Name: *quantity*
Type: *Integer*

4.4.12 Store Controller

This control object manages changes in Cash Store attributes and the Drinks Store attributes.

Operations

Name: ChangeStoreQuantity
Documentation: This operation will either:
- instruct the Cash Store to update the quantity of a coin denomination to the new value supplied and update the total cash held in the Cash Store; or
- instruct the Drinks Store to update the drinks stock for a drinks brand requested to a new value supplied.

Parameters:

Name: *type*
Type: *Integer*

Name: *identifier*
Type: *Integer*

Name: *value*
Type: *Integer*

Name: CloseDown
Documentation: This operation will close down the store management function of the vending machine. This involves saving the attributes of the stores to the property files.

Name: DispenseDrink
Documentation: This operation instructs the the Drinks Store to dispense one drink, and then updates the Machinery Simulator Panel. It returns TRUE or FALSE to indicate whether dispensing was successful.
Return Class: *Boolean*

Parameters:

Name: *selected_brand*
Type: *Integer*

Name: GetTotalCash
Documentation: This operation returns the total cash held in the Cash Store.
Return Class: *Integer*

Name: GiveChange
Documentation: This operation instructs the Cash Store to issue a number of coins of a specific denomination, and then updates the Machinery Simulator Panel. It returns TRUE or FALSE to indicate whether the change issue was successful.
Return Class: *Boolean*

Parameters:

Name: *index*
Type: *Integer*

Name: *number_coins*
Type: *Integer*

Name: InitializeCashStore
Documentation: This operation initializes the Cash Store.

Name: InitializeDrinkStore
Documentation: This operation initializes the Drink Store.

Name: InitializeStores

Documentation: This operation initiates the initialisation of the Drink Store and the Cash Store from data read-in from an input file.

Name: SaveCashProperty
Documentation: This operation saves the attributes of the Cash Store to the input file.

Name: SaveDrinksProperty
Documentation: This operation saves the attributes of the Drinks Store to the input file.

Name: SetPrice
Documentation: This operation will be used to set the price of a drink brand as determined by the Maintainer.

Parameters:

Name: *index*
Type: *Integer*

Name: *price*
Type: *Integer*

Name: StoreCoin
Documentation: This operation will instuct the Cash Store to store the Coin sent as input, and the update the display on the Machinery Simulator Panel.
Return Class: *Boolean*

Parameters:

Name: *current_coin*
Type: *Coin*

Name: StoreController
Documentation: This constructor creates an instance of the Store Controller. It will then create the Cash Store, Drinks Store and Door.

Parameters:

Name: *cash loader*
Type: *PropertyLoader*

Name: *drinks loader*
Type: *PropertyLoader*

Name: TransferAll
Documentation: This operation is to facilitate the transfer of all cash in Cash Store to the maintainer.
Return Class: *Integer*

Name: Initialize
Documentation: This operation initializes the Store Controller and the two stores it controls.

Name: GetStore
Documentation: This operation returns a store of a specified type (ie: Cash or Drinks).
Return Class: *Store*

Parameters:

Name: *type*
Type: *Integer*

4.4.13 Store Item

This entity object implements a generic storage item class. It performs actions like; returning content (Store Item identification), setting quantity, returning quantity, increment quantity, decrement (release) quantity .

Relationships

Object Name: *Store Object*
Kind of Relationship: *AssociationRole*

Subclasses

Drinks Store Item
Cash Store Item

Attributes

Name: quantity
Type: *Integer*
Documentation: Quantity of Store Objects contained in the Store Item.

Operations

Name: StoreItem
Documentation: This constructor creates a new instance of the Store Item. It receives the quantity of the Store Item and a reference to the Store Object it contains.

Parameters:

Name: *quantity*
Type: *Integer*

Name: *store_object*
Type: *Store Object*

Name: GetContent
Documentation: Returns the object type that is stored in the Store Item.
Return Class: *Store Object*

Name: SetQuantity
Documentation: This operation sets the total number of the store item held.

Parameters:

Name: *quantity*
Type: *Integer*

Name: GetQuantity
Documentation: Returns the total number of the store item held.
Return Class: *Integer*

Name: Increment
Documentation: This operation increments the item quantity by one, and returns whether successful.
Return Class: *Boolean*

Name: Decrement
Documentation: This operation decrements the item quantity by one, and returns whether successful.
Return Class: *Boolean*

Name: SetContent
Documentation: This operation sets the Store Object held by the Store Item.

Parameters:

Name: *object*
Type: *Store Object*

4.4.14 *Store Object*

This entity object represents the generic object that is contained in a Store Item.

Relationships

Object Name: *Store Item*
Kind of Relationship: *AssociationRole*

Subclasses

Drinks Brand
Coin

Attributes

Name: name
Type: *String*
Documentation: Name of the Store Object.

Operations

Name: GetName
Documentation: This operation returns the name of the Store Object.
Return Class: *String*

Name: SetName
Documentation: This operation sets the name of the Store Object.
Return Class:

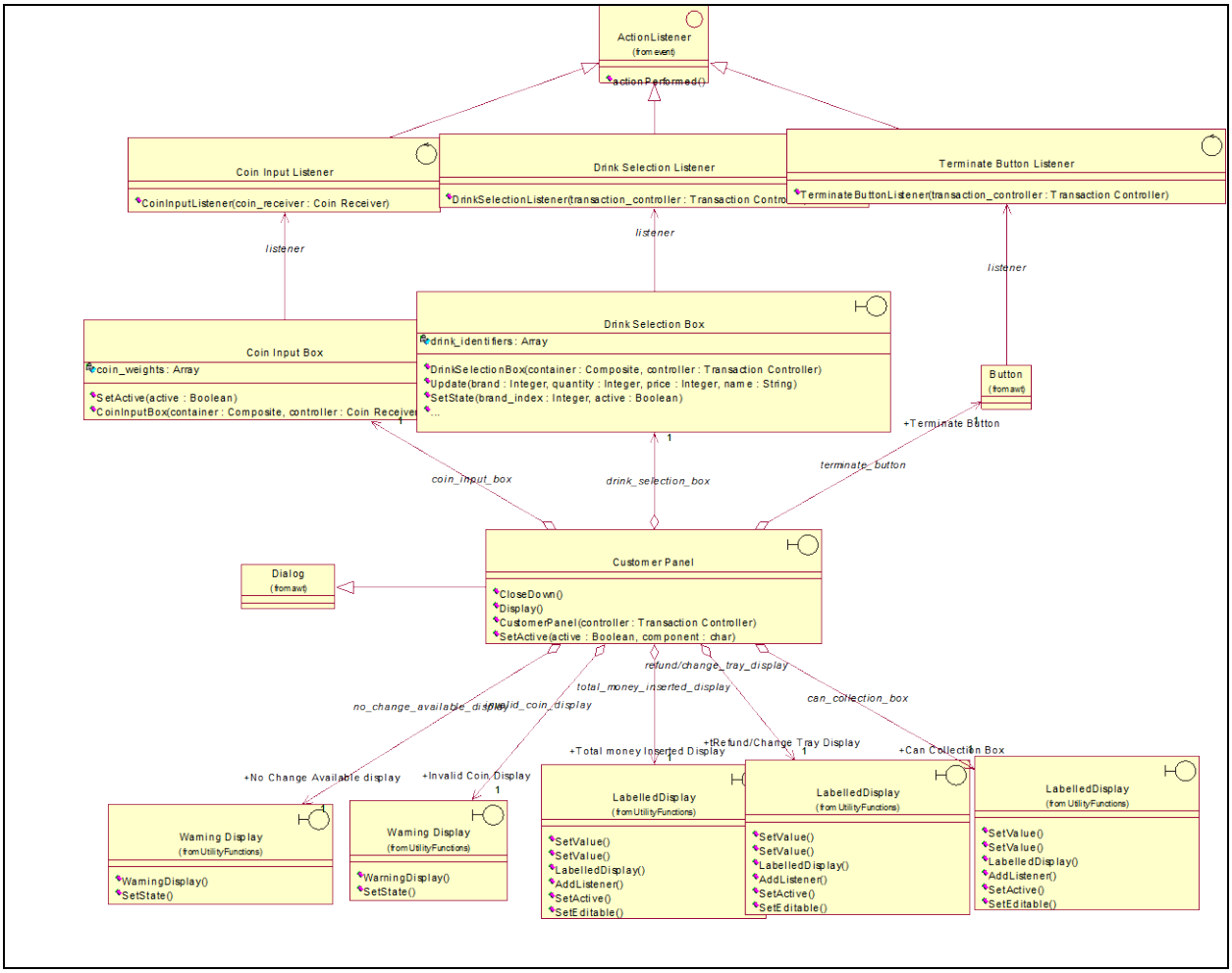
Parameters:

Name: *name*
Type: *String*

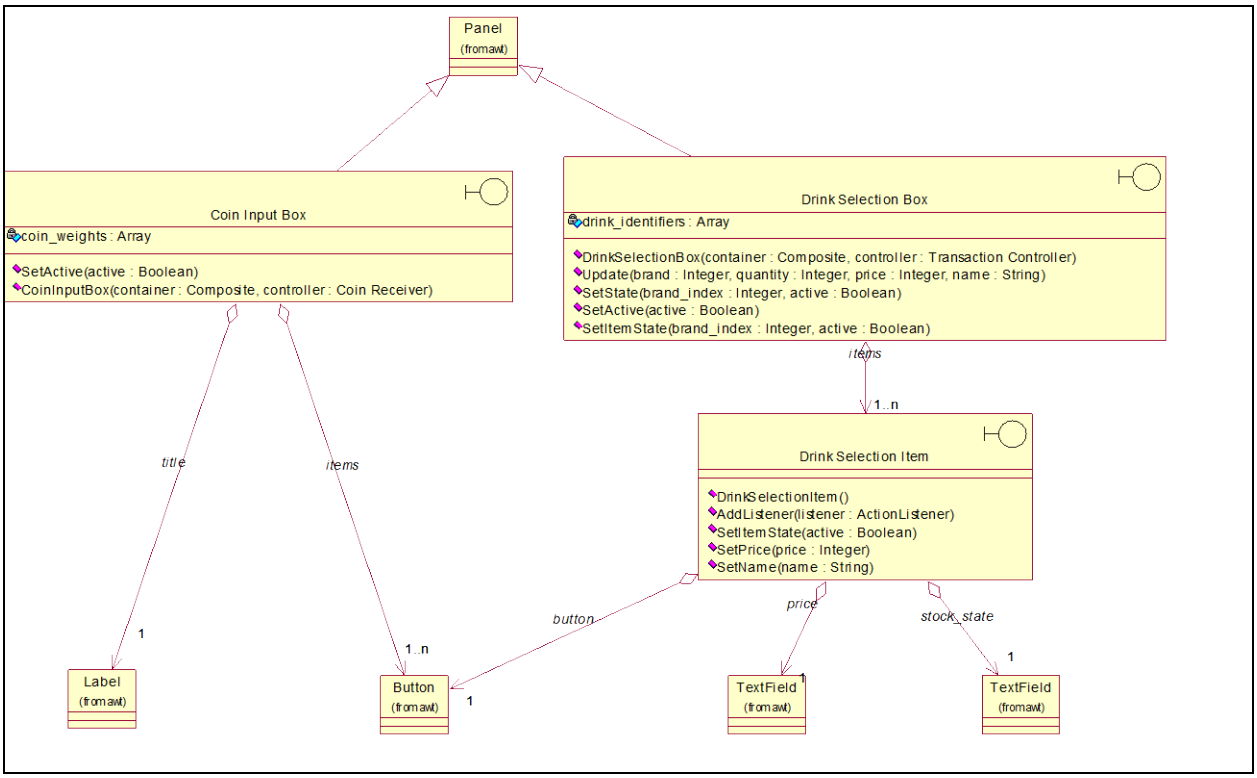
4.5 Customer Functions

The Customer Functions package contains the objects that facilitate the customer transactions of the vending machine. Static views of the main objects are presented below followed by specifications of each object in the package.

4.5.1 Static Structure of Customer Panel



4.5.2 Static Structure of Customer Panel Components



4.5.3 *Change Giver*

This control object manages the giving of change to the Customer.

Operations

Name: ResetChange
Documentation: This operation is used to reset the Refund/Change Tray display on the Customer Panel.

Name: GiveChange
Documentation: This operation manages the issuing of change to the Customer.

Return Class: *Boolean*

Parameters:

Name: *change_required*
Type: *Integer*

Name: DisplayChangeStatus
Documentation: This operation is used to display the appropriate message on the No Change Available Display depending on the current change availability.

Name: ChangeGiver
Documentation: This constructor creates an instance of the object.

Parameters:

Name: *transaction_controller*
Type: *Transaction Controller*

4.5.4 *Coin Input Box*

This interface object is part of the Customer Panel. It is used to enter Coins into the vending machine.

Relationships

Object Name: *Panel*
Kind of Relationship: *InheritsRelationship*

Object Name: *Coin Input Listener*
Kind of Relationship: *AssociationRole*

Object Name: *Label*
Kind of Relationship: *AggregateRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Superclasses

Panel
Container
Component

Attributes

Name: coin_weights
Type: *Array*
Documentation: Array of the weights of the Coins represented by each button in the Coin Input Box.

Operations

Name: SetActive
Documentation: This operation activates the Coin Input Box if the parameter is TRUE. Otherwise, the Coin Input Box is deactivated.

Parameters:

Name: *active*
Type: *Boolean*

Name: CoinInputBox
Documentation: This constructor creates an instance of the object.

Parameters:

Name: *container*
Type: *Composite*

Name: *controller*
Type: *Coin Receiver*

4.5.5 *Coin Input Listener*

This control object implements the coin denomination selection (button presses by the customer) on the Customer Panel when coins are entered for a drink to be dispensed. Action will be performed in corresspondance to the button pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Coin Input Box*

Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: CoinInputListener
Documentation: This constructor creates an instance of the Coin Input Listener.

Parameters:

Name: *coin_receiver*
Type: *Coin Receiver*

4.5.6 *Coin Receiver*

This control object manages the input and storage of Coins.

Attributes

Name: coins
Type: *Array*
Documentation: List of the Coins entered during the transaction.

Name: total_inserted
Type: *Integer*
Documentation: Total amont of money entered so far during current transaction.

Operations

Name: StartReceive
Documentation: Commence receiving a series of Coins. To do this the Coin Receiver instructs the Coin Input Box to become activated. It also updates the Total Money Inserted Display on the Customer Panel.

Name: ReceiveCoin
Documentation: When a Coin is received the following will occur:
1 The Coin Input Box will be instructed to become deactivated.
2 The weight of the Coin will be sent to the object Coin for it to determine the denomination and value.
3 The Total Money Inserted Display will be updated.
4 If an invalid coin is entered, the Invalid Coin Display will be instructed to display INVALID COIN.
5 The Transaction Controller will be informed to process the current transaction based on the money received.

Parameters:

Name: *weight*
Type: *Integer*

Name: ContinueReceive
Documentation: This operation will activate the Coin Input Box so that further coins can be received.

Name: StoreCash
Documentation: Instruct the Cash Store to update its totals and then re-set the Total Money Inserted Display to zero.
Return Class: *Boolean*

Name: StopReceive
Documentation: This operation will deactivate the Coin Input Box in order to stop receiving coins.

Name: RefundCash
Documentation: This operation handles the refunding of Coins entered so far to the Customer.
Return Class:

Name: CoinReceiver
Documentation: This constructor creates an instance of the object.

Parameters:

Name: *transaction_controller*
Type: *Transaction Controller*

Name: SetActive
Documentation: This operation activates or deactivates the Coin Input Box.

Parameters:

Name: *active*
Type: *Boolean*

4.5.7 *Customer Panel*

This panel simulates the vending machine's customer interface panel. It will enable the user (the Customer) to:

1 insert coins;
2 select brands;
3 terminate transaction.

It will also provide the following display functions:

1 display total money inserted;
2 indicate coin not valid;
3 indicate no change available;
4 display the value of the change to be collected;
5 display an icon representing the dispensed drink.

Relationships

Object Name: *TextField*
Kind of Relationship: *UsesRelationship*

Object Name: *Dialog*
Kind of Relationship: *InheritsRelationship*

Object Name: *Coin Input Box*
Kind of Relationship: *AggregateRelationship*

Object Name: *Drink Selection Box*
Kind of Relationship: *AggregateRelationship*

Object Name: *TextField*
Kind of Relationship: *AggregateRelationship*

Object Name: *LabelledDisplay*
Kind of Relationship: *AggregateRelationship*

Object Name: *TextField*
Kind of Relationship: *AggregateRelationship*

Object Name: *LabelledDisplay*

Kind of Relationship: *AggregateRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Object Name: *Warning Display*
Kind of Relationship: *AggregateRelationship*

Object Name: *Warning Display*
Kind of Relationship: *AggregateRelationship*

Object Name: *LabelledDisplay*
Kind of Relationship: *AggregateRelationship*

Superclasses

Dialog
Window
Container
Component

Operations

Name: CloseDown
Documentation: Remove the Customer Panel from the display.

Name: Display
Documentation: Display the Customer Panel. This will be acheived by displaying the frame of the panel and then sending messages to its constituent objects instructing them to display themselves.

Name: CustomerPanel
Documentation: This constructor creates an instance of the Customer Panel. It further creates Invalid Coin Display,No Change Available Display, Refund/Change Tray Display, Total Money Inserted Display, Coin Input Box, Drink Selection Box, Can Collection Box and Terminate Button.

Parameters:

Name: *controller*
Type: *Transaction Controller*

Name: SetActive
Documentation: This operation activates or deactivates the Customer Panel and its compenent objects.

Parameters:

Name: *active*
Type: *Boolean*

Name: *component*
Type: *char*

4.5.8 *Dispense Controller*

This control object is for handling the dispense drink use case.

Operations

Name: ResetCan
Documentation: This operation will be used to instruct the Can Collection Box to remove the

drinks can shape or drink brand name from being displayed.

Name: AllowSelection
Documentation: This operation will be used to activate/deactivate (as indicated through a parameter) the Drink Selection Box so that transactions can continue or will be disallowed.

Parameters:

Name: *allow*
Type: *Boolean*

Name: DispenseDrink
Documentation: This operation will be used to dispense a drink. It will :
1 Instruct the Drinks Store to dispense the drink. It will also instruct the Can Collection Box to display a can shape.
2 Instruct the Store Controller to update the Drinks Store Display on the Machinery Simulator Panel.
3 In case of fault detection, it will send a "fault detected" response to the Transaction Controller.
Return Class: *Boolean*

Parameters:

Name: *selected_brand*
Type: *Integer*

Name: UpdateDrinkSelection
Documentation: This operation is used to display the lattest stock and price information on the Drink Selection Box.

Parameters:

Name: *index*
Type: *Integer*

Name: DispenseController
Documentation: This constructor creates an instance of the object.

Parameters:

Name: *transaction_controller*
Type: *Transaction Controller*

Name: UpdateDrinkPanel
Documentation: This operation updates the whole Drink Selection Box with current names, stocks and prices.

4.5.9 *Drink Selection Box*

This interface object is part of the Customer Panel. It is used by the Customer to select a drink.

Relationships

Object Name: *Panel*
Kind of Relationship: *InheritsRelationship*

Relationship Name: *theDrink Selection Box*
Object Name: *Drink Selection Listener*
Kind of Relationship: *AssociationRole*

Object Name: *Drink Selection Item*
Kind of Relationship: *AggregateRelationship*

Superclasses

Panel
Container
Component

Attributes

Name: drink_identifiers
Type: *Array*
Documentation: Array of integers providing identifiers for each selection button.

Operations

Name: DrinkSelectionBox
Documentation: This constructor creates an instance of the object.

Parameters:

Name: *container*
Type: *Composite*

Name: *controller*
Type: *Transaction Controller*

Name: Update
Documentation: Updates the stock status, name and price of a drink brands based on the values received.

Parameters:

Name: *brand*
Type: *Integer*

Name: *quantity*
Type: *Integer*

Name: *price*
Type: *Integer*

Name: *name*
Type: *String*

Name: SetState
Documentation: This operation activates or deactivates the drink selection buttons on the Customer Panel.

Parameters:

Name: *brand_index*
Type: *Integer*

Name: *active*
Type: *Boolean*

Name: SetActive
Documentation: This operation will activate or deactivate the drink selection buttons.

Parameters:

Name: *active*
Type: *Boolean*

Name: SetItemState
Documentation: This operation activates or deactivates the drink selection buttons on the Customer Panel. This operation also displays OUT OF STOCK messages where

appropriate.

Parameters:

Name: *brand_index*
Type: *Integer*

Name: *active*
Type: *Boolean*

4.5.10 Drink Selection Item

This boundary object enables a drink to be displayed and selected. It also displays the stock availability of the drink.

Relationships

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Object Name: *TextField*
Kind of Relationship: *AggregateRelationship*

Object Name: *TextField*
Kind of Relationship: *AggregateRelationship*

Operations

Name: DrinkSelectionItem
Documentation: This constructor creates an instance of the object.

Name: AddListener
Documentation: This operation attaches a listener to the item.

Parameters:

Name: *listener*
Type: *ActionListener*

Name: SetItemState
Documentation: This operation activates or deactivates the drink selection item button. This operation also displays OUT OF STOCK messages where appropriate.

Parameters:

Name: *active*
Type: *Boolean*

Name: SetPrice
Documentation: This operation sets the price on the Drink Selection Item.

Parameters:

Name: *price*
Type: *Integer*

Name: SetName
Documentation: This operation sets the name on the Drink Selection Item.

Parameters:

Name: *name*
Type: *String*

4.5.11 Drink Selection Listener

This control object monitors the selection of drink (drink brand) by a customer in the Customer Panel. It performs an action in corresspondance to the drink selected.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Drink Selection Box*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: DrinkSelectionListener
Documentation: This constructor creates an instance of the object.

Parameters:

Name: *transaction_controller*
Type: *Transaction Controller*

4.5.12 Terminate Button Listener

This control object monitors the Terminate Button on the Customer Panel and informs the Transaction Controller when it is pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Button*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: TerminateButtonListener
Documentation: This constructor creates an instance of theTerminate Button Listener.

Parameters:

Name: *transaction_controller*
Type: *Transaction Controller*

4.5.13 Transaction Controller

This control object coordinates the customer transactions for selection of a drink brand, coin input, storage of coins and termination requests for ongoing transactions.

Attributes

Name: change_given
Type: *Boolean*
Documentation: Boolean set to TRUE when change is successfully issued during the transaction.

Name: drink_dispensed
Type: *Boolean*
Documentation: Boolean set to TRUE when the drink is successfully dispensed during the transaction.

Name: price
Type: *Integer*
Documentation: Price of the selected drink.

Name: selection
Type: *Integer*
Documentation: Identifier of the selected drink.

Operations

Name: StartTransaction
Documentation: This operation will start the customer transaction. It receives the identification for the selected drink brand (item) from the Customer Panel. The following actions are performed in the operation:
1 The price of the selected item is obtained.
2 The Refund/Change Tray Display is reset.
3 The Can Collection Box is reset.
4 The Drink Selection Box is deactivated to disallow the selection of further drinks when the transaction is in progress.
5 The Coin Receiver will be instructed to start receiving the coins.

Parameters:

Name: *identifier*
Type: *Integer*

Name: ProcessMoneyReceived
Documentation: This operation processes the money received by the Coin Receiver during the progress of a transaction. The following actions are performed during this operation:
1 The current total money inserted is obtained from the Coin Receiver.
2 If the received money is more than or equal to the price of the drink, operation CompleteTransaction of the Transaction Controller is triggered.
3 If the received money is less than the price of the drink, the Coin Receiver is instructed to continue receiving the coins.

Parameters:

Name: *total*
Type: *Integer*

Name: CompleteTransaction
Documentation: This operation is performed when the Transaction Controller is informed that coin entry is complete and the money received is sufficient to dispense the drink. The following actions are perfromed.
1 Dispense the drink.
2 Give change if necessary.
3 Store the Coins that have been entered into the Cash Store.
4 Reset the Drink Selection Box to allow further transactions.

Name: TerminateFault
Documentation: If the Transaction Controller is informed that a fault was discovered while dispensing a drink, giving change or storing Coins, it will use this operation to deactivate the Drink Selection Box and instruct the Coin Receiver to refund the

money inserted by the customer.

Name: TerminateTransaction
Documentation: If the Transaction Controller receives a request to terminate the current transaction then following will occur:
1 If there is no transaction in progress or coin input is completed then the Customer Panel will be instructed to deactivate the Drink Selection Box.
2 If coin input is not yet complete, the Coin Receiver will be instructed to stop coin input and refund the money entered so far.
3 The Drink Selection Box is then reset to allow further transactions.

Name: CancelTransaction
Documentation: This operation will cancel an ongoing customer transaction.

Name: CloseDown
Documentation: This operation will close down the transaction control function of the vending machine.

Name: TransactionController
Documentation: This constructor creates an instance of the Transaction Controller.

Parameters:

Name: *main_controller*
Type: *Main Controller*

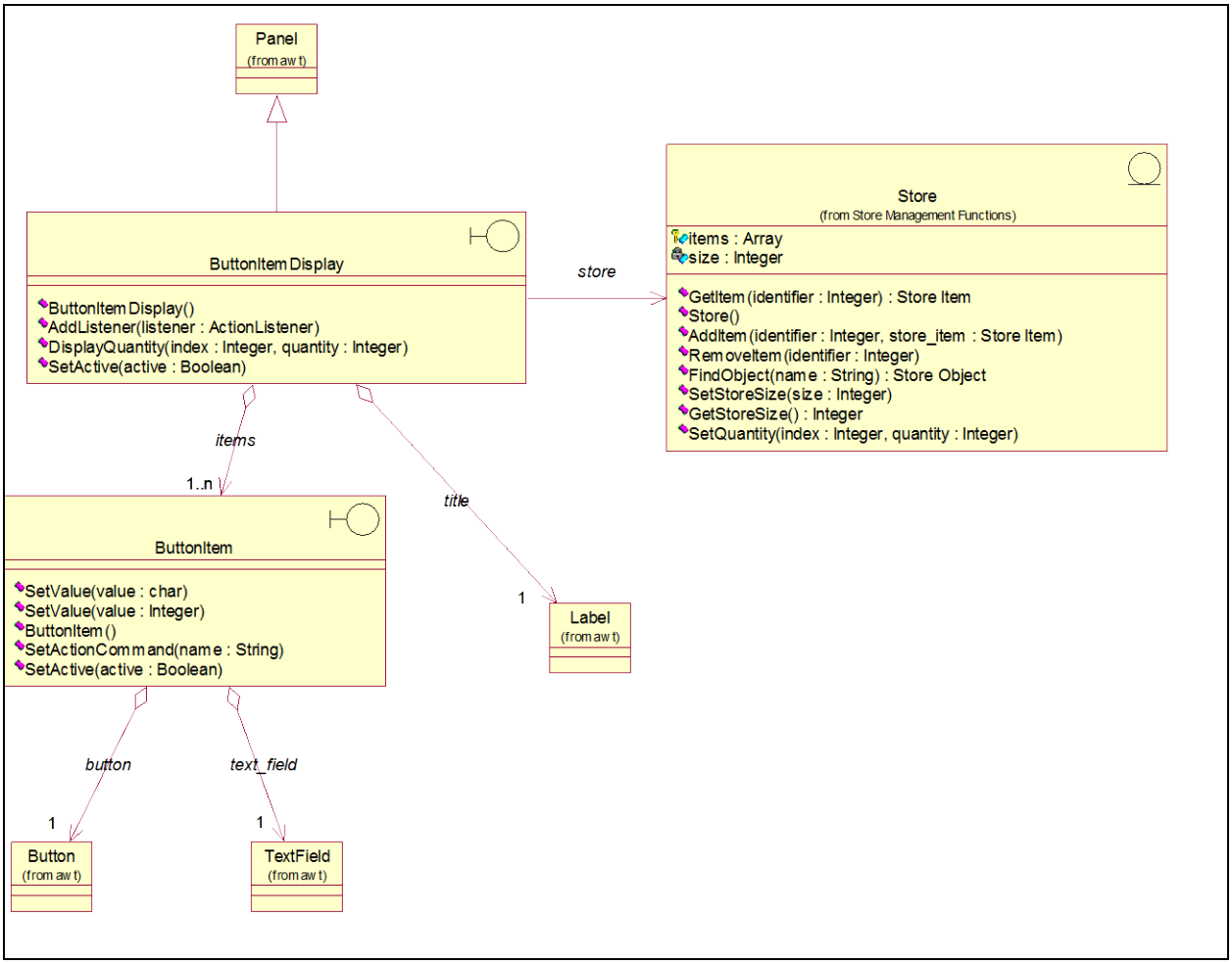
Name: DisplayCustomerPanel
Documentation: This operation displays and intialises the customer panel.

Name: RefreshCustomerPanel
Documentation: This operation refreshes the Customer Panel when the maintainer logs-out.

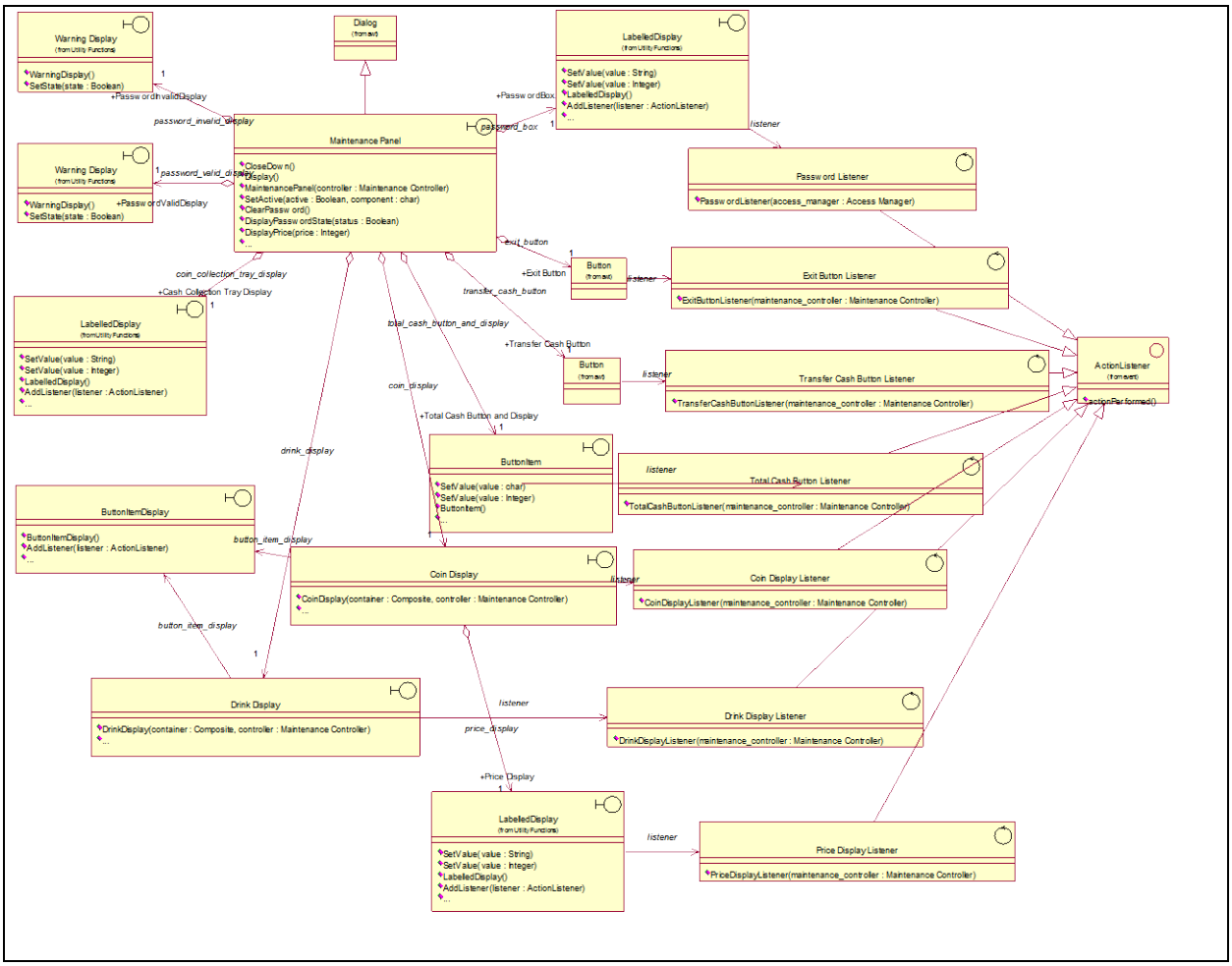
4.6 Maintenance Functions

The Maintainer Functions package contains the objects that facilitate the maintainer's interaction with the vending machine. Static views of the main objects are presented below followed by specifications of each object in the package.

4.6.1 Static Structure of ButtonItemDisplay



4.6.2 Static Structure of Maintenance Panel



4.6.3 Access Manager

This control object manages and controls entry to the system by the Maintainer.

Operations

Name: ProcessPassword
Documentation: Send a received password string to the Password object for validation. Then instruct the Password Box to become deactivated.

Parameters:

Name: *password*
Type: *String*

Name: AccessManager
Documentation: This constructor will create a new instance of the Access Manager object and one instance of the Password object.

Parameters:

Name: *main_controller*
Type: *Maintenance Controller*

4.6.4 *ButtonItem*

This boundary object displays one item in a ButtonItemDisplay.

Relationships

Object Name: *Total Cash Button Listener*
Kind of Relationship: *AssociationRole*

Object Name: *TextField*
Kind of Relationship: *AggregateRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Operations

Name: SetValue
Documentation: This operation sets the text to be displayed.

Parameters:

Name: *value*
Type: *char*

Name: SetValue
Documentation: This operation sets the integer value to be displayed.

Parameters:

Name: *value*
Type: *Integer*

Name: ButtonItem
Documentation: This constructor creates an instance of the object.

Name: SetActionCommand
Documentation: This operation attaches the specific item (Drink or Coin) to the button.

Parameters:

Name: *name*
Type: *String*

Name: SetActive
Documentation: This operation activates the ButtonItem if the parameter is TRUE. Otherwise, the ButtonItem is deactivated.

Parameters:

Name: *active*
Type: *Boolean*

4.6.5 *ButtonItemDisplay*

This boundary object is to display stock information when a button is pressed.

Relationships

Object Name: *Panel*
Kind of Relationship: *InheritsRelationship*

Object Name: *ButtonItem*
Kind of Relationship: *AggregateRelationship*

Relationship Name: *theButtonItemDisplay*

Object Name: *Label*
Kind of Relationship: *AggregateRelationship*

Object Name: *Drink Display*
Kind of Relationship: *AssociationRole*

Object Name: *Coin Display*
Kind of Relationship: *AssociationRole*

Object Name: *Store*
Kind of Relationship: *AssociationRole*

Superclasses

Panel
Container
Component

Operations

Name: ButtonItemDisplay
Documentation: This constructor creates an instance of the object.

Name: AddListener
Documentation: This operation attaches a listener to the ButtonItemDisplay.

Parameters:

Name: *listener*
Type: *ActionListener*

Name: DisplayQuantity
Documentation: This operation displays a quantity on to a specific ButtonItem.

Parameters:

Name: *index*
Type: *Integer*

Name: *quantity*
Type: *Integer*

Name: SetActive
Documentation: This operation activates the ButtonItemDisplay if the parameter is TRUE. Otherwise, the ButtonItemDisplay is deactivated.

Parameters:

Name: *active*
Type: *Boolean*

4.6.6 *Coin Display*

This interface object is part of the Maintenance Panel. It is used by the Maintainer to display the total number of Coins of each denomination that are currently held in the Cash Store. The object contains buttons to represent (and select) the coin denominations used in the vending machine.

Relationships

Object Name: *Coin Display Listener*
Kind of Relationship: *AssociationRole*

Object Name: *LabelledDisplay*
Kind of Relationship: *AggregateRelationship*

Object Name: *ButtonItemDisplay*
Kind of Relationship: *AssociationRole*

Operations

Name: CoinDisplay
Documentation: This constructor creates an instance of the object.

Parameters:

Name: *container*
Type: *Composite*

Name: *controller*
Type: *Maintenance Controller*

Name: SetActive
Documentation: This operation activates the Coin Display if the parameter is TRUE. Otherwise, the Coin Display is deactivated.

Parameters:

Name: *active*
Type: *Boolean*

Name: DisplayQuantity
Documentation: Displays the total number of coins of a denomination received from a requestor.

Parameters:

Name: *identifier*
Type: *Integer*

Name: *total*
Type: *Integer*

4.6.7 *Coin Display Listener*

This control object monitors the events in the Coin Display and performs actions in response to the buttons being pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Coin Display*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: CoinDisplayListener
Documentation: This constructor creates an instance of the Coin Display Listener.

Parameters:

Name: *maintenance_controller*
Type: *Maintenance Controller*

4.6.8 *Drink Display*

This interface object is part of the Maintenance Panel. It is used by the Maintainer to display the total number of cans of each Drinks Brand that are currently held in the Drinks Store. This object contains buttons which represent (and to select) the drinks brands served by the vending machine.

Relationships

Object Name: *Drink Display Listener*
Kind of Relationship: *AssociationRole*

Object Name: *ButtonItemDisplay*
Kind of Relationship: *AssociationRole*

Operations

Name: DrinkDisplay
Documentation: This constructor creates an instance of the object.

Parameters:

Name: *container*
Type: *Composite*

Name: *controller*
Type: *Maintenance Controller*

Name: DisplayQuantity
Documentation: This operation displays the stock value received for the currently selected brand. The display will be done on a text field associated with the Drink Display object.

Parameters:

Name: *identifier*
Type: *Integer*

Name: *stock*
Type: *Integer*

Name: SetActive
Documentation: This operation activates the Drink Display if the parameter is TRUE. Otherwise, the Drink Display is deactivated.

Parameters:

Name: *active*
Type: *Boolean*

4.6.9 *Drink Display Listener*

This control object monitors the events in the Drink Display and performs actions in response to the buttons being pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Drink Display*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: DrinkDisplayListener
Documentation: This constructor creates an instance of the Drink Display Listener.

Parameters:

Name: *maintenance_controller*
Type: *Maintenance Controller*

4.6.10 *Exit Button Listener*

This control object monitors the Exit Button and performs actions in response to the button being pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Button*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: ExitButtonListener
Documentation: This constructor creates an instance of the Exit Button Listener.

Parameters:

Name: *maintenance_controller*
Type: *Maintenance Controller*

4.6.11 *Maintenance Controller*

This control object handles the system maintenance use case.

Operations

Name: LoginMaintainer
Documentation: When the Maintenance Controller receives a message saying that the Maintainer has successfully logged-in, the following will occur:
1 Messages will be sent to activate the following elements of the Maintenance Panel: Coin Display, Drink Display, Total Cash Display, Exit Button, Transfer Cash Button and Total Cash Button.
2 The Door will be instructed to set its status as unlocked, and then inform the Machinery Simulator Panel to update the Door Status Display.
3 The Simulator Control Panel will be instructed to deactivate the Activate Customer Panel Button.
4 The Transaction Controller will be instructed to terminate and suspend Customer transactions. Operation TerminateTransaction of Transaction Controller will be used to accomplish this.

Parameters:

Name: *passwordCorrect*
Type: *Boolean*

Name: SetPrice

Documentation: This operation will be used to set the price of a drink brand as determined by the Maintainer.

Parameters:

Name: *price*
Type: *Integer*

Name: DisplayCoin
Documentation: This operation will be used to get the total number of coins of a selected denomination.

Parameters:

Name: *coin*
Type: *Integer*

Name: GetTotalCash
Documentation: This operation sends the total cash held in the cash store to the Maintenance Panel.

Name: MaintainerLogout
Documentation: When the Maintenance Controller receives a message saying that the Maintainer has correctly logged-out, the following will occur:
1 Check the door status of Door to determine whether the vending machine door is locked. If the door is unlocked, then the exit request is ignored.
2 Re-set the Maintenance Panel (initial values set, buttons activated/deactivated) .
3 Update the Customer Panel and permit Customer transactions to re-start.

Name: MaintenanceController
Documentation: This constructor creates an instance of the Maintenance Controller.

Parameters:

Name: *main_controller*
Type: *Main Controller*

Name: DisplayDrinks
Documentation: This operation will get the drink stock value and prices (for a specific brand) for display.

Parameters:

Name: *brand*
Type: *Integer*

Name: CloseDown
Documentation: This operation will close down the maintenance functions of the vending machine.

Name: DisplayMaintenancePanel
Documentation: This operation displays and initialises the Maintenance Panel.

Name: TransferAll
Documentation: This operation is to facilitate the transfer of all cash in Cash Store to the maintainer.

4.6.12 *Maintenance Panel*

This panel simulates the vending machines maintainer control panel. It will enable the user (the Maintainer) to:
1 log-on using a password;

2 display number of Coins of each denomination in the Cash Store;
3 display total value of cash in the system;
4 display number cans of each Drinks Brand held in the Drinks Store;
5 change the price of any Drinks Brand;
6 collect all cash held in system;
7 formally exit from the panel.

Relationships

Object Name: *Dialog*
Kind of Relationship: *InheritsRelationship*

Object Name: *LabelledDisplay*
Kind of Relationship: *AggregateRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Object Name: *LabelledDisplay*
Kind of Relationship: *AggregateRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Object Name: *Warning Display*
Kind of Relationship: *AggregateRelationship*

Object Name: *Button*
Kind of Relationship: *AggregateRelationship*

Object Name: *Warning Display*
Kind of Relationship: *AggregateRelationship*

Object Name: *Coin Display*
Kind of Relationship: *AggregateRelationship*

Object Name: *ButtonItem*
Kind of Relationship: *AggregateRelationship*

Object Name: *Drink Display*
Kind of Relationship: *AggregateRelationship*

Superclasses

Dialog
Window
Container
Component

Operations

Name: CloseDown
Documentation: Remove the panel objects and the maintenance panel from the display.

Name: Display
Documentation: Display the Maintenance Panel. This will be acheived by displaying the frame of the panel and then sending messages to its constituent objects instructing them to:
1 display themselves;
2 set initial values; and
3 deactivate all constituent objects except Password Box and Cash Collection

Tray Display.

Name: MaintenancePanel
Documentation: This constructor creates an instance of the Maintenance Panel. It further creates Drink Display, Price Display, Cash Collection Tray Display, Total Cash Display, Password Box, PasswordValidDisplay, PasswordInvalidDisplay,Coin Display, Exit Button, Transfer Cash Button, Total Cash Button.

Parameters:

Name: *controller*
Type: *Maintenance Controller*

Name: SetActive
Documentation: This operation activates or deactivates the Maintenance Panel and its compenent objects.

Parameters:

Name: *active*
Type: *Boolean*

Name: *component*
Type: *char*

Name: ClearPassword
Documentation: This operation removes the entered password from the display.

Name: DisplayPasswordState
Documentation: This operation displays a message indicating the VALID or INVALID password status.

Parameters:

Name: *status*
Type: *Boolean*

Name: DisplayPrice
Documentation: This operation displays the price of the selected drinks brand.

Parameters:

Name: *price*
Type: *Integer*

Name: DisplayTotalCash
Documentation: This operation displays the received value as the total cash held in the Cash Store of the vending machine.

Parameters:

Name: *total*
Type: *Integer*

Name: DisplayCoins
Documentation: This operation displays the amount of money to be issued on the Cash Collection Tray Display.

Parameters:

Name: *cash*
Type: *Integer*

4.6.13 Password

This entity object stores the correct password.

Attributes

Name: password
Type: *String*
Initial Value: *<actual password text>*
Documentation: Contains a text string holding the "actual" password that users need to enter before they are permitted to log-in to the Maintenance Panel.

Operations

Name: ValidatePassword
Documentation: Compare the received Password with the authorised password string stored in the object, and then return a message to the Access Manager saying whether the password is valid or invalid.
Return Class: *Boolean*

Parameters:

Name: *password*
Type: *String*

Name: Password
Documentation: This constructor creates an instance of the Password object.

4.6.14 Password Listener

This control object monitors the events in the Password Box and performs actions in response to data being entered into the text field.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *LabelledDisplay*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: PasswordListener
Documentation: This constructor creates an instance of the Password Listener.

Parameters:

Name: *access_manager*
Type: *Access Manager*

4.6.15 Price Display Listener

This control object monitors the events in the Price Display and performs actions in response to data being entered into the text field.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *LabelledDisplay*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: PriceDisplayListener
Documentation: This constructor creates an instance of the Price Display Listener .

Parameters:

Name: *maintenance_controller*
Type: *Maintenance Controller*

4.6.16 Total Cash Button Listener

This control object monitors the Total Cash Button on the Maintenance Panel and informs the Maintenance Controller when it is pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *ButtonItem*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: TotalCashButtonListener
Documentation: This constructor creates an instance of the Total Cash Button Listener.

Parameters:

Name: *maintenance_controller*
Type: *Maintenance Controller*

4.6.17 Transfer Cash Button Listener

This control object monitors the Transfer Cash Button and performs the required action in response to the button being pressed.

Relationships

Object Name: *ActionListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Button*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: TransferCashButtonListener
Documentation: This constructor creates an instance of theTransfer Cash Button Listener .

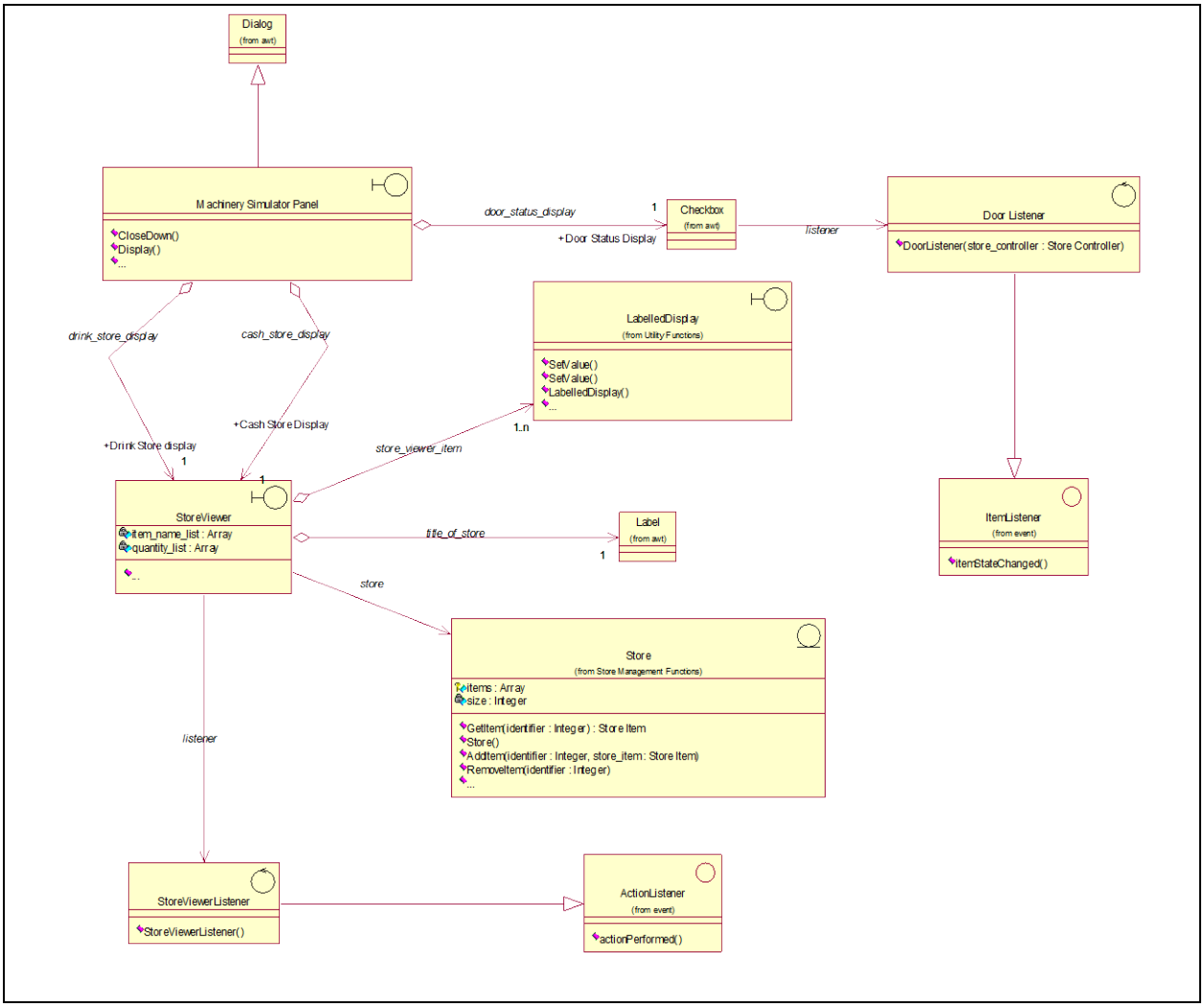
Parameters:

Name: *maintenance_controller*
Type: *Maintenance Controller*

4.7 Machinery Functions

The Machinery Functions package contains the objects that facilitate the controller's interaction with the vending machine. Static views of the main objects are presented below followed by specifications of each object in the package.

4.7.1 Static Structure of Machinery Simulator Panel



4.7.2 Door

This object represents the door of the vending machine. It is opened so that the machine can be restocked with cans and cash.

Attributes

Name: is_closed
Type: *Boolean*
Initial Value: *TRUE*
Documentation: Boolean set to TRUE if the door is locked and FALSE otherwise.

Operations

Name: SetState

Documentation: Update the door status attribute with the data (open/locked) received.

Parameters:

Name: *state*
Type: *Boolean*

Name: GetState
Documentation: Send/return the status of the door (unlocked/locked) to the requestor.

Name: Door
Documentation: This constructor creates an instance of the Door.

Name: IsClosed
Documentation: Returns TRUE to the requestor if the door is locked, and FALSE otherwise.
Return Class: *Boolean*

4.7.3 *Door Listener*

This control object monitors the door status request (close door) when the Controller uses the Machinery Simulator Panel. It performs an action in response to the request.

Relationships

Object Name: *ItemListener*
Kind of Relationship: *InheritsRelationship*

Object Name: *Checkbox*
Kind of Relationship: *AssociationRole*

Superclasses

ItemListener
EventListener

Operations

Name: DoorListener
Documentation: This constructor creates an instance of the object.

Parameters:

Name: *store_controller*
Type: *Store Controller*

4.7.4 *Machinery Controller*

This object controls the Change State use case.

Operations

Name: Initialize
Documentation: This operation creates the Door.

Name: CloseDown
Documentation: This operation will close down the machinery functions of the vending machine.

Name: DisplayMachineryPanel
Documentation: This operation displays and initialises the Machinery Simulator Panel.

Name: IsDoorClosed
Documentation: This operation determines whether the Door is closed - a Boolean is set to TRUE if the door is locked and FALSE otherwise.
Return Class: *Boolean*

Name: SetDoorStatus
Documentation: This operation will be used to instruct the Door object to change the staus of the vending machine door to unlocked (state = FALSE) or locked (state = TRUE).

Parameters:

Name: *state*
Type: *Boolean*

Name: DisplayDoorStatus
Documentation: This operation displays the current Door status (open or closed) on the Door Status Display.

Name: DisplayDrinkStock
Documentation: This operation will get the stock values of drinks brands from the Drinks Store and display them on the Machinery Simulator Panel.

Name: DisplayCoinStock
Documentation: This operation will get the stock values of coin denominations from the Cash Store and display them on the Machinery Simulator Panel.

Name: StoreCoin
Documentation: This operation will instuct the Cash Store to store the Coin sent as input, and the update the display on the Machinery Simulator Panel.
Return Class: *Boolean*

Parameters:

Name: *current_coin*
Type: *Coin*

Name: DispenseDrink
Documentation: This operation instructs the the Drinks Store to dispense one drink, and then updates the Machinery Simulator Panel. It returns TRUE or FALSE to indicate whether dispensing was successful.
Return Class: *Boolean*

Parameters:

Name: *selected_brand*
Type: *Integer*

Name: GiveChange
Documentation: This operation instructs the Cash Store to issue a number of coins of a specific denomination, and then updates the Machinery Simulator Panel. It returns TRUE or FALSE to indicate whether the change issue was successful.
Return Class: *Boolean*

Parameters:

Name: *index*
Type: *Integer*

Name: *number_coins*
Type: *Integer*

Name: MachineryController
Documentation: Creates an instance of the Machinery Controller.

Parameters:

Name: *main_controller*
Type: *Main Controller*

4.7.5 *Machinery Simulator Panel*

This panel simulates the physical actions of the maintainer. It enables the user to:

1 display and enter new values for the number of cans of each Drinks Brand held by the Drinks Store;
2 display and enter new values for the number of Coins of each denomination held by the Cash Store;
3 display whether the vending machine Door is unlocked, and enable it to be locked.

Relationships

Object Name: *Dialog*
Kind of Relationship: *InheritsRelationship*

Object Name: *Checkbox*
Kind of Relationship: *AggregateRelationship*

Object Name: *StoreViewer*
Kind of Relationship: *AggregateRelationship*

Object Name: *StoreViewer*
Kind of Relationship: *AggregateRelationship*

Object Name: *StoreViewer*
Kind of Relationship: *AggregateRelationship*

Superclasses

Dialog
Window
Container
Component

Operations

Name: CloseDown
Documentation: Remove the panel objects and Machinery Simulator Panel from the display.
Return Class:

Name: Display
Documentation: Display the Machinery Simulator Panel. This will be acheived by displaying the frame of the panel and then sending messages to its constituent objects instructing them to:
1 display themselves;
2 set initial values; and
3 deactivate themselves.

Name: MachinerySimulatorPanel
Documentation: This constructor creates an instance of the Machinery Simulator Panel. It further creates Drinks Store Display, Cash Store Display and Door Status Display. It receives and keeps reference for Store Controller.

Parameters:

Name: *controller*
Type: *Store Controller*

Name: SetActive
Documentation: This operation activates or deactives the Machinery Simulator Panel and its component objects.

Parameters:

Name: *active*

Type: *Boolean*

Name: *component*
Type: *char*

Name: SetDoorStatus
Documentation: This operation sets the door state to "open" or "closed".

Parameters:

Name: *state*
Type: *Boolean*

4.7.6 *StoreViewer*

This boundary object displays the contents of a store (Drinks Store or Cash Store) and allows them to be changed.

Relationships

Object Name: *StoreViewerListener*
Kind of Relationship: *AssociationRole*

Object Name: *Label*
Kind of Relationship: *AggregateRelationship*

Object Name: *LabelledDisplay*
Kind of Relationship: *AggregateRelationship*

Object Name: *Store*
Kind of Relationship: *AssociationRole*

Attributes

Name: item_name_list
Type: *Array*

Name: quantity_list
Type: *Array*

Operations

Name: StoreViewer
Documentation: This constructor will create a new instance of the StoreViewer.
Return Class:

Name: SetActive
Documentation: This operation activates the Store Display if the parameter is TRUE. Otherwise, the Store Display is deactivated.

Parameters:

Name: *active*
Type: *Boolean*

Name: Update
Documentation: Update the display fields with the data provided.

4.7.7 *StoreViewerListener*

This control object monitors data entered into a StoreViewer, when the Controller uses the Machinery Simulator Panel. When data is entered, it intiates the process to store the data.

Relationships

Object Name: *ActionListener*

Kind of Relationship: *InheritsRelationship*

Object Name: *StoreViewer*
Kind of Relationship: *AssociationRole*

Superclasses

ActionListener
EventListener

Operations

Name: StoreViewerListener
Documentation: This constructor creates a new instance of the Store Viewer Listener.

5. Deployment

The system will be deployed as a stand-alone application running on a PC workstation.