**House Mate Controller Service Design Document**

Date: 24th Oct, 2017

Author: Stephen Akaeze

Reviewers: Ritong Chen

Introduction

This document describes the high-level design approach and implementation of the House Mate Controller Service(HMCS) and the problems that it solves. This is accomplished using class diagrams, sequence diagrams, use cases, class descriptions etc. to provide a UML view of the HMCS.

Overview

The House Mate Controller Service(HMCS) augments the House Mate Model Service(HMMS). The HMMS creates digital representations of Houses and their occupants. By default, Houses (comprising appliances, sensors, rooms) and occupants created by HMMS simulate a “basic” living space. By basic, the buildings do not implement IoT. IoTs define smart homes as they can accomplish the following,

* Improve the occupants’ safety
* Improve the occupants’ comfort
* Monitors occupant status and location
* Restock home resources
* Execute real time commands from occupants etc.

HMCS is the central mind that attaches the IoT intelligence and features to the Houses and elements created by the HMMS.

Requirements

This section defines the requirements for the House Mate Controller Service.

The House Mate Controller Service should support the following functions:

* Monitor Sensor and Appliances for status updates.
* Apply rules that respond to the status updates from sensors and appliances and generate actions.
* Sensor input includes voice commands received via the Ava devices. Note that Ava devices are now considered appliances since they can provide voice feedback to occupants.
* In response to actions, generate and send control messages to Appliances.

The House Mate Controller Service should use the interface of the House Mate Model Service

to monitor the status of each of the IOT devices installed within the houses. In response to

inputs, the Controller Service will use rules to invoke actions. The actions will be executed

through the appliance controls.

All rule execution and resulting actions should be logged.

Additional requirements

* The HMCS should implement command and observer pattern between the HMMS and HMCS
* Use the HMMS knowledge graph to track the location and status of occupants
* All rules execution and resulting actions should be logged

Sensor, Stimulus, Rule, Action

The following Stimulus table defines the behavior for the HMCS. The HMCS will monitor all sensors and appliances for each of the houses and rooms. For each stimulus, apply the appropriate rule and action.

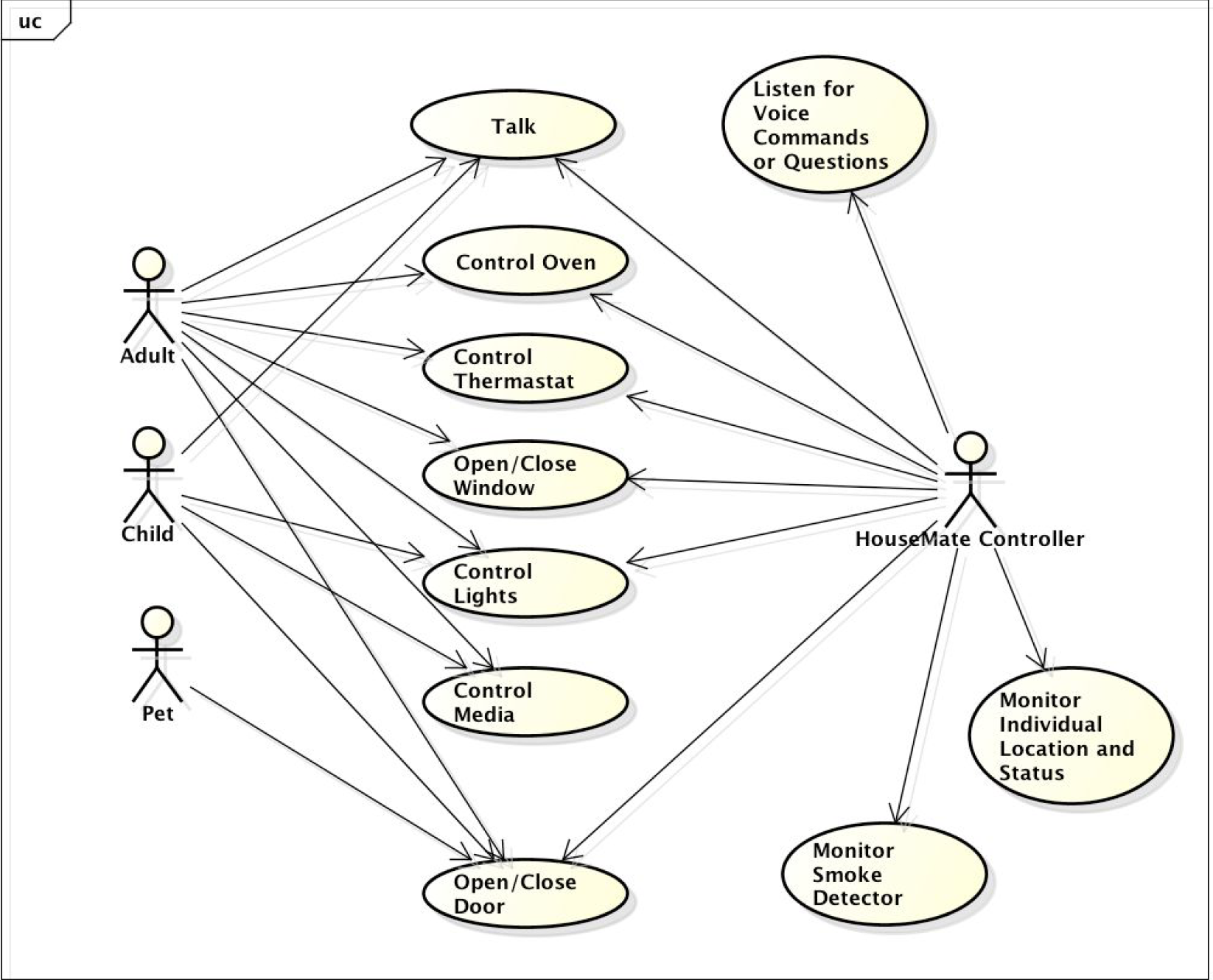
**Stimulus Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sensor/Appliance** | **Stimulus** | **Rule** | **Action** |
| Ava | Command: “open  door” | open the door to the  room | set door status to  open. |
| Ava | Command: “close  door | close the door to the  room | set door status to  closed |
| Ava | Command: “lights off” | turn off all the lights  in the room | set light status to off |
| Ava  (light color change ) | Generic Command:  “appliance\_type  status\_name value” | send the command to appliance with  matching type in the  current room. | forward command to  appliance and echo  command via Ava. |
| Ava | Question: “where is  <occupant\_name>?”  For example, “where  is Rover?” | use the Knowledge  Graph to determine  location of occupant. | send response to  Ava. “Rover  is\_located\_in kitchen” |
| Camera | Occupant detected. | turn on the lights in  the room, and  increase the  thermostat | send command to  turn lights on,  increase the  temperature of the  thermostat, update  location of occupant  in the KG |
| Camera | Occupant leaving | if no more occupants  are in the room, then  turn the lights off, and  lower the thermostat | turn off lights,  decrease the  temperature of the  thermostat, Update  the location of  occupant in the KG. |
| Camera | Occupant is inactive | if the only occupant,  dim the lights and  update the status of  the occupant to  resting. | update occupant  status in KG to  resting. |
| Camera | Occupant is active | update occupant  status to active | update occupant  status in KG to be  active. |
| Smoke Detector | Mode Fire | if occupants are in  the house, turn on all  lights in the house  and ask occupants to  leave the house. If  room has a window  and is on the first floor, recommend  exiting through the  window.  Call 911 to let them  know there is a fire. | send command to  turn on lights  send AVA text to  speech: “Fire in the  Kitchen, please leave  the house  immediately”. Call 911 |
| Oven | TimeToCook goes to  0 | if oven is on, turn  oven off and alert  occupants that food  is ready. | Turn oven off.  send Ava text to  speech. “Food is  ready” |
| Refrigerator | Beer count changes. | If beer count is less  than 4, ask Occupant  if they would like to  order more beer. If  occupant says yes,  order more beer. | Send email to store  requesting more  beer |

Use Cases

The HMCS supports two primary use cases:

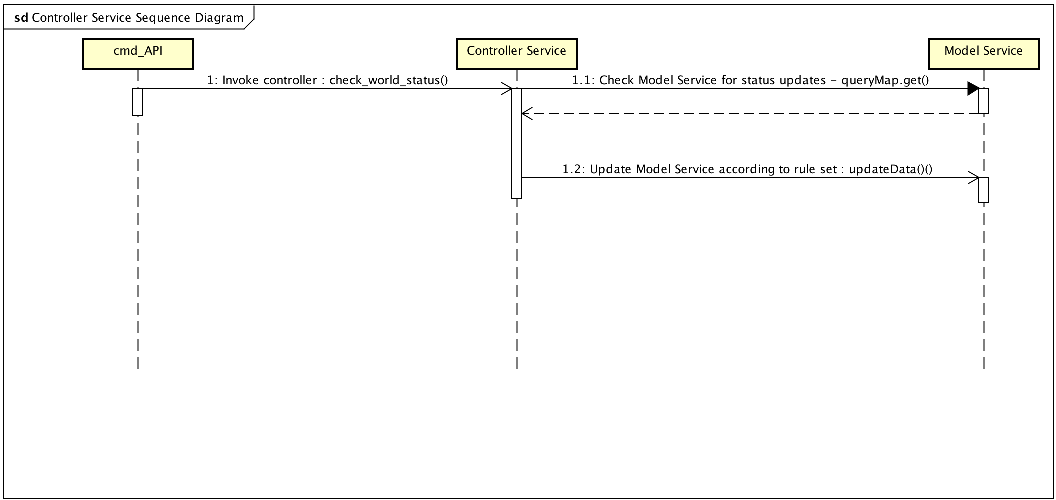
1. The HMCS constantly monitors the HMMS knowledge graph for new commands, occupant location or potential safety indicator
2. The HMCS executes the commands and updates the KG accordingly



As shown by the diagram above, The HMCS constantly identifies commands and updates that it executes to implement IoT within the HMMS defined home. Simply put, HMCS adds “user defined” intelligence to the HMMS elements.

Sequence Diagram

The Diagram below



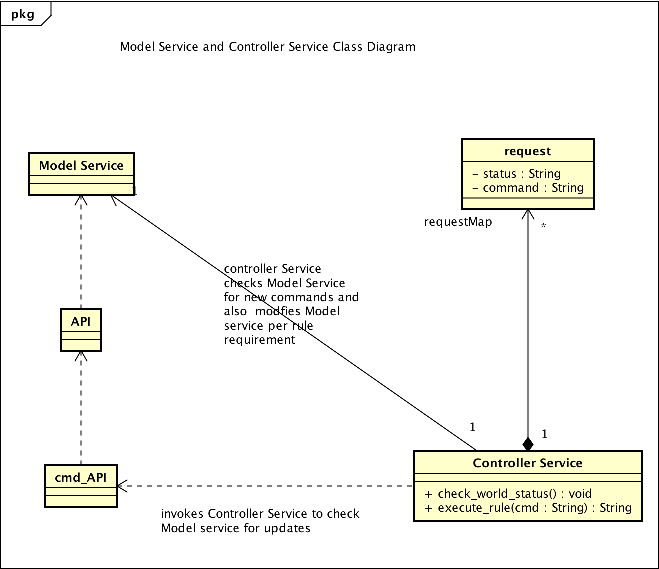
As shown by the sequence Diagram above, for every HMMS specification line read by cmd\_API

* cmd\_API invokes Controller Service to check for new commands/updates with check\_world\_status().
* Controller service uses queryMap.get() to check for new commands from sensors and appliances. queryMap.get executes as automatic response by either returning all commands.
* Controller service identifies any new commands by the command pool returned by queryMap.get()
* Controller Service saves the new command
* Controller Services executes the new commands according to in-built rules
* If demanded by rule per any command, the Controller Service uses updateData() to update the HMMS

Implementation

**Class Diagram**

The following diagram defines the House Mate Controller Service Classes contained with the package “cscie97.asn3.housemate.controller”.



**Class Dictionary**

This section specifies the class dictionary for the House Mate Controller service. The classes should be defined within “cscie97.asn3.housemate.controller”.

**Request Class**

As its name implies, the request class stores all commands issued by appliances and sensors. This is a command pattern implementation where an instance is created per request and also saved. It comprises two attributes,

**Properties**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Type** | **Description** |
| command | String | Public member that stores the sensor or appliance command. For example, “AVA1 input close\_door”. |
| status | String | Public member that stores the status of the command which can either be “open” or “closed”. |

**Controller Service Class**

The Controller Service is a singleton design Class which uses command pattern to store new commands, executes the command applies observer pattern to update the knowledge graph/ HMMS elements. The HMCS is invoked by cmd\_API. The relationship between the Controller Service (HMCS), cmd\_API and Model Service (HMMS) is described in the sequence diagram on page 5.

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| Check\_world\_status | ():void | Public member that does the following   * Checks for new commands * Stores new command * Sends command to execute\_rule() for further processing * Updates stored command status as closed if command was correctly executed |
| execute\_rule | (cmd:String):String | Public member that executes the command and uses obkect pattern to update the KG or HMMS element(s) |

**Association**

|  |  |  |
| --- | --- | --- |
| Association Name | Type | Description |
| requestMap | Map<Integer, Request> | Public associations that stores all new commands, whether valid or invalid. The request is mapped by a hashcode() generated from its command |

Implementation details

The above described design implementation for the House Mate Controller Service meets the requirements as follows

1. With every HMMS update, the HMCS checks the Knowledge graph/HMMS for any new commands and updates
2. The execute\_rule() method takes a command argument and executes it according to preset rules as described in the stimulus table
3. “AVA” type devices are classified as appliances.
4. All commands are logged and status indicated
5. All resulting actions from commands are printed to standard output. This approach simplifies viewing all commands and respective resulting actions in context
6. Control messages are sent to appliances as indicated by rule.
7. A request instance is created for every new command satisfying the command pattern requirement
8. The execute\_rule() method is able to send several control messages to appliances per rule requirement. This satisfies the observer pattern.

Exception Handling

The HMMS handles majority of any errors or exceptions. However, the HMCS throws one exception which is described below

ControllerCommandException: This exception is thrown in either of the following events

* The specified command is generated by the wrong appliance or sensor
* The command cannot be matched to any specific rule or stimuli in the stimulus table

Testing

1. Functional testing - The Controller Service will successfully identify and execute all valid commands with the accurate syntax and matching stimuli
2. Performance testing – The Controller service performance is dependent on the processing speed, memory speed and available memory of the host computer. Its speed is also dependent on the number of HMMS elements that need to be updated after a request is processed
3. Exception handling: The Controller service is case sensitive but it always identifies undocumented commands or documented commands generated by the wrong devices.

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.