Smart City Controller Design Document

Date: 10/13/20
Author: Stephen Sheldon
Reviewer(s): Ethan Anthony, Paul Sotherland

Introduction

This document provides an overview of the Smart City Controller Service. This document first gives an overview of what problem the Smart City Controller Service solves. The requirements for the Smart City Service are broken down next. A use case diagram is shown followed by an explanation of all actors and use cases that the Controller Service encompasses. Implementation is discussed first by displaying a class diagram that is used to show all classes contained within the package "csci97/smartcity/controller" and then by a class dictionary that breaks down each individual class into its properties, associations, and methods. Implementation details explains how the different parts of the implementation interact, gives an instance diagram, a sequence diagram and also explains how the implementation addresses the requirements. The section on exception handling outlines how exceptional handling is dealt in the Controller Service. The Testing section explains how the Controller Service will be tested. Risks are finally addressed at the end to identify any associated risks with the design process.

Overview

The Controller Service interacts with the Ledger, Model and Authentication Services.

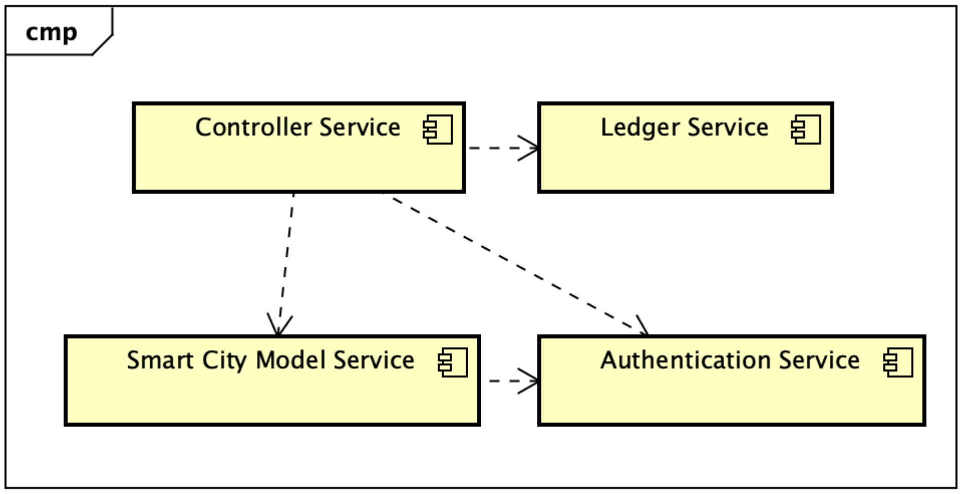

Figure 1: Component Diagram of the Smart City

The Controller Service interacts with the model service by monitoring the sensors of IoT Devices and people within the Smart City and also controls these IoT Devices. The Controller

Service handles monitoring of events received by the IoT sensors and responds to them. The Controller Service also has the ability to listen to voice commands within the Model Service and respond to them with the appropriate IoT Device.

The controller service handles financial transactions that occur within the Smart City. It utilizes the Ledger Service to manage transactions between accounts and check account balances.

Requirements

This section provides a summary of the requirements for the Smart City Controller Service.

Smart City Controller Service

1. Has ability to monitor sensors, IoT devices and persons within a city in the Model Service.
2. Monitors Model Service for IoT events by using the Observer Pattern and reacts to them based upon a set of rules.
3. Has the ability to listen to voice commands that occur in the Model Service and then react to them with the appropriate IoT devices.
4. Has the ability to respond to general questions about the state of the city.
5. Actions performed by the Controller service are implemented via the Command Pattern.
6. Can create transactions with the Ledger Service.

Natural Disaster Event

- Announces "There is a <emergency_type> in <city> please find shelter immediately" across all IoT devices in the city. Emergency type can be fire, flood, earthquake or severe weather.
- Sends half of the robots in the city to the latitude and longitude of the emergency.
- Orders the remaining half of robots within the city to "Help people find shelter".

Traffic Accident Event

- Makes an announcement via the microphone of the reporting IoT device to "Stay calm, help is on its way".
- Sends the nearest two robots to the latitude and longitude of the accident.

High CO2 Event

- If three high CO2 events are received from different devices in the city then all cars are disabled in the city.
- A high CO2 event is a level over 1000.

Low CO2 Event

- If the Smart City is currently in a state of high CO2 levels and receives three consecutive events to lower CO2 level of different devices to below 1000 then all cars are enabled in the city.

Litter Event

- Makes an announcement via the speaker of the reporting IoT device to “Please do not litter”.
- Sends the robot closest to the litter event to the latitude and longitude of the litter event to have it clean it up.
- Charges the person associated with the litter event 50 units for littering.
- If person is a visitor then they are not charged for littering.

Broken Glass Event

- Finds the robot nearest to the location of the broken glass event and sends them to the latitude and longitude of the broken glass event.

Person Seen Event

- Updates the location of the person that was seen.

Missing Child Event

- Checks to see if person with <person_id> exists.
- If person does exist then the IoT device that generated the event announces “Person <person_id> is at lat <lat> and <long>, a robot is retrieving now, stay where you are”.
- The closest robot to the missing person is located.
- The robot makes an announcement “Retrieve person <person_id> and bring to <lat> long <long> of microphone”.
- The robot’s latitude and longitude is updated to the location of the IoT device that generated the event.
- The missing person’s latitude and longitude is updated to the location of the IoT device that generated the event.

Parking Event

- Charges the account associated with the person in the vehicle for one hour of parking.
- Must trigger the creation of a transaction with the ledger service to charge the account.

Bus Route Event

- Generates a response originating at the receiving IoT device, “Yes, this bus goes to Central Square”.

Board Bus Event

- The IoT bus will announce “hello, good to see you <person_id>”, where <person_id> is the ID of the person who boards the bus.
- If the person is a resident and has a positive account balance, then the person’s account is charged for the rate of the bus.

- Triggers the creation of a transaction with the ledger service if person is a Resident.
- If person is a Visitor, then they are not charged a fee to ride the bus.

Movie Info Event

- The kiosk that event originated from makes an announcement over the speaker "Casablanca is showing at 9 pm".
- The kiosk text changes to display https://en.wikipedia.org/wiki/Casablanca_(film)#/media/File:CasablancaPoster-Gold.jpg.

Movie Reservation Event

- Gets the information that corresponds to the person_id associated with event.
- Checks for positive balance in ledger account of person.
- If person is a resident with a positive balance greater than or equal to 10 then their account is charged 10 units.
- If the person is a resident with an account balance less than 10 then they are notified that they do not have the funds to buy a movie ticket.
- If the person is a visitor, then they are not charged a fee for making a reservation.

Nighttime Event

- Turns all streetlights within the city to a maximum brightness of 100.

Use Cases

The Smart City Controller Service supports the following use cases:

Actors:

The Controller Service acts as an observer to the Model Service and interacts with the Model, Ledger and Authentication Service. The following diagram is used to help visualize the responsibilities of the controller service. In that way you can think of the Controller Service in this instance as acting on itself.

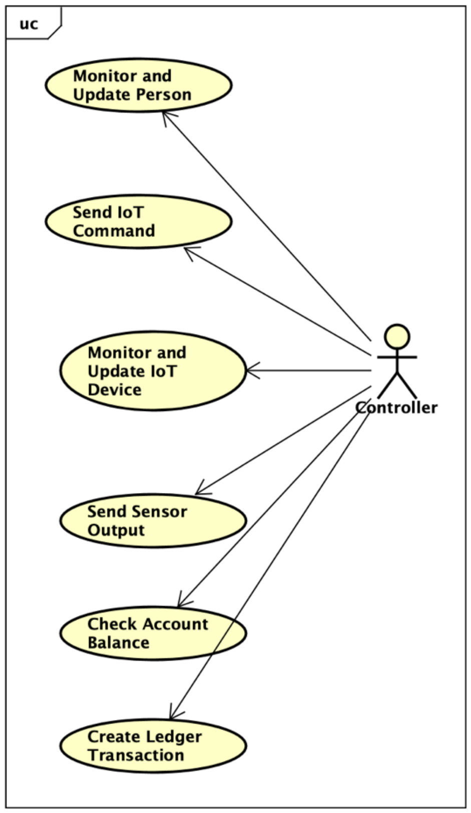
uc
Monitor and Update Person
Send IoT Command
Monitor and Update IoT Device
Controller
Send Sensor Output
Check Account Balance
Create Ledger Transaction

Figure 2: Use Case Diagram for Controller Service

Monitor Person

- The Controller Service has the ability to monitor a person within the Smart City. It does this by acting as an observer of the Model Service. The Model Service sends a notification to all Observers when an event occurs such as updating a person's location.

Send IoT Command

- The Controller Service sends IoT commands in response to sensor events that occur. The Model Service notifies Observers of new events and the Controller, which acts as an Observer, then reacts. The Controller can send IoT Commands such as telling an IoT device to make an announcement.

Monitor IoT Device

- The Controller acts as an active Observer of the Model Service. The Model Service knows that an Observer may exist but doesn't know who exactly that observer is. The Model Service regardless of this knowledge attempts to notify any observers of events that occur within the smart city such as a car parking in a parking space.

Update IoT Device

- The Controller has the ability to update IoT Devices and does so based on specific events it observes.

Send Sensor Output

- The Controller has the ability to send sensor output to the speaker of an IoT Device. It can make announcements over the speaker of any IoT device within the Smart City.

Check Account Balance

- The Controller service has the ability to check the account balance of an account within the Ledger Service. This is used to perform checks to verify if an account has the available funds needed to create a transaction generated from an event.

Create Ledger Transaction

- The Controller Service can create Transactions via the Ledger Service. Transactions are created with events that have a fee associated with them such as a Resident taking the bus.

Implementation

Class Diagram

The following is a class diagram for all classes contained within the package "cscie97.smartcity.controller".

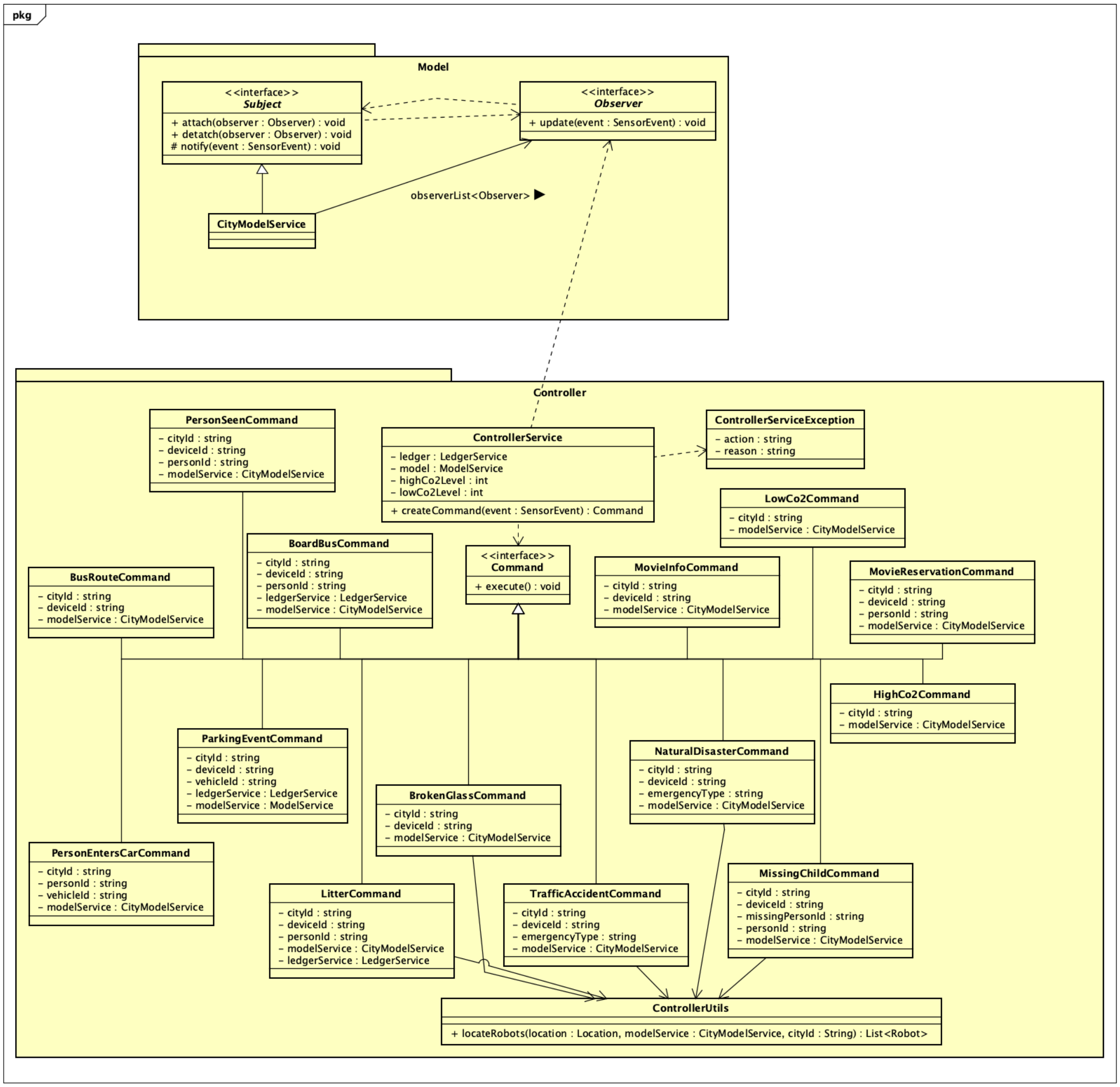

Figure 3: Class Diagram for Controller Service

Class Dictionary

This section specifies the class dictionary for the Smart City Controller Service. The classes should be defined within the package “cscie97.smartcity.controller”.

Subject

The Subject Interface is an interface used to implement the Observer design pattern. The Subject class has abstract methods to be implemented in the City Model Service for attaching new Observers to the subject, removing Observers from the subject and notifying all Observers of SensorEvents.

Methods

Method Name	Signature	Description
attach	(observer:Observer):void	Takes an Observer Object and adds it to the observerList.
detatch	(observer:Observer):void	Removes Observer object observer from the observerList.
notify	(event:SensorEvent):void	Loops through all Observer objects in the observerList and calls the update method passing the SensorEvent event in as a parameter.

Observer

The Observer interface contains the abstract method update. The update method is to be implemented but any class that implements the Observer interface. Specifically, the Controller Service will implement the Observer interface.

Methods

Method Name	Signature	Description
update	(event:SensorEvent):void	Takes a SensorEvent. Implementation is specified by implementing concrete class. The ControllerService will implement this method contains logic to decide what to do with SensorEvents and generate Command objects to respond to them.

ControllerService

The ControllerService handles the logic for processing SensorEvent received when the ModelService subject notifies it of a new SensorEvent. The ControllerService implements the Observer interface and is responsible for providing a concrete implementation of the interface's update method. The ControllerService creates Command objects based upon the type of SensorEvent is receives and then executes those Command objects.

The ControllerService per the requirements implements the ability to create transactions, respond to voice commands, and IoT events by creating Command object responses with the createCommand method.

Properties

Property Name	Type	Description

ledger	LedgerService	The LedgerService that is associated with the Controller when the ControllerService is instantiated. Used for checking account balances and creating transactions.
model	ModelService	The ModelService that is associated with the Controller when the ControllerService is instantiated. Used for retrieving data from the Model and updating data in the Model.
highCo2Level	int	Tracks how many high Co2 events have been seen.
lowCo2Level	int	Tracks how many low Co2 events have been seen.

Methods

Method Name	**Signature**	**Description**
createCommand	(event:SensorEvent):Command	Takes a SensorEvent event and returns the appropriate concrete Command class to the Controller Service.

ControllerServiceException

The ControllerServiceException is returned from the Controller Service in response to error conditions or an event that is not handled by the Controller Service. The Controller Service Exception captures the action that triggered the exception and the reason as to why is was triggered.

Properties

Property Name	**Type**	**Description**
action	string	The action performed that triggered the exception.
reason	string	The reason the exception was thrown.

Command

The Command Interface contains the abstract methods execute and log. The Command interface is utilized in implementation of the Command pattern.

Methods

Method Name	**Signature**	**Description**
execute	void	Implemented by concrete classes.

NaturalDisasterCommand

The NaturalDisasterCommand is a command response to SensorEvents that detect a natural disaster. Natural disasters can be either a fire, flood, earthquake or severe weather. When executed it announces over all devices that there is an emergency and to please find shelter. It then sends half of all robots within a city to the location of the emergency and orders the remaining half of the robots to help people find shelter.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device where event originated from.
emergencyType	string	The type of emergency.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Implementation from the Command interface. Makes an announcement over all IoT devices within the city for people to find shelter. Sends half of the robots within the city to the location of the emergency and has the remaining half of the robots help people find shelter.

TrafficAccidentCommand

The TrafficAccidentCommand is a command response to SensorEvents that detect a traffic accident. When executed an announcement is made via the microphone of the reporting IoT device to “stay calm, help is on its way.” The two nearest robots to the IoT device are then sent to the location of the IoT device.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device where event originated from.
emergencyType	string	The type of emergency.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Implementation from the Command interface. Makes an announcement over the IoT device speaker that reported the event. Finds the two nearest robots to the traffic accident and updates their location to the latitude and longitude of the traffic accident.

HighCo2Command

The HighCo2Command is a command response that is triggered when three events Co2 events are received with CO2 levels equal to or greater than 1000.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Disable all cars in the city.

LowCo2Command

The LowCo2Command is a command response that is triggered when three co2 events are received with values less than 1000.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Enable all cars in the city.

LitterCommand

The LitterCommand is a command response that is triggered when a SensorEvent detects that a person has thrown garbage to the ground.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device that has detected the event.
modelService	ModelService	Reference to Model Service.
ledgerService	LedgerService	Reference to Ledger Service.
personId	string	Person ID of person that initiated event.

Methods

Method Name	Signature	Description
execute	void	The IoT device that detected the event broadcasts the message "Please do not litter" over its speaker. Sends the nearest robot to the location of the reported event to clean up the trash. Charges the person whom littered 10 units from their ledger account.

BrokenGlassCommand

The BrokenGlassCommand is a command response to a SensorEvent that has detected broken glass.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device that has detected the event.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description

execute	void	Sends the nearest robot to the location of the broken glass.

PersonSeenCommand

The PersonSeenCommand is a command response to a SensorEvent that has detected a person in a new location.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device that event originated from.
personId	string	Person ID of person that was seen.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Updates the location of the person to the location that the person was seen at.

MissingChildCommand

The MissingChildCommand is a command response to a SensorEvent that has requested help for finding a missing child.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device that has detected the event.
modelService	ModelService	Reference to Model Service.
missingPersonId	string	Person ID of person that is missing.
personId	string	Person ID of person that initiated event.

Methods

Method Name	Signature	Description
execute	void	Checks to see if the requested person exists. Announces to requester via IoT device speaker “person <person_id> is at lat <lat> long <long>, a robot is retrieving now, stay where you are”. The location of the missing person is updated to the location of the requesting person. The location of the nearest robot to the missing person is updated to the location of the requesting person.

ParkingEventCommand

The ParkingEventCommand is a command response to a SensorEvent that has detected a vehicle has parked into an IoT parking space. Note that it is necessary for a person to enter a car for that car to park in a parking space as the person’s account is needed to be charged for the parking space fee.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device where event originated from.
vehicleId	string	Device ID of vehicle that has parked in parking space.
modelService	ModelService	Reference to Model Service.
ledgerService	LedgerService	Reference to Ledger Service.

Methods

Method Name	Signature	Description
execute	void	Retrieves ledger account of person who has parked vehicle. Creates a transaction with the Ledger Service to charge the person’s account for 1 hour of time. The hourly fee is associated with the IoT Parking Space.

BusRouteCommand

The BusRouteCommand is a command response to a SensorEvent that has detected a person asking if a bus goes to central square.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device that has detected the event.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Creates a SensorOutput with value “Yes, this bus goes to Central Square.” The SensorOutput is passed to the IoT device in which the person interacted with and an announcement is made to the person.

BoardBusCommand

The BoardBusCommand is a command response to a SensorEvent that has detected a person boarding a bus.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device that has detected the event.
ledgerService	LedgerService	Reference to Ledger Service.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Creates a SensorOutput that announces to the person, “Hello, good to see you <person_id>”. If the person is a resident with a positive account balance that is greater or equal to the fee to ride the bus, then their associated ledger account is

		charged. If the person does not have sufficient funds, then they are notified and denied from riding the bus. If the person is a visitor, then they are allowed to ride the bus for free.

MovieInfoCommand

The MovieInfoCommand is a command response to a SensorEvent that has detected a person asking a kiosk “What movies are showing tonight?”.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device that has detected the event.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Creates a SensorOutput to tell the user “Casablanca is showing at 9 pm”. Updates the display on the kiosk to https://en.wikipedia.org/wiki/casablanca_(film)#/media/File:CasablancaPoster-Gold.jpg.

MovieReservationCommand

The MovieReservationCommand is a command response to a SensorEvent that has detected a person asking a kiosk to “reserve 2 seats for the 9 pm showing of Casablanca”.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
deviceId	string	Device ID of device that has detected the event.
personId	string	Person ID of person that initiated the event.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Retrieves the Person object of the person that has requested to reserve seats. Checks for positive balance greater than or equal to 10 units in person's ledger account. If the person is a resident, then a ledger transaction is created to charge the person's account 10 units and they are notified "your seats are reserved; please arrive a few minutes early". If the person is a resident with insufficient funds, then they are notified that they have insufficient funds. If the person is a visitor, then they are not charged a fee.

PersonEntersCarCommand

The PersonEntersCarCommand is a command response to a SensorEvent that has detected that a person has entered a car.

Properties

Property Name	Type	Description
cityId	string	City ID of City where event has occurred.
personId	string	Person ID of person who has entered car.
vehicleId	string	Device ID of vehicle that has been entered.
modelService	ModelService	Reference to Model Service.

Methods

Method Name	Signature	Description
execute	void	Sets the car's ledger account to be the person's ledger account.

ControllerUtils

Provides utility methods to locate all robots within a city and sort them by distance.

Methods

Method Name	Signature	Description

locateRobots	(location:Location,modelService:CityModelService, cityId:string):List<Robot>	Returns a list of robots sorted by distance from closest to furthest away from Location parameter.

Implementation Details

The following sequence diagram shows how the Observer and Command patterns are implemented in the Smart City Model and Controller services.

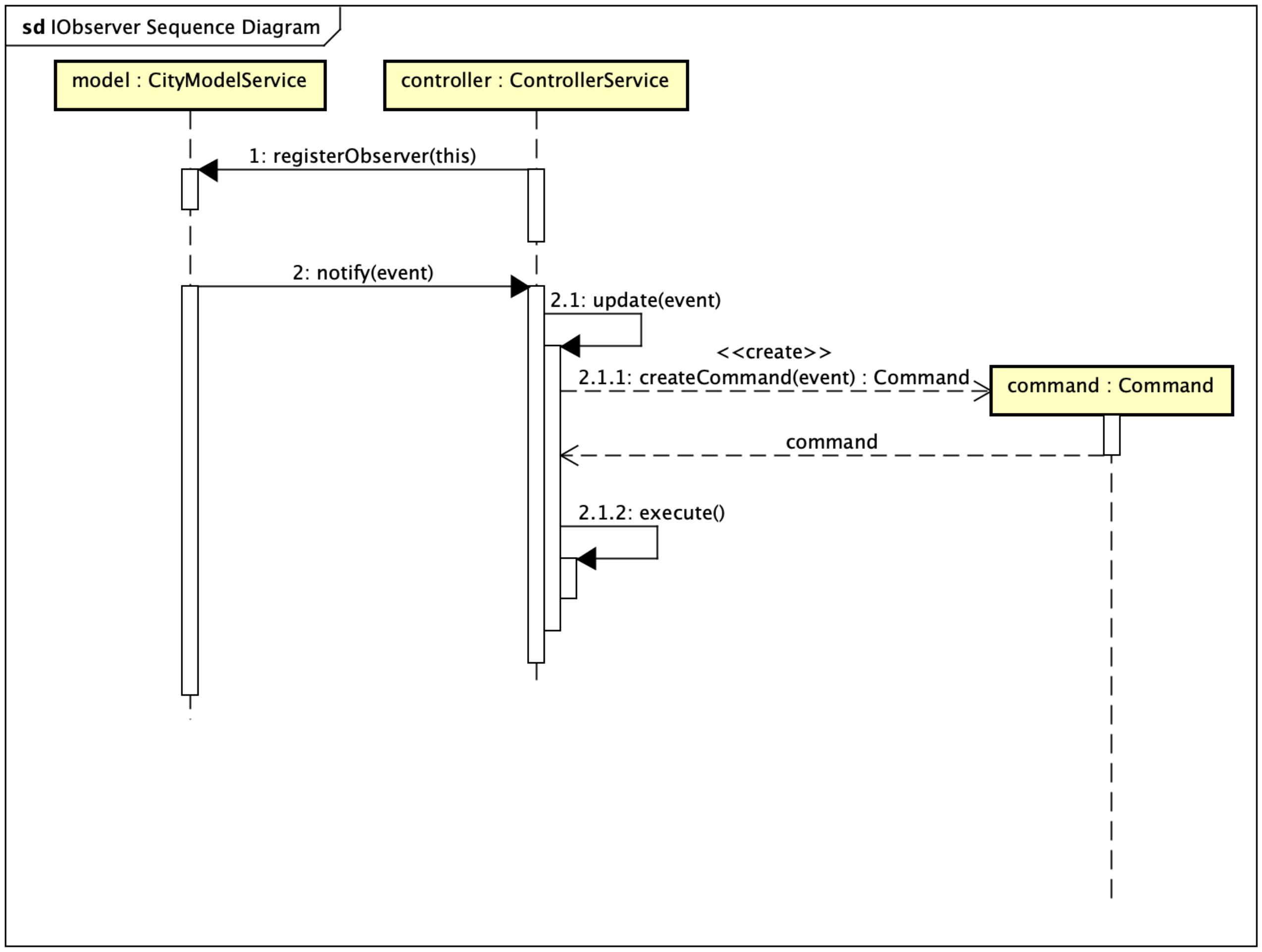
sd IObserver Sequence Diagram
model : CityModelService
controller : ControllerService
1: registerObserver(this)
2: notify(event)
2.1: update(event)
<<create>>
2.1.1: createCommand(event) : Command
command : Command
command
2.1.2: execute()

Figure 4: General Observer and Command Pattern Sequence Diagram

The CityModelService acts as the subject in the observer pattern. It implements the Subject interface and overrides the three abstract methods registerObserver, removeObserver and notifyObserver. I decided to use an interface to favor composition versus inheritance in order to adhere to the dependency inversion principle.

The CityModelService registers itself as an observer of the ModelService. When the ModelService receives a new SensorEvent it then notifies all its observers of the event. In the above sequence diagram, it notifies the ControllerService of the event. The ControllerService receives the notification and calls update passing in the event. Update then calls the createCommand method passing in the event. CreateCommand contains the logic as to what type of command to create and return. The correct command for the event is created and returned to the ControllerService. The ControllerService then executes the command.

The following sequence diagram gives a more detailed sequence of events that take place in response to a natural disaster event.

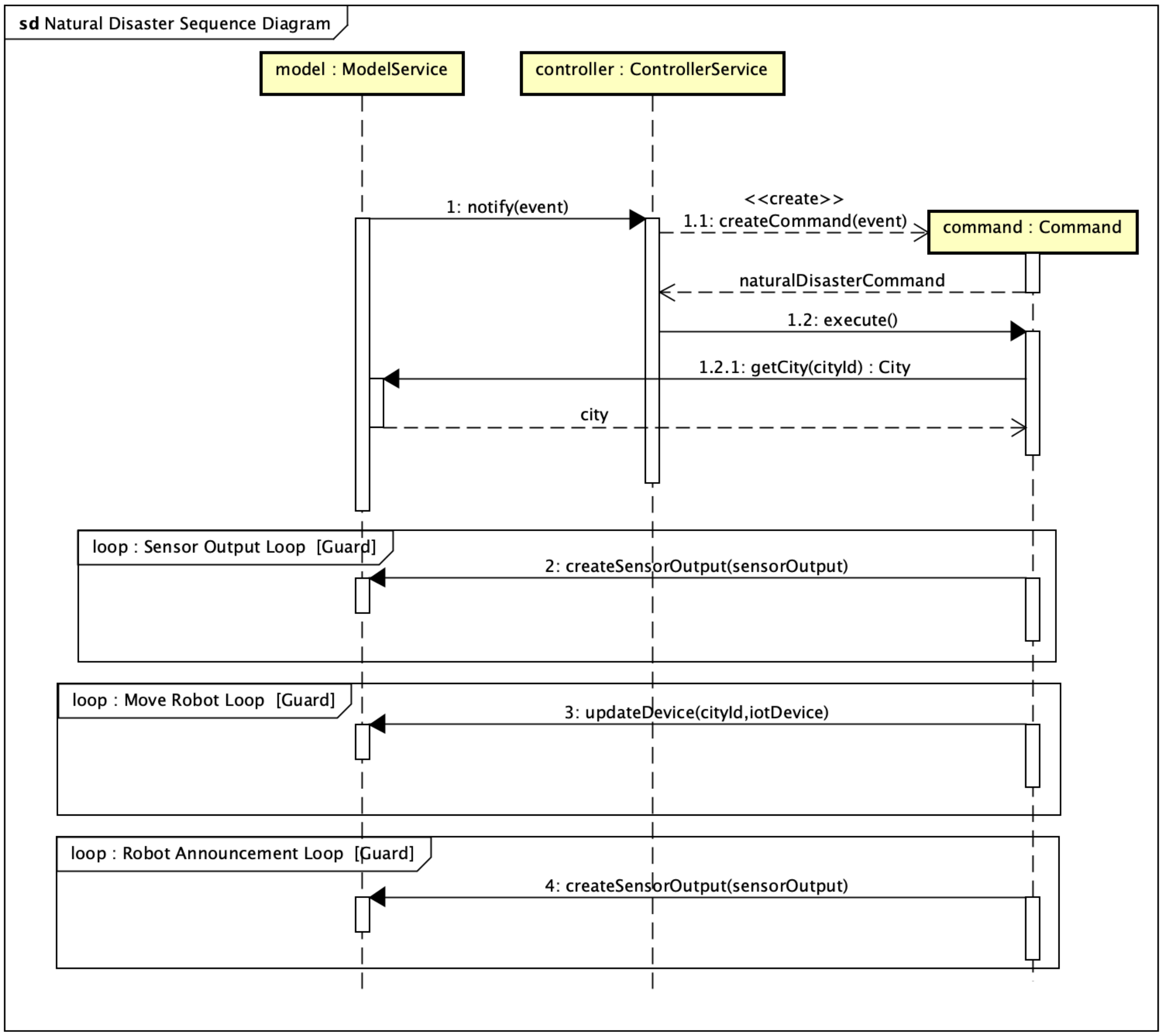
sd Natural Disaster Sequence Diagram
model : ModelService
controller : ControllerService
command : Command
1: notify(event)
<<create>>
1.1: createCommand(event)
naturalDisasterCommand
1.2: execute()
1.2.1: getCity(cityId) : City
city
loop : Sensor Output Loop [Guard]
2: createSensorOutput(sensorOutput)
loop : Move Robot Loop [Guard]
3: updateDevice(cityId,iotDevice)
loop : Robot Announcement Loop [Guard]
4: createSensorOutput(sensorOutput)

Figure 5: Sequence Diagram for Natural Disaster Event

The createCommand method will identify the correct Command response and, in this case, returns a NaturalDisasterCommand. The Controller Service will then call execute on this command. The SensorEvent that the Command received stores the cityId of the city associated with the generated event. When the command is executed it reaches out to the Model Service to retrieve a copy of the city and all devices and people in it. The command then loops through all IoT devices in the city sending them a SensorOutput that contains the following string for them to make as an announcement – "There is a <emergency_type> in <city> please find shelter immediately." The command then iterates through half of the total robots in the city and calls updateDevice on them in order to update their location to the location of the disaster. Finally, the command sends a SensorOutput to the remaining half of the robots in the city telling them to "Help people find shelter".

Exception Handling

The Controller Service validates events that it observes happen in the Model Service. If the event is one that is not handled by the Controller Service, then an exception is thrown stating such and it is written to the log. If a specific error occurs for a valid event, such as no robots in a city able to respond to a natural disaster, then classes that implement the Command interface will throw a ControllerServiceException stating the action that was attempted and the reason why the exception is thrown. This information is then logged into log.txt.

Testing

For this assignment testing will be done via a test script. Details on how to run tests and expected output can be found in the testing.pdf file. Manual unit tests are utilized to verify that the Controller Service is working as intended. The test script consists of valid scenarios and also invalid scenarios that are utilized to verify error handling. The test script also does functional to verify that requirements are met by generating events for every event scenario that the Controller can handle and then verifying that the correct response is executed. Integration testing is inherently done in event scenarios that require the controller service to interact with the Model and Ledger services. Future integration testing will include the Authentication Service once that service is created in assignment 4.

Risks

Several event scenarios involve financial transactions. It's important that access to the Ledger Service is isolated only to the Controller Service so that outside actors cannot access it and modify accounts or create transactions. It is important that for events that do involve financial transactions that accounts are verified to have a high enough balance for the transaction to occur and that transactions update account balances. At the moment the system uses in-memory data storage so if the system goes down all data for the Smart City is lost. In the future this data should be persisted to a database to avoid the loss of data.