House Mate Model Service

Date: Oct 3rd, 2017

Author: Stephen Akaeze

Reviewers(s): Chris Auclair and Jaehyon Rhee

INTRODUCTION

This document provides the design approach and implementation details for the House Mate Model Service(HMMS).

OVERVIEW

The HMMS is a digital representation of any physical building(s) and its occupants. Any HMMS building can comprise the following elements:

* Occupant
* House
* Room
* Sensors
* Appliances

The HMMS is responsible for configuring and manipulating the above mentioned digital elements/models.

REQUIREMENTS

**House**

The “House” element is a digital model of a physical house and comprises the following properties:

* globally unique identifier
* Address (street, city, state)
* zero or more occupants
* one or more rooms
* zero or more IOT Devices

**Room**

The “Room” element is a digital model of a room and comprises the following attributes

* Type of Room (Kitchen, Closet, etc.)
* Floor of the house that the room is on
* Unique name of the room within the scope of the house
* Number of windows

**Occupant**

Occupant represents a person or animal. The HMMS can track and locate all occupant(s) locations and status. Persons can be either Adults or Children. Animals are usually pets. Occupants can be known (family member or friend) or unknown (e.g. guest or burglar). All occupants have a name for reference. Occupants also have a status, either active or sleeping.

***Note that the same occupant can be recognized by more than one house.***

**Sensor**

Sensors are IoT devices and capture and share data about the conditions within the house. Examples of Sensors include:

● Smoke Detector

● Camera: monitors location of occupants

Each sensor records data specific to its type. The data recorded by the sensor is automatically sent to the House Mate System. Each sensor has a unique identifier. Sensors are also located within a room of the house. In summary Sensors have the following features.

● unique identifier

● state

● room, location within the house

● sensor type

**Appliance**

An Appliance is similar to a Sensor since it is able to record and share data about itself or its surroundings. An Appliance differs from a Sensor since it can be also be controlled. Examples of Appliances include:

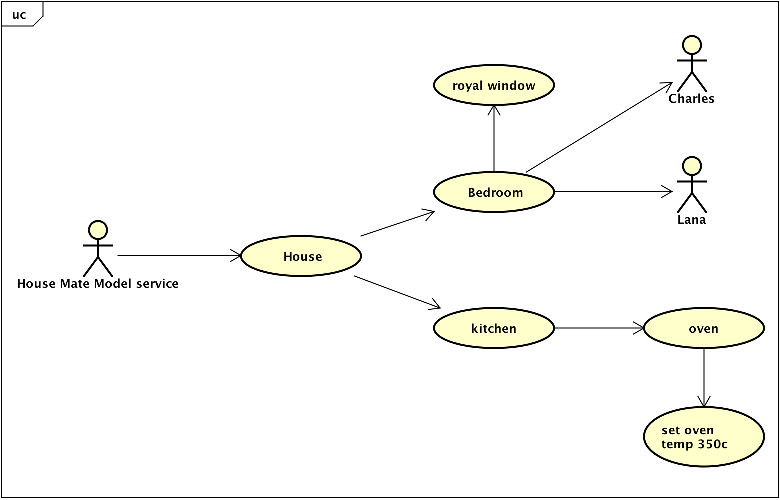
● Thermostat (adjust room temperature)

● Window (open, close)

● Door (open, close, lock)

USE CASES

The following use case diagram documents the use case examples of the HMMS.

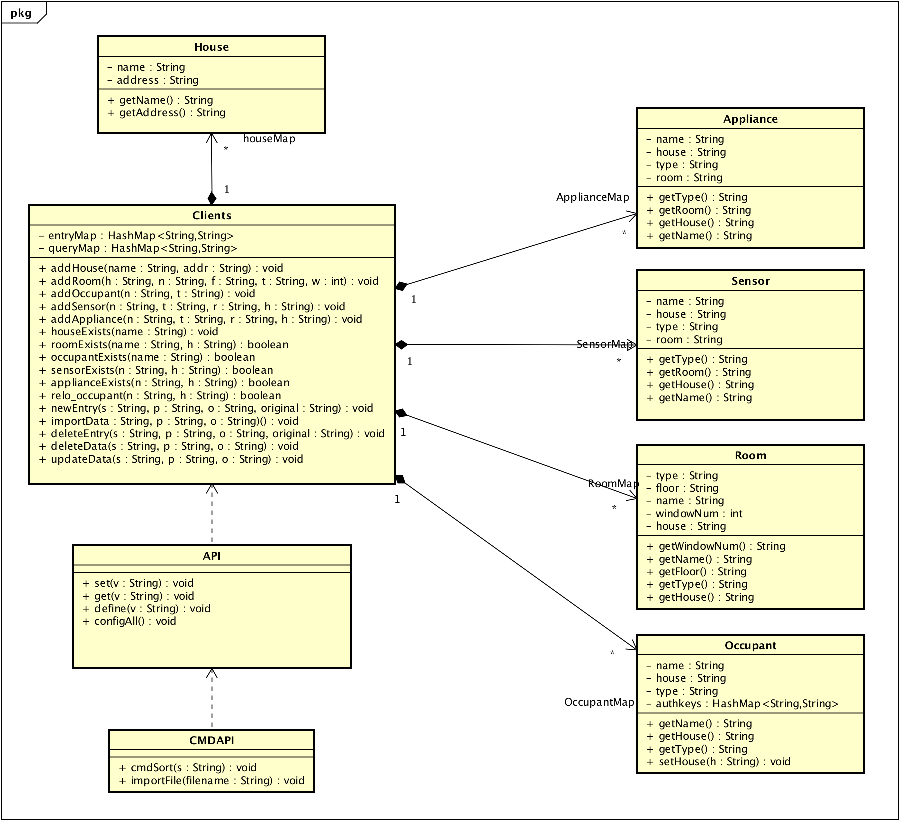


As shown above, the HMMS is able to create a digital representation of a house with two rooms (bedroom and kitchen), 1 window (royal window), 1 appliance (oven) and two occupants in the bedroom. As shown, the HMMS can also control the temperature of the oven. The HMMS is constantly monitoring the locations of Lana and Charles. The HMMS’s potential real-life applications can include home automation systems that than remotely manage and configure several aspects of a physical home.

IMPLEMENTATION

**Class Diagram**

The following class diagram defines the House Mate Model Service classes contained within the package “cscie97.asn2.housemate.model”.



CLASS DICTIONARY

This section specifies the class dictionary for the House Mate Model Service. These classes should be defined within “cscie97.asn2.housemate.model”.

**CMDAPI**

The CMDAPI class

* read HMMS CLI commands from a specified file
* the CLI command to the appropriate API command

***Methods***

|  |  |  |
| --- | --- | --- |
| Method name | Signature | Description |
| importFile | (filename: String): void | Public method for reading CLI command lines from filename. |
| cmdSort | (s: String): void | Public method for evaluating the CLI commands and calling the applicable API class command to execute commands. |

**API**

The API class contains methods that perform the following

* Defining or configuring a HMMS class instance.
* Setting the status or value of a HMMS class instance.
* Displaying the status or value of HMMS class instance.

***Methods***

|  |  |  |
| --- | --- | --- |
| Method name | Signature | Description |
| define | (v: String): void | Public method for creating a HMMS element class instance based on command “v” |
| set | (v: String): void | Public method for setting the status or value of an existing appliance/occupant element class instance based on command “v” |
| get | (v: String): void | Public method for displaying the status or value of an existing HMMS element class instance based on command “v” |
| configAll | ():void | Public method for called by get() to display the configuration of all existing House instances |

**Clients**

The Clients class is the container for all created home, room, occupant, appliance and sensor instances. It also serves as a knowledge graph to store sensor/appliance/occupant status/values. It also implements a singleton design and its instance is called by the getInstance() cmd. To improve performance, an additional queryMap association is created to contain all possible queries for all previous entries to the knowledge graph. This enables O(1) performance due to the efficiency of the HashMap.

***Methods***

|  |  |  |
| --- | --- | --- |
| Method name | Signature | Description |
| addHouse | (name : String, addr : String) : void | Public method that adds new house object to houseMap |
| addRoom | (h : String, n : String, f : String, t : String, w : int) : void | Public method that adds new room object to roomMap |
| addOccupant | (n : String, t : String) : void | Public method that adds new occupant object to occupantMap |
| addSensor | (n : String, t : String, r : String, h : String) : void | Public method  that adds new sensor object to sensorMap |
| addAppliance | (n : String, t : String, r : String, h : String) : void | Public method that adds new appliance object to applianceMap |
| houseExists | (name : String) : void | Public method that checks if house already exists in houseMap |
| roomExists | (name : String, h : String) : boolean | Public method that checks if room already exists in roomMap |
| occupantExists | (name : String) : boolean | Public method that checks if occupant already exists in occupantMap |
| sensorExists | (n : String, h : String) : boolean | Public method that checks if sensor already exists in sensorMap |
| applianceExists | (n : String, h : String) : boolean | Public method that checks if appliance already exists in applianceMap |
| relo\_occupant | (n : String, h : String) : boolean | Public methods that checks if occupant exists in occupantMap and relocates the existing occupant to a house |
| newEntry | (s : String, p : String, o : String, original : String) : void | Public method that updates the queryMap with a new entry |
| importData | (s : String, p : String, o : String) : void | Public method that updates entryMap with key "s+p+?" mapped to String "s+p+o"  . It also updates queryMap with all possible "?" and s, p and o combinations. |
| deleteEntry | (s : String, p : String, o : String, original : String) : void | Public method that deletes an existing queryMap entry. It deletes "original" from the SET mapped by s+p+o |
| deleteData | (s : String, p : String, o : String) : void | Public method that deletes all existing queryMap entries of any String s+p+o |
| updateData | (s : String, p : String, o : String) : void | This updates queryMap with new s+p+o values |

***Associations***

|  |  |  |
| --- | --- | --- |
| Association Name | Type | Description |
| houseMap | Map<String,String> | Public association containing all HMMS house instances |
| roomMap | Map<String,String> | Public association containing all HMMS room instances |
| sensorMap | Map<String,String> | Public association containing all HMMS sensor instances |
| applianceMap | Map<String,String> | Public association containing all HMMS appliance instances |
| occupantMap | Map<String,String> | Public association containing all HMMS occupant instances |
| entryMap | Map<String,String> | Public association containing all previous Clients.updateData() arguments |
| queryMap | Map<String,Set<String>> | Public association containing queries pointing to all previous Clients.updateData() arguments |

**House**

The House class represents instances of houses. Each House has a unique name and address.

***Methods***

|  |  |  |
| --- | --- | --- |
| Method name | Signature | Description |
| getName | (): String | returns the name of the any House Object |
| getAddress | (): String | returns the name of the any House Address |

***Properties***

|  |  |  |
| --- | --- | --- |
| Property Name | Type | Description |
| name | String | House instance Unique identifier |
| address | String | House instance address |

**Room**

The Room Class represents instances of rooms. Each room has a unique name, type, floor, house and windows.

***Methods***

|  |  |  |
| --- | --- | --- |
| Method name | Signature | Description |
| getName | (): String | returns the room's unique identifier |
| getHouse | (): String | returns the name of the House where the room is located |
| getFloor | (): String | returns the room's floor |
| getType | (): String | returns the room's type |
| getWindowsNum | (): String | returns the room's window number |

***Properties***

|  |  |  |
| --- | --- | --- |
| Property Name | Type | Description |
| name | String | Private unique identifier for room instances |
| house | String | Private name of house containing room instance |
| floor | String | Private floor in house where room instance is located |
| type | String | Private type of room instance |
| windowNum | int | Private number of windows in room instance |

**Sensor**

The Sensor Class represents instances of sensors. Each sensor has a unique name, house, room and type.

***Methods***

|  |  |  |
| --- | --- | --- |
| Method name | Signature | Description |
| getName | (): String | returns the sensor's unique identifier |
| getHouse | (): String | returns the name of the House where the sensor is located |
| getRoom | (): String | returns the sensor's room |
| getType | (): String | returns the sensor's type |

***Properties***

|  |  |  |
| --- | --- | --- |
| Property Name | Type | Description |
| name | String | Private unique identifier for sensor instances |
| house | String | Private name of house containing sensor instance |
| floor | String | Private floor in house where sensor instance is located |
| type | String | Private type of sensor instance |

**Appliance**

The Appliance Class represents instances of appliances. Each appliance has a unique name, house, room and type.

***Methods***

|  |  |  |
| --- | --- | --- |
| Method name | Signature | Description |
| getName | (): String | returns the appliance's unique identifier |
| getHouse | (): String | returns the name of the House where the appliance is located |
| getRoom | (): String | returns the appliance's room |
| getType | (): String | returns the appliance's type |

***Properties***

|  |  |  |
| --- | --- | --- |
| Property Name | Type | Description |
| name | String | Private unique identifier for appliance instances |
| house | String | Private name of house containing appliance instance |
| floor | String | Private floor in house where appliance instance is located |
| type | String | Private type of appliance instance |

**Occupant**

The Occupant Class represents instances of occupants. Each occupant has a unique name, house and type.

***Methods***

|  |  |  |
| --- | --- | --- |
| Method name | Signature | Description |
| getName | (): String | returns the occupant's unique identifier |
| getHouse | (): String | returns the name of the House where the occupant is located |
| getType | (): String | returns the occupant's type |
| setHouse | (h: Strings): void | Sets the occupant house to h |

***Properties***

|  |  |  |
| --- | --- | --- |
| Property Name | Type | Description |
| name | String | Private unique identifier for occupant instances |
| house | String | Private name of house containing occupant instance |
| type | String | Private type of occupant instance |
| authKeys | HashMap<String,String> | Private map of house authentication keys belonging to an occupant |

IMPLEMENTATION DETAILS

In addition to the above Class dictionary and Class diagram

* CMDAPI class reads a file, extracts command lines and sends the command line to API
* API class accepts a command line, deciphers the command line and calls the appropriate Clients class function to execute the function.
* Clients Class contains all instances of house, room, occupant, sensor and appliance classes as well as the methods for manipulating these instances.

The implementation meets the requirements of the House Mate Model Service because the design is able to

* create all instances of House, Room, Occupant, Sensor and Appliance classes.
* set or change an appliance/sensor/occupant status using a knowledge graph
* display/return details of all existing instance configurations, status or values.
* It checks for accurate command syntax.

EXCEPTION HANDLING

There are two primary Exception handlers that are implemented in HMMS design. They are as follows:

**ImportException:** is thrown during the following situations

* Sample input file does not exist

**CommandException:** is thrown during the following situations

* Invalid command syntax
* Creating any instance that already exists.
* Setting the value of a non-existent instance.
* Getting the value of a non-existent instance.

TESTING

Implement a test driver called TestDriver that implements a static main() method. The main() method should accept 1 parameters, an input sample input file. The main method will call the CMDAPI.importFile() method, passing sample input file. The importFile() method invokes the cmdSort() method which identifies the command type and continues the execution of sample input file commands. The TestDriver class should be defined in the package “cscie97.asn2.test”.

RISKS

Because of the in memory implementation, the number of house, room, occupant, sensor and appliance instances are limited by the memory allocated to the JVM.