House Mate Controller Design Document

Date: 10/20/2015
Author: Gerald Trotman
Reviewer(s):

Introduction

This document defines the design for the House Mate Controller Service API, a model that centralizes and implements commands once imported from the Model Service.

Overview

After having stored the states of our appliances and sensors as well as occupants and their location within a given room, our next problem we needed to solve is how to simulate an interaction where commands are acted upon our model and stimulate change to our system. Since we are selling this product as a “Smart Home” being able to demonstrate its interactivity is fundamental to providing business value.

Requirements

This section defines the requirements for the Mobile Application Store Product API.

Now that we have set the configurations in place for out Model Service, we now have to turn our attention on our Controller Service. In our requirements:

1. We are to set up a “listener” that interprets our CommandImporter as commands which either “set”, “define”, “show”, or “add” to the status and value of our current appliances or sensors
2. We are required to keep track of the location of our occupants with a KnowledgeGraph
3. We are required to execute commands in the form of our Command Pattern implementation.
4. We are also required to log the execution of our commands

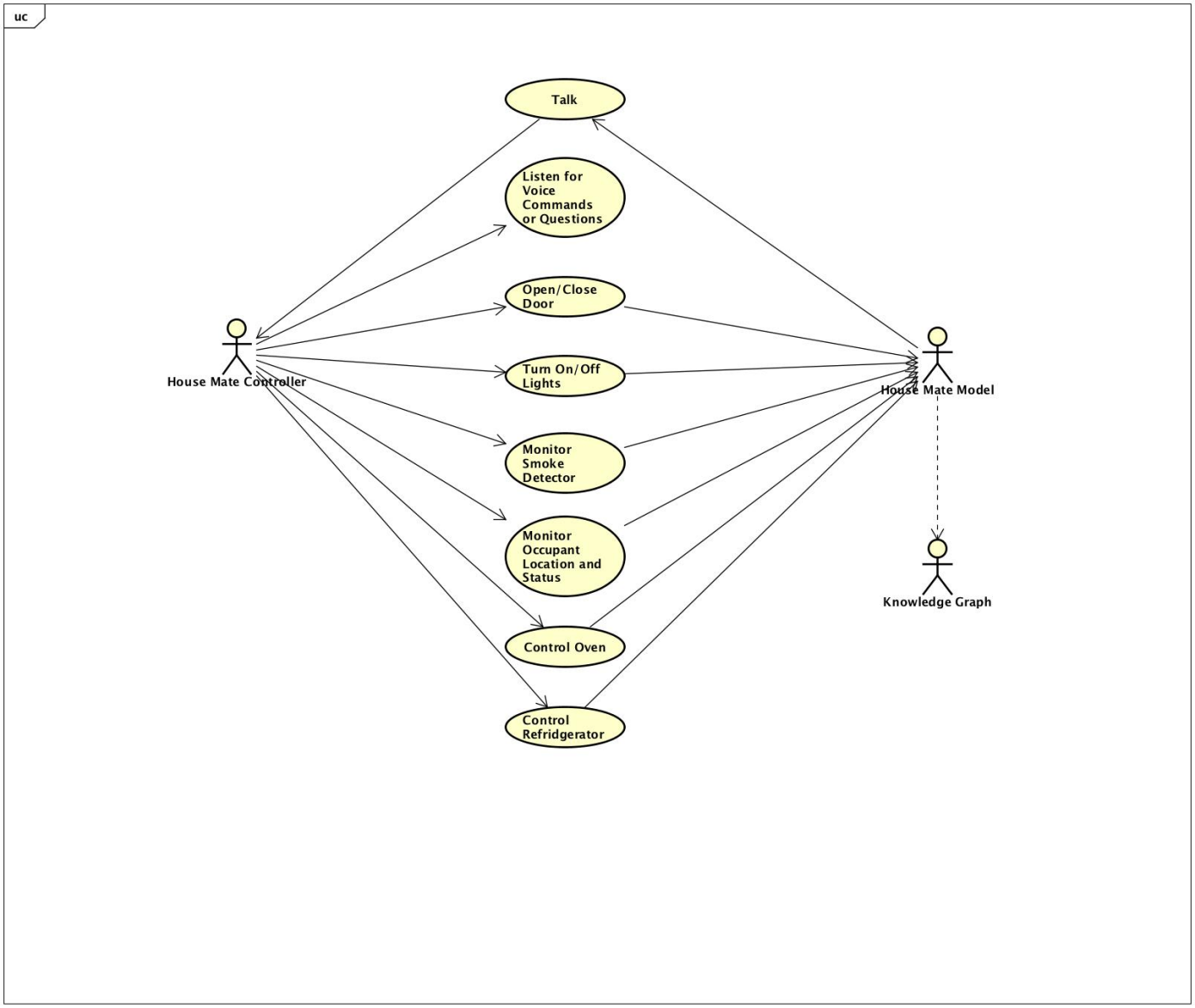
For the full detaile list of commands that we are asked to implement, please refer to our HouseMateControllerServiceRequirements.pdf.

Use Cases

This design supports the following use cases:

1. As the House Mate Controller
    a. It listens for a command being spoken to by the House Mate Model via Ava
    b. It is notified by the House Mate Model via Ava to open or close the door
    c. It is notified by the House Mate Model via Ava to turn the light on or off
    d. It is notified by the House Mate Model that the Smoke Detector
        i. Ava requests that the Occupants need to leave
        ii. The House Mate Model Service to talk to the Knowledge Graph and clear out all of the Occupants
    e. It is notified by the House Mate Model via Camera of the Occupants location and/or state
        i. The House Mate Model Service talks to the Knowledge Graph to get the location of the occupant
            1. There occupant leaves the room
                a. Turn the lights off
                b. Turn off the thermostat
            2. The occupant enters the room
                a. Turn the lights on
                b. Turn the thermostat on
            3. The occupant is active
            4. The occupant is inactive
                a. Dim lights
                b. Occupant is set to resting

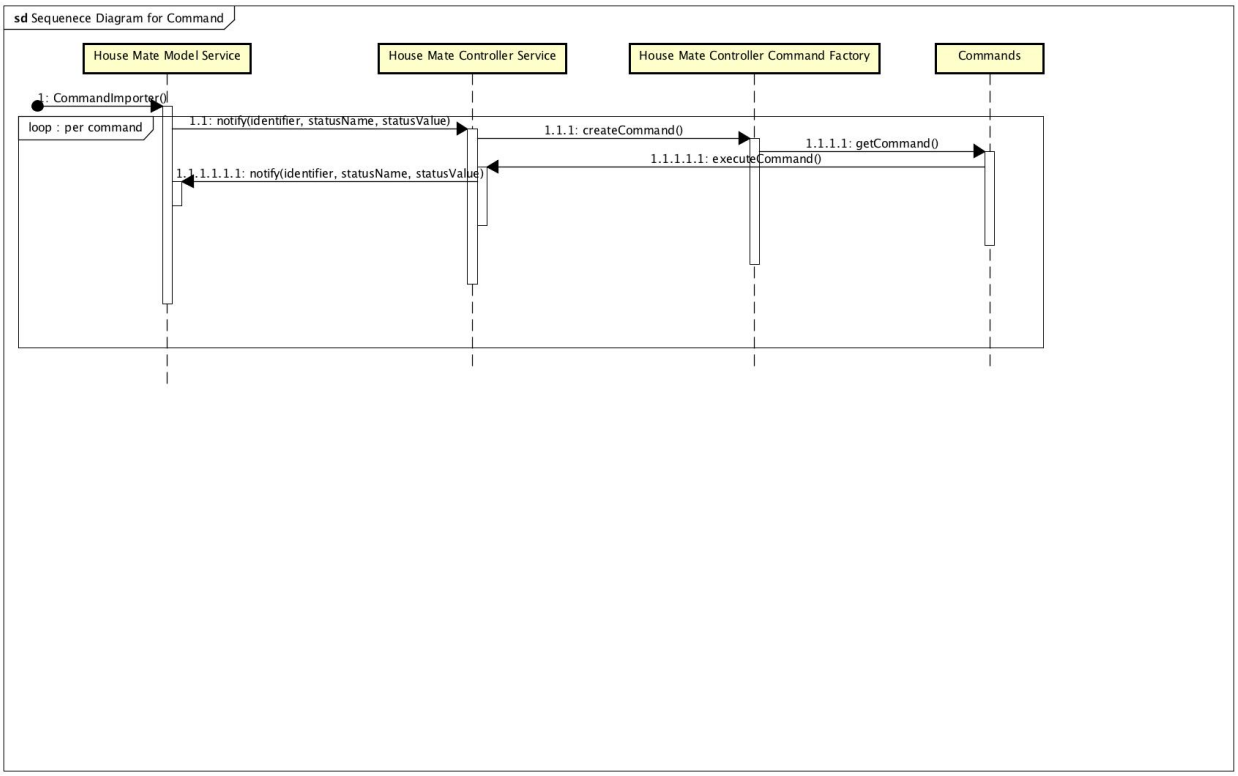
House Mate Controller Service Design Diagram
CSCI E-97 Assignment #3



Implementation

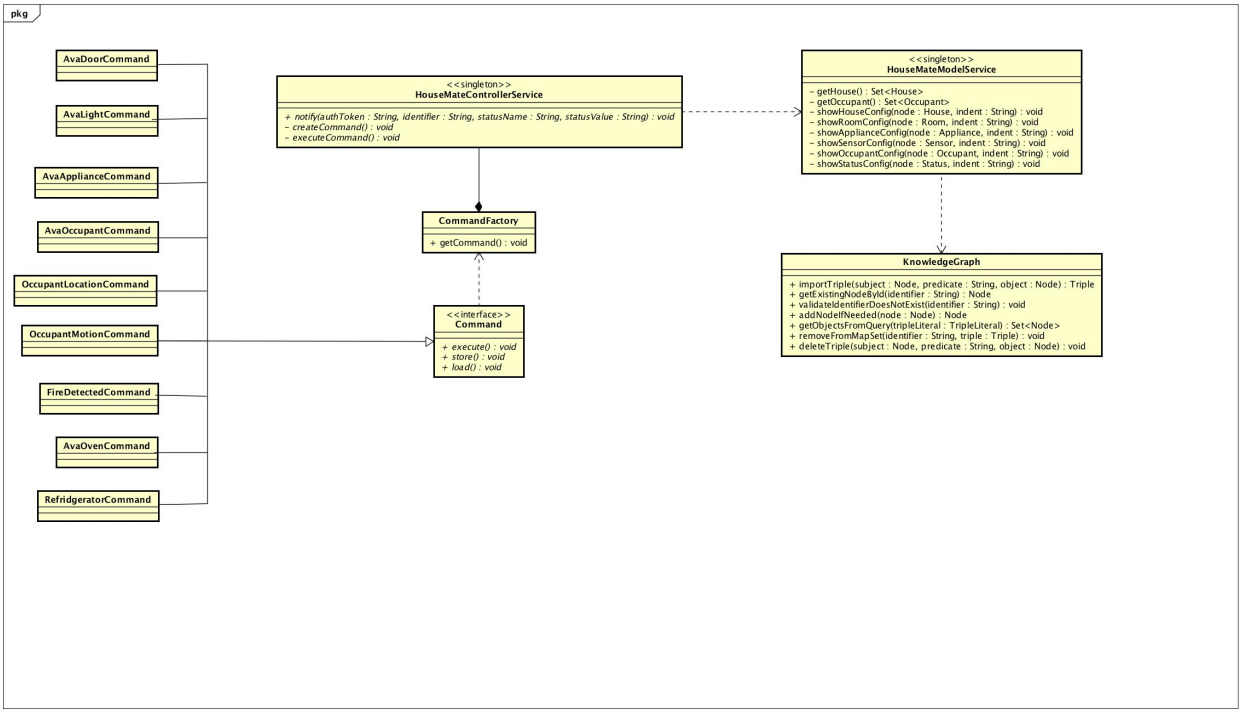
Sequence Diagram

The sequence diagram below steps you through the process from where commands are imported into the Model Service all the way back to when the Controller Service executes the commands that update the Model Service.



Class Diagram

The following class diagram defines the classes defined in this design.



Class Dictionary

*This section specifies the class dictionary for the House Mate Controller Service. The classes defined within the package "*cscie97.asn3.housemate.controller".

HouseMateControllerService <singleton>

The House Mate Control Service is a singleton class that coordinates the execution of commands from the CommandFactory and the implementation within the HouseMateModelService.

Methods

Method Name	Signature	Description
notify	(authToken:String, identifier:String, statusName:String, statusValue:String):void	Public method called by the HMMS when the CommandImporter implements various commands that require state change

createCommand	():void	The factory pattern method for creating our commands
executeCommand	():void	The method that does the actual implementation of our commands on the HouseMateModelService().

CommandFactory

Within this class is the creation of our various commands to be acted upon

Methods

Method Name	**Signature**	**Description**
getCommand	():void	This method interprets the tokenized command from the importer to the notifier which then identifies the appropriate command to execute within our factory of commands.

Command <interface>

This interface defines the structure for our concrete commands as well as storing and loading their execution. The command pattern specifies that we implement an execute method (and any storing or loading therein) that

Methods

Method Name	**Signature**	**Description**
execute	():void	This runs the actions within the concrete command.
store	():void	As the command is executed, this will store a history of their request on disk
load	():void	This gets called to recover executions that were stored

AvaDoorCommand

This concrete class sets the door status to open or closed

Methods

Method Name	Signature	Description
execute	():void	Calls a method that sets the door status to either open or closed.

AvaLightCommand

This concrete class sets the lights on and off

Methods

Method Name	Signature	Description
execute	():void	Calls a method that sets the light status to either on or off.

AvaApplianceCommand

This concrete class is a generic Ava command that gives the status of a device in a given room

Methods

Method Name	Signature	Description
execute	():void	Calls a method that gives the status of a generic IOT for Ava to dictate.

AvaOccupantCommand

This concrete command responds to Ava being asked about occupant location

Methods

Method Name	Signature	Description
execute	():void	Calls a method that gets the whereabouts of and occupant from the KnowledgeGraph.

OccupantMotionCommand

This is a concrete command that detects whether the occupant is either active or inactive within a given room

Methods

Method Name	Signature	Description
execute	():void	Calls a method that updates the occupants status to either active or inactive which dims the lights and updates the KG of the occupant location

OccupantLocationCommand

This is a concrete command that detects whether an occupant enters or leaves a room

Methods

Method Name	Signature	Description
execute	():void	Calls a method that turns on the light in the room and increases the thermostat or turns off the lights and decreases the thermostat depending on if the occupant leaves or enters the room. It also updates the KG of the occupant location

FireDetectedCommand

This is a concrete command that turns on all the lights, opens the window on the first floor and asks all the occupants to leave.

Methods

Method Name	Signature	Description
execute	():void	Calls a method that sets the lights to on, opens the window if it is on the first floor and dictates to Ava for occupants to leave.

AvaOvenCommand

This is a concrete command that turns off the oven and alerts the occupants that food is ready

Methods

Method Name	Signature	Description
execute	():void	Calls a method that sets the oven value to off and Ava alerts the occupants the food is ready

RefridgeratorCommand

This is a concrete command that orders more beer if the refrigerator value is below threshold

Methods

Method Name	Signature	Description
execute	():void	Calls a method that observes beer count. If it falls below threshold, it orders more beer.

Implementation Details

Explain details of the implementation.
Remember to reference the requirements from the body of the design document to show how your design is addressing the requirements.

The implementation of the House Mate Controller Service addresses the management of a pattern of commands by facilitating a CommandFactory that creates commands and allows the Controller to execute the corresponding command based on the tokens that stimulated it. The Knowledge Graph gets used and updated according to the commands that require the whereabouts of the Occupant.

Testing

For testing you would implement a TestDriver class that reads in the housematesetup2.text file (and just about any .txt file for that matter) and should be packaged
See Risks

Risks

Document any risks identified during the design process.
Are there parts of the design that may not work or need to be implemented with special care or additional testing?
There were several risks involved during my design process. For me specifically was the risk involved with trying to use a different House Mate Model than my own after beginning the design of the House Mate Controller. The risk wasn't too high only because it made for a lot of clarification of my House Mate Controller design. Otherwise, it did affect time.

Another risk in the design had to do more with the implementation of executing the commands immediately versus having a system in place for queuing. This is the difference between a single threaded versus multithreaded implementation which is okay for our purposes but certainly not ideal for an actual working command controller with dozens of commands that may need implementation at once.

Unfortunately, I did not leave myself enough time to execute my Controller design against a working House Mate Model but it did give me much more confidence that I could easily implement the controller's required commands.