House Mate Controller Service Design Document

Date: October 28, 2015
Author: Kush Patel
Reviewer(s): Sarah Leinicke, Joshua Mcelfresh

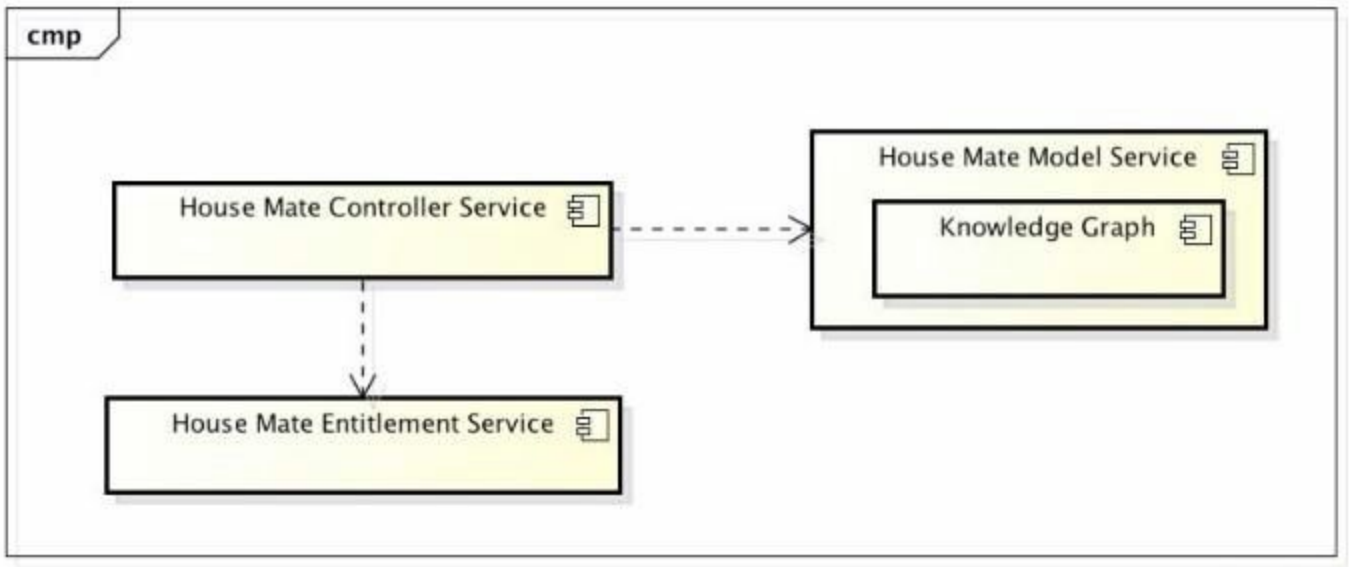
Introduction

This document describes a design and implementation of the House Mate Controller Service (HMCS); a controller service that works in conjunction to the House Mate Model Service (HMMS). Together these systems control and maintain the state of many houses and occupants assisting them in day to day household activities.

Overview

The HMMS is a singleton that merely maintains the state of all the occupants, houses, rooms, and devices. It's a great way to check the state of the system, but not control it. This is where the HMCS comes into play; a system that can interact with different states of the occupants and houses as well as different commands provided to it. Automating things such as turning off the lights as you leave a room or fall asleep or alerting the user that the food in the oven has been cooked. Users can also interact with it by passing commands. For example, a user can say "open window" and the HMCS will determine the room the occupant is in, and open the windows to that room.

The following design shows how the HMCS fits in with the rest of the system.



This is a very high level design overview of the entire House Mate system. The HMCS implements a command pattern with which it interacts with the HMMS (shown below in detailed class diagram).

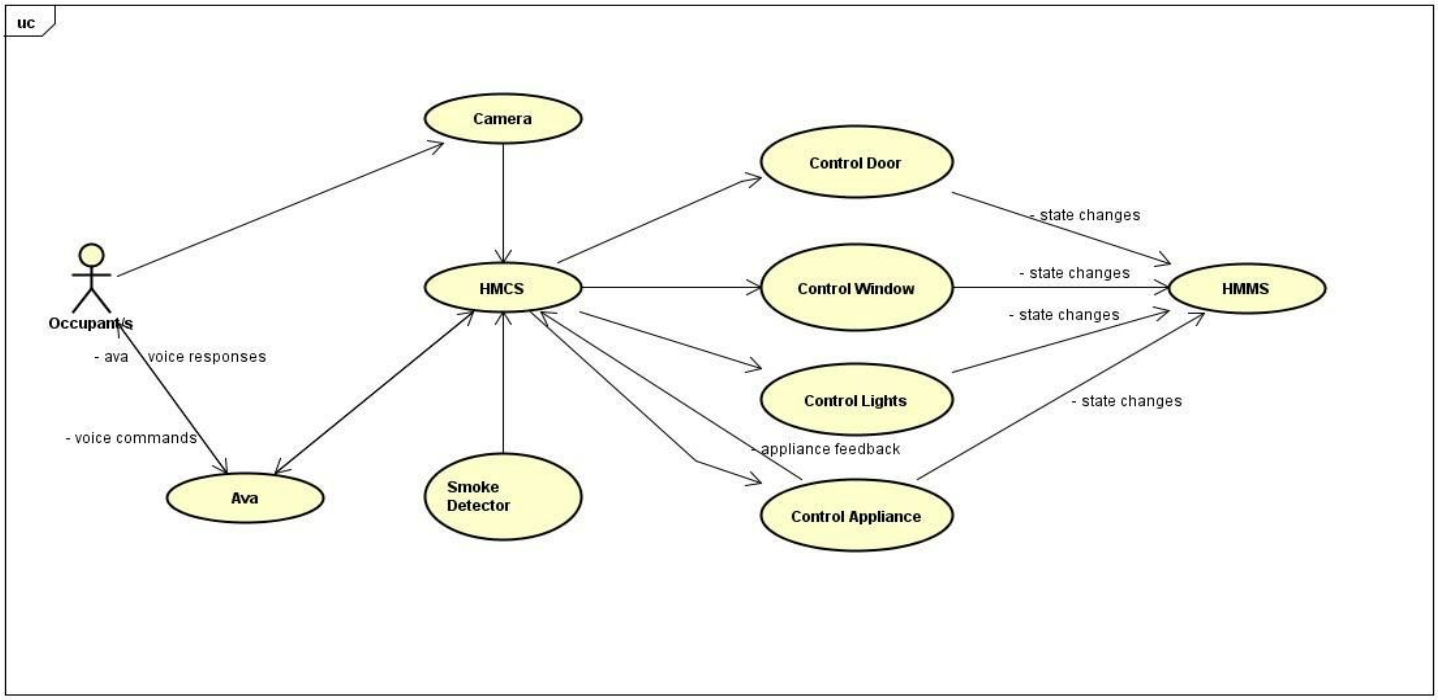
Requirements

The HMCS is responsible for monitoring the sensors and appliances within the home. Upon changes in the state of the sensors or appliances, the HMCS must execute the proper commands that complete the system interaction for the respective state change. For example: if the HMCS is given the command to “open window” the HMCS must notify the HMMS and change the state of the window to open. Specifically, the HMCS must support the following functions:

- Monitor Sensor and Appliances for state changes
    - Respond to the state updates in sensors and appliances and apply rules to generate actions
- Accept voice commands via the Ava appliances and provide voice feedback when necessary
    - Act on given voice commands and control appliances accordingly.

Use Cases

Occupants interact with the HMCS via input appliances, Ava, to control other appliances around the house. Below is a sample flow of the interaction between the actors and the system.



Use Case: Occupant/s

1. Occupant says command “open windows” to Ava
2. HMCS receives command from Ava and calls Control Window
3. Control Window opens window and passes state change to HMMS notifying the windows are open.

Use Case: Sensor (Smoke Dectector)

1. Smoke Detector senses “FIRE” and sends information to HMCS
2. HMCS calls Control Lights
3. Control Lights turns on all Lights and notifies the HMMS for state change
4. HMCS sends voice response via Ava to occupants notifying them about the fire
5. HMCS dials 9-1-1 and contacts emergency services

Implementation

The HMCS is designed with a command pattern; used to invoke the rules/actions based on the stimulus. The HouseMateCommandImporter is the primary interaction with the HMMS, and is now also the primary interaction to the HMCS using the command line interface system provided. Information from the commands provided are parsed and passed onto either HMMS or the HMCS or both system for either system to make the interaction. As stated in the HMMS design, the layout of the house and logged occupant list are stored in this singleton.

Prior to the design of the HMMS, the KnowledgeGraph (KG) was developed; thus the design for the KG was further developed into the HMMS. The design for the HMCS requests using the KG to track the location and status of the occupants. Using the subject, predicate, object design to store: occupant, location/status, value(the location or the current status). Upon implementation of the HMMS, occupants were given these properties in the form of location : Location (composed of house name and room name) and status : String. Based on this deviation from the design, the final implementation of the HMCS obtains status and location from the HMMS.

The HMCS implements the command pattern to invoke the different actions based on the stimulus provided. Each command has both an execute method as well as a log event method designed to do as expected: execute the action required, and log the event upon completion.

To interact with the system via the command importer, follow the below pattern for each command line:

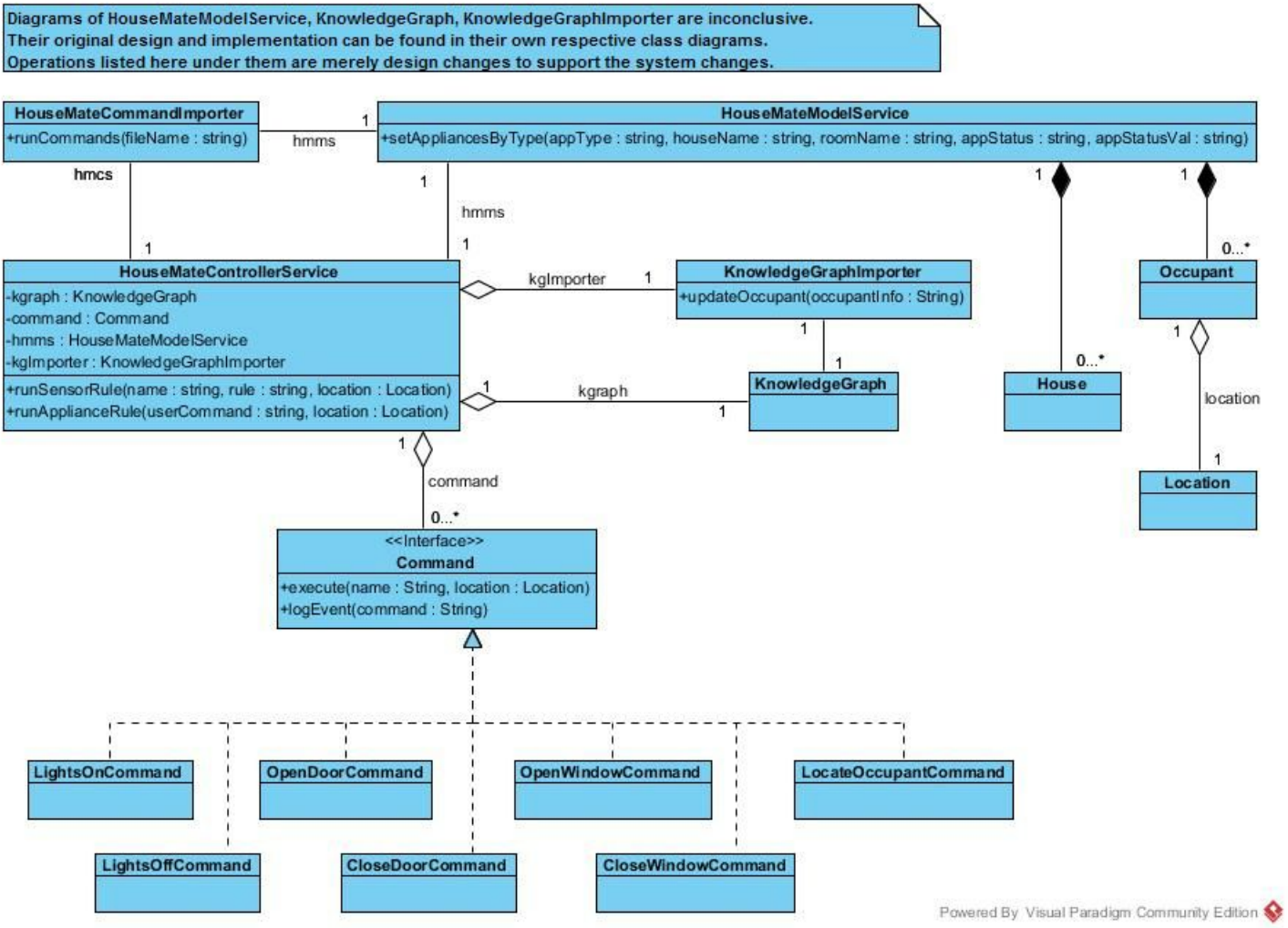
- # for comment
- define for new <object>
    - house, room, occupant, sensor, appliance
- set for setting a <property>
    - occupant, sensor, appliance
- show for displaying the configuration for the <object>
    - appliance, sensor, configuration(house / room)

When the sensor interacts with an occupant by observing them, the command must include one of the following clauses: OCCUPANT_DETECTED, OCCUPANT_LEAVING, OCCUPANT_ACTIVE, OCCUPANT_INACTIVE.
When Ava* interacts with an occupant via voice communication the occupant’s commands must contain common terms such as: where, open, close, on, off, door, window, or light.

* Ava is regarded as an appliance and must follow appliances formatting as it can listen and speak.

Class Diagram

The following class diagram elaborates on the design and implementation of the HMCS. The design also shows the connection to the HMMS and KG however does not elaborate on the design. See prior design documents for HMMS and KG design and implementation.



Class Dictionary

***NOTE: All methods are public.*

HouseMateControllerService

The HMCS is the service that makes the overall House Mate system interactive with the occupants. The command importer simulates the information the HMCS would receive from either a sensor or an appliance and takes the responsible action to assist the occupant in house activities.
A camera sensor can detect an occupant in a room. The sensor would provide information describing itself to the HMCS as well as which occupant has entered the room. The HMCS would respond by turning on the lights in that room, and updating the location of the occupant.
An Ava appliance can receive a command; specifically a query by a secondary occupant in the house. Asking for the location for the first occupant. Because the HMCS tracks the location of the occupant, it can respond with the location of the occupant. Also, if the occupant leaves and enters a different home which is also monitored by the HMCS, then the Ava sensor can also respond with that house's name and room. Thus letting a mother know that her son is at her neighbor's house in the living room.

Methods

Method Name	Signature	Description
runSensorRule	(name : string, rule : string, location : Location) : void	Takes information prased from the command importer and invoke the correct command based on the inputs from the sensor.
runApplianceRule	(userCommand : string, location : Location) : void	Takes information prased from the command importer and invoke the correct command based on the inputs from the appliance.

Properties

Property Name	Type	Description
kgraph	KnowledgeGraph	Used to store the location and status of the occupants (later during implementation, the KG is no longer queried for the information however it is still updated for future use)
kgImporter	KnowledgeGraphImporter	Used to update occupant information in KG
hmms	HouseMateModelService	Stores all the data for the House Mate system including occupants and configurations for the house, rooms, sensors, appliances

command	Command	Interface defined individually during the execution of the commands once the rules which need to be applied are determined by the HMCS

HouseMateCommandImporter

The HouseMateCommandImporter imports all the commands provided in the given filename. It parses each command extracting data from it and then invokes a call to either the HMMS or HMCS or both based on the required change/action that must take place. Commands follow a certain pattern and without following the pattern described above under Implementation, commands can result in an error. However, a single errored command will not error out the run of the entire file; it will simply print out when an error occurred and what caused the error.

Methods

Method Name	Signature	Description
runCommands	(fileName : String)	Reads in all the lines from the filename provided; ignores lines with a "#" as they can be comments for a user reading the command list. Then runs each command step by step, so "show" commands should be at the end once all the configuration has been loaded.

HouseMateModelService

The HMMS is a singleton class that gets initialized once the application starts which is important since its storing all the data and configuration for the whole House Mate system and so data isn't accidentally set in a different new HMMS. The detailed design and implementation of all the existing methods properties and associations can be found in the HMMS design document.

Methods

Method Name	Signature	Description
setApplianceByType	(appType : string, houseName : string, roomName : string, appStatus : string, appStatusVal : string) : void	Method to set a status value for all appliance of the same configuration in a given location. Commonly used to turn on all lights, or open all windows in a room.

KnowledgeGraphImporter

This class is responsible for loading data into the KG. Per the design's request to store the location and statuses of the occupants in the KG, the following changes were made to this class for such functionality. Upon completing the development of the HMCS, the KG required too much refactor to store this information and

because the HMMS already contained existing functionality, the design was modified to use them instead. However, the following methods were still added to the system during design and implementation.

Methods

Method Name	Signature	Description
updateOccupant	(occupantInfo : String) : void	An occupant can be added or updated in the KG via this method. The format for occupant info is as such: <name> location <housename:roomname> or <name> status <active/inactive>

Command

Command is an interface; a structure to build all the variable commands from. Each implementation of the command is unique and will execute different actions in the system. But implementing the command pattern, allows the HMCS to only initiate a single command and and only use a single implementation as it completed the action required.

Methods

Method Name	Signature	Description
execute	(name : String, location : Location) void	All commands may or may not use both a name and a location depending on the action they execute, but this method definition provides any possible information needed. Null is acceptable if the command is not expected to use that argument.
logEvent	(command : String) void	All implementations of the command is unique and require different log messages. For user reference, the log will also output the specific command that was used to invoke the action.

Realizations

Realization Name	Description
LightsOnCommand	This command uses location (house name and room name) and sets all the appliances with type: “light” as on
LightsOffCommand	This command uses location (house name and room name) and sets all the appliances with type: “light” as off
OpenDoorCommand	This command uses location (house name and room name)

	and sets all the appliances with type: “door” as open
CloseDoorCommand	This command uses location (house name and room name) and sets all the appliances with type: “door” as closed
OpenWindowCommand	This command uses location (house name and room name) and sets all the appliances with type: “window” as open
CloseWindowCommand	This command uses location (house name and room name) and sets all the appliances with type: “window” as closed
LocateOccupantCommand	This command searches for an occupant with the given “name” and responds with the last known location.

Implementation Details

This project requires the implementation of a service used to configure and manage an instance of a House Mate. A system that automates daily house activities as well as providing real time data / status updates when needed. The HMMSController is the command parser; any errors with the commands or the files are handled here; otherwise it calls the appropriate method in the HMMS to complete the action required for the command. The HMMS is a single entity, aka singleton that stores all the data for this current House Mate and prints out any information that has been requested. Once the application is initialized, the singleton is initialized and keeps that static state for the remainder of the application. The HMMS supports a Command Line Interface for configuring the houses, rooms, sensors, appliances, and occupants, as well as printing data / configurations for each of them. Essentially each object is just a data storage item, and the HMMS configures the correlation between them, and then stores the highest level objects (house and occupants) in itself.

Testing

Functional Testing:
Two different test files were loaded into the system and used to test different feature sets of the HMCS design.

- **housesetup.txt**
  - This is the given file for completion testing to validate the system meeting the requirements.
  - the output file is: houseSetupOutput.txt
- **houseTestOpenClose.txt**
  - This file was created with different interactions with the HMCS. The setup involves setting up a house with rooms, doors, windows, lights, cameras, and Ava sensors. The HMCS should accept commands from a user.
  - Actor joe_smith was set to open doors, open windows, and turn on lights as he moved from room to room.
  - Also, camera sensors should detect joe_smith leaving a room and entering a new one and thus automate the process of turning on the lights which is also tested.
  - After testing these states, the test is setup to test a final interaction of another occupant entering the house. Jill arrives home from work, and as she enters the house into the dining room, the lights are enabled and her location is set.
  - Joe hears jill enter the home, and asks Ava where she is, testing that jill's location has updated her to be in the dining room.
  - Jill doesn't see joe anywhere on floor1, so she asks Ava the location of joe. the system tracked joe entering the bed_room1 and so notifies jill he is in bed.

Performance Testing: This application is very small scale, and even though the test files used are larger than previous test files, no noticeable delay has presented.

Exception Handling: These test files were run before completion of the HMCS and as components are completed, the correct interaction results. But until the develop was complete, valid exceptions were thrown which validated the exceptions are accurate as well as assisted in the development of the HMCS as well as bug finding.

Risks

This HMCS has not been develop to integrate with all sensors and appliances currently. Though the requirements requested implementations for fire detection as well as appliance completion (timer ending for oven), these were not a chosen into this implementation. The users may try to interact with such sensors and appliances without knowledge of the limitations of the HMCS.
Also, as mentioned in the design of the HMMS, the HMCS also run completely in memory with instances of the HMMS. So as the system gets larger, manages more houses with complex IOTs and floor plans, the system could overload the allocated memory for this application.