Housemate Controller Design Document

Date: 3/15/2017

Author: Bry Power

Reviewer(s):

# Introduction

Overview

This document provides the design details for the House Mate Controller Service, which is part of a larger system called The House Mate System. The House Mate system consists of a collection of sensors that can be used to monitor and control appliances throughout the house.

The goal of this system is to enable control and modification over houses, occupants, rooms, appliances and sensors through voice commands. The House Mate Controller Service is responsible for monitoring the state of the sensors and appliances within the home. In addition, the Controller Service is able to generate actions to control the appliances based on rules, in response to status updates from the sensors and appliances.

Requirements

The House Mate Controller Service subscribes to updates from sensors and appliance in the Model Service as a registered observer. When notified of an update the controller is passed command objects by the Observable objects and executes the actions based on rules.Sensor input includes voice commands received via the Ava devices, occupants movements detected by cameras, and smoke being detected by the some detector. Macro command objects are employed to group related commands together based on the rules. The command objects will then use the controller service api to update or query the model system.

# Use Cases

The Housemate Controller supports the use cases of sensors and devices in the house. When a sensor needs to update the state of an object in the house, it uses the controller service.

The actors and their actions include:

1. Ava
   1. Open / close the door – in response to a detected voice command
   2. Turn lights on/ off – in response to a detected voice command
   3. Send a voice alert – speak a message to the occupants
2. Camera
   1. Set occupant status – in response to detecting occupants leaving/ entering the room or falling asleep/ waking up.
3. Smoke Detector
   1. Turn lights on/ off
   2. Call 911
   3. Send an Ava voice alert
4. Oven
   1. Turn itself on/ off – in response to the timer reaching 0
   2. Send an Ava voice alert
5. Refrigerator
   1. Order food from the store
   2. Send an Ava voice alert

The following use case diagram illustrates the use cases supported by the House Mate Controller service:



# Implementation

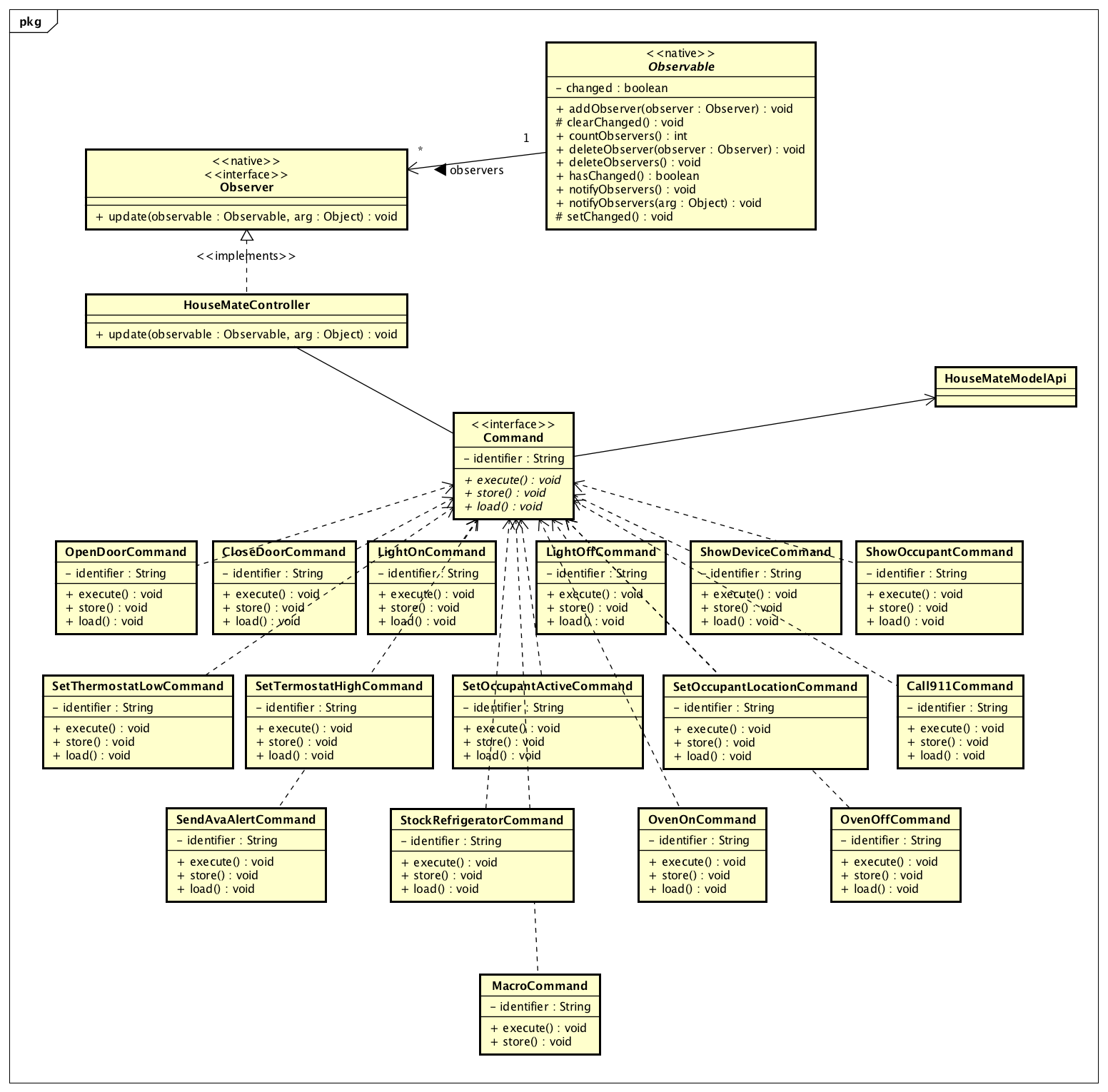
*This section of the document will describe the implementation details for ...*

*The implementation section should cover the following topics:*

* *What are the classes, and their properties, associations and methods?*
* *What are the important interfaces and how they will be implemented?*
* *How are the requirements addressed?*

# Class Diagram

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



# Class Dictionary

*This section specifies the class dictionary for the class … defined within the package …*

## *HouseMateController*

*Implements the Observer interface. Accepts notification from the Model that a change has been requested by an Observable Device. Is passed a command object and executes the action.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| update | (observable : Observable, arg : Object) : void | Public method called by an Observable device and passed the String identifier of the object to perform an action on. |

## *Command*

*Interface for all command objects.*

***Methods***

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| execute | () : void | Public method called by an Observable device and passed the String identifier of the object to perform an action on. |
| store | ():void | As each command is executed, it is stored on disk. |
| load | ():void | In case of a system failure, the commands are loaded and execute in the correct order |

***Properties***

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| identifier | String | The String identifier of the device or occupant to be updated |

***Associations***

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| modelApi | ModelApi | Singleton instance of the house mate model API. Used to update the state of the entity referenced by the identifier. |

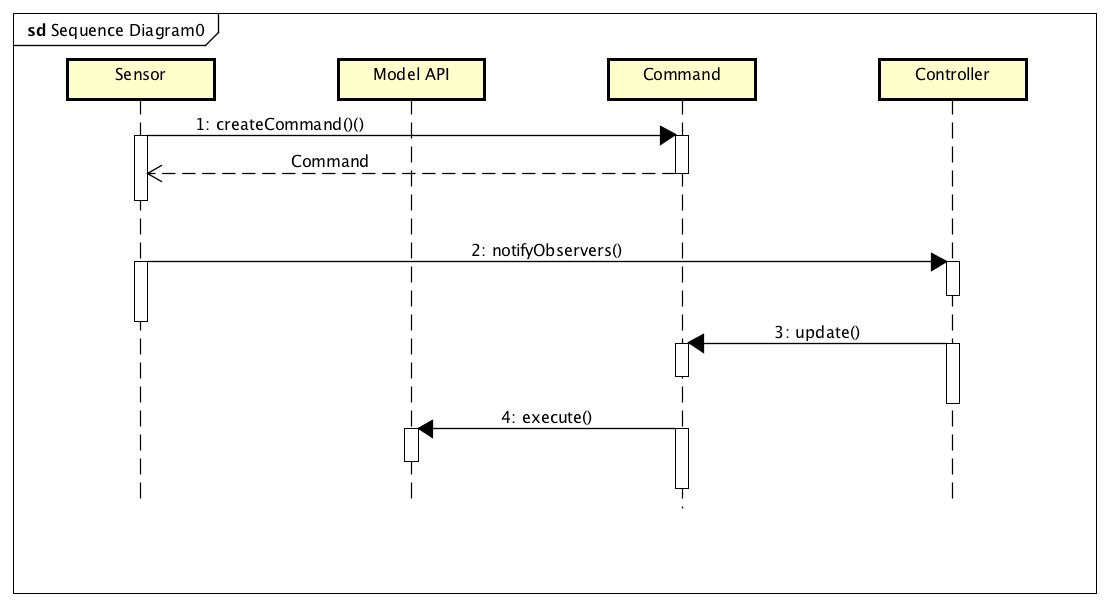
***Classes implementing this interface***

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Identifier** | **Api Method** |
| OpenDoorCommand | Door | setDevice() |
| CloseDoorCommand | Door | setDevice() |
| LightOnCommand | Light | setDevice() |
| LightOffCommand | Light | setDevice() |
| ShowDeviceCommand | Device | showDevice() |
| ShowOccupantCommand | Occupant | showOccupant() |
| SetThermostatLowCommand | Thermostat | setDevice() |
| SetThermostatHighCommand | Thermostat | setDevice() |
| SetOccupantActiveCommand | Occupant | SetOccupant() |
| SetOccupantLocationCommand | Occupant | SetOccupant () |
| SendAvaAlertCommand | Ava | setDevice() |
| Call911Command | Ava | setDevice() |
| OvenOnCommand | Oven | setDevice() |
| OvenOffCommand | Oven | setDevice() |
| StockRefrigeratorCommand | Refrigerator | setDevice() |
| MacroCommand | Various | Various |

# Implementation Details

The HouseMateController implements the Java Observable interface and is subscribed to the devices and sensors as an observer so that when a sensor or device detects a change in the system that should result in an action being initiated, it creates the corresponding Command object and passes it to the all observers using the native Java Observable class method notifyObservers which will include the HouseMateController. In the update method the controller will call execute on the Command execute method. The execute method will use the Model Api to update the system.

The following sequence diagram illustrates the interaction between the House Mate Controller Service and the House Mate Model Service:



# Exception Handling

*Provide details on your exception handling. What types of exceptions are expected and how are they handled by the design? Describe your exception classes and their properties.*

# Testing

*Provide a testing strategy for testing the component.*

* *Functional*
* *Performance*
* *Regression*
* *Exception Handling*

# 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?*