Script commande

14/06/2023

Table des matières

Installation du service système....3
Installation des dépendances systemes....3
Création de l'environnement python3....3
Initialisation du service système....5
Activation du service au démarrage système....5
Arrêt démarrage du service système....6
Démarrage manuel du service....6
Arrêt manuel du service....6
Configuration du service....6
Fichier de configuration config.json....6
Section «mqtt»....6
Section «psql»....7
Section «ems»....7
Section «coordination»....7
Section «logging»....9
Fichier de configuration systemd....12
Définition des utilisateurs et mot de passe....12
Structure du programme....14
Généralités....14
Schéma de principe général....14
Description....15
Module broker.py....17
Traitement des messages en provenances des écrans....18
Traitement des messages en provenances de virya....18
Schéma général des traitements en réception....19
Module ems.py....20
Cycle d'exécution de l'ems....20
Schéma général des cycles EMS....21
Module emsbroker.py....22
Structure générale des typologies....23
Structure générale des drivers....24
Scénarios d'évolutions....26
Ajout d'un équipement domotique....26
Ajout d'une interface utilisateur....26
Ajout d'une typologie....26

Astrolabe CAE

Installation du service système

L'installation est prévue pour un système linux Debian de version 9 ou supérieure.

Installation des dépendances systemes

sudo apt-get install git-all

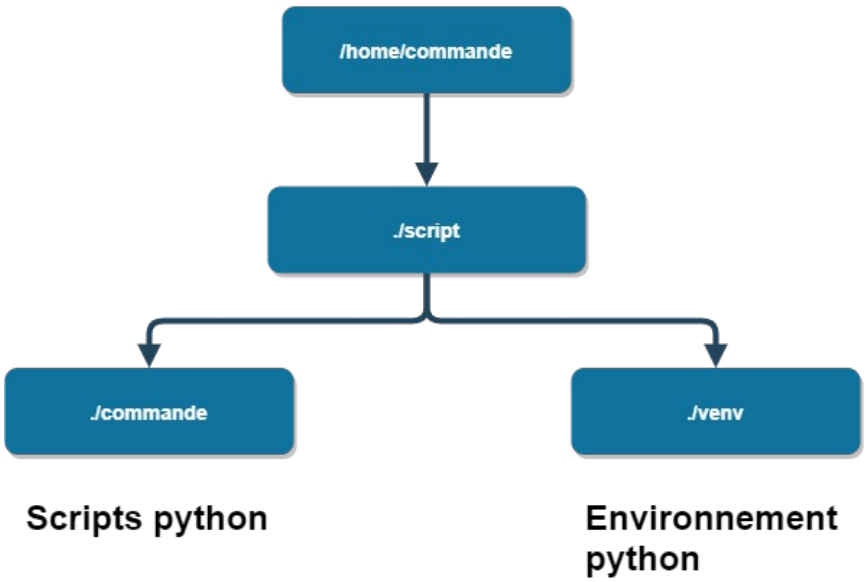
sudo apt-get install python3-venv

sudo apt-get install python3-dev libpq-dev build-essential

Création de l'environnement python3

Le script est prévu pour fonctionner dans un environnement virtuel python3. Le script est placé dans le répertoire ~/script/commande de l'utilisateur **commande.** Le script peut être déplacé sans difficulté, mais il conviendra dans ce cas de modifier les chemins du fichier de démarrage de service **systemd** (elfecommande.service).

La structure de répertoires du projet est la suivante :



Astrolabe CAE

Création du répertoire projet

```
mkdir script
```

```
cd script
```

Création de l'environnement virtuel :

```
python3 -m venv venv
```

Activation de l'environnement :

```
source ./venv/bin/activate
```

Installation des dépendances nécessaires au script dans l'environnement virtuel :

```
pip install psycopg2
```

```
pip install paho-mqtt
```

Notes : les versions de psycopg2 et paho-mqtt au moment de la réalisation sont les suivantes :

Paho-mqtt	1.6.1
Psycopg2	2.9.6

Installation des scripts python proprement dit :

à partir du répertoire /*home/*commande/script

```
git clone https://github.com/Energies-citoyennes-en-Pays-de-Vilaine/commande.git
```

Astrolabe CAE

Initialisation du service système

Copier le fichier elfecommande.service dans le repertoire systeme /*etc/*systemd/system

```
sudo cp elfecommande.service /etc/systemd/system/elfecommande.service
```

Configurer les utilisateurs et les mots de passe MQTT et PostgresQL dans le fichier /etc/systemd/system/elfecommande.service

dans la section Service :

- Verifier les chemins d'accès au script de commande dans les variables ExecStart et WorkingDirectory
- Ajouter les variables d'environnements **commande_db_pass**, **commande_db_user** pour l'accès au serveur PostgreSQL
- Ajouter les variables d'environnements **commande_broker_user**, **commande_broker_pass** pour l'accès au broker MQTT.

Extrait de la section service du fichier /etc/systemd/system/elfecommande.service :

```
[Service]
ExecStart=/home/commande/script/venv/bin/python3 -m commande.py
WorkingDirectory=/home/commande/script/commande
Environment="commande_db_pass=blabla"
Environment="commande_db_user=usermachin"
Environment="commande_broker_pass=blabla"
Environment="commande_broker_user=usertruc"
```

Recharger la liste des daemons systèmes :

```
sudo systemctl daemon-reload
```

Activation du service au démarrage système

```
sudo systemctl enable elfecommande.service
```

Arrêt démarrage du service système

Démarrage manuel du service

```
sudo systemctl start elfecommande.service
```

Arrêt manuel du service

```
sudo systemctl start elfecommande.service
```

Configuration du service

Fichier de configuration config.json

Le fichier de configuration config.json doit être positionné dans le répertoire du script et correspondre au répertoire défini dans la variable **WorkingDirectory** du fichier de définition systemd

Section «mqtt»

défini les options associées à la connexion avec le broker MQTT

option	description	exemple
host	Adresse IP ou nom d'hôte du broker mqtt	192.168.30.100
port	Port de connexion avec le broker mqtt	1883
keepalive	Temps en secondes entre deux tests de connexions	15
maxretry	Nombre de tentatives de connexion en cas d'échec de connexion	360

retrydelaysec	Delai entre deux tentatives de connexion après un échec.	10

Section «psql»

défini les options associées à la connexion avec le serveur de base de données Postgresql pour les données de coordination.

option	description	exemple
host	Adresse IP ou nom d'hôte du serveur	192.168.30.118
port	Port de connexion avec le serveur	5432
database	Base de données par défaut	preprod_bdd_coordination

Section «ems»

défini les options associées aux tables de sorties de l'ems dans la base de données.

option	description	exemple
database	Base de données de la table de sortie	preprod_sortie_ems
table	Nom de la table comportant les données de sorties de l'ems	result
thread	Nombre de thread de traitement en parallèle	1

Section «coordination»

défini les options associées aux tables de données de coordination dans la base de données.

option	description	exemple
database	Nom de la base de données de coordination	preprod_bdd_coordination
equipement_domotique_table	Nom de la table comportant les équipements domotiques	bdd_coordination_schema.equipement_domotique

equipement_domotique_type_table	Nom de la table comportant les types d'équipements domotiques	bdd_coordination_schema.equipement_domotique_type
equipement_domotique_table_root	Racine des noms de tables contenant les détails des équipements domotiques	bdd_coordination_schema.equipement_domotique_
equipement_pilote_ou_mesure_table	Nom de la table des équipements pilotés	bdd_coordination_schema.equipement_pilote_ou_mesure
equipement_pilote_machine_generique	Nom de la table des équipements pilotés génériques	bdd_coordination_schema.equipement_pilote_machine_generique
equipement_pilote_typologie_installation_domotique	Nom de la table des typologies d'équipements d'installation domotique	bdd_coordination_schema.equipement_pilote_typologie_installation_domotique"
equipement_pilote_vehicule_electrique_generique	Nom de la table des détails concernant les véhicules électriques	bdd_coordination_schema.equipement_pilote_vehicule_electrique_generique

permanent_topic	Liste des topics MQTT écoutés en permanence. Le topic des évenements écrans et viriya	`[`
		`"hasp/#",`
		`"viriya/#"`
		`]`

screen_usage_assoc	Liste des associations entre numéro de page d'un écran et typologie d'installation domotique	`{`
		`"2":[111],`
		`"3":[112],`
		`"4":[113],`
		`"5":[221],`
		`"6":[131],`
		`"7":[151,155]`
		`}`

screen_indicator_topic	Nom du topic mqtt pour envoyer une requête de rafraîchissement des horaires pour un écran	`INDICATEURS/refresh`
screen_connexion_page	Numero de la page du status de	9

	connexion sur les écrans	
screen_connexion_label_id	Identifiant du label de connexion sur la page connexion	8
screen_connexion_label	Texte envoyé aux écrans comme status de connexion	écran connecté
screen_connexion_label_color	Couleur (RVB) du texte de status de connexion	#002080
screen_status_page		8
screen_led_id	Identifiant du voyant « led » sur les pages des écrans	10
screen_status_led	Liste des associations entre page d'écran et identifiant du voyant « led » sur la page de résumé	{ "2":12, "3":13, "4":14, "5":15, "6":16, "7":17 },
equipement_continu	Liste des typologies d'équipements pilotés qui fonctionne en mode « continu »	[131,151,155]

Section «logging»

défini les options associées au niveau de journalisation des messages dans le fichiers log système /var/log/syslog.

option	description	exemple
level	Niveau des messages de log journalisé dans le journal systéme. Les niveaux possibles sont : CRITICAL ERROR	INFO

	WARNING INFO DEBUG	

Fichier de configuration actuel (preprod)

```
{
    "mqtt": {
        "host": "192.168.30.100",
        "port": 1883,
        "keepalive":15,
        "maxretry":360,
        "retrydelaysec":10
    },
    "pgsql": {
        "host": "192.168.30.118",
        "port": "5432",
        "database":"preprod_bdd_coordination"
    },
    "ems": {
        "database":"preprod_sortie_ems",
        "table":"result"
        "thread":1
    },
    "coordination": {
        "database":"preprod_bdd_coordination",

"equipement_domotique_table":"bdd_coordination_schema.equipement_domotique",

"equipement_domotique_type_table":"bdd_coordination_schema.equipement_domotique_type",

"equipement_domotique_table_root":"bdd_coordination_schema.equipement_domotique_",
```

```
"equipement_pilote_ou_mesure_table":"bdd_coordination_schema.equipement_pilote_ou_mesure",

"equipement_pilote_machine_generique":"bdd_coordination_schema.equipement_pilote_machine_generique",

"equipement_pilote_typologie_installation_domotique":"bdd_coordination_schema.equipement_pilote_typologie_installation_domotique",

"equipement_pilote_vehicule_electrique_generique":"bdd_coordination_schema.equipement_pilote_vehicule_electrique_generique",


      "permanent_topic":[
      "hasp/#",
      "viriya/#"
      ],
       "screen_usage_assoc":{
            "2":[111],
            "3":[112],
            "4":[113],
            "5":[221],
            "6":[131],
            "7":[151,155]
       },
       "screen_indicator_topic":"INDICATEURS/refresh",
      "screen_connexion_page":9,
      "screen_connexion_label_id":8,
      "screen_connexion_label":"ecran connecté",
       "screen_connexion_label_color":"#FF2080",
      "screen_status_page":8,
        "screen_led_id":10,
        "screen_status_led":{
            "2":12,
```

```
            "3":13,
            "4":14,
            "5":15,
            "6":16,
            "7":17
        },
        "equipement_continu": [131,151,155]
    },
    "logging": {
        "level":"INFO"
    }
}
```

Fichier de configuration systemd

Le fichier défini l'utilisation du script par le système d'exploitation Debian. Cependant les script défini également les utilisateurs / mot de passe pour l'accès au serveur PostgresSQL et MQTT.

Définition des utilisateurs et mot de passe

Les utilisateurs et mots de passe des services PostgreSQL et mqtt sont des variables d'environnements système. Ces variables sont défini dans la section « [Service] » du fichier de configuration systemD

```
[Service]
Environment="commande_db_pass=blabla"
Environment="commande_db_user=usermachin"
Environment="commande_broker_pass=blabla"
Environment="commande_broker_user=usermachin"
```

Variable	Rôle
commande_db_user	Nom d'utilisateur pour la connexion PostgreSQL
commande_db_pass	Mot de passe de l'utilisateur PostgreSQL
commande_broker_user	Nom d'utilisateur pour la connexion MQTT
commande_broker_pass	Mot de passe pour la connexion MQTT

le fichier complet :

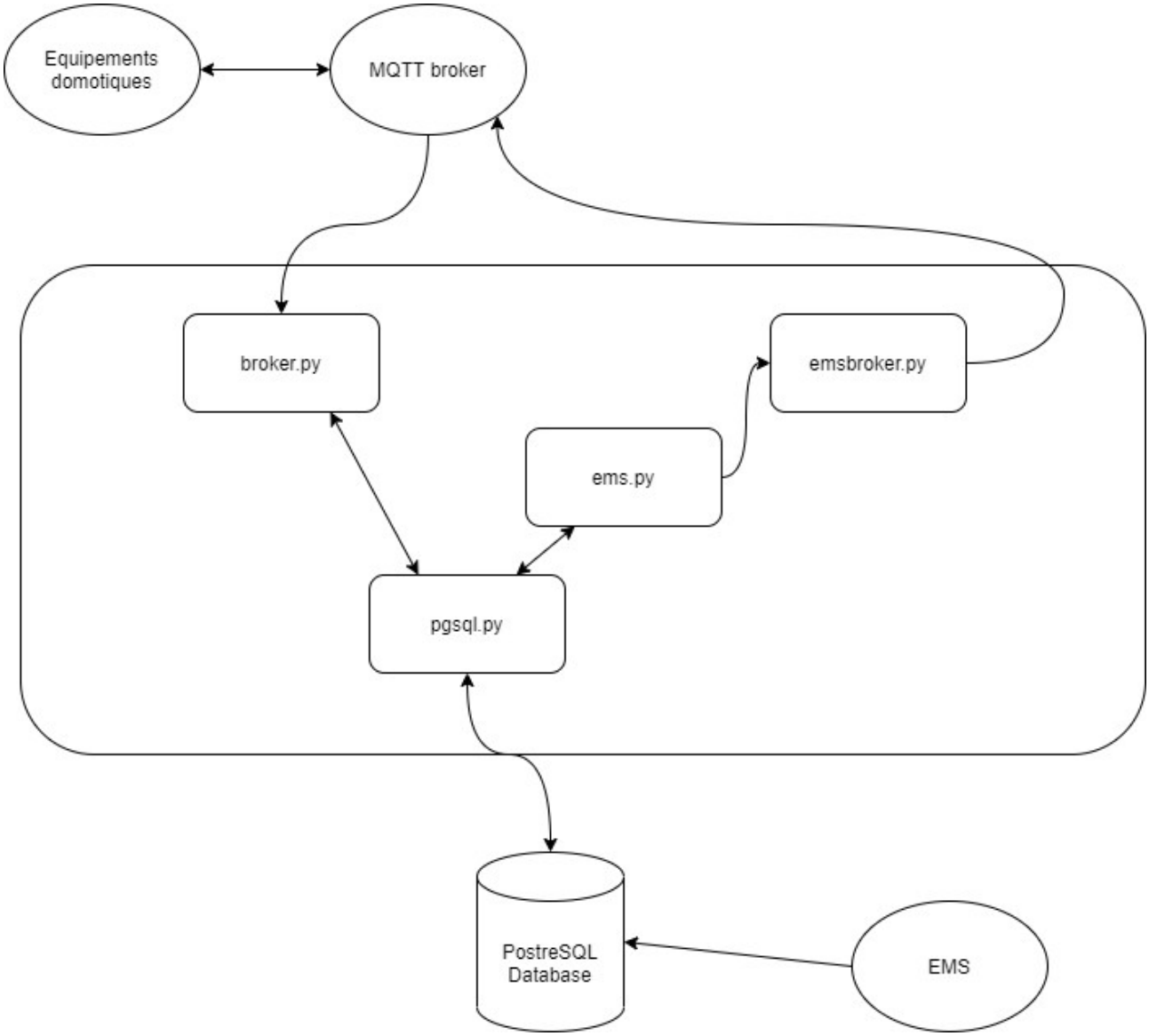
```
[Unit]
Description=ELFE Python Service
After=network.target

[Service]
ExecStart=/home/commande/script/venv/bin/python3 -m commande.py
WorkingDirectory=/home/commande/script/commande
Environment="commande_db_pass=xxxx"
Environment="commande_db_user=xxxx"
Environment="commande_broker_pass=xxxx"
Environment="commande_broker_user=xxxx"

[Install]
WantedBy=multi-user.target
```

Astrolabe CAE

Structure du programme

Généralités

Schéma de principe général



Description

De manière schématique, le script commande comporte 4 éléments principaux. Les modules broker.py, ems.py et emsbroker.py s'exécutent dans des process (thread) séparés.

broker.py

Gère l'interfaçage avec le broker MQTT pour les messages provenant des interfaces utilisateurs viriya et écrans. En particulier le module broker.py écoute en permanences les topics mqtt permanents définis dans le fichier de configuration.

Sur réception d'un message, il route le message vers le driver adéquat pour réaliser le traitement demandé par l'interface utilisateur (mise à jour Viriya, utilisation d'un écran)

ems.py

Gère l'interface avec les données de sorties de l'EMS. Periodiquement, le module interroge la table sortie de l'EMS dans la base de données PostgreSQL . Pour chaque équipement domotique présent dans la table, le module ems charge la typologie de l'équipement domotique et exécute les commandes adéquates en fonction de l'activation ou non de l'équipement par l'EMS.

Le module dispose d'un mécanisme de mémorisation des données de sortie de l'EMS. En cas de défaillance de la connexion avec la base de données de l'EMS, le module ems.py prend en compte les informations transmises au cycle précédent. Les données de sorties de l'EMS décrivent l'activation ou la désactivation des équipements pilotés pour une durée de 24H avec des cycles de 15 minutes.

Le module déclenche l'analyse des données de l'EMS 30 secondes après sont démarrage, puis à chaque frontière de 15 minute (à 12h00 12h15 12h30 ...)

emsbroker.py

Gère l'interface avec le broker MQTT pour les messages à destination et provenant des équipement domotiques. Sur demande des modules broker.py ou ems.py le script transmets les

messages aux équipements domotique et fournit un service d'écoute temporaire des équipements pour traiter les réponses aux commandes.

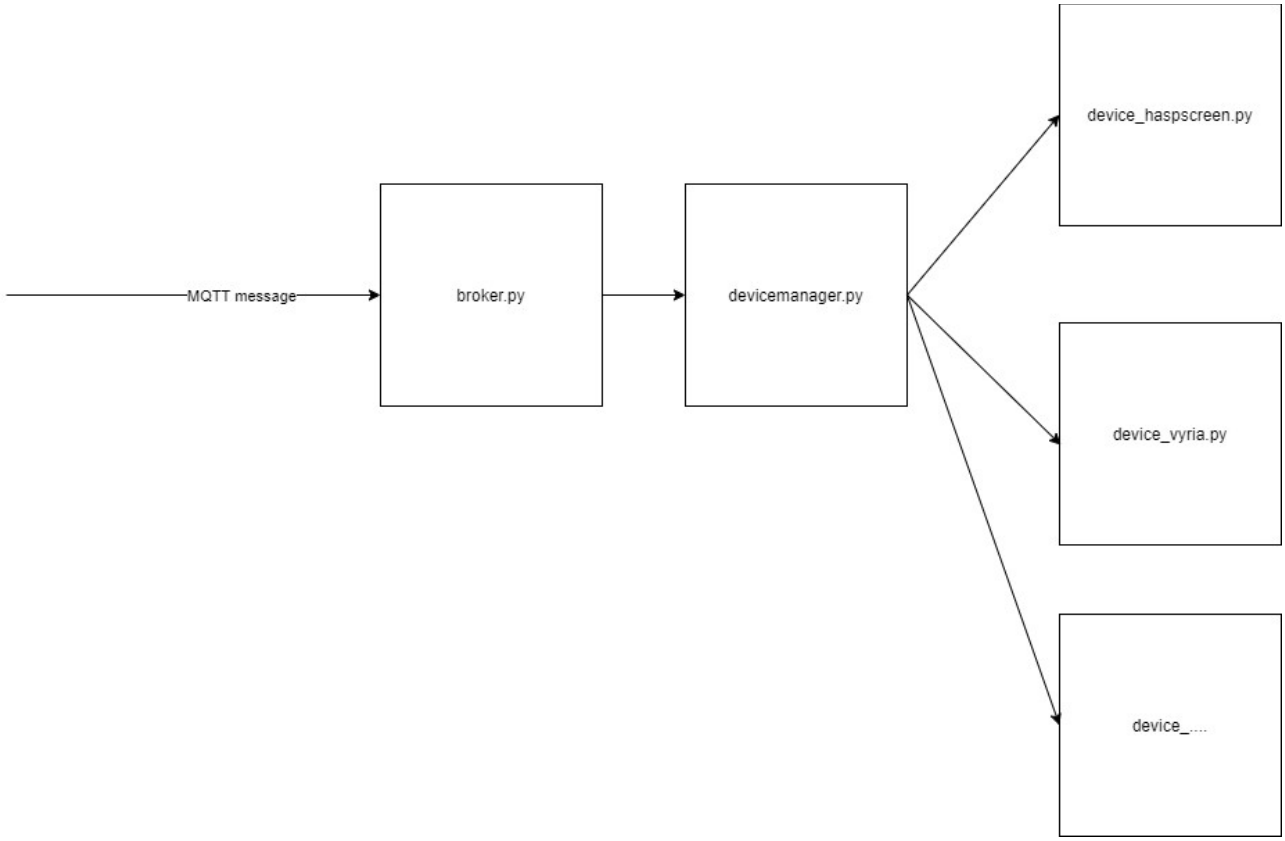
pgsql.py

Gère l'interface avec le serveur PostgreSQL. Transmet les demandes d'exécution de requêtes SQL et retourne les résultats fournis par le serveur SQL.

Module broker.py

Le module gère l'interface des interfaces utilisateurs a travers le broker MQTT.

Sur réception d'un message, le topic est analysé et transmis au DeviceManager (manager.py). En fonction de la racine du topic, le DeviceManager transmet le message au drivers concerné



Traitement des messages en provenances des écrans

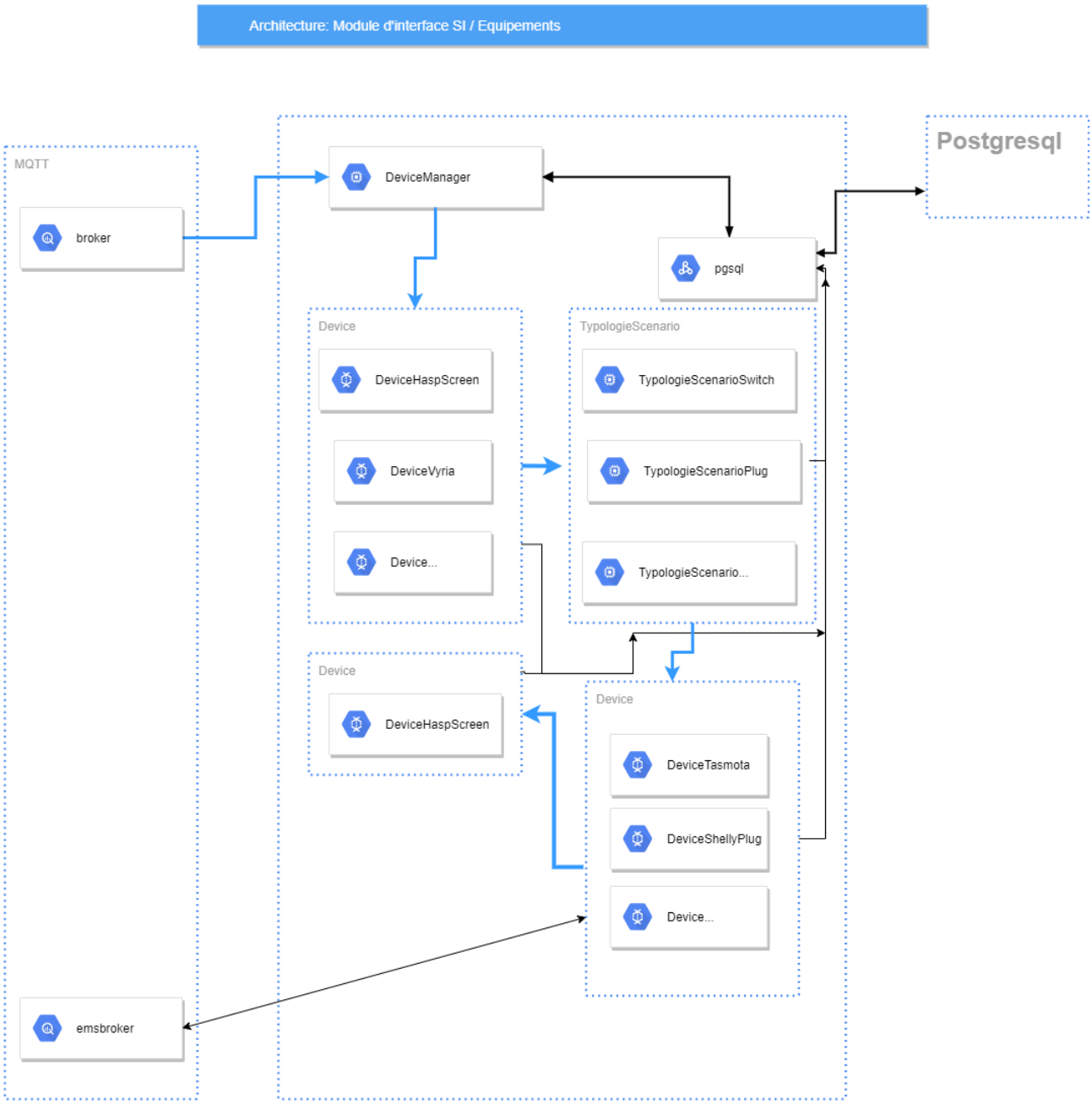
1. Extraction des informations concernant la page, l'identifiant de l'élément de la page et le n° de série à partir du message MQTT.
2. Recherche de l'utilisateur associé à l'écran dans la table des équipements domotiques.
3. Recherche de l'équipement piloté à partir de l'utilisateur et la typologie associé à la commande de l'écran
4. Chargement et initialisation de la typologie
5. Chargement des équipements domotiques
6. Exécution de la typologie pour activer ou désactiver le mode piloté
7. Recherche de l'écran associé à l'utilisateur
8. Mise à jour des information de l'écran

Traitement des messages en provenances de virya

1. Extraction de l'identifiant d'équipement piloté à partir du message MQTT
2. Recherche de l'utilisateur associé à l'équipement piloté
3. Chargement et initialisation de la typologie
4. Chargement des équipements domotiques
5. Exécution de la typologie pour activer ou désactiver le mode piloté à partir des informations en base de données
6. Recherche de l'écran associé à l'utilisateur
7. Mise à jour des informations de l'écran

Astrolabe CAE

Schéma général des traitements en réception



Module ems.py

Le module gère le traitement périodique des données en sortie de l'ems. Toute les 15 minutes le module lit la table résultat de l'ems :

- Si des nouvelles données sont disponibles, le module met à jours les informations dans un cache interne
- En cas d'échec de lecture, le module exécute un cycle avec les données en cache

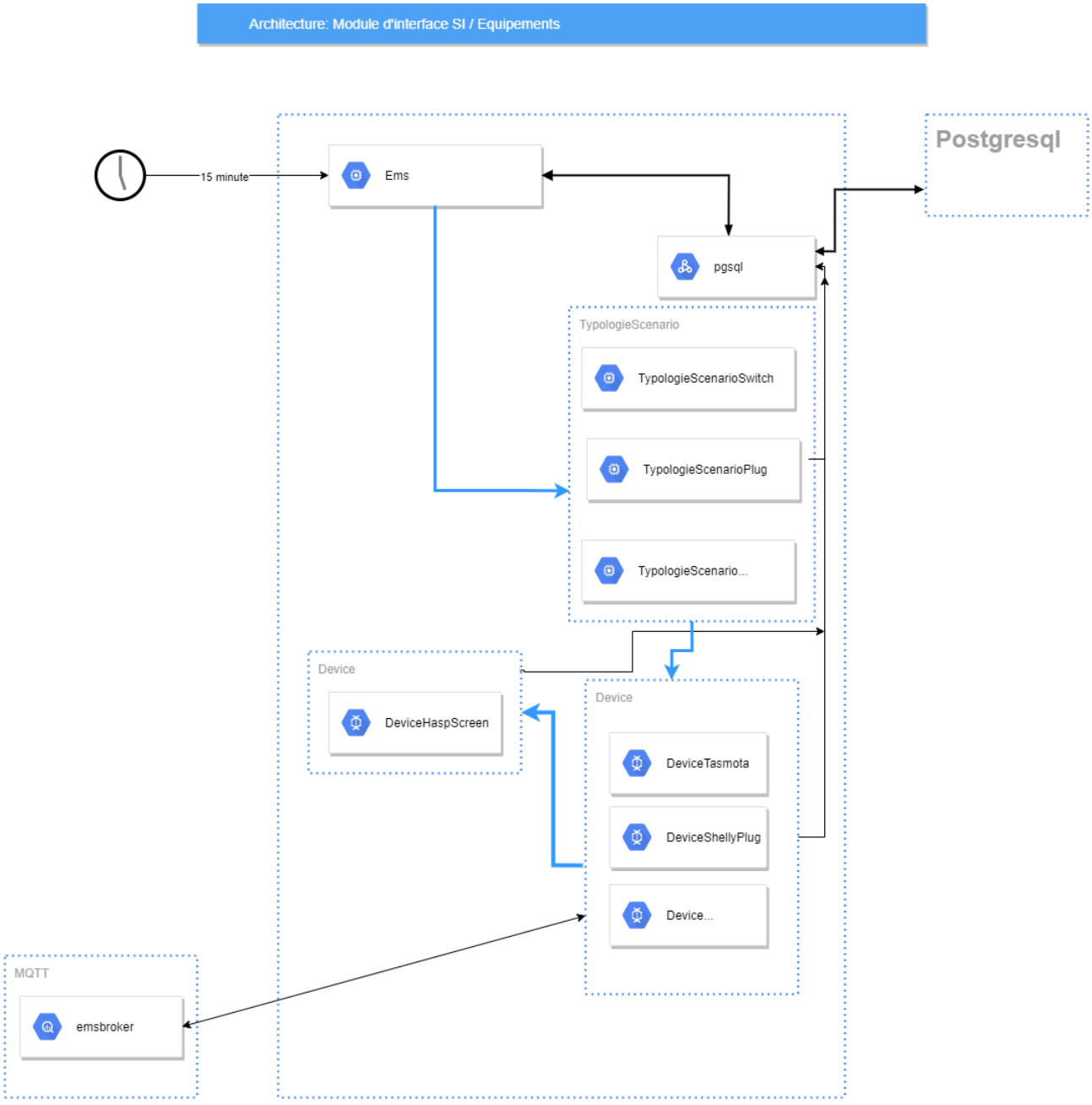
Le module exécute ensuite un cycle complet

Cycle d'exécution de l'ems

Pour chaque équipements pilotés, le module vérifie si il a des consignes en provenance de l'ems. Si une consigne est présente (on ou off) et que l'équipement est en mode piloté. Le cycle suivant est exécuté :

1. Chargement de la typologie associée à l'équipement piloté
2. Initialisation de la liste des devices correspondants au équipements domotique associés à l'équipement piloté.
3. Exécution de la typologie en fonction de la consigne de l'ems (on ou off) et du mode de la typologie (continue ou non)
4. La typologie détermine le device associé à l'équipement domotique et demande au device d'exécuter la commande.
5. Le device utilise le module emsbroker pour s'inscrire temporairement au topic MQTT de transmission des messages de réponse de l'équipement domotique.
6. Le device transmet les ordres nécessaires à l'équipement domotiques
7. Le device analyse les messages de réponses du device
8. Le device supprime l'inscription temporaire au topic MQTT
9. la typologie met à jour la machine à état de l'équipement piloté en base de données

Schéma général des cycles EMS



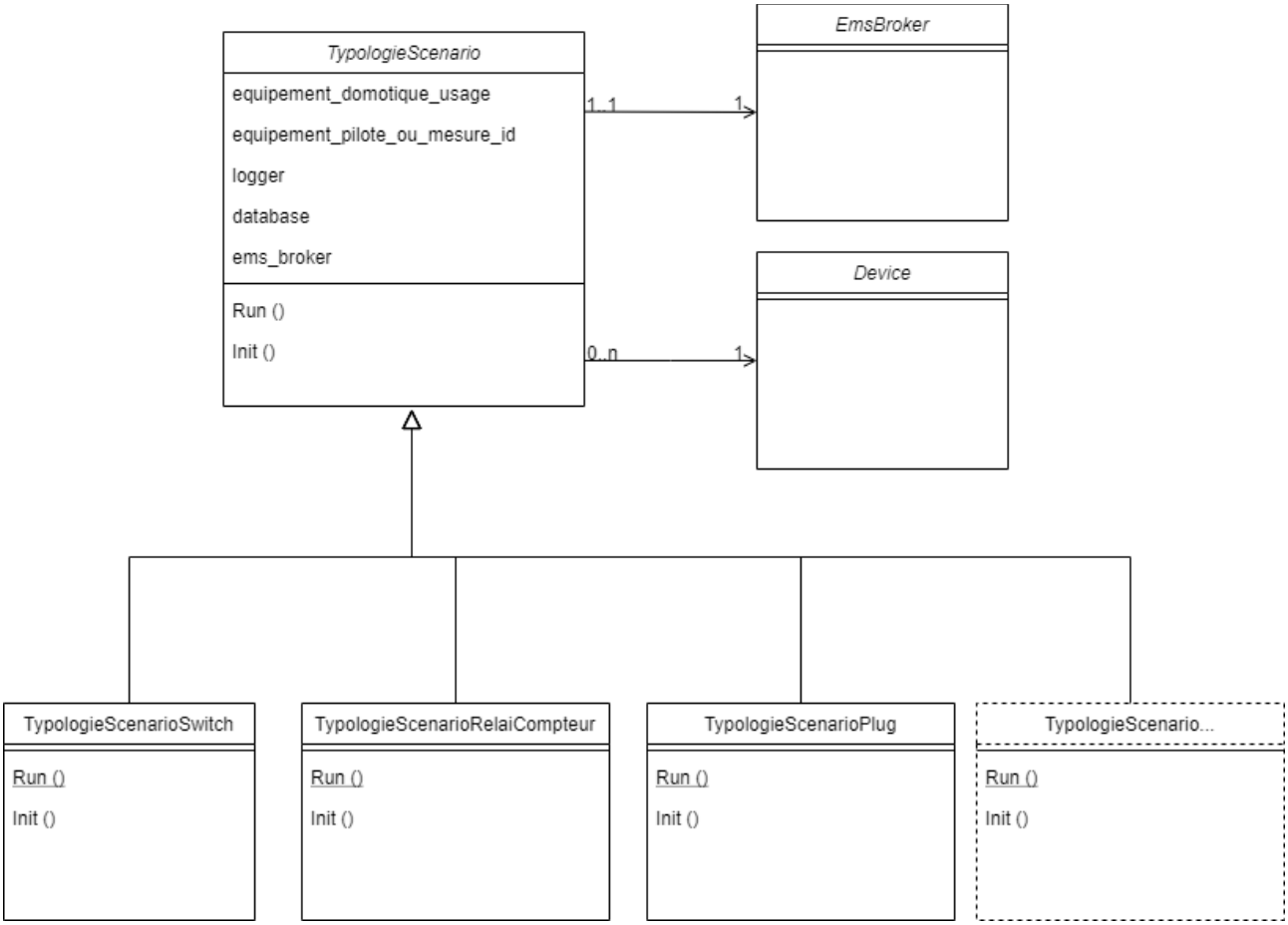
Module emsbroker.py

Le module emsbroker fournit une interface avec le broker MQTT et un service qui permet d'enregistrer temporairement un topic et un device associé. Ce service permet de recevoir au niveau d'un object de type device les réponses MQTT émisent par un équipement en réaction a un message de commande MQTT.

Structure générale des typologies

De manière a pouvoir être manipulé de façon générique, les typologies sont dérivées de la classe de base TypologieScenario. Cette classe de base fournie un ensemble générique de fonctions accessibles par l'ensemble des typologies. Les typologies fournissent 2 services spécialisés :

- Init () qui correspond à l'initialisation lors d'un passage pilote / manuel
- Run () qui correspond à l'exécution d'un cycle ems.

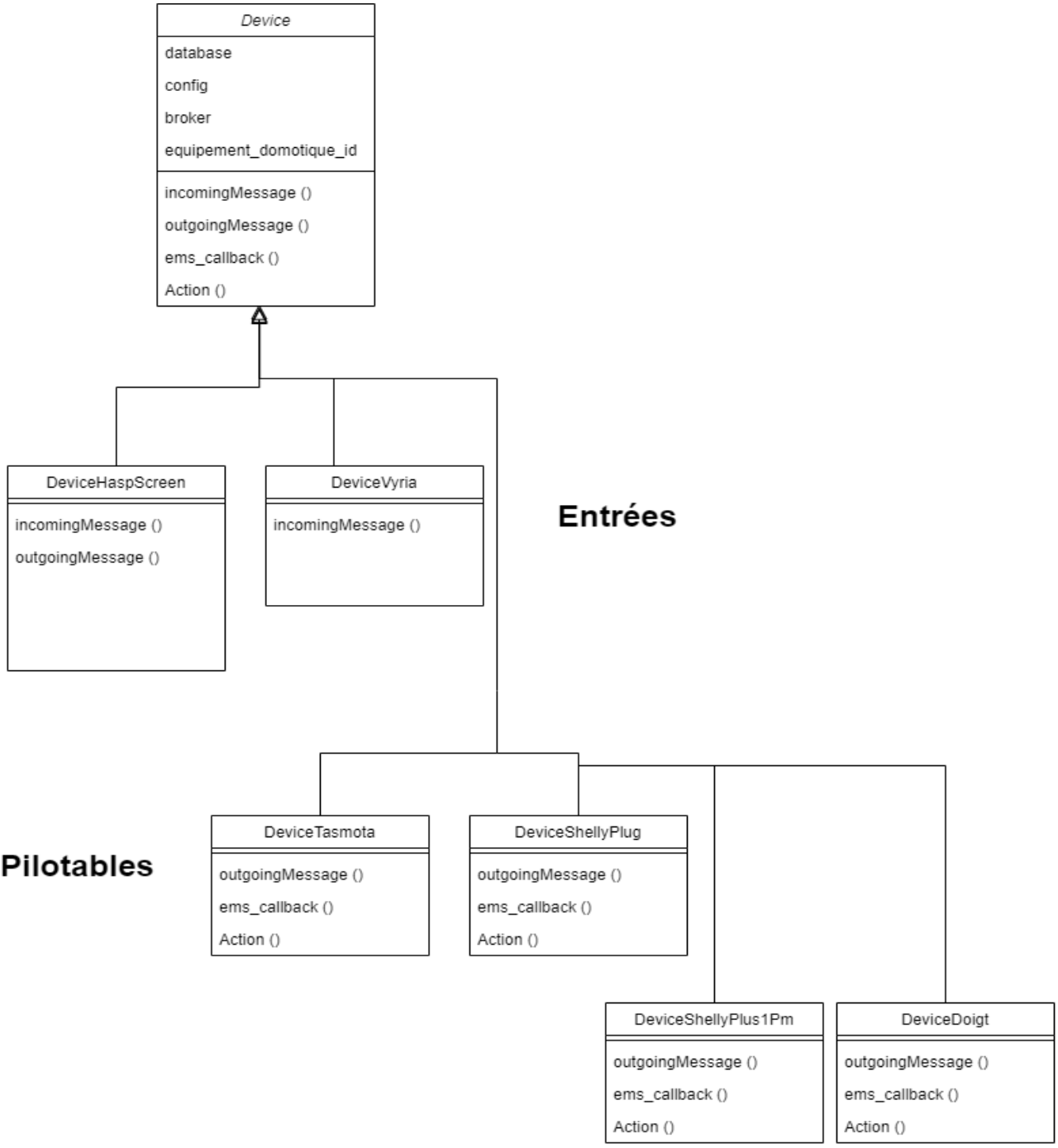


Structure générale des drivers

De manière a pouvoir être manipulé de façon générique, l'ensemble des équipements domotique sont modélisés par des devices dérivés de la classe de base Device.

La classe de base Device fournie une ensemble de fonction générique utilisable par toute les classe dérivées. Chaque device implemente au besoin les services spécialisés suivant :

- **incomingMessage** () traitement d'un message asynchrone en reception.
- **OutgoingMessage** () emission d'un message vers l'équipement domotique.
- **ems_callback** () traitement d'un message synchrone en réception d'une réponse d'un équipement.
- **Action** () demande d'exécution d'une commande particulière sur l'équipement domotique (on ou off)


Device
database
config
broker
equipement_domotique_id
incomingMessage ()
outgoingMessage ()
ems_callback ()
Action ()
DeviceHaspScreen
incomingMessage ()
outgoingMessage ()
DeviceVyria
incomingMessage ()
Entrées
Pilotables
DeviceTasmota
outgoingMessage ()
ems_callback ()
Action ()
DeviceShellyPlug
outgoingMessage ()
ems_callback ()
Action ()
DeviceShellyPlus1Pm
outgoingMessage ()
ems_callback ()
Action ()
DeviceDoigt
outgoingMessage ()
ems_callback ()
Action ()

Scénarios d'évolutions

Ajout d'un équipement domotique

A priori tout type d'équipement pilotable par MQTT peut être modéliser dans la mesure ou il accepte des commandes de type on/off. La modélisation de l'équipement devra être réalisée en implémentant une classe python spécifique, dérivée de la classe de base Device. Tant que l'équipement gère des commandes de type on / off, il devrait être compatible avec les typologies existantes. Cette classe spécialisée devra être enregistrée dans le device_factory pour associer un type d'équipement en base de donnée avec la classe correspondante python.

Ajout d'une interface utilisateur

De la même manière qu'un équipement domotique, une interface utilisateur de type écran ou interface web peut être modélisée en spécialisant la classe de base Device. La racine du topic mqtt doit être spécifique si on veut conserver le routage dans le device_manager. Le device_manager devra être modifié pour prendre en compte une règle de routage supplémentaire. Le topic de réception des commandes devra être ajouté dans les topics permanent du fichier de configuration.

Ajout d'une typologie

Une typologie correspond au « scénario » d'activation d'équipements domotiques. Dans la mesure ou elle correspond globalement à l'activation ou la désactivation d'un équipent domotique. Une typologie sera modélisée en spécialisant la classe de base TypologieScenario. Cette classe spécialisée devra être enregistrée dans le typologyscenario_factory pour associer une typologie d'équipement en base de données avec la classe correspondante python.