Store Controller Service Design Document

Date: 10/24/2021
Author: Burak Ufuktepe
Reviewer: Matthew Quesada

Contents

Introduction

This document defines the design for the Store Controller Service which is responsible for monitoring the state of the sensors and appliances and also controlling the appliances based on status updates from the sensors and appliances within the store.

Overview

The Store Controller Service provides the overall management for the stores and acts as a medium for communication between the Store Model Service, the Ledger Service, and the Authentication Service. The UML component diagram in Figure 1 shows how the Store Controller Service fits into the Store 24x7 System.

The Store Controller Service monitors the sensors, appliances, and customers. Additionally, it manages the appliances in response to status updates from sensors and appliances. Furthermore, the Store Controller Service utilizes the Authentication Service to perform transactions.

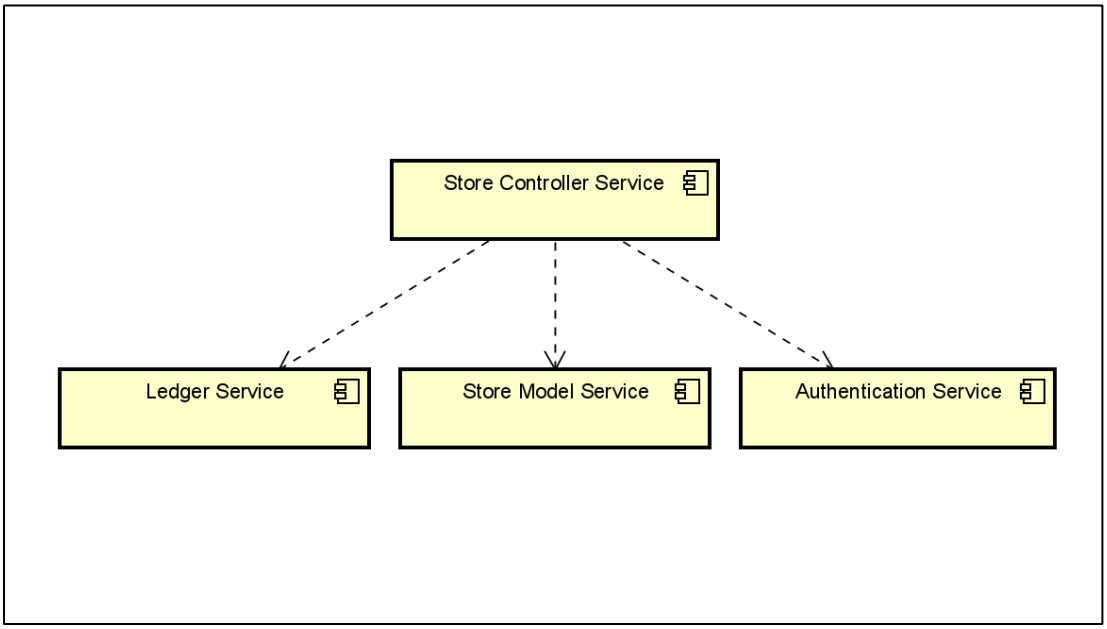

Figure 1: UML Component Diagram for the Store 24x7 System

Requirements

This section provides a summary of the requirements for the Store Controller Service. The following functions should be supported by the Store Controller Service:

- Listen for status updates from sensors and appliances.
- Listen for voice commands received via microphones.
- Generate actions in response to status updates from sensors and appliances based on a set of rules.
- Send control messages to appliances in response to events.
- Check account balances and submit transactions for checkout utilizing the Ledger Service.
- Use the Store Model Service API to monitor the status of the IoT devices.

Sensors

- Cameras alert the Store Controller Service in case of emergencies such as fire, flood, earthquake, or an armed intruder.
- Cameras detect:
  - When a product is added/removed to/from a basket or dropped on the floor.
  - When a customer enters an aisle.
- Microphones detect the sound of breaking glass.
- Microphones listen for customer questions/requests.

Appliances

- Robots assist customers to their cars or to leave the store in case of an emergency.
- Robots can clean the floor, fetch products for customers, and restock shelves.
- Turnstiles can weigh customer baskets and identify approaching customers.
- Speakers can respond to customer questions.

Customers

- Can ask questions or make requests through microphones.
- Can add/remove items to/from their basket.

Use Cases

The following Use Case diagram describes the use cases supported by the Store Controller System.

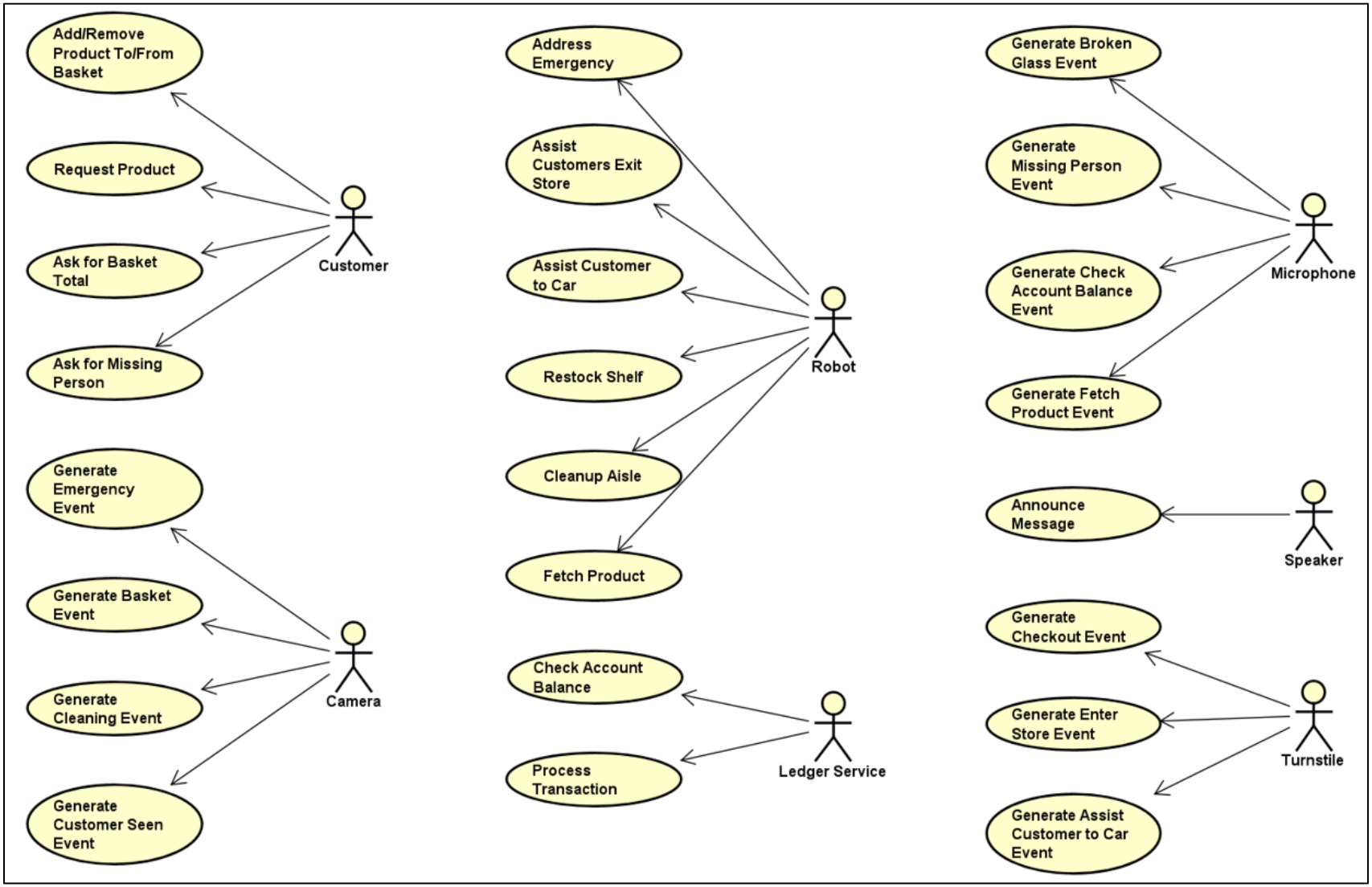

Figure 2: Store Controller Service UML Use Case Diagram

Actors

The actors of the Store Controller System include Customers, Cameras, Robots, Microphones, Speakers, Turnstiles, and the Ledger Service.

Customers

Customers are the shoppers that enter the store. They browse the aisles and add items to their baskets. Once they are done shopping, they leave the store through one of the smart turnstiles where checkout happens automatically.

Robots

Robots restock shelves and clean the store. They also assist customers to their cars, fetch products for customers, and address emergencies.

Cameras

Cameras, which are located throughout the store, are AI-powered devices. They can detect customers, emergencies, and actions such as adding a product to a basket.

Microphones

Microphones listen for customer questions/requests and generate events to address these questions/events. They are also able to detect the sound of breaking glass.

Turnstiles

Turnstiles are devices that let customers in the store and generate checkout events. They announce welcome and goodbye messages when customers enter and leave the store.

Ledger Service

The ledger service processes transactions, maintains account balances, and manages the integrity of the blockchain.

Use Cases

Add/Remove Product To/From Basket

When a customer adds a product to his/her basket, a camera generates a Basket Event. As a result, the product is removed from the shelf and added to the customer's basket and a robot restocks the shelf if the inventory level falls below a certain threshold. Conversely, when a product is removed from the customer's basket, it is added back to the shelf.

Request Product

A customer can request a product using once of the microphones throughout the store. This initiates a Fetch Product Event and a robot finds the product and brings it to the customer.

Ask for Basket Total

A customer can ask for the total value of basket items using one of the microphones in the store. This will generate a Check Account Balance Event. The value of the items in the basket will be calculated and the account balance will be checked and a speaker will inform the customer about the basket total and his/her account balance.

Ask for Missing Person

A customer can ask for the location of another customer using one of the microphones in the store. This will generate a Missing Person Event. The customer will be located by the Store Model Service and a speaker will inform the customer who asked the question.

Address Emergency

Robots address emergencies such as fire, flood, earthquake, or armed intruder. Emergencies are detected by cameras that generate Emergency Events.

Assist Customers Exit Store

Robots assist customers to leave the store in case of emergencies such as fire, flood, earthquake, or armed intruder. Emergencies are detected by cameras that generate Emergency Events

Assist Customer to Car

Robots assist customers to their cars if the total weight of products in their baskets exceeds 10 lbs. Assist Customer to Car Events are generated by turnstiles.

Restock Shelf

After a customer adds a product to his/her basket from a shelf, a camera generates a Basket Event. As a result, a robot finds the same product in the STORE_ROOM, moves the product from the STORE_ROOM to the FLOOR and restocks the shelf.

Cleanup Aisle

Robots can clean up aisles in response to Cleaning Events or Broken Glass Events. Cleaning Events are generated by cameras when they detect products on the floor whereas Broken Glass Events are generated by microphones when they detect a sound of breaking glass.

Fetch Product

Robots fetch products for customers when customers make a request using gone of the microphones throughout the store. Microphones generate a Fetch Product Event in response to this request.

Generate Emergency Event

Cameras generate Emergency Events when they detect emergencies such as fire, flood, earthquake, or armed intruder.

Generate Basket Event

Cameras generate Basket Events when a customer adds/removes a product to/from his/her basket.

Generate Cleaning Event

Cameras generate Cleaning Events when they detect a product on the floor.

Generate Customer Seen Event

Cameras generate Customer Seen Events when they detect a customer entering an aisle.

Generate Broken Glass Event

Microphones generate Broken Glass Events when they detect the sound of breaking glass.

Generate Missing Person Event

Microphones generate Missing Person Events when a customer asks for the location of a customer.

Generate Check Account Balance Event

Microphones generate Check Account Balance Events when customers ask for the total value of basket items.

Generate Fetch Product Event

Microphones generate Fetch Product Events when customers make a fetch request to a microphone.

Announce Message

Speakers announce messages when customers enter and leave the store and when there is an emergency. Speakers also respond to customer questions and requests.

Generate Checkout Event

Turnstiles generate Checkout Events when customers approach turnstiles.

Generate Enter Store Event

Turnstiles generate Enter Store Events when they detect customers waiting to enter the store.

Generate Assist Customer to Car Event

Turnstiles generate Assist Customer to Car Events when they detect that the total weight of products in the basket exceeds 10 lbs.

Check Account Balance

The Ledger Service can retrieve a customer's account balance in response to a Check Account Balance Event or an Enter Store Event.

Process Transaction

The Ledger Service processes transactions in response to Checkout Events by creating transactions and submitting them to the blockchain.

Design

The **Observer Pattern** is used to allow the Store Controller Service to listen for events emitted by the devices of the Store Model Service. The Store Model Service implements the Subject interface which defines registerObserver, removeObserver, and notifyObservers methods. On the other hand, the Store Controller Service implements the Observer interface and registers itself as an observer to the Store Model Service. When an event occurs, the Store Model Service notifies its observers by calling their update methods. The Observer Pattern is shown in Figure 4.

Showing the **Command Pattern** in Figure 4 is not feasible, hence it is explained here and an example for updating the location of a customer is shown in Figure 3. The Command Factory is responsible for creating Command objects from Events for the Store Controller Service (Invoker). The Event object holds a reference to a Store Model Service (Receiver) object. The Command objects access the Store Model Service API using the Event objects which are passed as arguments to the execute methods of the Command objects. The Store Controller Service (Invoker) calls the execute methods of the Command objects. Please note that in Figure 3 only the classes that are necessary for updating the location of a customer is shown.

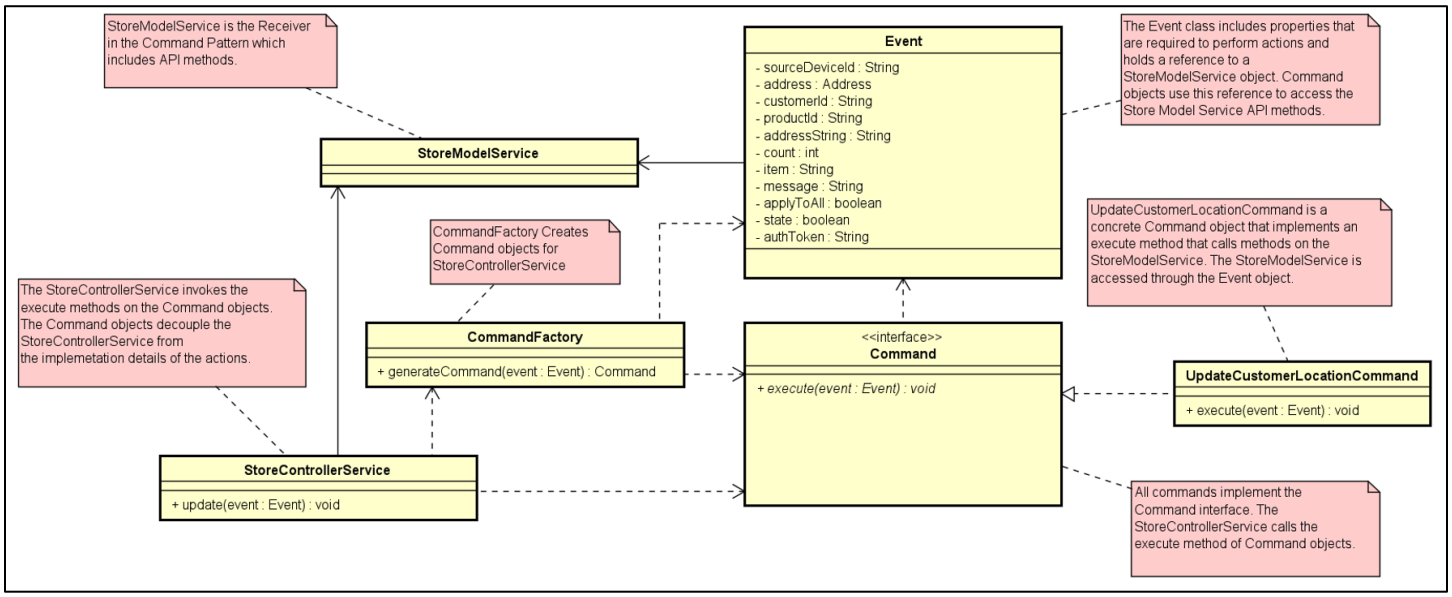

Figure 3: Command Pattern for Updating the Location of a Customer

Class Diagram

The following class diagram defines the classes defined in this design. All command objects implement the Command interface and have an execute method as shown in the Class Diagram. However, in order to simplify the Class Diagram, most of the StoreModelService and LedgerService classes are not shown in this class diagram as those were already documented in the design documents of the Store Model Service and the Ledger Service.

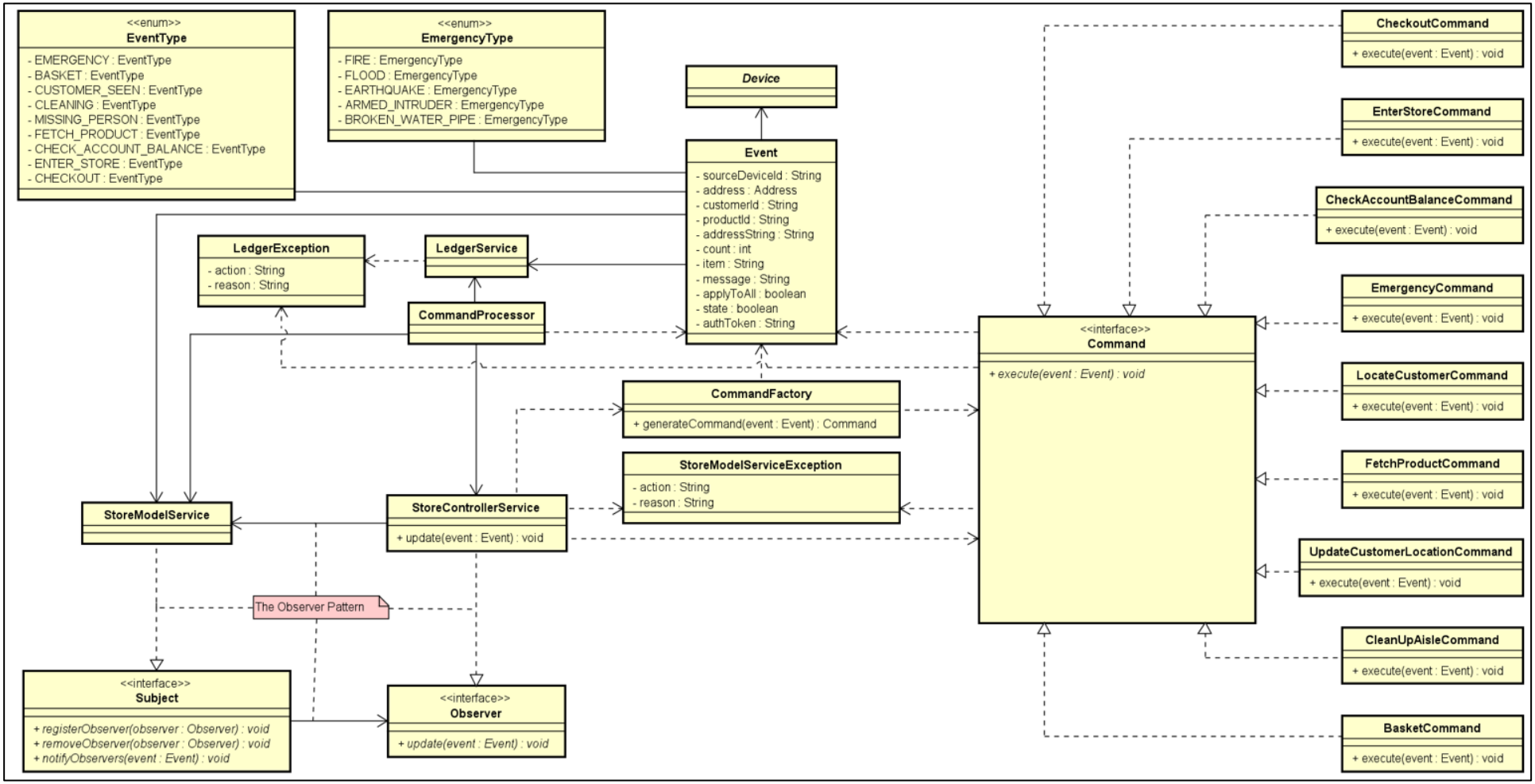

Figure 4: Store Controller Service Class Diagram

Sequence Diagram

Figure 5 shows the sequence diagram for checking an account balance. Even though in reality the Microphone would generate the check account balance event, for the purposes of this assignment, the events are by the CommandProcessor class based on the given script. The check account balance event is passed tot the Store Controller Service which utilizes a Command Factory to generate the CheckAccountBalanceCommand. And the CheckAccountBalanceCommand utilizes the Store Model Service and the Ledger Service to perform the required actions.

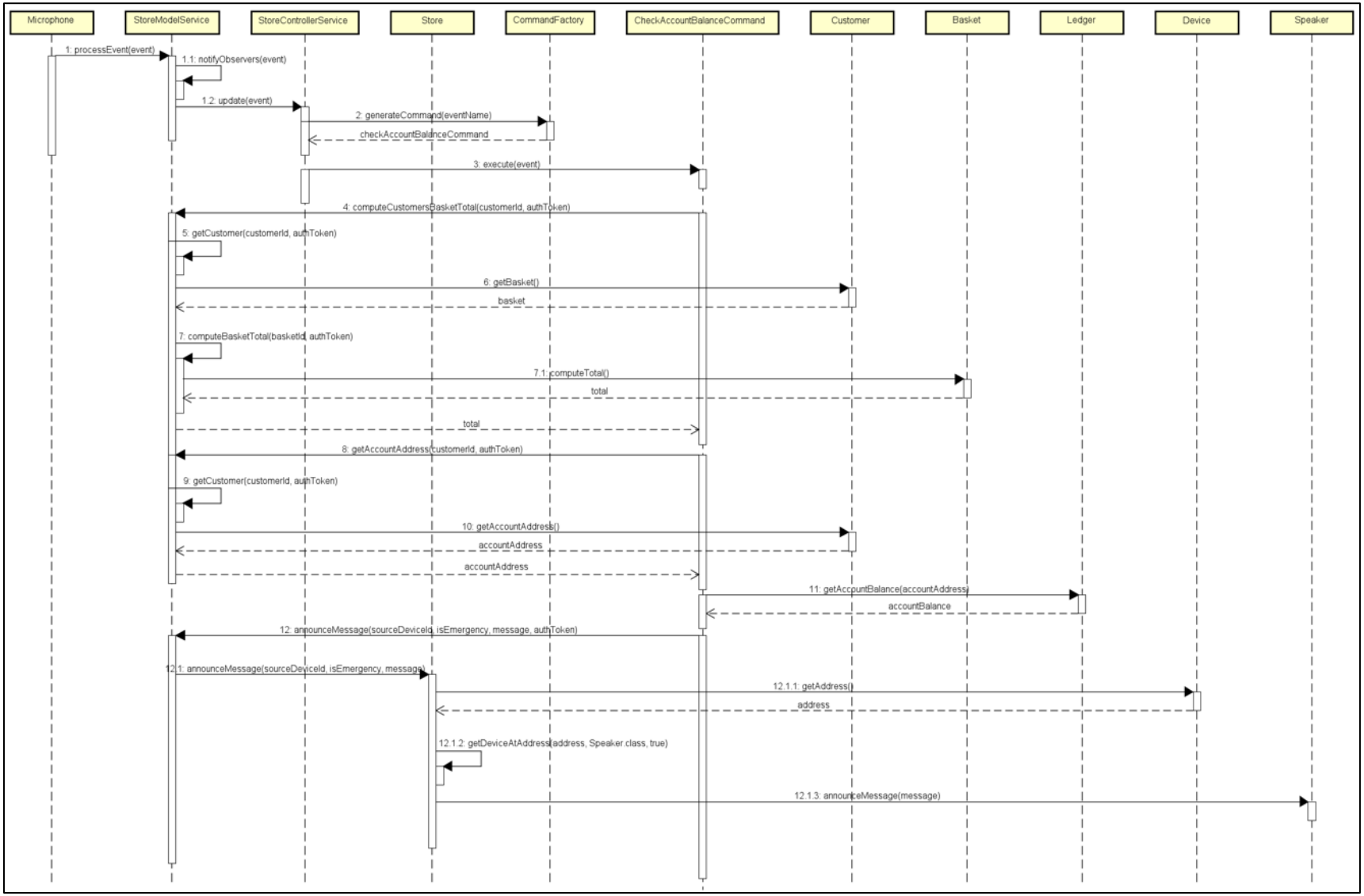

Figure 5: Sequence Diagram for Adding a Product to a Basket

Class Dictionary

This section specifies the class dictionary for the Store Controller Service which is defined within the package cscie97.controller.

StoreControllerService

The Store Controller Service monitors store events utilizing the Observer Pattern. It implements the Observer interface and is registered as an observer in the Store Model Service. Actions that are generated in response to status updates from the Store Model Service are implemented using the Command Pattern. To decouple the Store Controller Service from the implementation details of creating Commands, a Command Factory is used. The Command Factory has a static method that generates commands based on the given event name.

Methods

Method Name	Signature	Description
update	(event:Event):void	Call the generateCommand static method of the CommandFactory class to generate a Command instance and invoke the execute method of the Command instance. Throw a StoreModelServiceException in response to error conditions in execute methods of Command objects.

Associations

Association Name	Type	Description
storeModelService	StoreModelService	StoreModelService instance that can be used to un-register the Store Controller Service.

CommandFactory

The CommandFactory generates concrete Command objects based on the event type. The Store Controller Service is a client of the CommandFactory. The generateCommand method of the CommandFactory includes implementation details for creating Commands from events. The command objects are identified using the eventCommandMap which maps event names to command class names. If additional events and commands need to be considered in the future, only updating the eventCommandMap will be sufficient, the generateCommand method does not need to be updated.

Methods

Method Name	Signature	Description
generateCommand	(eventName:String):Command	Generate a Command instance for the given event name. Utilize the eventCommandMap to identify the command object. Throw a StoreModelServiceException if the given event name is invalid or if the corresponding command class does not exist.

Properties

Property Name	Type	Description
eventCommandMap	Map<String, String>	Maps event names to command class names. Used for identifying the command object for a given event name.

BasketCommand

Provides implementation details for adding a product to a basket or removing a product from a basket. This action is performed in response to a Basket Event. The execute method is invoked when a basket event occurs where products are added to or removed from a customer's basket. The inventory is also updated, the new inventory level is computed and a restocking action is initiated if the inventory level drops below a certain threshold (i.e. 50%).

Methods

Method Name	Signature	Description
execute	(event:Event):void	Call the Store Model Service API to add a number of products to a basket or remove from a basket for the given basket, product and quantity that are stored in the given event object. Throw a StoreModelServiceException if an error occurs in StoreModelService methods.

UpdateCustomerLocationCommand

According to the requirements, cameras generate Customer Seen Events which trigger actions to update the location of customers. UpdateCustomerLocationCommand provides implementation details for updating a customer's location in response to a Customer Seen Event. The execute method is invoked when a customer enters an aisle.

Methods

Method Name	Signature	Description
execute	(event:Event):void	Create an address for the customer's new location and then call the Store Model Service API to update the customer's location. Throw a StoreModelServiceException if the given address or the customer ID in the event object is invalid.

LocateCustomerCommand

Provides implementation details for identifying a customer's location. This action is performed in response to a Missing Person Event. The execute method is invoked when a customer asks for the location of another customer.

Methods

Method Name	Signature	Description
execute	(event:Event):void	Call the Store Model Service API to retrieve the location of the customer and utilize the AnnounceCommand to announce the message.

EmergencyCommand

Provides implementation details for opening all turnstiles, announcing an emergency message, and controlling robots for addressing the emergency and assisting customers exit the store. According to the requirements, these series of actions need to be perfomed in response to an Emergency Event.

Methods

Method Name	Signature	Description
execute	(event:Event):void	Call the Store Model Service API to open all turnstiles in the store, announce the emergency message on all speakers in the store. Then find a robot that is located in the same aisle as the emergency. If no robot is found, then get any robot. Instruct the rest of the robots to assist customers exit the store. Finally, close all the turnstiles.

Method Name	Signature	Description
		Throw a StoreModelServiceException if the given emergency is not a recognized emergency or if the given aisle ID in the event object is invalid.

FetchProductCommand

Provides implementation details for fetching one or more products to a customer. This action occurs in response to a Fetch Product Event per the requirements document.

Methods

Method Name	Signature	Description
execute	(event:Event):void	Call the Store Model Service API to find the requested product and bring it to the customer. Utilize the AnnounceCommand to announce a message to the customer in the following conditions: ➢ If the customer who is making the request is not registered. ➢ If the given product does not exist in the store. Throw a StoreModelServiceException if the given customer ID in the event object does not exist.

CleanUpAisleCommand

Provides implementation details for cleaning up an aisle. This action occurs in response to a Cleaning Event or a Broken Glass Event per the requirements document.

Methods

Method Name	Signature	Description
execute	(event:Event):void	Call the Store Model Service API to have a robot clean up the aisle. Throw a StoreModelServiceException if the given aisle ID in the event object does not exist in the store

CheckAccountBalanceCommand

Provides implementation details for checking a customer's account balance, computing the basket total, and announcing a message to the customer. Per the requirements document, these actions occur in response to a Check Account Balance Event where the customer asks for the total value of items in his/her basket total.

Methods

Method Name	Signature	Description
execute	(event:Event):void	Call the Store Model Service API to compute the total value of items of a registered customer's basket. If the customer is not registered, announce a message to inform the customer that registration is required to make a check account balance request. Call the Ledger Service API to retrieve the customer's account balance. Announce a message for the total value of basket items and the customer's account balance. Throw a LedgerException if no block has been committed yet or if the account address does not exist. Throw a StoreModelServiceException if the given customer ID in the event object does not exist or if the customer does not have an assigned basket.

CheckoutCommand

Provides implementation details for computing the basket total of a customer, retrieving the customer's account balance, creating and processing a transaction, computing the total weight of items in the basket, opening the turnstile, announcing a goodbye message, closing the turnstile, having a robot assist a customer to his/her car if the basket items weigh more than 10 lbs. These actions occur in response to a checkout event emitted by a turnstile per the requirements document.

Methods

Method Name	Signature	Description
execute	(event:Event):void	Call the Store Model Service API to check if the customer is registered and has an assigned basket. Then compute the basket total. If the basket total is positive, retrieve the customer's account balance and check if the customer has enough funds. Announce a message if the customer does not have enough balance. Otherwise, create and process the transaction using the Ledger Service API. Open the turnstile, announce a goodbye message, and close the turnstile. Compute the total weight of basket items. If the weight is over 10 lbs, a robot will assist

Method Name	Signature	Description
		the customer to his/her car. Throw a LedgerException for the following conditions: ➢ If no block has been committed yet ➢ If the account address does not exist ➢ If the min fee requirement is not satisfied ➢ If the amount is negative ➢ If the note is longer than the allowed note length ➢ If the payer or receiver account does not exist ➢ If the payer account does not have enough funds Throw a StoreModelServiceException if the given customer ID in the event object does not exist, if the customer does not have an assigned basket or if the customer is not located in the store.

EnterStoreCommand

Provides implementation details for checking a customer's account balance, granting/denying access to the store, opening the turnstile, announcing a welcome message, assigning a basket to the customer, and closing the turnstile. These actions occur in response to an Enter Store Event emitted by a turnstile per the requirements document.

Methods

Method Name	Signature	Description
execute	(event:Event):void	Call the Store Model Service API to retrieve the customer's account balance and grant access to the store if the customer has a positive account balance. Otherwise, deny access. After granting access, open the turnstile, announce a welcome message, assign a basket to the customer if the customer is registered. Finally, close the turnstile. Throw a LedgerException if no block has been committed yet or if the account address does not exist. Throw a StoreModelServiceException if the given customer ID in the event object does not exist

Event

Stores properties related to events emitted by the sensors of the Store Model Service.

Associations

Association Name	Type	Description
storeModelService	StoreModelService	StoreModelService instance that is used to access the Store Model Service API.
ledger	Ledger	Ledger instance that is used to access the Ledger Service API.
address	Address	Provides location information about the event
device	Device	Device object that is related to the event.

Properties

Property Name	Type	Description
eventType	EventType	Identifies the type of event.
sourceDeviceId	String	Unique identifier of the source device which emitted the event
customerId	String	Unique identifier of the customer that is related to the event.
productId	String	Unique identifier of the product that is related to the event.
addressString	String	Represents a location in the form of storeId:aisleId, aisleId:shelfId or aisleId.
count	int	Represents the quantity for a product.
emergencyType	EmergencyType	Identifies the type of emergency.
item	String	Item on the floor that needs to be cleaned.
message	String	Message to be announced.
state	boolean	Identifies the state of the turnstile.
authToken	String	Authorization token.

Observer (Interface)

The observers of the Store Model Service implement the Observer interface. The update method is called whenever sensors and appliances emit events.

Methods

Method Name	Signature	Description
update	(event:Event):void	Provides a method signature for the method that will be called by the Subject to update the observers.

Subject (Interface)

The Subject interface provides method signatures for registering, removing, and updating observers. The Store Model Service implements the Subject interface.

Methods

Method Name	Signature	Description
registerObserver	(observer:Observer):void	Provides a method signature for registering observers.
removeObserver	(observer:Observer):void	Provides a method signature for removing objects from being observers.
notifyObservers	(event:Event):void	Provides a method signature for notifying observers when the Subject's state changes.

Command (Interface)

The Command interface declares an interface for all commands. It provides a method signature for a single method called execute which is called by the StoreControllerService to perform actions in response to events. The execute method takes an Event object as an argument that holds a reference to a StoreModelService instance and Ledger instance and includes properties related to the event.

Methods

Method Name	Signature	Description
execute	(event:Event):void	Provides a method signature for invoking a command.

EventType (enum)

Identifies the type of event.

Properties

Property Name	Type	Description
EMERGENCY	EventType	Emergency event
BASKET	EventType	Basket event
CUSTOMER_SEEN	EventType	Customer seen event
CLEANING	EventType	Cleaning event
MISSING_PERSON	EventType	Missing person event
FETCH_PRODUCT	EventType	Fetch product event
CHECK_ACCOUNT_BALANCE	EventType	Check account balance event
ENTER_STORE	EventType	Enter store event
CHECKOUT	EventType	Checkout event

EmergencyType (enum)

Identifies the type of emergency in a store.

Properties

Property Name	Type	Description
FIRE	EmergencyType	Fire emergency
FLOOD	EmergencyType	Flood emergency
EARTHQUAKE	EmergencyType	Earthquake emergency

Property Name	Type	Description
ARMED_INTRUDER	EmergencyType	Armed intruder emergency

Design Details

The Store Controller Service uses the Observer pattern to get notified about the events that are emitted by devices in stores. The Store Controller Service implements the Observer interface. The Observer interface defines a method signature for a method called update which takes an Event argument. On the other hand, the Store Model Service implements the Subject interface which has three methods; registerObserver, removeObserver, and notifyObservers.

For the purposes of this assignment, events are fed into the Command Processor. When the Command Processor receives a create event command, it creates an Event object which includes properties related to the event. The Command Processor then passes this Event object to the Store Model Service's processEvent method which in turn calls the notifyObservers method to notify the Store Controller Service about the Event.

When the Store Controller Service receives the Event, it extracts the event name from the Event object and passes the event name to the CommandFactory class. The Command Factory Class looks at the event name and returns a Command object. The execute methods of command objects are invoked by the Store Controller Service. All Command objects communicate with the Store Model Service API to generate actions in response to events.

Exception Handling

The error conditions in the CommandProcessor methods result in a CommandProcessorException. The CommandProcessorException contains the failed command and the reason for the failure. Also, the line number of the command is included if the commands are read from a file.

For validating method arguments, IllegalArgumentExceptions are used and these exceptions are caught by the StoreModelService class. The StoreModelService class gets the message from these exceptions and throws StoreModelServiceExceptions.

StoreModelServiceExceptions can also result from error conditions in the Store Model Service methods. It captures the attempted action and the reason for failure.

The Store Controller Service catches LedgerExceptions that are thrown from the execute methods of the Command objects. The Store Controller Service extracts the action and reason properties from LedgerExceptions and using these properties throws StoreModelServiceExceptions. The Store Controller Service also throws StoreModelServiceExceptions that it receives from the Command objects.

Testing

A test driver class is implemented with a static main() method which accepts a command file. The test driver can be found at com\cscie97\store\test\TestDriver.java. The main() method calls the processCommandFile(file:string) method of the CommandProcessor.

Also two test scripts are provided; test_1.script and test_2.script. Both scripts include a sample store configuration, ledger configuration, sample events, and queries.

test_1.script demonstrates functionality whereas test_2.script demonstrates exception handling.

Risks

The system does not have any authentication service. Hence, anyone can generate events and control the appliances in the stores. Furthermore, sensitive information such as customer names, emails, or blockchain account addresses can easily be exposed. Therefore, this issue needs to be addressed for security reasons.