House Mate Entitlement Service Document

Date: November 11th 2015
Author: Anna Levin

Introduction:

This document details the design used for the implementation of the controller system for an automated home system. This system component is known as the House Mate Controller.

Overview:

The House Mate System is designed for monitoring and interacting with fully automated homes. It is achieved by implementing a concept known as the Internet of Things where ordinary household appliances are connected to the Internet and can be controlled by a remote device (e.g. a smart phone).

The House Mate System, consists of three major components and two smaller ones:

- The House Mate Model Service (HMMS)
- The House Mate Controller
- The House Mate Entitlement Service
- The Importer (pg 15) and UserInterface (pg 24)

The HMMS creates a knowledge graph of homes, their occupants, and any device that's part of the Internet of things. It has to be able to retrieve information and update the model accordingly using the correct commands.

The House Mate Controller monitors the sensors and controls the appliances in the house. When there is a status change in one of the appliances/senors it will respond by making the appropriate status changes in any affected appliance (e.g. if a sensor picks up someone entering an empty room it will turn on all the lights). It is also where all voiced based commands and queries are handled.

The House Mate Entitlement Service handles user privileges and makes sure a user can't pass in a command that they don't have the privilege for. It also makes sure that the user can't act on a device that is part of a resource they don't have the right access to.

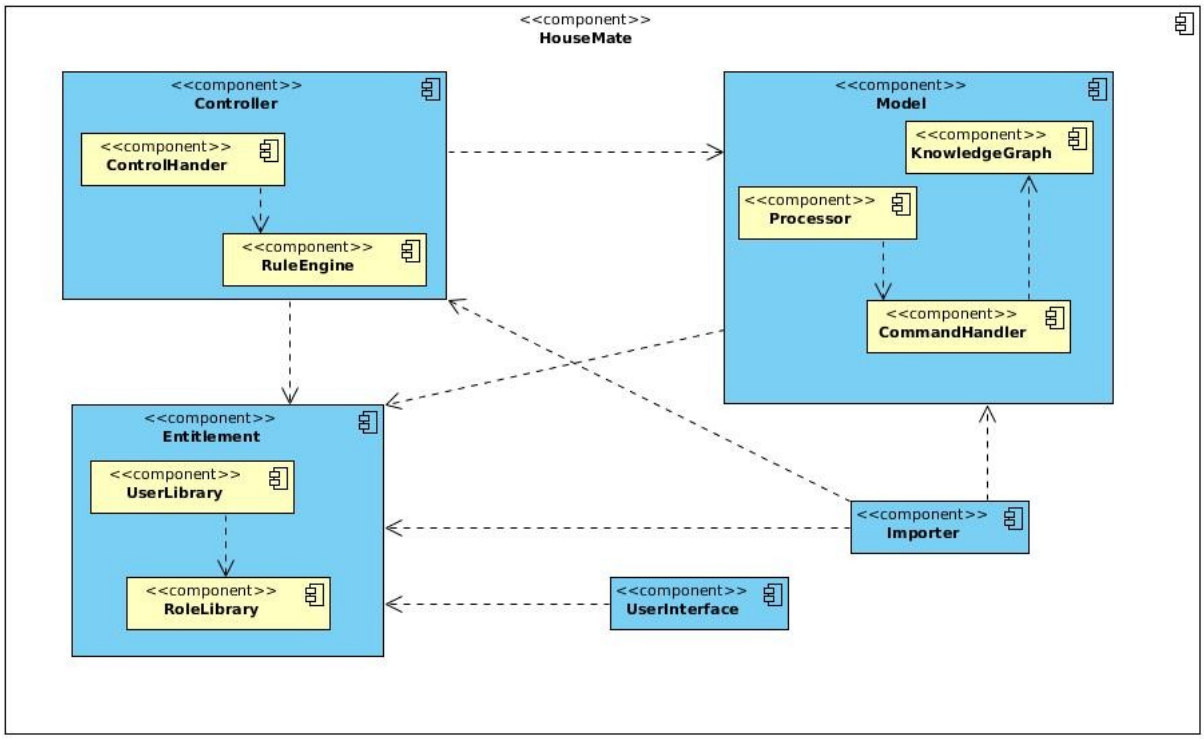


Fig 1: Top level component diagram

The focus of this project is on the Entitlement Service. The Entitlement Service handels the management of commands based on who issued them and that users privileges. A user can access any IOT Devices that are part of their resources and privaleges. The role (adult, child, etc) determines what privileges they have. For instance, an adult would have the privilege of using the oven in a resident type resource while a child wouldn't have that.

For full control over the House Mate System (e.g. model updating) the user must have administrative privileges. The administrator can influence any IOT Device that's part of the House Mate System as well as add/remove new instances to the system.

If the admin adds a new occupant instance then a new user instance gets added to the system, including their role and privileges (Fig 2 pg 33). However the user will have an empty resource library until the user adds their house relation(s) to the system.

In order for any user to interact with the House Mate System, they need to get an Access Token first. To get one they simply have to log in either by entering their user name and password or via voice print.

Requirements:

References and Sources

The following sources were used to determine the requirements:

- REF-01 : HouseMateEntitlementServiceRequierments.pdf

Identified Requirements

Analysis requirements derived from the HouseMateEntitlementServiceRequierments.pdf:

1. Access token are needed in order to access model from the controller or user interface [REF-01 pg 6]
2. New occupant results in roles being created for said occupant and default voice recognition [REF-01 pg 2]

Course requirements as specified from the HouseMateEntitlementServiceRequierments.pdf:

1. Authentication tokens are one time use only and are globaly unique [REF-01 pg 5-6]
2. Have Exception to catch invalid commands and tokens as well as when a user is unable to log in [REF-01 pg 6]:
   a) AccessDeniedException (user doesnt have those permissions)
   b) InvalidAccessTokenException (invalid token passed in)
   c) AuthenticationException (log in issue)

Use Cases:

The same use cases from before still apply:

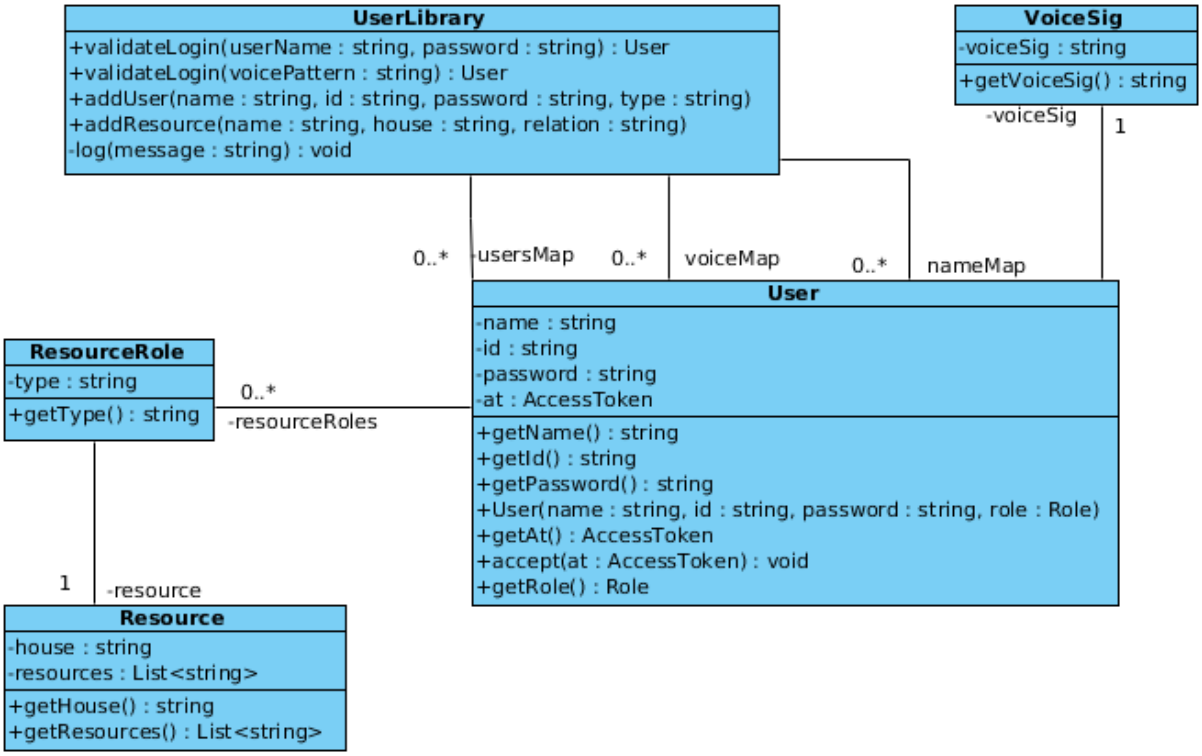
- Add new model instance to system
- Delete model instance from system
- Update model
  - Change the status/settings of sensors/appliances
    - When status/setting of sensors/appliances consult rule engine to see if additional actions need to be taken
      - Execute any additional actions
- Retrieve information on the environment or particular aspect.
- Retrieve location/status of occupant
- Retrieve information on sensors/appliances

However with the addition with the entitlement service, there are two new additional use cases that are unique to the admin role:

- Add new User instance
- Update User with new Resource Role

Class Diagram

UserLibrary



Name	Value
Name	UserLibrary
Teamwork Create Date Time	Dec 31, 1969 7:00:00 PM
Point Connector End To Compartment Member	true

Summary

Name	Description
UserLibrary	Contains all the users for the system. Since there are two ways to login, there are two maps being used. One uses the password as the key to the User while the other uses VoiceSig to map to the user.
VoiceSig	Voice signature of the user
User	Contains information on the user (id, password, voice sig, role, etc). Used to validate commands and make sure that the user has the permission to execute a particular command.
ResourceRole	Represents the user relation to a particular resource (e.g. child guest vs child resident)

Resource	The id of a particular resource (in this case house) and all of its sub resources (the appliances)

Details

UserLibrary

Name	Value
Description	Contains all the users for the system. Since there are two ways to login, there are two maps being used. One uses the password as the key to the User while the other uses VoiceSig to map to the user.
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Operations

public validateLogin (userName : string, password : string) : User		
Parameters	**userName**	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	password	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Description	Validate the password and username	
Transit To	N/A (gatekeeper -> userLibrary)	
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

public validateLogin (voicePattern : string) : User		
Parameters	**voicePattern**	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Description	Validate the voice pattern	
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

public addUser (name : string, id : string, password : string, type : string)		
Parameters	**name**	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	id	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	password	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	type	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

public addResource (name : string, house : string, relation : string)		
Parameters	**name**	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	house	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	relation	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

private log (message : string) : void		
Parameters	**message**	
	Multiplicity	Unspecified
	Type	string
	Direction	inout
Description	Prints out a message that a new user has been created and provides the time stamp	
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

VoiceSig

Name	Value

Description	Voice signature of the user
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Attributes

private voiceSig : string			
Type	string		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

Operations

public getVoiceSig () : string	
Leaf	false
Ordered	false
Unique	true
Query	false

User

Name	Value
Description	Contains information on the user (id, password, voice sig, role, etc). Used to validate commands and make sure that the user has the permission to execute a particular command.
Active	false

Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Attributes

private name : string			
Type	string		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

private id : string			
Type	string		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

private password : string	
Type	string
Allow Empty Name	false

Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

private at : AccessToken			
Type	AccessToken		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

Operations

public getName () : string	
Leaf	false
Ordered	false
Unique	true
Query	false

public getId () : string	
Leaf	false
Ordered	false
Unique	true
Query	false

public getPassword () : string

Leaf	false
Ordered	false
Unique	true
Query	false

public User (name : string, id : string, password : string, role : Role)		
Parameters	**name**	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	id	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	password	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	role	
	Multiplicity	Unspecified
	Type	Role
	Direction	in
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

public getAt () : AccessToken	
Leaf	false
Ordered	false
Unique	true
Query	false

public accept (at : AccessToken) : void		
Parameters	at	
	Multiplicity	Unspecified
	Type	AccessToken
	Direction	in
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

public getRole () : Role	
Transit To	User Sequence Diagram.2.1.1: getRole() : Role (gatekeeper : Gatekeeper -> user : User)
	Admin Sequence Diagram.4: getRole() : Role (gatekeeper : Gatekeeper -> user : User)
Leaf	false
Ordered	false
Unique	true
Query	false

ResourceRole

Name	Value
Description	Represents the user relation to a particular resource (e.g. child guest vs child resident)
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Attributes

private type : string

Type	string		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

Operations

public getType () : string	
Leaf	false
Ordered	false
Unique	true
Query	false

Resource

Name	Value
Description	The id of a particular resource (in this case house) and all of its sub resources (the appliances)
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Attributes

private house : string	
Type	string
Allow Empty Name	false

Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

private resources : List			
Template Type Bind Info	N/A		
Type	List		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

Operations

public getHouse () : string	
Leaf	false
Ordered	false
Unique	true
Query	false

public getResources () : List	
Leaf	false
Template Type Bind Info	N/A
Ordered	false
Unique	true

Query	false

Class Diagram

Importer

Name	Value
Name	Importer
Teamwork Create Date Time	Dec 31, 1969 7:00:00 PM
Point Connector End To Compartment Member	true

Summary

Name	Description
Importer	Takes in a file name and reads and converts it into a format that can be processed. There is a load file method for each type of file that could be read in. At the moment there are only three such methods.

Details

Importer

Name	Value
Description	Takes in a file name and reads and converts it into a format that can be processed. There is a load file method for each type of file that could be read in. At the moment there are only three such methods.
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Operations

public loadFile (fileName : string) : List

Parameters	fileName	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Description	Takes in the file name, opens it, reads in each line, and stores each line in an ArrayList of strings. When it finishes reading the file, the ArrayList gets returned.	
Leaf	false	
Template Type Bind Info	N/A	
Ordered	false	
Unique	true	
Query	false	

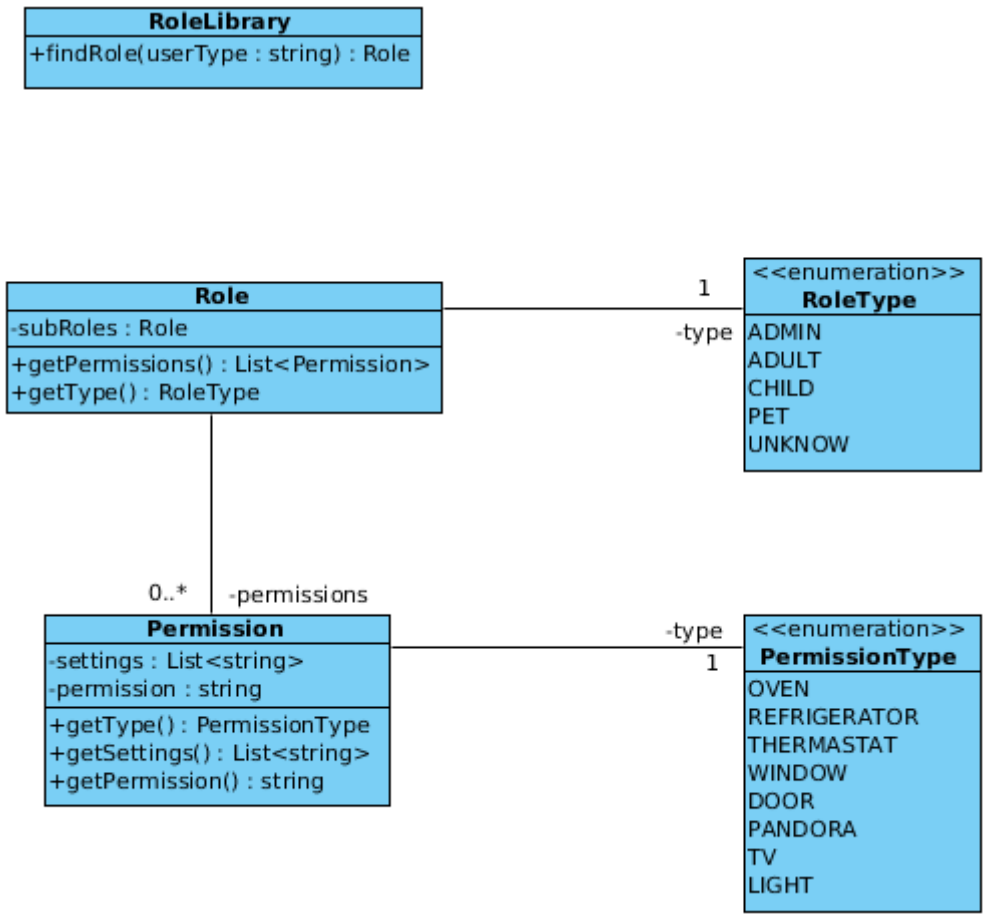
public loadRuleFile (fileName : string) : List		
Parameters	fileName	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Description	Read in a rule file and parses it so that rules can be generated. File must contain one stimulus, one or more clauses, and one or more actions. Returns an ArrayList<ArrayList<String>> where each ArrayList<String> represents a rule	
Leaf	false	
Template Type Bind Info	N/A	
Ordered	false	
Unique	true	
Query	false	

public loadAuthenFile (fileName : string) : List		
Parameters	fileName	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Leaf	false	
Template Type Bind Info	N/A	

Ordered	false
Unique	true
Query	false

Class Diagram

RoleLibrary



Name	Value
Name	RoleLibrary
Teamwork Create Date Time	Dec 31, 1969 7:00:00 PM
Point Connector End To Compartment Member	true

Summary

Name	Description
RoleLibrary	Handles the assignment of Role to User. Returns a Role that coresponds to the string indicator passed in (e.g. “adult” results in a Role of type Adult being returned). There are five types of Roles and they each only have one available

	instance: Admin, Adult, Child, Pet, Unknown
RoleType	
Role	The type of role the user has and what permissions they have as a result. For instance, the admin will have full permission to every aspect of the House Mate System while an Adult will only have full permission to the resources of any house they are a resident off. The singleton pattern is used to ensure that there is only one type of each role. It implements the Composite pattern since it uses a hierachy when handling the Permissions. For example, an Admin will have the same and more privlages as an Adult while the Adult will have the privileges that a Child has plus some additional ones. The result of that would be the Admin having Adult as a sub Role and the Adult having the Child as its.
Permission	Contains the type of device and what device settings are available. Also contains a basic description
PermissionType	An Enum that idicates what kind of appliance the Permission is giving access to.

Details

RoleLibrary

Name	Value
Description	Handles the assignment of Role to User. Returns a Role that coresponds to the string indicator passed in (e.g. “adult” results in a Role of type Adult being returned). There are five types of Roles and they each only have one available.
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Operations

public findRole (userType : string) : Role		
Parameters	userType	
	Multiplicity	Unspecified

	Type	string
	Direction	inout
Description	Takes in a string describing the type of user (e.g. adult, child) and returns the appropriate role for said user.	
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

RoleType

Name	Value
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false
Stereotypes	enumeration

Enumeration Literals

ADMIN

ADULT

CHILD

PET

UNKNOW

Role

Name	Value
Description	The type of role the user has and what permissions they have as a result. For instance, the admin will

	have full permission to every aspect of the House Mate System while an Adult will only have full permission to the resources of any house they are a resident off. The singleton pattern is used to ensure that there is only one type of each role. It implements the Composite pattern since it uses a hierachy when handling the Permissions. For example, an Admin will have the same and more privlages as an Adult while the Adult will have the privileges that a Child has plus some additional ones. The result of that would be the Admin having Adult as a sub Role and the Adult having the Child as its.
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Attributes

private subRoles : Role			
Type	Role		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	0..*		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

Operations

public getPermissions () : List	
Transit To	User Sequence Diagram.2.1.2: getPermissions() : List<Permission> (gatekeeper : Gatekeeper -> role : Role)
	Admin Sequence Diagram.3.1.1: getPermissions() : List<Permission> (gatekeeper : Gatekeeper -> role : Role)

Leaf	false
Template Type Bind Info	N/A
Ordered	false
Unique	true
Query	false

public getType () : RoleType	
Leaf	false
Ordered	false
Unique	true
Query	false

Permission

Name	Value
Description	Contains the type of device and what device settings are available. Also contains a basic description
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Attributes

private settings : List			
Template Type Bind Info	N/A		
Type	List		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		

Is ID	false
Leaf	false

private permission : string			
Type	string		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

Operations

public getType () : PermissionType	
Leaf	false
Ordered	false
Unique	true
Query	false

public getSettings () : List	
Leaf	false
Template Type Bind Info	N/A
Ordered	false
Unique	true
Query	false

public getPermission () : string	
Leaf	false
Ordered	false
Unique	true
Query	false

PermissionType

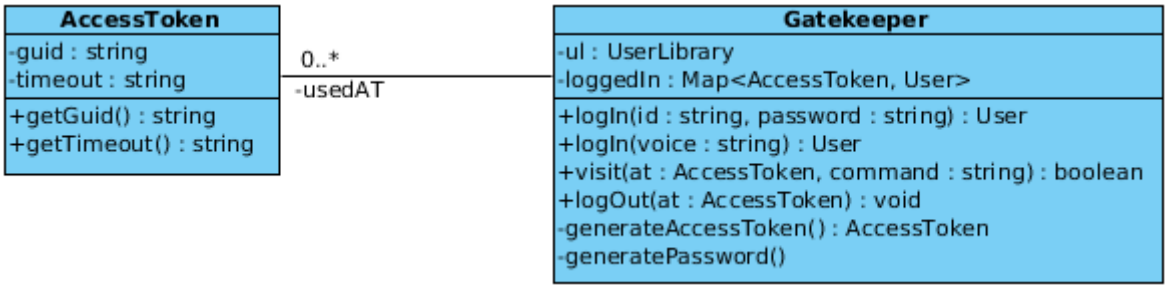
Name	Value
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false
Stereotypes	enumeration

Enumeration Literals

OVEN
REFRIGERATOR
THERMASTAT
WINDOW
DOOR
PANDORA
TV
LIGHT

Class Diagram

UserInterface



Name	Value
Name	UserInterface
Teamwork Create Date Time	Dec 31, 1969 7:00:00 PM
Point Connector End To Compartment Member	true

Summary

Name	Description
AccessToken	A globally unique identifier with a one time use. Enables the user to pass in commands and queries if they are logged into the system. Has a limited lifespan if user remains inactive.
Gatekeeper	Keeps track of all the active users that are currently logged on the system while only allowing registered users to access the system. An AccessToken is required to access any user related information for any purpose (e.g. seeing if a command is valid). Implements the Visitor pattern.

Details

AccessToken

Name	Value
Description	A globally unique identifier with a one time use. Enables the user to pass in commands and queries if they are logged into the system. Has a limited lifespan if user remains inactive.
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false

Root	false

Attributes

private guid : string			
Type	string		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

private timeout : string			
Type	string		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

Operations

public getGuid () : string	
Leaf	false
Ordered	false
Unique	true
Query	false

public getTimeout () : string

Leaf	false
Ordered	false
Unique	true
Query	false

Gatekeeper

Name	Value
Description	Keeps track of all the active users that are currently logged on the system while only allowing registered users to access the system. An AccessToken is required to access any user related information for any purpose (e.g. seeing if a command is valid). Implements the Visitor pattern.
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Leaf	false
Root	false

Attributes

private ul : UserLibrary			
Type	UserLibrary		
Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	Unspecified		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

private loggedIn : Map	
Template Type Bind Info	N/A
Type	Map

Allow Empty Name	false		
Getter	false	Setter	false
Derived	false		
Multiplicity	0..*		
Aggregation	None		
Derived Union	false		
Read Only	false		
Is ID	false		
Leaf	false		

Operations

public logIn (id : string, password : string) : User		
Parameters	**id**	
	Multiplicity	Unspecified
	Type	string
	Direction	in
	password	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Description	Standard login class (e.g. needs username and password)	
Transit To	N/A (TestDriver -> gatekeeper)	
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

public logIn (voice : string) : User		
Parameters	**voice**	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Description	Voice based login. Uses voice recognition to identify	

	who is giving a verbal command/query through Ava
Leaf	false
Ordered	false
Unique	true
Query	false

public visit (at : AccessToken, command : string) : boolean		
Parameters	at	
	Multiplicity	Unspecified
	Type	AccessToken
	Direction	in
	command	
	Multiplicity	Unspecified
	Type	string
	Direction	in
Description	Checks to see if a command is valid. Uses to AccessToken to see which User has that token. If their is a user assigned to said token, then it will check to see what privileges that user has and compares the command to it. Returns true if everything checks out.	
Transit To	User Sequence Diagram.2.1: visit(at : AccessToken, command : string) : boolean (commandHandler : CommandHandler -> gatekeeper : Gatekeeper)	
	Admin Sequence Diagram.3.1: visit(at : AccessToken, command : string) : boolean (commandHandler : CommandHandler -> gatekeeper : Gatekeeper)	
Leaf	false	
Ordered	false	
Unique	true	
Query	false	

public logOut (at : AccessToken) : void		
Parameters	at	
	Multiplicity	Unspecified
	Type	AccessToken
	Direction	inout
Description	logout of system	

Transit To	N/A (TestDriver -> gatekeeper)
Leaf	false
Ordered	false
Unique	true
Query	false

private generateAccessToken () : AccessToken	
Description	Generate a new access token with a guid for the user.
Leaf	false
Ordered	false
Unique	true
Query	false

private generatePassword ()	
Description	Generates a guid password. The password is a guid due to the fact that Users are stored in a Map with a password for their key.
Leaf	false
Ordered	false
Unique	true
Query	false

Implementation Details

Sequence of Events

There are different variables that will alter the sequence of events. User type could determine if there are extra steps that could be taken when issuing a command. The method of logging (voice vs username and password) in to the system will also produce a different sequence of events. For now we will be covering the following sequence diagrams:

- Admin Sequence Diagram
- User Sequence Diagram
- Voice Command Sequence Diagram

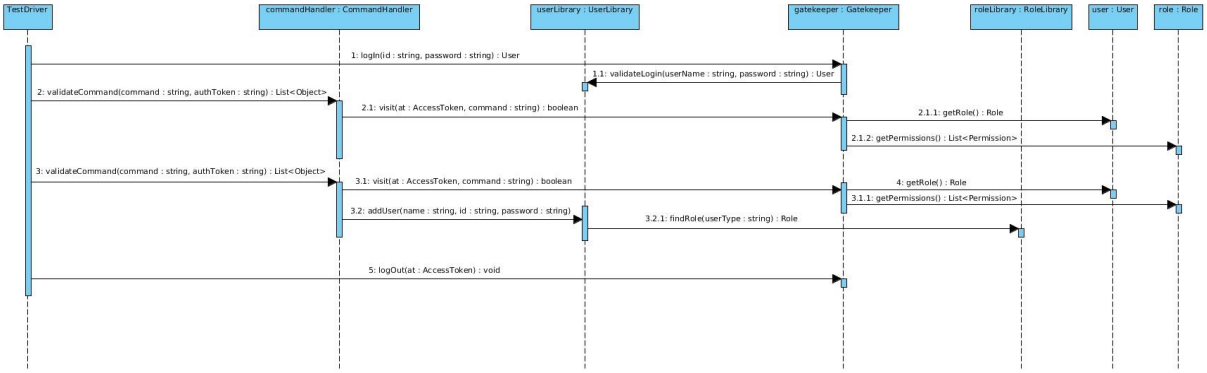


Fig 2: Admin Sequence Diagram

The Fig 2 sequence diagram demonstrates the process in which an admin would interact with the system.

1. The admin makes a log in request from the Gatekeeper.
2. The Gatekeeper uses the user name and password to verify that it is a legitimate user accessing the system by using the UserLibrary.
3. After confirmation, the admin can start passing commands into the system through the CommandHandler.
4. The command is then passed to the Gatekeeper for verification before it gets executed
5. The Gatekeeper then gets that users role, privileges and resources to confirm that the user has the required specifications for making such a command. However for the admin, the remaining checks are skipped after confirming the role.
6. Once verification is confirmed, the program continues on as normal.
7. If the command is to add a new occupant instance, then the Gatekeeper will first contact the UserLibrary to make sure that a new user instance is added to the system.
   a) A generated password, id (the occupants name), and voice voice pattern (--name--) are passed in along with the occupant name and type
      - The occupant type is what determines the role
8. The process continues until the admin logs out of the system.

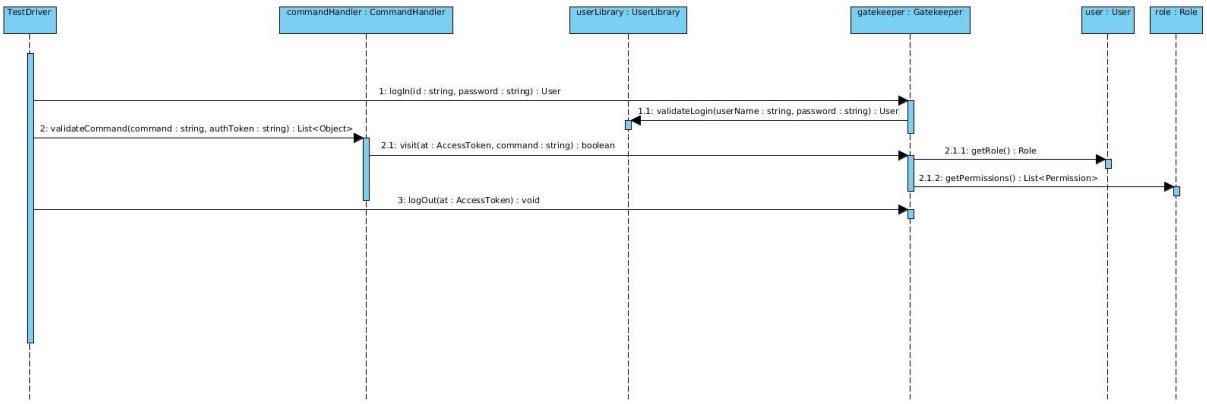


Fig 3: User Sequence Diagram

If you compare the diagram in Fig 3 to the admin sequence diagram (Fig 2), you will see that the two are very similar. Fig 3 shows the sequence of events for a normal user, and unlike the admin, a regular user cannot add a new occupant to the model. There is a greater chance of a command not being executed due to the user not having that resource or privilege.

Regardless of the additional restrictions between user and admin, the events are otherwise the same and continue to repeat until the user logs out.

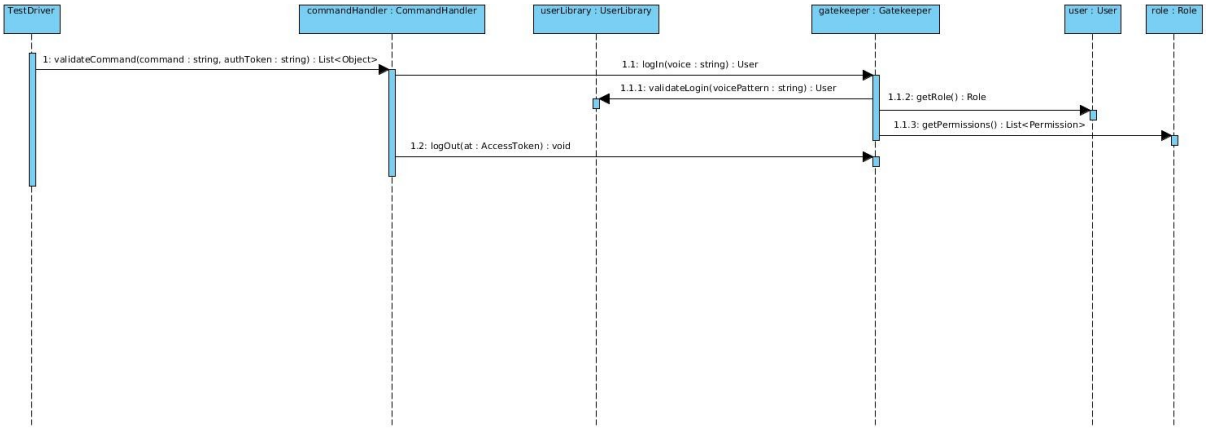


Fig 4: Voice Command Sequence Diagram

The log in/out method for vocal commands given through Ava is a little different (Fig 4).

1. A command gets passed in (most likely from the admin) to the CommandHandler
2. The CommandHandler checks to see if its a status update from an Ava stating that an occupant is requesting a status change of one or more IOT devices
3. Using the “voice pattern”, the CommandHandler contact the Gatekeeper with a log in request
4. If successful, the CommandHandler passes the command to the Gatekeeper for verification that the user has permission to make that particular request.
5. After the command is executed or denied, the CommandHandler contacts the Gatekeeper one more time to log out the user before continuing on.

Had there been more time, as well as a way to demonstrate, the Gatekeeper would have been multi-threaded. This would allow multiple users to access and interact with the system at the same time. There also would have been a better security system set up for maneging the passwords.

Design Patterns

Singleton Pattern:

The singleton pattern restricts instantiation of a class to one object. It is used prominently in this project by the following classes:

RuleEngine
CommandHandler
ControlerHandler
KnowledgeGraph
UserLibrary
Gatekeeper
DomainModel
Role[1]

1There are actually five instances of Role but each represent one type. As a result the Singleton pattern is used to make sure each role type is instantiated only one.

Factory Method Pattern:

The factory method is used to restrict the creation of certain classes to keep them within their parent or base class. KnowledgeGraph, UserLibrary, and DomainModel both uses this method for their sub classes.

Command Pattern:

The command pattern involves using an object to encapsulate all information needed to perform an action or trigger an event at a later time. In this project, ExecCommand is the object that contains the information required to change the status of a particular appliance and when to carry out the command.

Visitor Pattern:

The visitor pattern lets you preform an operation without affecting the element(s) passed in. The Gatekeeper class attempts to use this pattern when analyzing the commands.

Composite Pattern:

This pattern uses partitioning by composing objects into a tree like structure to represent a hierarchy. Classes using this method are known to carry attributes of the same class type. The Role class uses this pattern to keep the permissions organized.

Testing

Exception Handling:

Exceptions still function as normal. However the latest exceptions can best be tested by using the interactive feature of the program. There a person can log in as a user with non administrative privileges and test the exceptions by passing in commands that would activate them. Using a define, add, or remove command would be the easiest way to do this. You can also pass in a set command with the intention of triggering an Exception, but that requires remembering the user type (e.g. adult, child, etc) and what resource roles that user has.

Regression, and Performance:

Regression testing was limited to making sure that the House Mate Model System wouldn't execute any commands that the user doesn't have permission for.

The log system was used to make sure that new user instances were properly being generated. When a new occupant was added it would print to the file their name, id, password, and role.

For more in depth testing, JUnit is recommended since it will let the user know in which events did the expected result not match up with the value returned.

Functional:

Functional testing can easily be done by passing a file in as a argument. It is best to have a show command at the beginning and end of a block of set commands to show the changes. If you have interactive mode activated as well, then you could use a user that had been created to experiment with their privileges and resources.

Users can also be uploaded with a file using the method addUsersFromFile(AccessToken at, String fileName) in Gatekeeper and the will also be added to the model. A sure file called UserList.usr is currently provided. However in order to run the file, it needs to be passed in through the command line argument

(see README file for more details). If you want to make your own user file, then the folowing syntax should be used

User Syntax:

User::
name{
<name>
}
id{
<id>
}
password{
<password>
}
type{
admin/adult/child/pet/unknow
}

Risks

Aside from the fact that there are no security measures being used to handle the passwords the system currently assumes that when an occupant-house relation gets added that the User instance for the occupant already exists. Plans to add an Exception for that aspect is being planned. For now make sure that the occupant is already part of the model before adding such a relation.