House Mate Model Service Design Document

Date: October 7, 2015

Author: Kush Patel

Reviewer(s): Carolina Nobre & Benjamin Trey

# Introduction

The House Mate Model Service (HMMS) is an evolved system from the Knowledge Graph implementing the singleton design and storing data. The structure of this design provides a simple yet effective way to communicate data via commands.

These commands are the driving component to the HMMS which would allow homes to be automated for each individual user interacting with them. This system would allow all registered occupants in a home to control different aspects of their environment as well as get information. Examples such as preheating the oven to 350 for brownies, or determining which smoke detector is beeping really drive the need for a system like this in every home.

Overview

*Overview of the problem to be solved. What is the problem and why is it being solved? How will the resulting solution provide business value?*

*Consider adding a diagram that explains how this component fits into the overall System with some descriptive text explaining the diagram.*

# Requirements

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

*Provide your understanding of the requirements, both functional and nonfunctional. Reference the provided Requirements and System Architecture documents. Don’t cut and paste from the requirements document.*

*Product Manager and others can read this to understand what requirements your design will support. There is already a requirements doc, so keep this brief and to the point, highlighting the important requirements that the design is addressing. Structure in a way to provide a requirements checklist for your design.*

# Use Cases

*Enumerate the use cases supported by the design,*

*This design supports the following use cases:*

*Include a Use Case Diagram.*

# Implementation

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

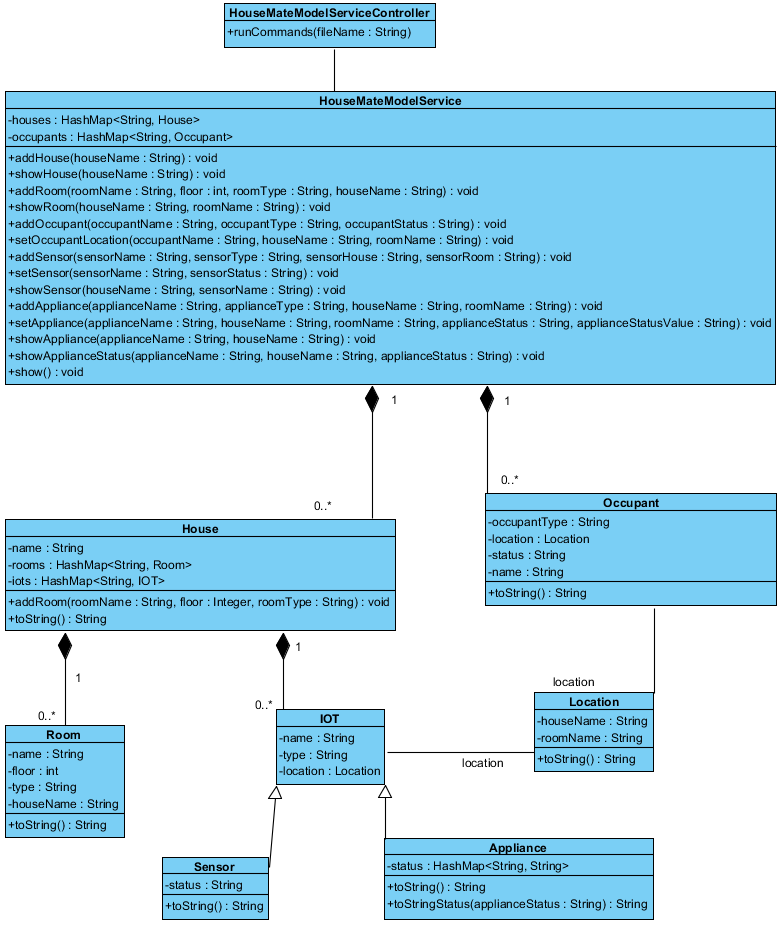
*The implementation section should cover the following topics:*

* *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 House Mate Model Service implantation and all classes used for this solution and their affiliations to the overall system.



# Class Dictionary

*\*\*NOTE: All methods are public.*

**HouseMateModelServiceController**

The HMMSController is responsible for reading each individual command and parsing them; then calling the appropriate method in the HMMS to either input the configuration, or print out information. The key to the runCommand feature is the format of the input; given that these would be controlled commands via an API, the format for each line should always be consitant; thus the command parser does error out due to poor input format.

**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.

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| addHouse | (houseName : String) : void |  |
| showHouse | (houseName : String) : void |  |
| addRoom | (roomName : String, floor : int, roomType : String, houseName : String) : void |  |
| showRoom | (houseName : String, roomName : String) : void |  |
| addOccupant | (occupantName : String, occupantType : String, occupantStatus : String) : void |  |
| setOccupantLocation | (occupantName : String, houseName : String, roomName : String) : void |  |
| addSensor | (sensorName : String, sensorType : String, sensorHouse : String, sensorRoom : String) : void |  |
| setSensor | (sensorName : String, sensorStatus : String) : void |  |
| showSensor | (houseName : String, sensorName : String) : void |  |
| addAppliance | (applianceName : String, applianceType : String, houseName : String, roomName : String) : void | Add an appliance to a |
| setAppliance | (applianceName : String, houseName : String, roomName : String, applianceStatus : String, applianceStatusValue : String) : void | Configures an appliance: either update existing values or adding new status such as: temperature =400. Or add new one: ovenclean=needed |
| showAppliance | (applianceName : String, houseName : String) : void | Returns the appliance and all the status and values for it. |
| showApplianceStatus | (applianceName : String, houseName : String, applianceStatus : String) : void | Returns the specific value for the status request on the specific appliance. |
| show | (): void | Shows the configuration for all the houses and everything in the system |

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| houses | HashMap<String, House> | Map of all the houses registered and configured in the system |
| occupants | HashMap<String, Occupant> | Map of all the occupants in the system, don’t necessarily have to be affiliated with a house, but are registered and configured. |

**House**

Class 1 description

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| addRoom | (roomName : String, floor : Integer, roomType : String) : void | Method to add rooms to a house. Since each room is specific to a house, and can’t move from house to house. |
| toString | ():String | Custom toString to print information when “show” is called |

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| Name | String | Unique name of the house |
| Rooms | HashMap<String, Room> | Map containing all the rooms in a house; each with a unique name to distinguish between similar rooms |
| IOTs | HashMap<String, IoT> | Map containing all the IoTs in a house; each with a unique name to distinguish between similar devices |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| Houses | HashMap<String, House> | HMMS has many houses in its system each with a unique name to identify that house and then all the configurations affiliated for that house. |

**Occupant**

Occupant is any person / animal that is registered in the HMMS system. They do not necessarily have to be residents of the home, but can be anyone whose voice and facial recognition is in the system. This way the system can track all occupants in and out of a house, and identify any suspicious characters.

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| toString | ():String | Custom toString to print information when “show” is called |

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| Name | String | The unique name of the occupant |
| OccupantType | String | Adult, child, animal |
| Location | String | Not required, but gets set as the occupant enters a house and moves around the house, their location gets updated. |
| Status | String | Active / Sleeping (could be many more statuses such as “injured”) thus tracking specific states of a person as well |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| Occupants | HashMap<String, Occupant> | Many people can be registered in the HMMS, whether they are residents of the house or friends or neighbors or anyone. This way the system can track all users it recognizes. |

**Room**

The room class is used to model a room in the house. It is used part of location to identify where occupants and/or IoTs are located within a house.

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| toString | ():String | Custom toString to print information when “show” is called |

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| name | String | Unique identifier of the room. |
| floor | Integer | The level of which this room is located in the house. |
| type | String | The kind of room, help user to identify bathroom at a neighbor’s house maybe. |
| houseName | String | The house in which this room is located. |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| Rooms | HashMap<String, Room> | A house has many unique rooms. Possible to have multiple bedrooms, but they are still unique. |

**IOT**

The generic class of al interactive devices in a house that can provide information about the house, or complete actions such as: starting laundry, prepping oven, launch Roomba to vacuum.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| name | String | The unique name of the device |
| type | String | The type of the device |
| location | Location | The location(house + room) of the device |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| House | HashMap<String, IOT> | A house a set list of devices, though there might be multiple similar devices like many smoke detectors, they all must have a unique identifier to say which one has error. |

**Location**

Not mentioned as part of the original design, but definitely a handy tool if generic locations want to be used in the future. Comprised of a houseName and roomName and so if a user may want to turn on the lights, but doesn’t know which lights are in this room, they could potentially use their location and match with all lights with the same “location” and turn them on.

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| toString | ():String | Custom toString to print information when “show” is called |

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| houseName | String | The house part of the location (since an occupant could be your neighbor looking to turn on your lights.) |
| roomName | String | The room in the house. |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| IoT | Location | IoTs are all installed in specific locations |
| Occupant | Location | Occupants have a specific location they are |

**Sensor**

Sensors are a subset of IoT devices that capture and share data about the house. Each sensor has a name, type, location, and status. Example: a smoke\_alarm could have status: OK or FIRE or Battery\_Low. Whatever the status, communicating with the HMMS, the show sensor command will give you the status.

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| toString | ():String | Custom toString to print information when “show” is called |

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| status | String | The state in which the sensor is in. |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| IOT | Subclass | IOT are the overarching devices that the house mate system controls / communicates with. Sensors are a subset or (subclass) and so they are an extension on it. |

**Appliance**

Appliances are also a subset of IoT devices but these are about to be controlled. These can have more elaborate states and values for their states. Example: if the oven is on, it would have a status: temperature with a value: 350.

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| toString | ():String | Custom toString to print information when “show” is called |

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| status | HashMap<String, String> | The map that stores a status and the value for the status. |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
|  | Subclass | IOT are the overarching devices that the house mate system controls / communicates with. Sensors are a subset or (subclass) and so they are an extension on it. |

# Implementation Details

*Explain details of the implementation.*

*How do the various parts fit together or interact?*

*For example:*

* *How are Feature instances created and managed?*
* *How is the management interface implemented?*
* *How are the features used to generated functional code?*

*Some implementation details may be addressed in the class dictionary, but for things that are not, describe them here.*

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

# 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?*