HouseMate Entitlement Service Design Document

Date: 11/10/18
Author: Brandon T. Wood
Reviewers: Nicholas Whalen and Drew FitzGeraldz

Introduction

This document defines the design and implementation plans for the HouseMate Entitlement Service. This service monitors the activity of sensors and appliances tracked by the HouseMate Service and restricts access to certain automations and actions. It will secure the platform against unlawful access and also protecting users from functions that they should not have access to. For example a child should not be able to order beer or turn on the oven. Below we explain the requirements, use cases and implementation of this service and how it relates to the other services it interacts with.

Overview

This HouseMate Entitlement will limit and enforce access to the model service by way of the controller service. It will do this using authentication keys and by keeping a datastore of users associated with them. In this way it will protect the system and users from accessing features or functions that might be undesired or unsafe. The Entitlement service will integrate with the Controller service in order to inspect changes and prevent those that do not conform to the permissions we want to enforce.

Requirements

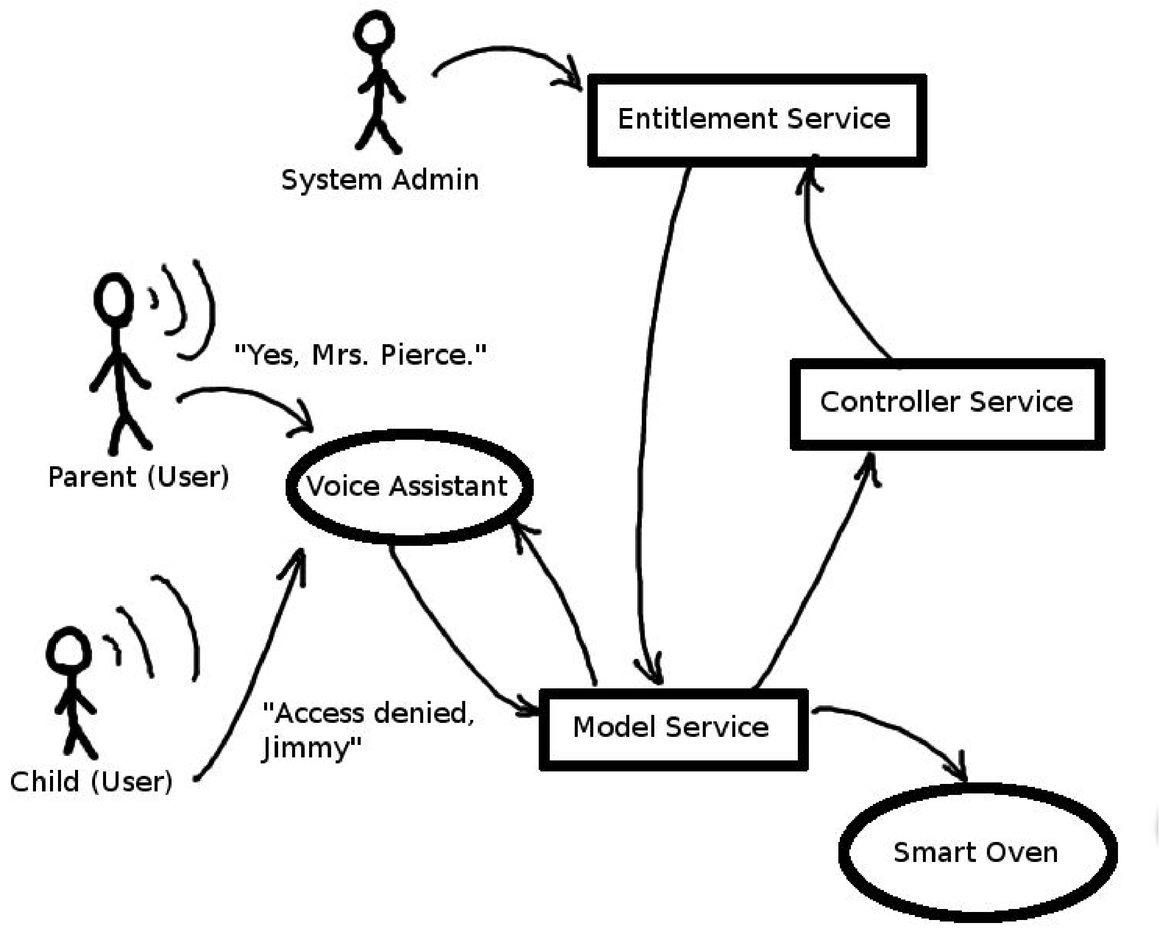
This service requires that we create:

1. Resources, locaically an extension of entities with unique UUIDs.
2. Permissions, housing a name, unique Id and description.
3. Roles, like permissions but assigned as a group.
4. Users, Represents users of the service with username and password.
5. Authentication, using access tokens required for actions to be completed

We should also be able to create and invalidate access tokens. The software also needs to make use of four different design patterns: singleton, abstract factory, composite and visitor.

Use Cases

The Entitlement Service is meant to block unwanted access or actions to the HouseMate system. Regular users who interact with the system mainly through the voice assistant give their vocal commands to the assistant. Who then passes the voice print credentials through to the Entitlement service via the model and controller services. If approved or denied the voice assistant will let the user know. Below we show a parent turning on the oven using the voice assistant, since they have permission to do so the voice assistant confirms this order with “Yes, Mrs. Pierce”. But when little Jimmy attempts the same his voice print is denied by the entitlement service so the voice assistant responds with “Access denied, Jimmy”.



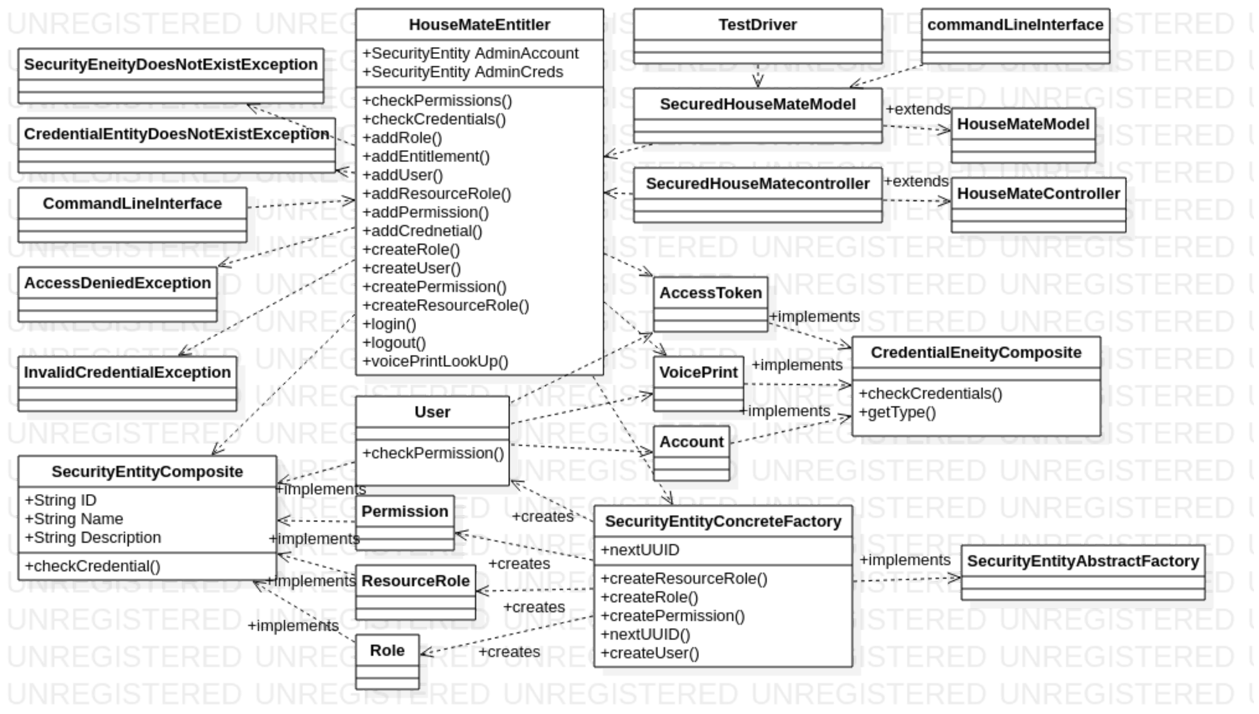
Additionally there is a CLI and API contained within the Entitlement Service for the System administrators to use. Here we have an Admin making some changes to the default roles and adding new permissions as part of an upgrade.

Implementation

In order to implement this based on previous requirements I started with the Entitler itself which made use of a concrete factory to produce the security entity composites: roles, users, permissions and resource roles. Which all used check permissions and would need to be tracked in a SecurityEntites map within the entitler service. Then I created a credential entity object and it's inheritors: VoicePrint, Account(username/passwords) and AccessTokens. These were needed for checking credentials, would be tied to the user object and also tracked in a series of maps within the entitler.

From here I had to integrate the existing HouseMateModel and HouseMateController with the Entitler Service. I created two decorator like classes around both of those classes prefixed with 'Secured', then reimplemented all the model methods with an additional AccessToken needed. This would allow me to secure any changes to being needed verification from the Entitler before they could be performed. Finally I added the needed exception classes, implemented auto login using default credentials for the Controller and created a CLI/file importer TestDriver.

Class Diagram



Class Dictionary

SecuredHouseMateModel Service

The Secured version of the Model decorates all the original methods of the Model but asks that you add an AccessToken that it always checks to be active before it continues. Also when a new occupant is defined a new user is created for them in the entitler and when an occupant is added to a house it updates the resource roles.

Property Name	**Type**	**Description**
titler	HouseMateEntitler	Singleton instance of the housemateentitler, allows for the authentication and tracking of user credentials.

SecuredHouseMateController Service

The only major changes from the base classes are that the controller service needs to login and validate it's credentials before executing it's rules. Also the AvaRules functionality had to be extended to support new requirements around the use of VoicePrint.

Property Name	**Type**	**Description**
token	AccessToken	Holds the current token for accessing the Model.
titler	HouseMatEntitler	Used to check credentials for accessing the model. Also generates a token.

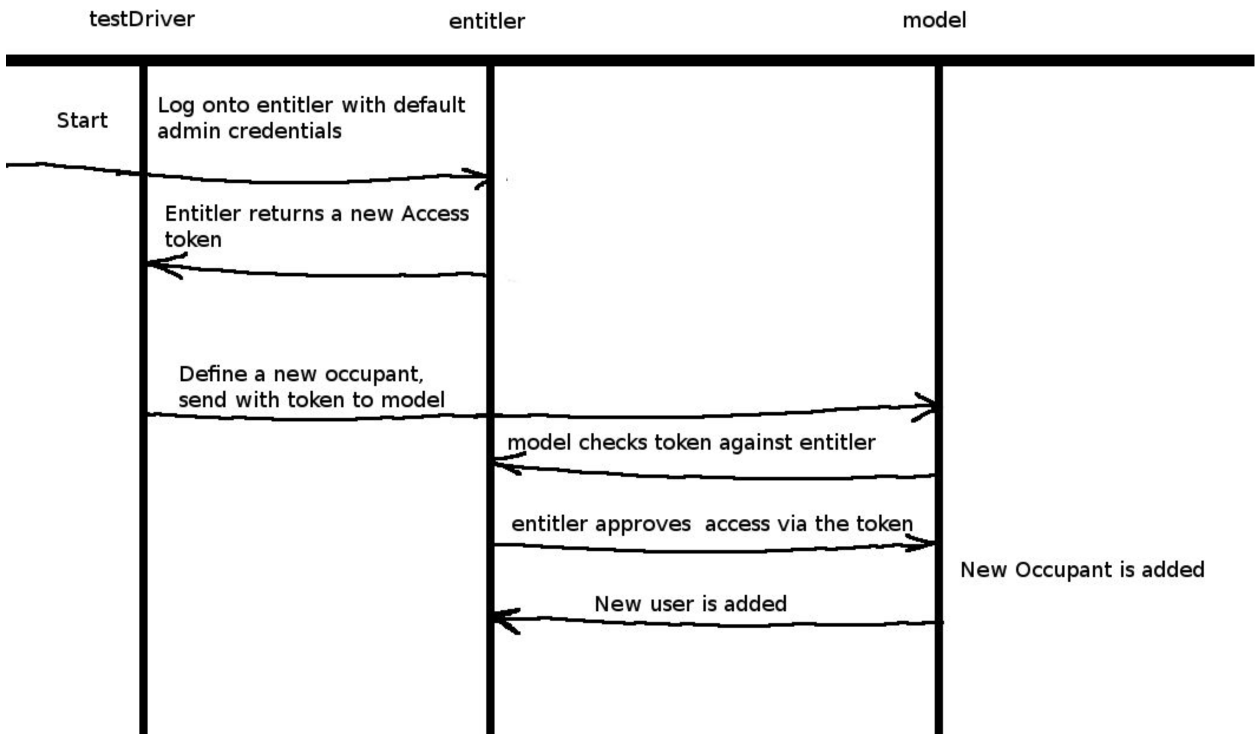
HouseMateEntitler

Method Name	**Signature**	**Description**
checkPermissions	boolean	Charged with taking a user and a description of a permission, then checking if that user has access to perform said description.

checkPermission	boolean	Charged with taking a userID and figuring out if this user has permission to perform the task passed in.
createUser	void	Adds a new user to the security role, based on house also adds associated ResourceRoles and security tokens to the user.
createRole	void	Adds a new security role to the entitlement service.
createEntitlement	void	Adds a new entitlement to the entitlement service.
createResourceRole	void	Adds a resource role to the entitlement service, a default role given to users upon their creation. Based on House rules.
createPermission	void	Adds a permission to the entitlement service. The base unit of authentication, they are the rules that restrict access to the model.
addRole	void	Takes a user Id and a role Id, makes it known to the system that this user has this role. Many to many relationship.
addPermissions	void	Takes a role Id and a permission Id, then makes it known that anyone with this role has this associated permission.
addEntitlement	void	Takes a role Id and an entitlement id, associates the role to the entitlement.
addResourceRole	void	Takes a userId and a HouseID, adds any corresponding resource role from the house to the user.
addCredentials	void	Takes a user id and a type/value, generates a new credential and adds it to the user given.
login	AccessToken	Given a userID and a password, checks if the password matches that users. If so returns a brand new access token for use.
logout	void	Invalidates given access token.
voicePrintLookUp	void	Takes a voice print, if that print is valid returns an Access Token.

Property Name	Type	Description
AdminUser	User	Stores the default admin user values.
DefaultResourceRole	ResourceRole	Stores the default resource role given to houses.
factory	SecurityEntityConcreteFactory	Used in the creation of all the securityentities.
instance	HouseMateEntitler	The singleton instance of the Entitler service.
SecurityEntities	HashMap	Stores all current SecurityEntites.
ResourceRoles	HashMap	Stores all current resource roles.
AccessTokens	HashMap	Stores all the currently active accesstokens.
VoicePrints	HashMap	Stores all the current voiceprints.

Sequence Diagram


testDriver
entitler
model
Start
Log onto entitler with default admin credentials
Entitler returns a new Access token
Define a new occupant, send with token to model
model checks token against entitler
entitler approves access via the token
New Occupant is added
New user is added

Exception Handling

There are three current throwable exceptions, the first being *InvalidCredentialsException* which is thrown in the case of receiving invalid credentials for a user or when trying to convert a voice print over to a credential that fails. The second is *AccessDeniedException* which is thrown when the permissions of a user do not match the action they are trying to perform which we block them from. The last being SecurityEntityDoesNotExistException, a fairly long name, only used when a Security Entity does not exist.

Testing

Included with the TestDriver class is a FunctionalityTestDriver that proves the functionality of validating credentials, checking permissions and there exceptions with commands that are meant to be caught or fail. If you want there is also an implemented commandLineInterface class for the entitler if you run that class as main. It will first ask you for login so give it Admin:trustNo1 which are the default credentials. Then you can freely input the desired functions from the requirements page.

Risks

This project requires a default admin account with a default password, this is a known area of exploitation by hackers or intruders. This software also uses plain text passwords for the CLI, meaning anyone with access to the git history of the user’s machine could find the passwords. Both could be fixed if we used some form of input masking, made the passwords for admins random and then forced users to change them on setup.