House Mate Entitlement Service

Date: Nov 17th, 2017
Author: Stephen Akaeze
Reviewers: Benjamin Jenkins and Bin Gao

Introduction

The purpose of the document to provide information describing the House Mate Entitlement Service (HMES) implemented in **cscse97.asn4.housemate.entitlement** package. It comprises design requirements, use case diagram, class diagram, class dictionary, sequence diagram, risks and implementation that help provide an improved understanding of the HMES and what it accomplishes within the House Mate Service.

Overview

The House Mate System is designed to fully automate the home. The House Mate Service allows residents to control their home environment through voice commands. Sensors monitor the location of individuals within the home. Smart lights, doors, windows, thermostats, and other appliances are controlled through the House Mate system. Some of these appliances are controlled automatically based on the location of the residents, while voice commands can be used to activate or override others. Security is an important aspect of the House Mate System. It is critical that the House Mate system only allow trusted parties to see the status of and control the IOT devices.

Requirements

The Entitlement Service is responsible for controlling access to the House Mate Model Service interface. The Entitlement Service provides a central point for managing Users, Resources, Permissions, Roles, Resource, Roles, and Access Tokens.

The Entitlement Service supports 2 primary actors: **Administrator** and **Occupants**.

The **Administrator** is responsible for managing the Resources, Permissions, Roles, and Users maintained by the Entitlement Service. The Administrator also provisions houses within the House Mate system and has full access to all resources within the House Mate System.

The **Occupants** are identified by their voice print. Occupants use voice commands to interact with the House Mate System. The Entitlement Service supports identifying and authenticating users using the user's voice print. When an Occupant issues a voice command, the Controller service sends the voice print to the Entitlement Service. The Entitlement Service finds the user with a matching voice print and returns an Authentication Token for the user. This Authentication token is used when invoking Model Service methods on behalf of the Occupant. In this way, only Occupants with the appropriate permissions are allowed to control appliances within the house.

The Entitlement service has to implement the following patterns

1. Use the **Visitor Pattern** to:
   a. support traversing the objects of the Entitlement Service to provide an inventory of all Users, Resources, Accesses, Roles, and Permissions.
   b. checking for access
2. Apply the **Abstract Factory Pattern** to create instances of the Entitlement Service domain classes.
3. Use the **Singleton Pattern** to return a pointer to an implementation of the Entitlement Service.
4. Use the **Composite Pattern** to manage the whole part relation of Roles and Permissions.

Use Cases

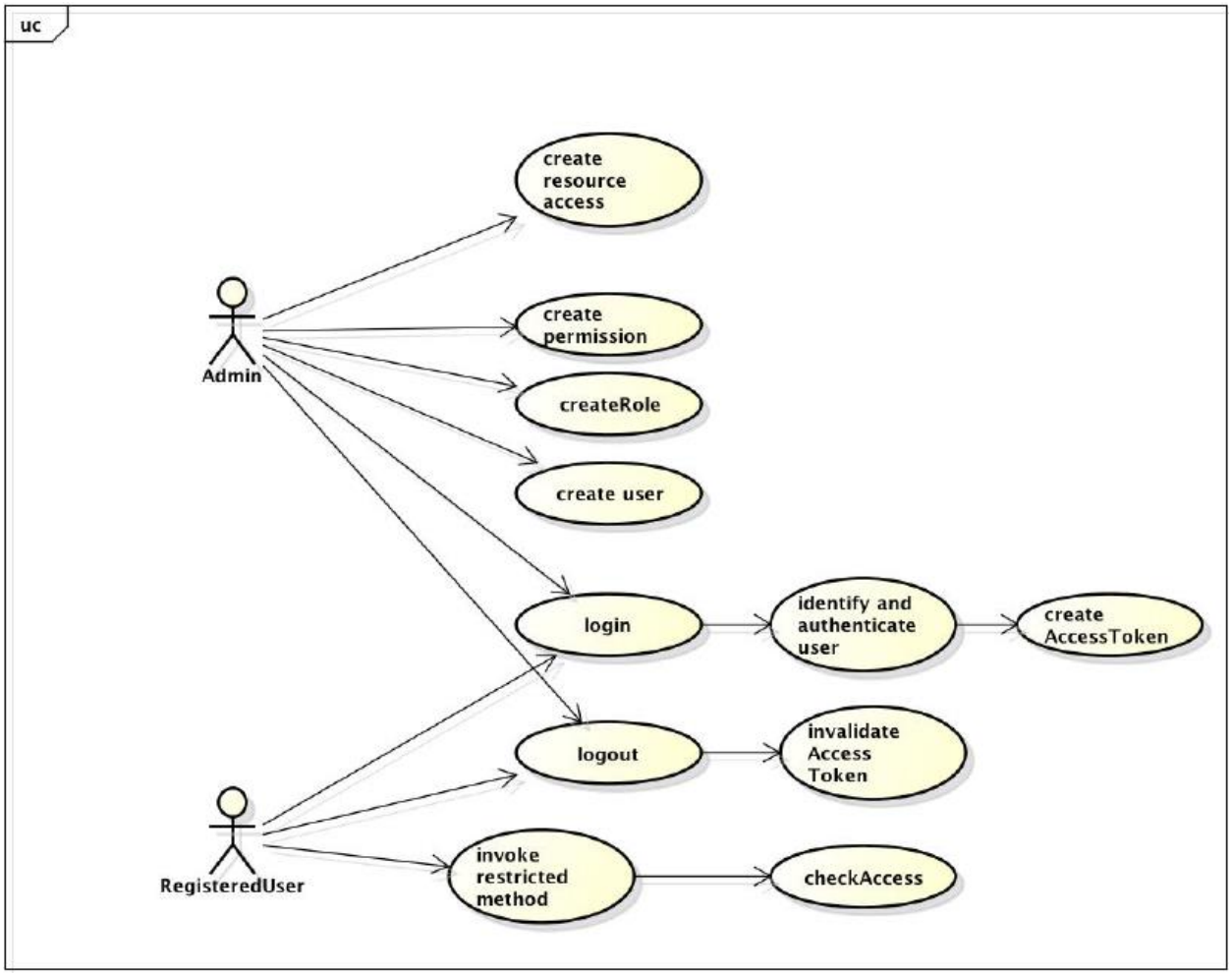


Fig. 1: HMES admin and Registered user(s) use cases

As shown in the above use case, the administrator has the full privilege of managing the Resources, Permissions, Roles, and Users maintained by the Entitlement Service. The administrator login brings the system online before any other HMMS commands can be executed. Upon logout, the system goes offline and prevents all further HMMS executions. The Administrator can also assign admin privileges to any user after the system has been fully setup.

Quick info

- An admin must be logged in(System online) before any HMMS command can be executed. An admin token is created for all admin logins.
- All AVA command voice print must be validated before the AVA command is executed. A voice token is created for all AVA requests.

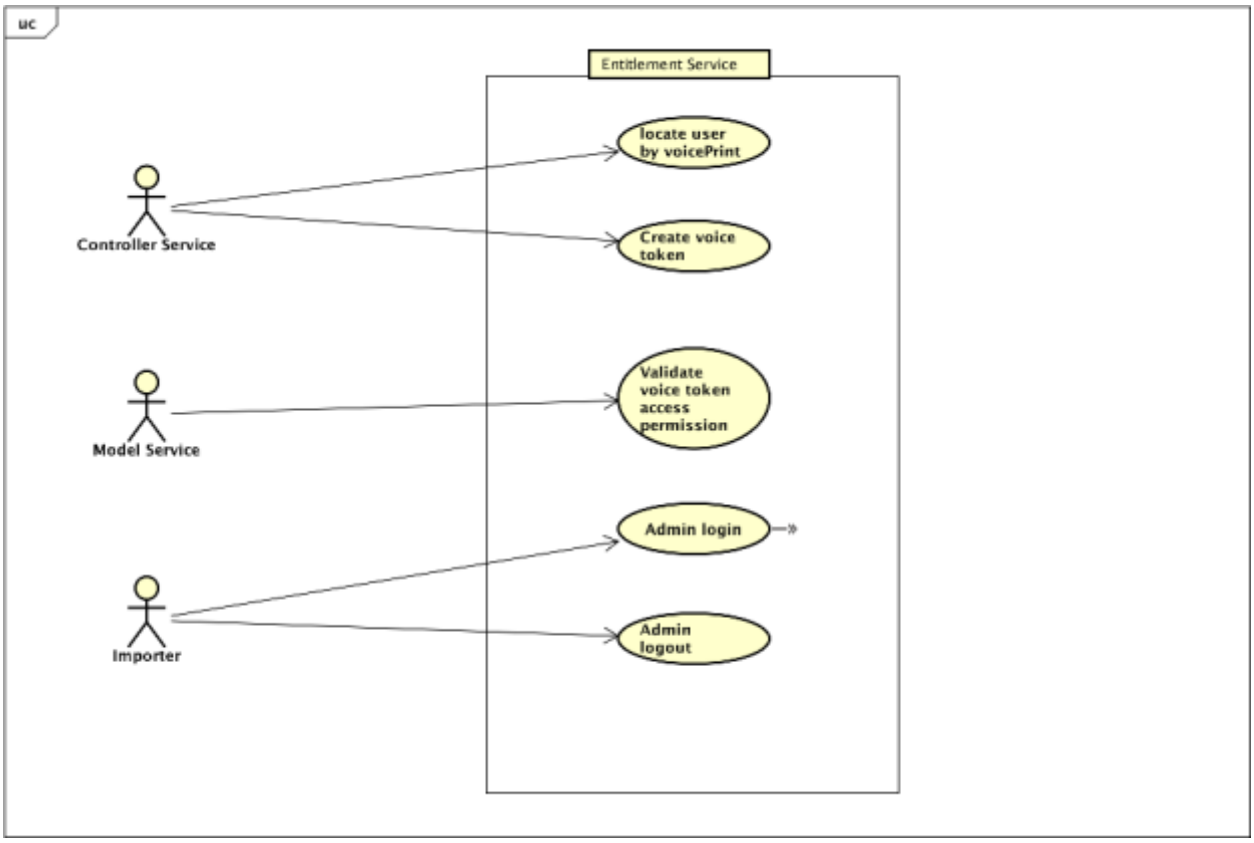


Fig. 2: Use cases of interactions between HMES, HMCS, HMMS and importer

As shown above, HMES can equally receive requests from other House Mate modules to guarantee smooth functionality. HMES can locate a registered user by their voice print, create admin or voice tokens, login an admin, validate a token and logout an admin

Class Diagram

The following class diagram defines the classes defined in this HMES design as implemented in the **cscse97.asn4.housemate.entitlement** package

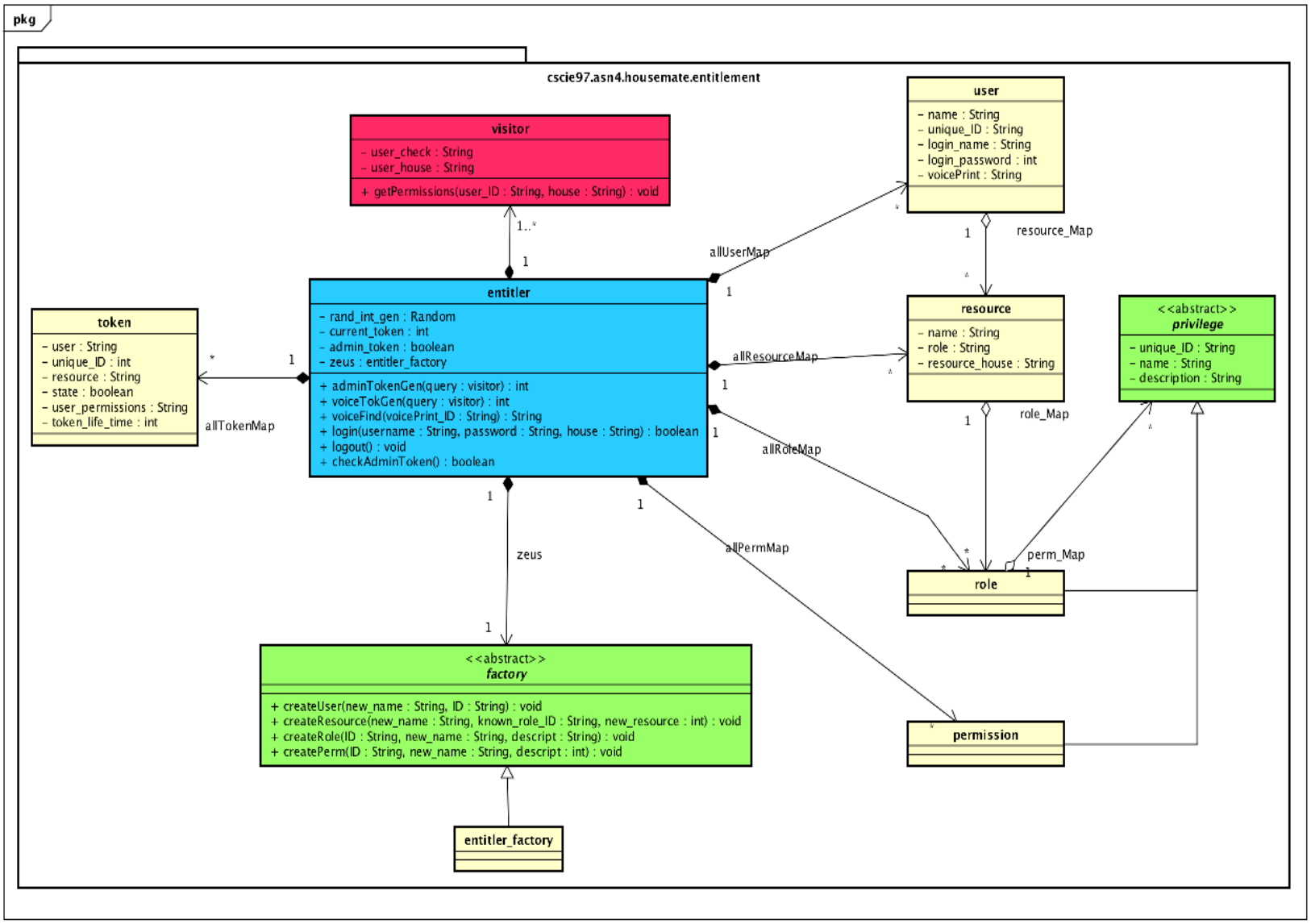


Fig 3: The HMES Class Diagram.

Class Dictionary

This section specifies the class dictionary for the House Mate Entitlement Service. The classes should be defined within “cscie97.asn4.housemate.entitlement”.

Entitler

This is the singleton class that comprises all the HMES components. It stores users, resources, permissions, roles etc. it also contains several methods used for operating on the earlier listed class instances

Methods

Method Name	Signature	Description
adminTokenGen	(query : visitor) : int	Public method for generating new tokens for administrator logins. The token is added to allTokenMap. The token ID is also stored in current_token. It also returns the token ID
voiceTokenGen	(query : visitor) : int	Public method for generating new tokens for all AVA commands. . The token is added to allTokenMap. It also returns the token ID
voiceFind	(voicePrint_ID : String) : String	Public method to accepting a voice print and returning its corresponding registered user or “unknown user” otherwise
login	(username : String, password : String, house : String) : boolean	Public method for verifying an administrator login for a specific house resource. It returns a Boolean indicating whether the login was a success. No HMMS commands can be executed without a valid admin login
logout	(): void	Public method for invalidating the previous admin token stored in current_token.
checkAdmin	() : boolean	Public method for verifying that the current token from the last admin login attempt contains administrative privileges to allow any HMMS command executions

Properties

Property Name	Type	Description
Rand_int_gen	Random	Random integer generator
current_token	int	Public token ID from the last administrator login
admin_token	boolean	Public Boolean which can revoke all central admin login attempts(No implementation).
zeus	factory	Public member to creating HMES domain classes such as user or resource

Associations

Association name	Type	Description
allUsermap	Map<String, user>	Public association storing all registered users
allResesourceMap	Map<String, resource>	Public association storing all created resource instances
allRolemap	Map<String, role>	Public association of all created role instances
allPermMap	Map<String, permission>	Public association of all created permission instances
allTokenMap	Map<String, token>	Public association of all created admin and voice tokens

Factory

The factory class implements abstract factory pattern to enable user, role, resources and permission instances to be specified and created after runtime

Methods

Method name	Signature	Description
createUser	(new_name : String, ID : String) : void	Public abstract method that creates user instances and stores them in allUserMap
createResource	(new_name : String, known_role_ID : String, new_resource : int) : String	Public abstract method that creates resource instances and stores them in allResourceMap
createRole	(ID : String, new_name : String, descript : String) : void	Public abstract method that creates role instances and stores them in allRoleMap

createPerm	(ID : String, new_name : String, descript : int) : void	Public abstract method for creating permission instances and storing them in allPermMap

Entitler_Factory

Entitler class inherits the methods factory class to create entitlement specific domain classes

Methods

Method name	Signature	Description
createUser	(new_name : String, ID : String) : void	Public method that creates user instances and stores them in allUserMap
createResource	(new_name : String, known_role_ID : String, new_resource : int) : String	Public method that creates resource instances and stores them in allResourceMap
createRole	(ID : String, new_name : String, descript : String) : void	Public method that creates role instances and stores them in allRoleMap
createPerm	(ID : String, new_name : String, descript : int) : void	Public method for creating permission instances and storing them in allPermMap

Token

The token class comprises all necessary attributes for creating comprehensive tokens to provide a reliable HMES implementation

Properties

Property Name	Type	Description
user	String	Public member for storing the user ID of the token assignee
unique_ID	int	Public member for storing the unique token ID
resource	String	Public member for storing the house resource that utilizes the token
state	boolean	Public member for storing the token validity. True mean valid
user_permissions	Set<String>	Public member for storing the unique IDs of all

		permissions associated with the specified registered user
token_life_time	int	Public member storing the number of importer commands executed before the current admin token stored in entitler.current_token becomes invalid

Visitor

The visitor class is created to enable token generation in the HMES. The visitor instance transverses through resources, roles and permissions of the user specified by “user-check’ in house resource “user_house’ to obtain all permission IDs associated with the registered user

Properties

Property Name	Type	Description
user_check	String	Public member containing the ID of the user whose credentials are to be trans versed to obtain his/her permission IDs
user_house	String	Public member storing the house Id where the user credentials is being obtained for.

Privilege

This is the abstract super class that provide a frame work for the role and permission classes.

Properties

Property Name	Type	Description
unique_ID	String	Public member the privilege instance unique ID
name	String	Public member storing the privilege instance name
description	String	Public member storing the privilege description

Permission

This stores the HMES permission details. It is a subclass of the privilege class

Properties

Property Name	Type	Description
unique_ID	String	Public member the permission instance unique ID
name	String	Public member storing the permission instance name
description	String	Public member storing the permission description

Role

This stores the HMES role details and contains a map of permissions associated with each role instance. It is a subclass of the privilege class

Properties

Property Name	Type	Description
unique_ID	String	Public member the role instance unique ID
name	String	Public member storing the role instance name
description	String	Public member storing the role description

Associations

Association name	Type	Description
perm_Map	Map<String, permission>	Public association storing all permissions assigned to the role instance

Resource

This stores the HMES resource details and contains a map of roles associated with each resource instance.

Properties

Property Name	Type	Description
unique_ID	String	Public member the resource instance unique ID
name	String	Public member storing the resource instance name

resource_house	String	Public member storing the house the resource instance is associated with

Associations

Association name	Type	Description
role_Map	Map<String, role>	Public association storing all roles assigned to the resource instance

User

This stores the HMES registered user details and contains a map of resources associated with each user instance.

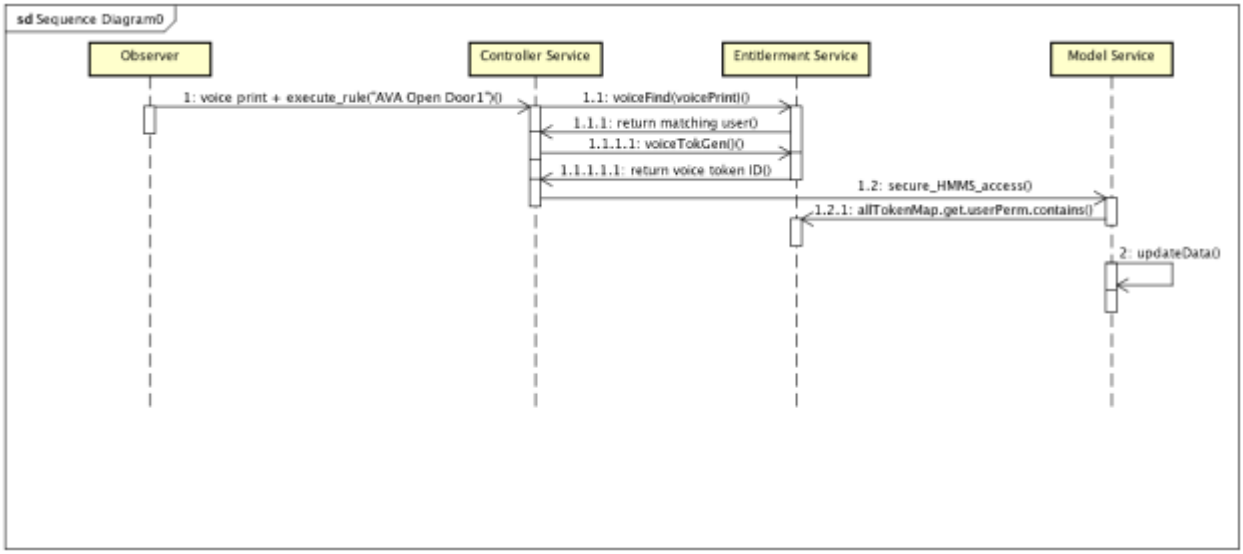
Properties

Property Name	Type	Description
unique_ID	String	Public member the user instance unique ID
name	String	Public member storing the user instance name
login_name	String	Public member storing the user's log in name
login_password	int	Public member storing the user's password
voicePrint	String	Public member storing the user's voicePrint

Associations

Association name	Type	Description
resource_Map	Map<String, resource>	Public association storing all resources assigned to the user instance

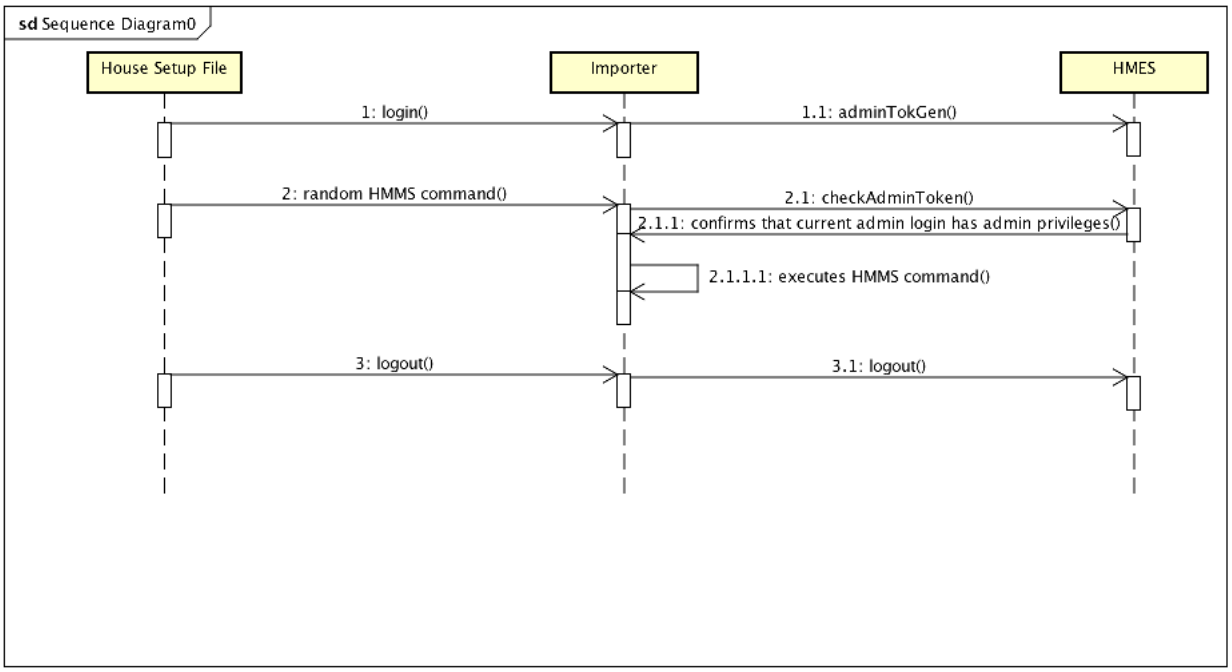
Sequence Diagram for AVA commands



As shown in the above sequence diagram

- For all new AVA commands, the observer (HMCS.check_world_status()) identifies a new AVA command
- HMCS queries the HMES for the user matching the AVA command voiceprint
- HMES identifies the user and returns the user to HMCS
- HMCS requests HMES to create for new voice token for the user
- HMES generates token and sends back the token ID
- HMCS makes HMMS command with associated command token
- HMMS confirms with HMES that token has all required permission for the command
- HMES confirms whether token has required permission(s) and informs HMMS
- HMMS proceeds to execute the command

Sequence Diagram for all HMMS/Importer commands



As shown in the above sequence diagram

- for all new importer/HMMS command, there has to be an administrative login before any new commands can be executed
- The importer obtains the next HMMS command from the house setup file
- Importer confirms from HMES that the current logged in user has admin privileges
- Then, importer proceeds to evaluate the command appropriately

Implementation

The specific implementation of the HMES is captured in the following class diagram and class dictionary. Below is a description of how the HMES implementation uses popular software design patterns for a more durable, faster and reliable implementation.

1) **Singleton Pattern**: This is implemented in the entitle class. This ensures a single instance of the entitle class for executing HMES requests

2) **Composite Pattern**: This is implemented between the role and permission classes. Both roles and permissions share the privilege super-class but every role contains a perm_Map Hash Map that can hold several permissions to demonstrate the whole-part relationship

3) **Abstract Factory Pattern**: This is implemented by the factory and entitle_factory classes. The “abstract” Factory methods such as “createUser()” bind the entitle_factory subclass to provide the entitlement specific implementation for the entitlement domain classes such as roles, resources, user and permissions.

4) **Visitor Pattern:** This is implemented with the entitler voiceTokGen() and adminTokGen ()methods. It is used to transverse through objects within the entitlement service to generate an inventory of permissions for a registered user.

5) **Token expiration**: This is implemented in the importer class. Importer update_admin_token() method called after every importer command execution to increment the current admin token’s life time. It also invalidates the admin token’s state if the token’s token_life-time value equals the importer class tokenLimit value.

Exception handling

The House Mate Entitlement Service is built to thrown an AccessDeniedException in the following occurrences or errors

- Creating a user instance for a non-existing occupant
- Invalid admin login credentials
- Current admin token does not have admin privileges or has expired
- Voice print does not match any registered HMES users

The HMMS and HCMS are also running in the background for a reliable House Mate System.

Testing

1) Functional testing - HMES will successfully identify and execute all valid commands with the accurate syntax and matching stimuli
2) Performance testing – HMES performance is dependent on the processing speed, memory speed and available memory of the host computer.
3) Exception handling: HMES does a great job of identifying potential issues in the House Mate System entitlement concerns

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.
- HMES provides unrestricted access to commands or notifications coming from appliances or sensors. There is a potential danger where a hacker could hack a specific appliance or sensor to gain access to the HMMS. This can be mitigated by ensuring that

all IoT devices employ trusted communication protocols and implement encryptions extensive AES encryptions