Smart City Model Service Requirements

Brett Bloethner

Introduction

This document provides system architecture details for the Smart City Software System's Smart City Controller Service. In this document you will find use case diagrams, class diagrams, sequence diagrams and implementation details that will help outline and explain the Smart City Controller Service design.

Table of Contents

1. Overview
2. Requirements
3. Use Cases & Use Case Diagram
4. Class Diagram
5. Class Dictionary
6. Implementation Details
7. Exception Handling
8. Testing
9. Risks
10. System Changes

Overview

The Smart City Controller Service can be considered the "brain" of the Smart City. One of the core requirements of the Smart City is that it needs to respond events that happen in the city. These events include things that people initiate, like buying movie tickets as well as metrics from sensors on smart city IoT devices. The Smart City Controller Service is the component responsible for coordinating how the city reacts to each of these events as they occur. The Controller Services ability to be notified of and respond to events is what allows it to coordinate responses to events in the city.

Requirements

This section describes the features required of the Controller Service. In order to effectively support the other components of the Smart City System, the Smart City Controller Service must…

Monitor the City's IoT Devices by…

- Implementing the Observer Pattern, allowing the Controller service to "subscribe" to a stream of events sent by the Smart City Model Service
- memorizing the last three CO2 readings form unique CO2 sensors in each city order to get an accurate overall CO2 reading for monitoring of air quality
- Appropriately parsing out voice command events to be responded to accordingly

- logging out all of the event and event responses processed by the Controller Service

Response to Events from the City's IoT Devices by…

- implementing the Command Pattern to allow effective queueing and execution of responses to events
- evaluating a set of predefined rules which determine if and how the Controller Service should respond to different types of events
- creating and executing commands, in response to events, that include generating and sending control messages back to the device from which the event was generated

Trigger Transactions in the Ledger Service by…

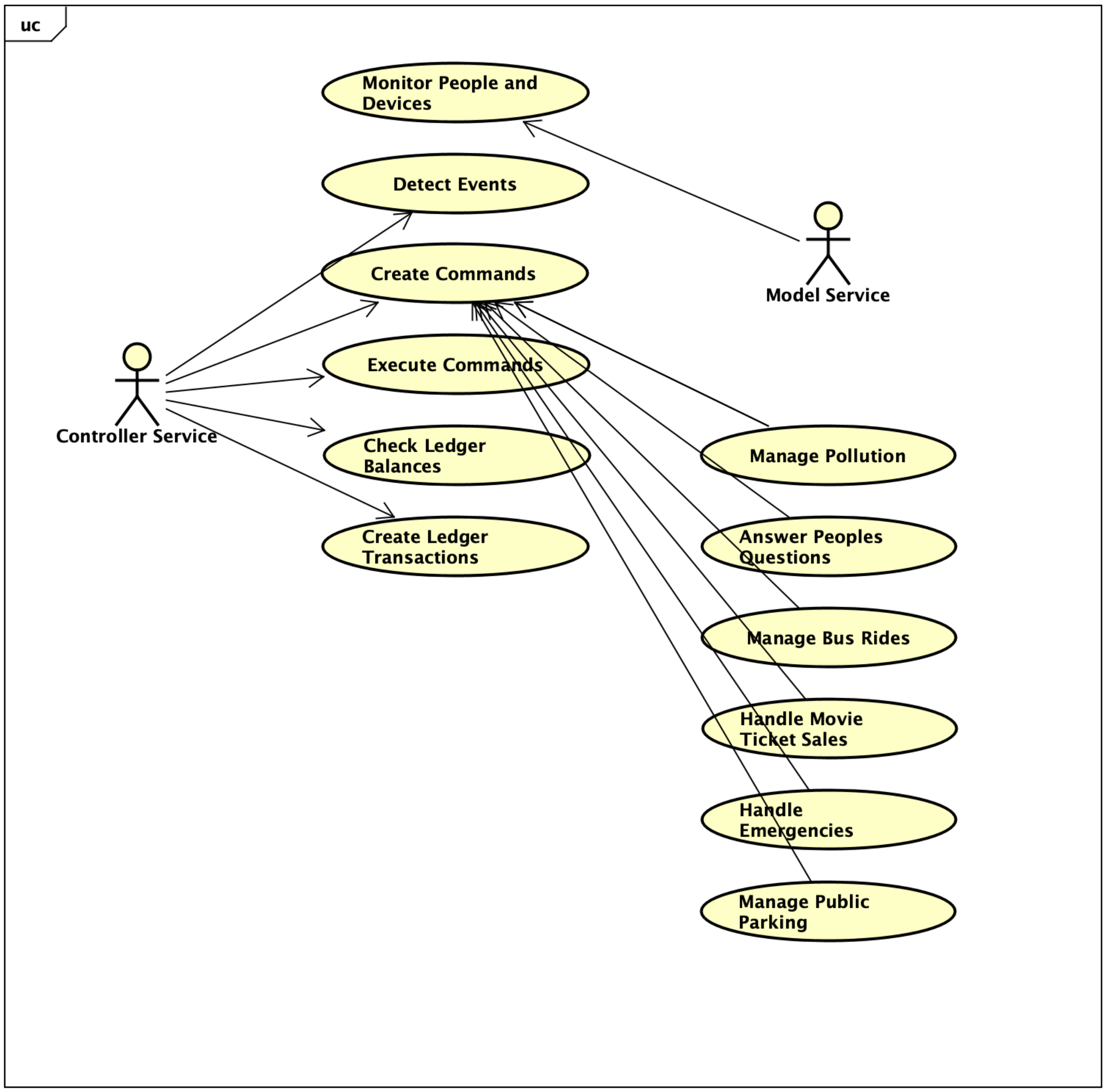
- sending transactions to the Smart City Ledger Service to be executed in response to events
- accurately handling an event in response to a successful transaction
- accurately handling an event in response to a failed transaction

Use Cases

Actors

- **Subject Model Service:** The Smart City Subject Model Service interacts with the Smart City Controller Service through one use case which is "Monitoring People and Devices." Through this use case, the Smart City Model Service provides events to the Smart City Controller Service for continuous people, device and city monitoring.
- **Controller Service:** The Controller Service the main actor in the Smart City Controller Service. Ironically it is the user of most of its own functionality. This is by design since its job is primarily to listen to event and process them using from start to finish.

Below is the use case diagram which outlines what the expected actors are and what abilities they'll have in the application.



Monitor People and Devices

- The notify() function on the Smart City Model Service gets called as a result of an event occurring
- The Smart City Model Service calls the update() function on each observer watching it

Detect Events

- The Smart City Controller service update() function is called by the Smart City Model Service, notifying the Controller Service that an even has just occurred

Create Commands

Manage Pollution

- A CO2 even is received via the update() function
- Determine whether or not the CO2 level is above or below 1000 and whether the last 3 CO2 events we of the same condition
- If the event is the 3rd consecutive event in that City with a level above 1000, get the City and disabled all cars in the city

- If the event is the 3rd consecutive event in that City with a level below 1000, get the City and enable all cars in the City

Answer Peoples Questions

- Determine whether or not the Event includes a qualified question
- If the question is about Movie Info, output movie info response via the Device that created the event
- If the question is about bur route info, output bus route info response via the Device that created the event

Manage Bus Rides

- Get the residents account and their ledger account
- Welcome them onto the bug and process transaction if they have enough funds
- Return an error if they do not

Handle Movie Ticket Sales

- Get resident and their ledger account
- Let them know seats are reserved if they have enough funds to purchase tickets

Handle Emergencies

- Determine type of emergency
- If its type 1
  - announce City wide announcement across all Devices
  - deploy half of robots to help with emergency
  - deploy other half of robots to help people take shelter
- It its type 2
  - emit message from the Device that reported the event
  - deploy the 2 nearest robots to the location of the event

Manage Public Parking

- Get vehicle by vehicle ID
- Process a ledger transaction and debit the vehicles ledger account 1 for each hour

Execute Commands

- Run the execute function on the concrete command
- Command will process the Even provided in accordance with the rule logic in the concrete command

Check Ledger Balances

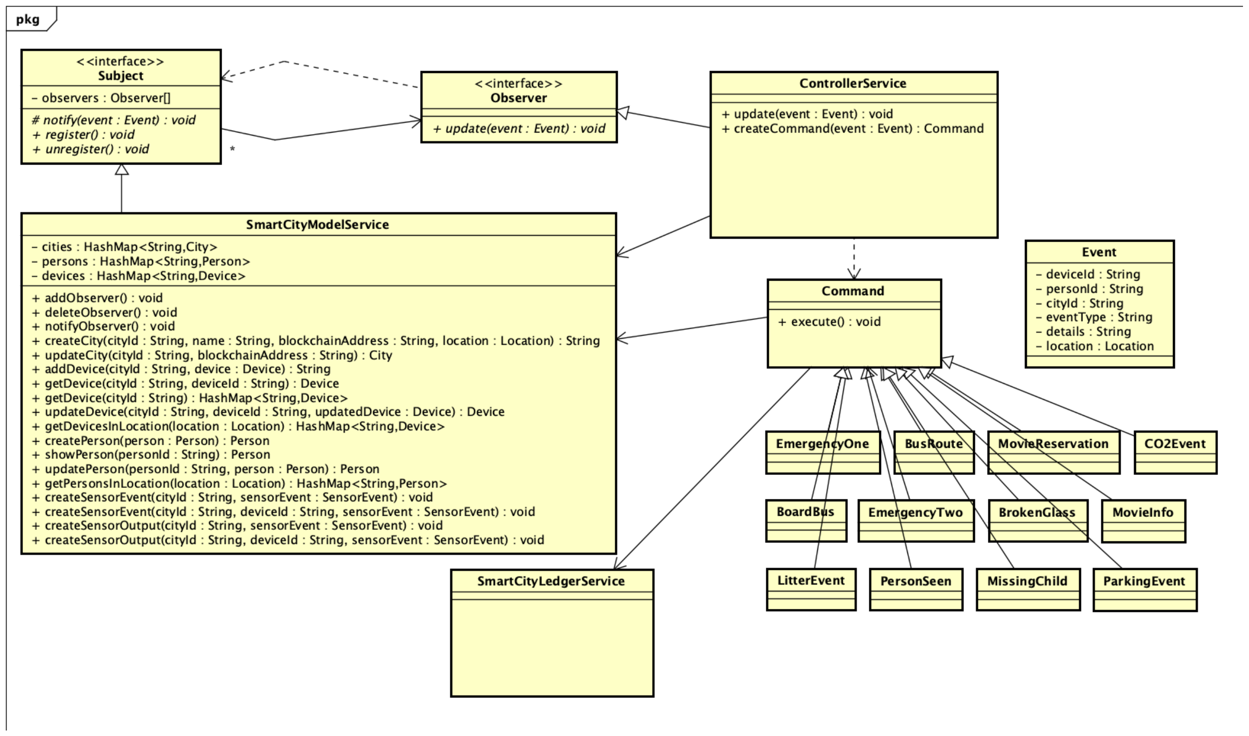
- Get ledger by resident, city or device account ID

Create Ledger Transactions

- Create a transaction using the spending account and receiving account
- Execute the transaction
- Return an error and stop the command executioin if the transaction fails

Class Diagrams

The following class diagram defines the Smart City Controller implementation classes contained within the package “com.cscie97.smartcity.controller”. Below is the class diagram outlining the design and relation of all the classes in the Smart City Controller Service.


pkg
<<interface>> Subject
- observers : Observer[]
# notify(event : Event) : void
+ register() : void
+ unregister() : void
*
<<interface>> Observer
+ update(event : Event) : void
ControllerService
+ update(event : Event) : void
+ createCommand(event : Event) : Command
SmartCityModelService
- cities : HashMap<String,City>
- persons : HashMap<String,Person>
- devices : HashMap<String,Device>
+ addObserver() : void
+ deleteObserver() : void
+ notifyObserver() : void
+ createCity(cityId : String, name : String, blockchainAddress : String, location : Location) : String
+ updateCity(cityId : String, blockchainAddress : String) : City
+ addDevice(cityId : String, device : Device) : String
+ getDevice(cityId : String, deviceId : String) : Device
+ getDevice(cityId : String) : HashMap<String,Device>
+ updateDevice(cityId : String, deviceId : String, updatedDevice : Device) : Device
+ getDevicesInLocation(location : Location) : HashMap<String,Device>
+ createPerson(person : Person) : Person
+ showPerson(personId : String) : Person
+ updatePerson(personId : String, person : Person) : Person
+ getPersonsInLocation(location : Location) : HashMap<String,Person>
+ createSensorEvent(cityId : String, sensorEvent : SensorEvent) : void
+ createSensorEvent(cityId : String, deviceId : String, sensorEvent : SensorEvent) : void
+ createSensorOutput(cityId : String, sensorEvent : SensorEvent) : void
+ createSensorOutput(cityId : String, deviceId : String, sensorEvent : SensorEvent) : void
Command
+ execute() : void
Event
- deviceId : String
- personId : String
- cityId : String
- eventType : String
- details : String
- location : Location
EmergencyOne
BusRoute
MovieReservation
CO2Event
BoardBus
EmergencyTwo
BrokenGlass
MovieInfo
LitterEvent
PersonSeen
MissingChild
ParkingEvent
SmartCityLedgerService

Class Dictionary

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

Controller Service

Class descriptioin

Methods

Method Name	Signature	Description

Properties

Property Name	Type	Description

Associations

Association Name	Type	Description

Command

Class descriptioin

Methods

Method Name	Signature	Description

Properties

Property Name	Type	Description

Associations

Association Name	Type	Description

TODO: EACH CONCRETE COMMAND CLASS WILL INCLUDE THE LOGIC TO FULFILL THE RULE FOR THE NECESSARY COMMAND/EVENT

Emergency One

Emergency Two

Board Bus

Bus Route

Litter Event

Movie Reservation

Movie Info

Broken Glass

Missing Child

Parking Event

CO2 Event

Person Seen

Event

Class descriptioin

Methods

Method Name	Signature	Description

Properties

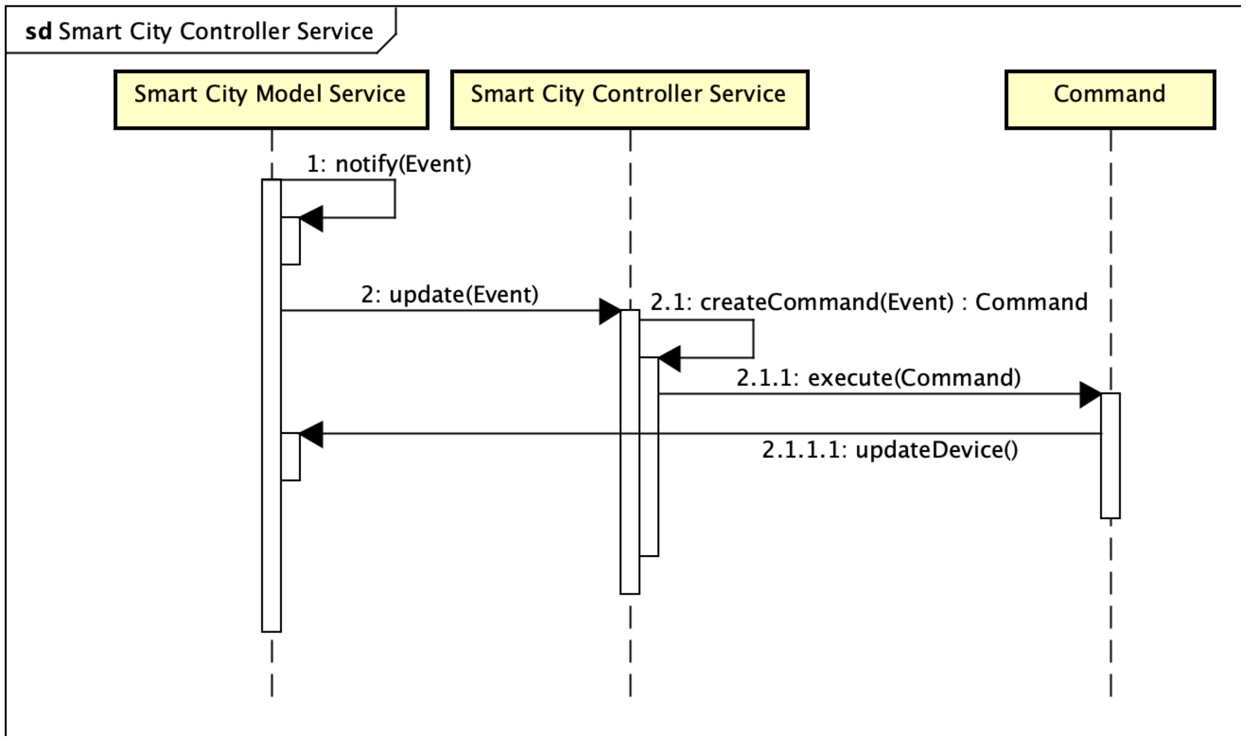
Property Name	Type	Description

Associations

Association Name	Type	Description

Implementation Details

The diagram below shows the general sequence of events executed by the Smart City Model Service once an event update is received.



All events start with the notify function. This is a core component of the observer patterns and allows the Smart City Model Service to tell the Smart City Controller Service when and event has occurred on one of the models. Next the Model Service used the event passed into Notify to call update on all of the objects observing the Model Service. This triggers the update function on the Smart City Controller Service. After the Controller Service receives the event, a command is created by the Controller Service by using the createCommand function. The createCommand function is basically a Command factory that will create a concrete command class based on whatever data is inside the Event object. Next, the execute function is called on the concrete command. This allows the rules in the concrete command to be processed and the even to responded to appropriately. Often times, a component of responding to an event will include calling back to the Device in order to update it or relay a message. This is expressed in the last step where updateDevice is called. However, it’s important to note that the specific function called on the Device might not be updateDevice and it’s even possible the Device might not need to be updated at all.

Exception Handling

Exception handling will be done primarily by way of making sure the Event objects include all the data needed to respond to an event. Exceptions will halt the command execution of a response to an event and will result in a error being logged rather than the event being responded to.

Testing

Testing

Risks

Risks

System Changes

System Changes