House Mate Entitlement Service Requirements

Author: Eric Gieseke

Date: 10/25/2017

Introduction

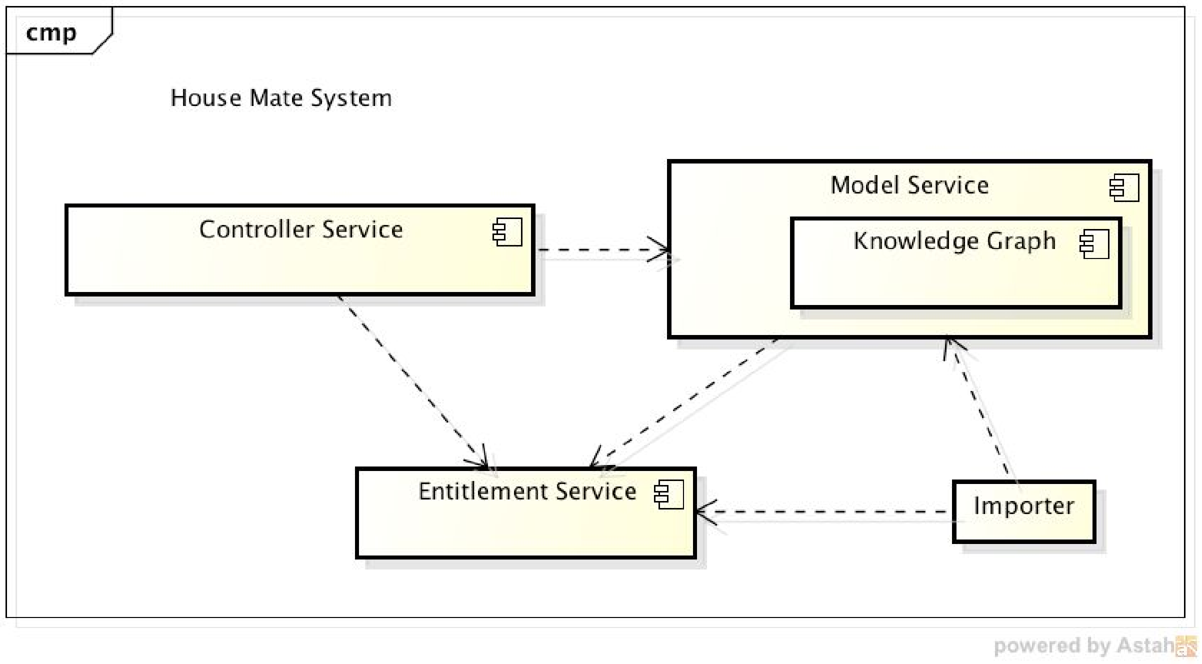
This document provides the requirements for the House Mate Entitlement Service.

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.

House Mate Integration



Caption: House Mate component diagram showing the Controller Service, Model Service, and Entitlement Service and their dependencies.

The Entitlement Service will be used to control access to the Model Service. The Model Service will be updated to use the Access Token passed in to each of the service methods and pass this to the Entitlement Service to validate that the associated user has the permission required to invoke the method.

When new Occupants are created, a corresponding user will be created within the Entitlement Service with a default voice template.

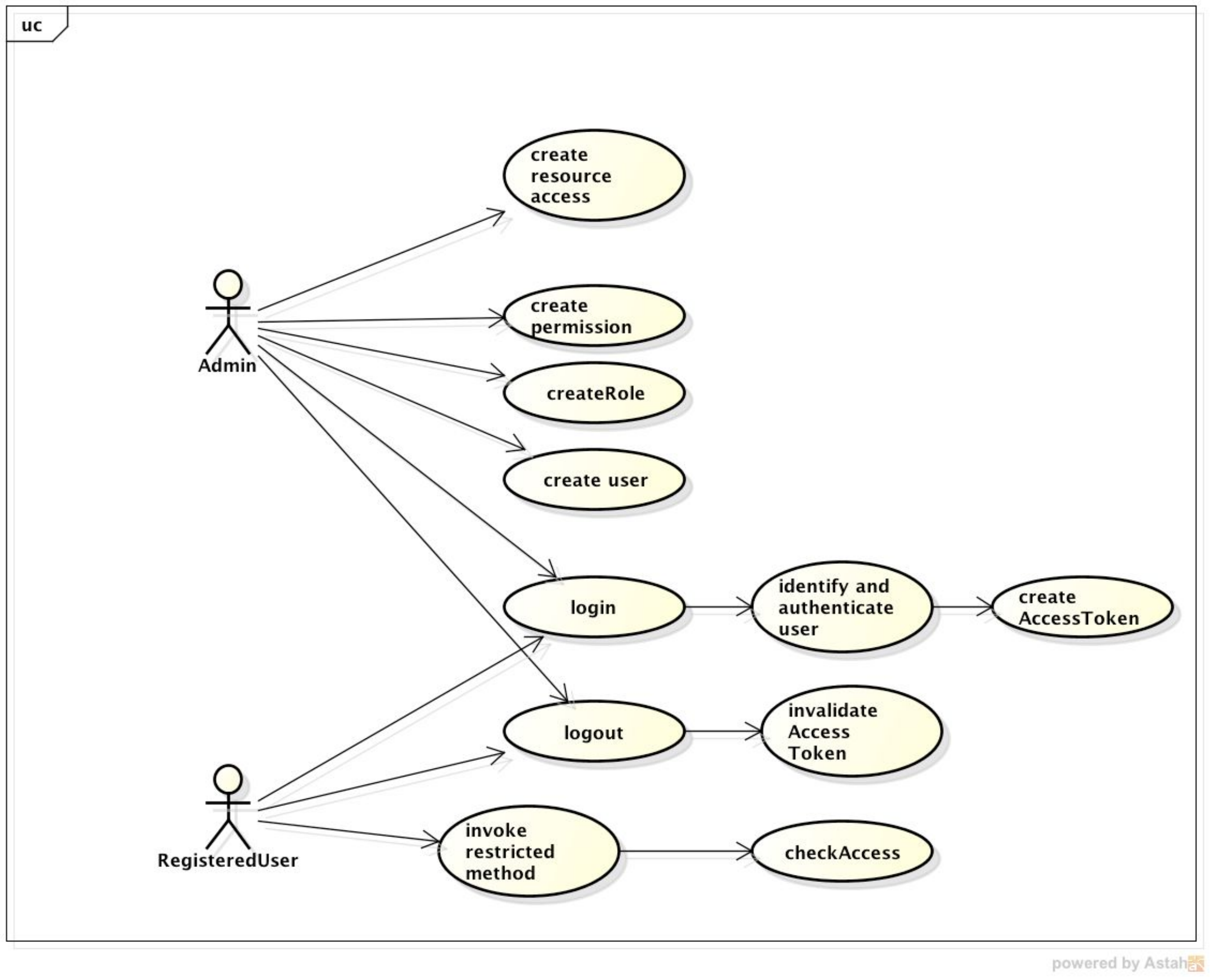
Also, when occupants are associated with a house, a Resource Role will be created within the Entitlement Service that binds the occupant with the house and the appropriate Role.

The command script processor will be extended to support the Entitlement Service commands as documented later in this document. Script commands must be preceded with a login request using an administrator username and password in order to obtain an Access Token.

The Control Service will use the Entitlement Service to request an authentication token to access the House Mate Service API. For voice commands received from the Ava device, the Controller Service will use a voice print to obtain an access token for the user making the request. The resulting Auth Token will be used for calling the House Mate Model Service methods. In this way, the entitlements associated with the Occupant making the request will be used to authorize or deny the request.

Entitlement Service

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, ResourceRoles, and Access Tokens.



Caption: Use Case Diagram for Entitlement Service

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 HouseMate system and has full access to all resources within the HouseMate System. Occupants are identified by their voice print.

Occupants use voice commands to interact with the HouseMate 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.

Entitlement Service API

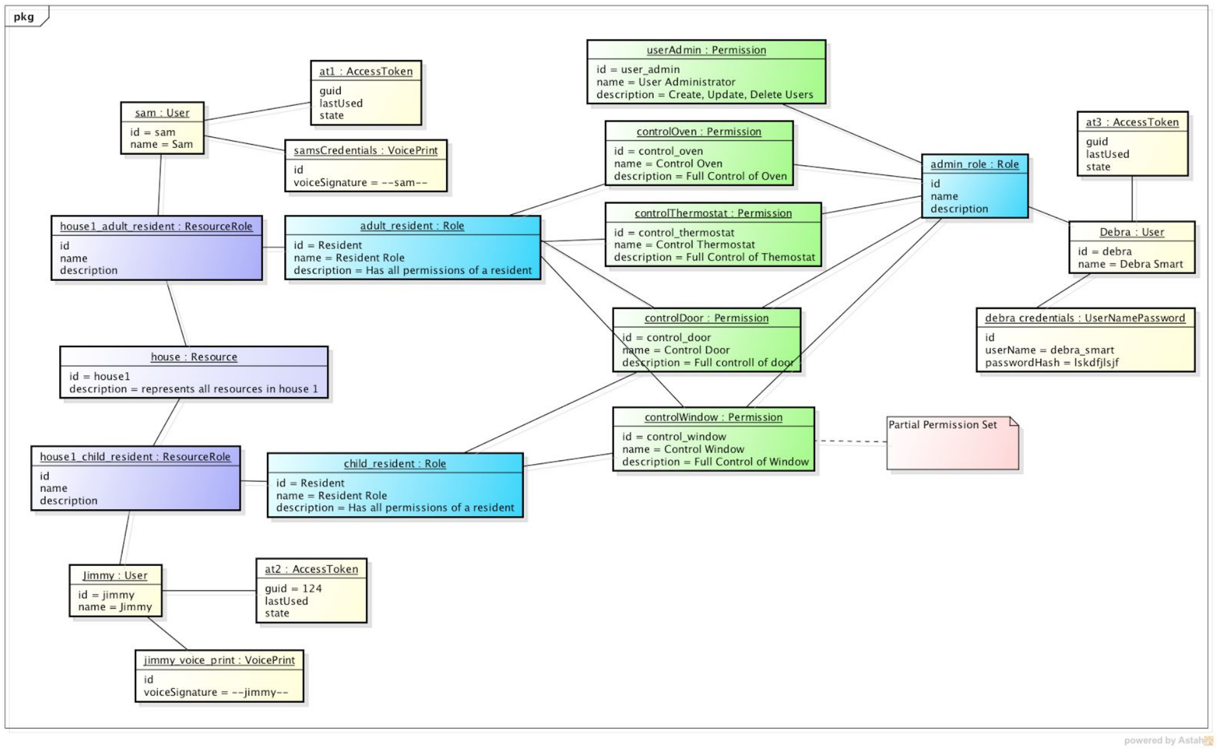
The Entitlement Service API supports the following functions:

- Creating Resources:
  - A Resource represents a physical and logical entity, for example a Sensor or Appliance.
  - A Resource has a unique ID, and a description.
- Creating Permissions:
  - Permissions represent an entitlement required to access a resource or function of the House Mate system.
  - Permissions have a unique id, name, and description.
  - A User may be associated with zero or more permissions.
- Creating Roles:
  - Roles are composites of Permissions.
  - Roles provide a way to group Permissions and other Roles.
  - Like Permissions, Roles have a unique id, name, and description.
  - Users may be associated with Roles, where the user has all permissions included in the Role or sub Roles.
  - Roles help simplify the administration of Users by providing reusable and logical groupings of Permissions and Roles.
- Creating Users:
  - Users represent registers users of the House Mate system.
  - Users have an id, a name and a set of Credentials. Credentials include a username and a password. To protect the password, the password should be hashed.
  - Users are associated with 0 or more Entitlements(Roles or Permissions).
- Authentication
  - The Authentication process provides users AccessTokens that can then be used to access restricted Service Methods.
  - If authentication fails, an AuthenticationException should be thrown.
  - If authentication succeeds, an AccessToken is created and returned to the caller.

    - The accessToken binds the User to a set of permissions that can be used to grant or deny access to restricted methods.
    - AccessTokens can timeout with inactivity.
    - AccessTokens have a unique id, an expiration time, and a state (active or expired).
    - Access tokens are associated with a User and a set of Permissions.
        - Login:
            - Login accepts a User’s credentials (username, password).
            - A check is made to make sure that the username exists, and then that the hash of the password matches the known hashed password.
        - VoicePrint
            - Voiceprint supports authentication through voice recognition
            - Voiceprint is simulated using a string in this format: “--<username>--”. For example the voice print for sam is “--sam--”.
            - The voice print signature is sufficient for identifying and authenticating a user.
- Logout:
    - Logout marks the given Access Token as invalid.
    - Subsequent attempts to use the AccessToken should result in a InvalidAccessTokenException.
- House Mate Model Service:
    - All methods defined within the House Mate Model Service should accept a AccessToken
    - Each method should validate that the AccessToken is non null and non empty.
    - The method should pass the accessToken to the Entitlement Service with the permission required for the method.
    - The AuthenticationService should check to make sure that the AccessToken is active, and within the expiration period, and then check that the user associated with the AccessToken has the permission required by the method.
    - The AuthenticationServce should throw an AccessDeniedException or InvalidAccessTokenException if any of the checks fail.

Exceptions should include useful information to help users understand the nature of the Exception.

The following instance diagram shows how the User, Role, Permission, Resource Role, AccessToken, and User Credentials are related.



Caption: Sample instance diagram for Entitlement Service.

Restricted methods include restricted methods on the Authentication, Provider and Renter Services.

Sample Authentication Data:

# define permissions

# define_permission, <permission_id>, <permission_name>, <permission_description>

define_permission, user_admin, “User Administrator”, “Create, Update, Delete Users”

define_permission, control_oven, “Control Oven”, “Full Control of Oven”

define_permission, control_thermostat, “Control Thermostat”, “Full Control of Thermostat”

define_permission, control_door, “Control Door”, “Full Control of Door” define_permission, control_window, “Control Window”, “Full Control of Window”

# define roles

# define_role, <role_id>, <role_name>, <role_description>

define_role, adult_resident, “Adult Resident Role”, “Has all permissions of an adult resident”

define_role, child_resident, “Child Resident Role”, “Has all permissions of a child resident”

define_role, admin_role, “Admin Role”, “Has all permissions of an administrator”

# add entitlement (permission, role) to role

# add_entitlement_to_role, <role_id>, <entitlement_id>

add_entitlement_to_role, admin_role, user_admin

add_entitlement_to_role, admin_role, control_oven

add_entitlement_to_role, admin_role, control_thermostat

add_entitlement_to_role, admin_role, control_door

add_entitlement_to_role, admin_role, control_window

add_entitlement_to_role, adult_resident, control_oven

add_entitlement_to_role, adult_resident, control_thermostat

add_entitlement_to_role, adult_resident, control_door

add_entitlement_to_role, adult_resident, control_window

add_entitlement_to_role, child_resident, control_door

add_entitlement_to_role, child_resident, control_window

# create_user

# create_user <user_id>, <user_name>

```
create_user, sam, Sam
create_user, jimmy, Jimmy
create_user, debra, “Debra Smart”
# add_user_credential
# add_user_credential <user_id>, <voice_print|password>, <value>
add_user_credential sam, voice_print, --sam--
add_user_credential jimmy, voice_print, --jimmy--
add_user_credential debra, password, secret



# add role to user
# add_role_to_user <user_id>, <role>
add_role_to_user debra, admin_role

# create resource role
# create_resource_role <resource_role_name>, <role>, <resource>
create_resource_role house1_child_resident,  child_resident, house1
create_resource_role house1_adult_resident, adult_resident, house1

 # add resource role to user
# add_resource_role_to_user <user_id>, <resource_role>
add_resource_role_to_user sam, house1_adult_resident
add_resource_role_to_user jimmy, house1_child_resident
```