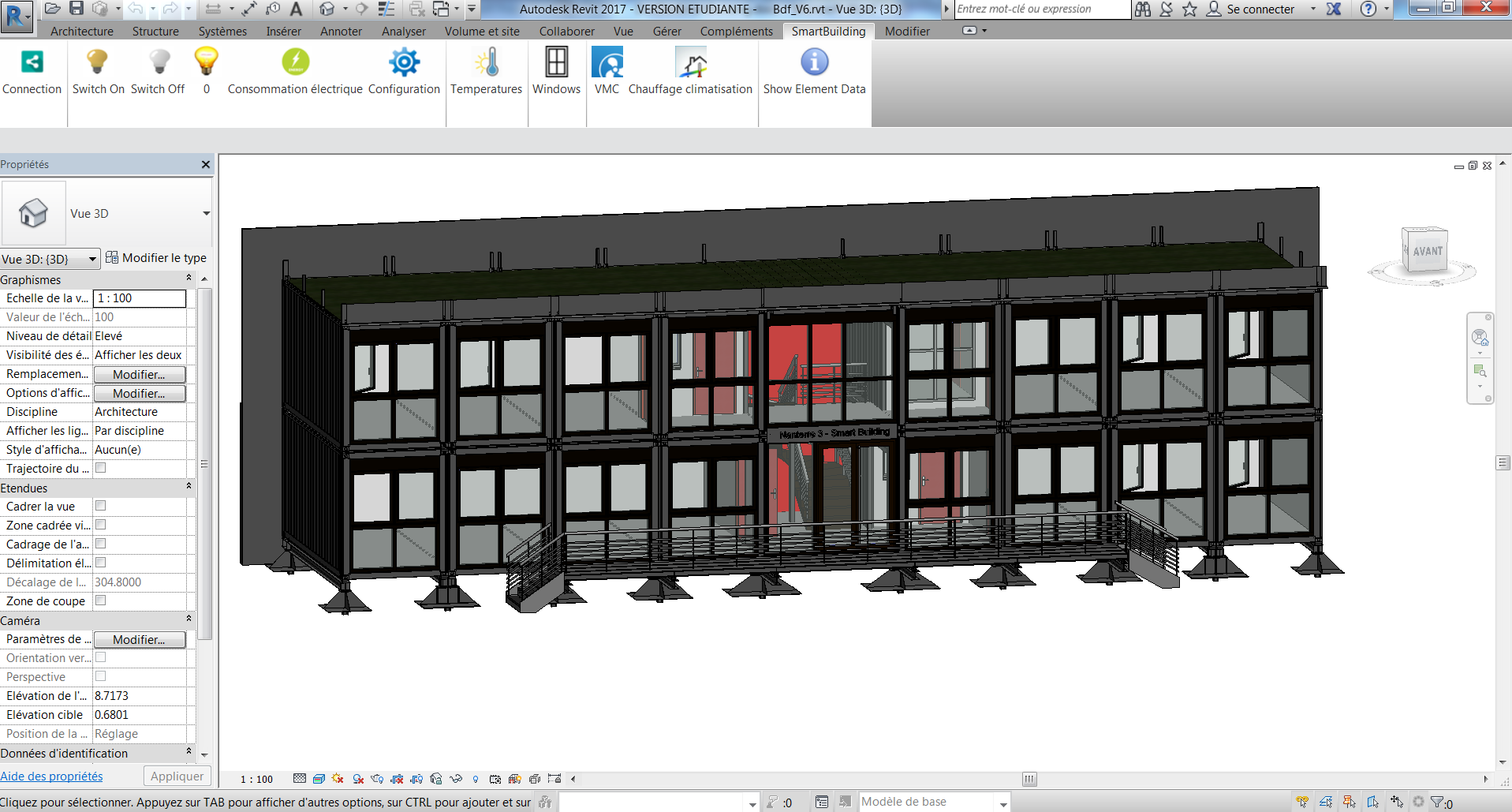
SmartBuilding CESI Nanterre

POC Jumeau numérique

Documentation technique



# Contexte

Les développements menés dans le SmartBuilding ont été des travaux exploratoires permettant :

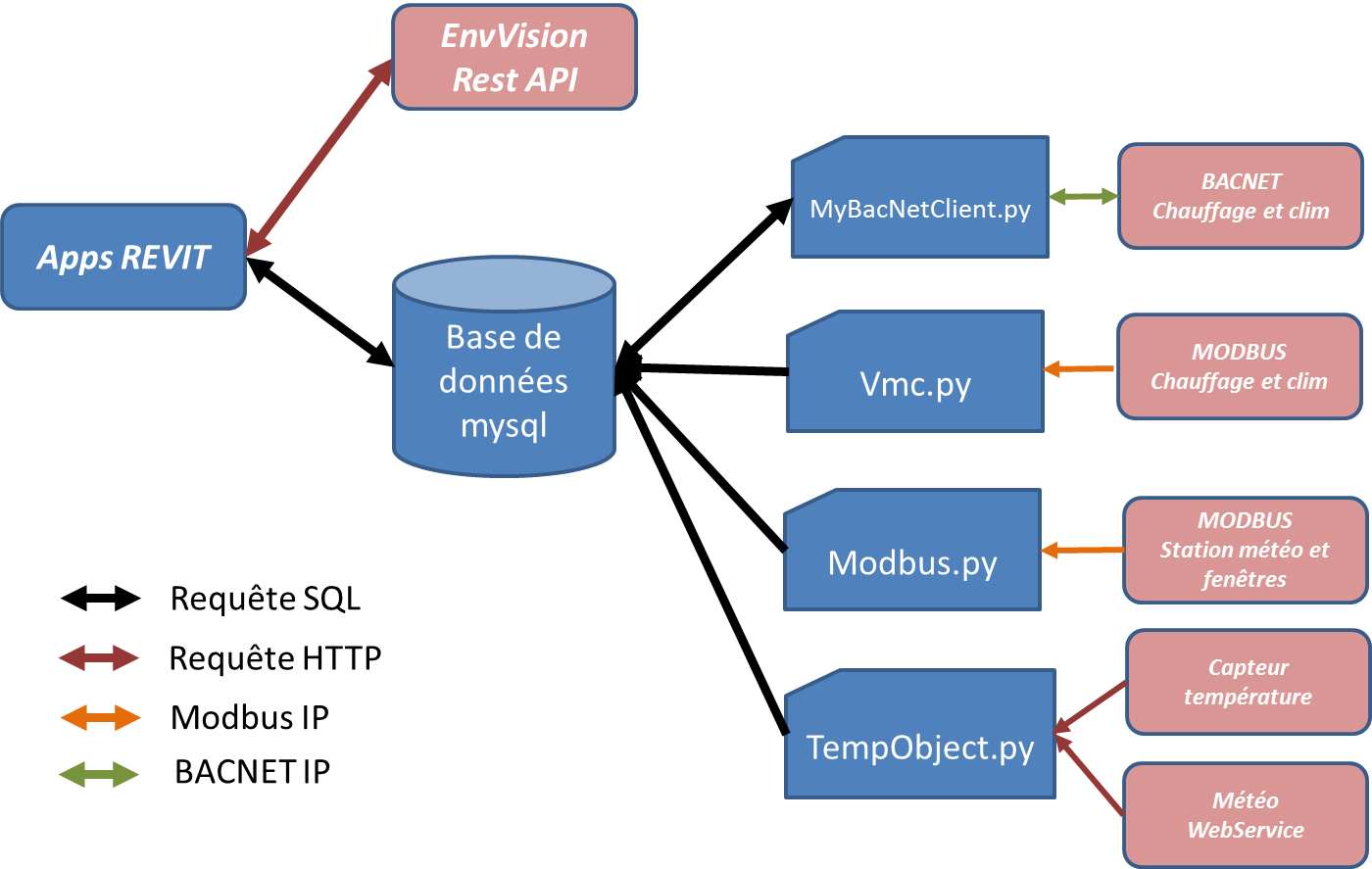
* Une exploration des différents protocoles
* Une connexion aux différents équipements
* Un stockage rapide mais efficace des données générées par le bâtiment
* Une interface de contrôle et des visualisations des données via une appli REVIT
* Une interface de contrôle et de visualisation des données via une modification de XBimExplorer

Ces travaux, bien que fonctionnels, ne sont en aucun cas les travaux finaux, le développement officiel étant pris en charge par un développeur à recruter

CESI n’ayant pas d’outil de contrôle de source, les sources sont stockées sur un github personnel : https://github.com/sebsegard/NumericalTwin

# Architecture

Voici l’architecture développée pour cette phase expérimentale



# Base de données

Avec l’outil mysqlworkbench, se connecter à l’adresse IP 10.192.12.5 :3306 avec les accès suivants :

**Login** : ssegard

**Password** : cesi

La base de données contient les données sous formes de tables non liées entre elles (données « à plat »)

Voici une description rapide des tables

|  |  |
| --- | --- |
| **Table** | **Description** |
| BacnetConsigne | Contient les consignes concernant le chauffage : salle d’application, température, mode de fonctionnement (chaud, froid), prohibition (droits utilisateurs), …. |
| BacnetValues | Contient les valeurs remontées par le système de chauffage (Températures, consigne, …) |
| FenetreEvents | Contient l’ensemble des changements d’états sur les fenêtres du bâtiment  Chaque ligne contient l’ensemble des fenêtres |
| FenetreMinutes | Contient une représentation de l’état des fenêtres sur chaque minutes Permet d’avoir un historique et éventuellement un calcul statistique |
| LightEnergyQuarter | Contient la consommation eléléctrique des luminaires par salle et par quart d’heure |
| MeteoHours | Contient les données renvoyées par la station météo (temperature, vent, hydro, pression, … ), avec un échantillonnage horaire |
| MeteoMinutes | Contient les données renvoyées par la station météo, avec un échantillonnage par minute |
| Room | Table contenant une description des salles du batiment La ligne 5-BUILDING représente le batiment dans sa globalité |
| TempHour | Contient un relevé de température et d’humidité par heure, par salle, et exterieur (via un service WEB météo) |
| TempMinute | Contient un relevé de température et d’humidité par minute, par salle, et exterieur (via un service WEB météo) |
| VmcHours | Contient les données issues de la VMC, donné prélevés une fois par heure |
| VmcHours | Contient les données issues de la VMC, donné prélevés une fois par minute |

Pour vérifier si un système fonctionne ou pas, il est conseillé d’effectuer une requete SQL pour récupérer le dernier enregistrement, et vérifier si son timestamp correspong à l’heure actuelle

Par exemple, pour vérifier l’état de la station météo :

SELECT \* FROM TEST.MeteoMinutes ORDER BY TimeStamp DESC;

Dans certaines tables, il y a une colonne validité (ou valid). C’est une information boolean indiquant l’état de confiance et la validité de l’information associé :

* Si validité vaut 1 : on peut avoir confiance dans la donnée
* Si validité vaut 0 : il y a un problème (souvent une exception au niveau du script)

# Apps Revit

## Introduction

L’application utilise l’API REVIT 2017, et a été testée avec RVIT 2017

L’appli a été développé en C# / WinForms

REVIT offre une API permettant de créer des applications tournant dans son contexte. Ces *applications* s’appellent des « ExternalApplications »

Les Apps REVIT sont localisées dans le dossier suivant :

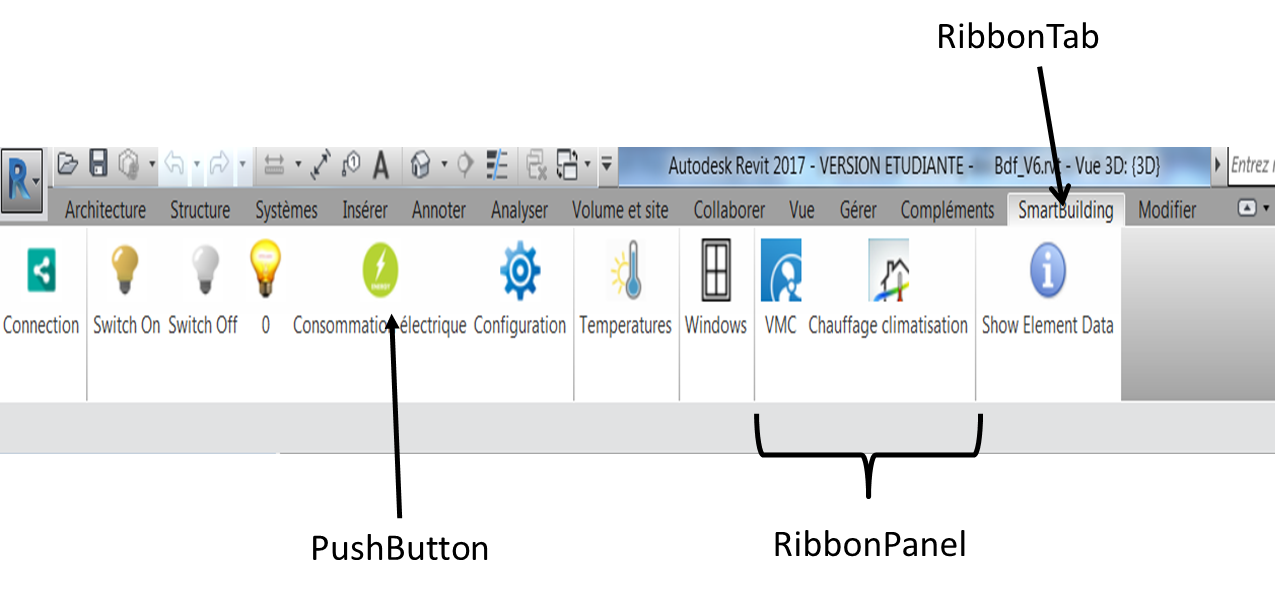
*C:\ProgramData\Autodesk\Revit\Addins\2017*

Pour enregistrer une application dans REVIT, il faut avoir une DLL implementant l’API, puis il faut créer un fichier .addin, référençant l’assembly (package) de *l’application*, et la classe principale (FullClassName).

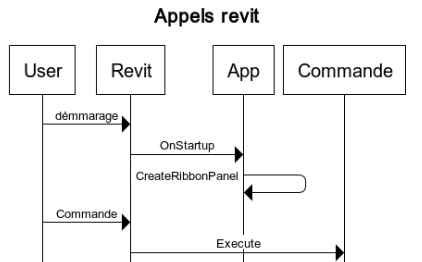
## Interfacage avec REVIT

## Une *application* REVIT(interface IExternalAPplication) est en fait un ensemble de *commandes* (Interface IExternalCommand)

### Partie IHM



### Séquences d’appels Revit



Revit fait office de Scheduler. L’appli ne communique pas directement avec les commandes, c’est Revit qui appelle les commandes

## Commandes

### Commande Température

**Path** : Temperatures.Temperatures

**Description**

La commande de température ouvre une fenêtre pop-up (nommée Temperatures.TemperatureForm) permettant d’afficher les courbes de températures.

Dans cette fenêtre, les données sont prélevées directement via une requete Sql (Table TempHour) et affiché dans un control de type « Chart » contenant plusieurs Séries (une par salle + extérieure)

### Commande Consommation energetique

**Path** : LightControl. LightEnergy

**Description :**

La commande de consommation énergétique des luminaires ouvre une fenêtre pop-up (nommée LightControl .LightEnergyForm) permettant d’afficher les courbes de consommation électrique des luminaires par salle.

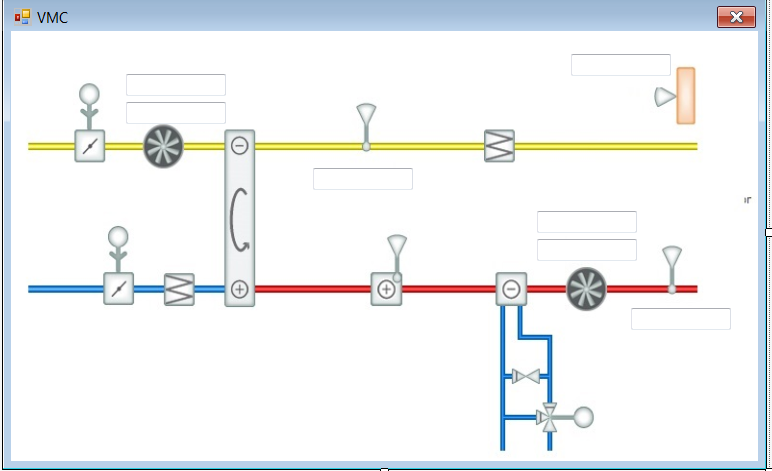
Dans cette fenêtre, les données sont prélevées directement via une requête Sql (Table LightEnergyQuarter) et affiché dans un control de type « Chart » contenant plusieurs Séries (une par salle + extérieure)

### Commande VMC

**Path** : CVC.VMC

**Description :**

La commande VMC ouvre une fenêtre pop-up (nommée CVC.CVCForm) permettant d’afficher les données de la VMC / CTA dans des *TextBox* prépositionnées dans une image.



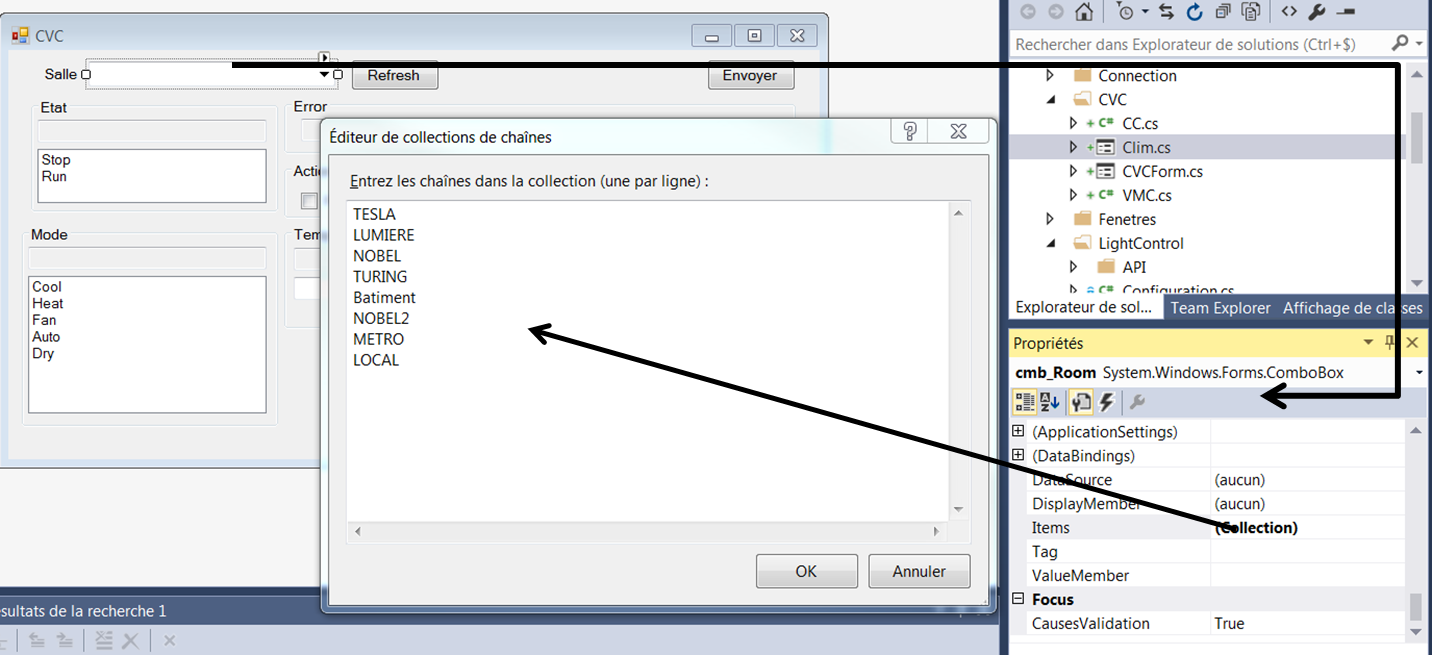
### Commande Chauffage et climatisation

**Path** : CVC.Clim

**Description :**

La commande Clim ouvre une fenêtre pop-up (nommée CVC.Clim) d’affichage et de contrôle du chauffage.

Dans cette fenêtre, il y a un une liste à choix permettant de choisir une salle ou le bâtiment.  
Cette liste est renseignée statiquement dans l’IDE et contient les valeurs suivantes :



A noter, l’index pour le bâtiment global est 5

Lorsque l’utilisateur choisit une salle, alors les données sont récupérées via une requete SQL (table BacnetValues) , traitées et affichées.

Lorsque l’utilisateur appui sur envoyer, alors les données sont lues dans l’interface, traités, et insérées via une requête SQL (table BacnetConsigne )

### Gestion des lumières

#### Principe

Les lumières sont pilotables via une API REST. Un luminaire est identifié par un numéro et un numéro area (salle)

La classe LightManager fait un mapping entre un élément REVIT (identifié par un identifiant unique Autodesk.Revit.DB.ElementId) et une « LightFixture », pouvant être une area ou un luminaire. Ce mapping est réalisé via un dictionnaire.

Il n’y a pas de filtre sur le type de l’élément REVIT. On pourrait ainsi selectionner un sol, une porte, … et lui associer une « LightFixture »

La classe *LightManager* sauvegarde ce mapping dans un fichier .xml et le recharge lors de son initialisation

Une « *LightFixture* » est constitué principalement de 2 paramètres *AreaId* et LuminaireId et est:

* Un luminaire, si *AreaId* et *LuminaireId* sont renseignés
* Un « plafond », c’est-à-dire une salle, si *AreaId* est renseigné et *LuminaireId =* UNKNOW\_LIGHT\_ID

Une LightFixture propose principalement 2 méthodes :

* On() : allume la salle en mode automatique ou met le luminaire à 100%
* Off() : eteint la salle ou met le luminaire à 0%

#### Commandes Off et On

**Path** : LightControl.SwitchOn et LightControl. SwitchOn

**Description :**

Allume ou éteint l’élément sélectionné, en procédant de la manière suivante :

* Récupération de l’élément REVIT Sélectionné (via une commande REVIT)
* Transformation de cet élément en LightFixture (via le dictionnaire de la classe LightManager)
* Appelle de la méthode On ou Off de la LightFixture

#### Commande Configuration

**Path** : LightControl.Configuration

**Description :**

Après avoir récupéré l’Elément REVIT sélectionné, ouverture une pop-up ConfigurationForm (avec en paramètre l’élément sélectionné).

Lors de son ouverture, la pop-up appelle le service REST pour récupérer les aréas disponibles et les affiche dans un arbre (méthode LoadTree() ).

Lorsque l’utilisateur clique sur un nœud représentant une area, le service REST est de nouveau appelé pour récupérer les luminaires associés à cet area (méthode ExpandArea), ainsi que les différents niveaux de luminosité configuré (auto, off, full,…)

Lorsque l’utilisateur clique sur Apply, alors :

* Si le nœud sélectionné est
  + une Area, alors La LightFixture est modifiée avec la nouvelle Area et UNKNOW\_LIGHT\_ID
  + Un luminaire, alors la LightFixture est modifiée avec la nouvelle area et le nouveau luminaire.
* Le fichier xml du LightManager est sauvegardé

### Gestion des fenêtres

#### Principe

Les fenêtres ont pour singularité d’avoir un impact directement sur la maquette BIM, en modifiant une propriété de l’objet fenêtre : la propriété « *Vent Opening Angle* »

Pour pouvoir obtenir un changement de position de fenêtre, il faut être capable de déterminer le changement d’état d’une fenêtre.

Il faut donc à un moment créer un tache (un thread) qui interroge périodiquement la base de données et savoir si une fenêtre change d’état et déclencher l’ouverture de celle-ci, à savoir la modification de la propriété

Par ailleurs, il est important de comprendre la dynamique des Threads, et que :

* Un thread crée par l’appli n’a pas le même contexte et les même droit qu’un thread créé par une commande.

Malheureusement, Revit « surveille » les threads, et empêche toute modification de données venant d’un thread inconnu et leve une exception. Pour rendre cette action possible, il est nécessaire de passer via un évènement de type ExternalEvent.

La gestion des fenêtres fonctionne de la manière suivante :

* Lors du démarrage, Création d’un évenement ExternalEvent nommé \_WindowsEvent
* Lors d’un click sur la commande Connexion, creation d’un thread pour détecter les changements liés aux fenêtres
* Lors du changement d’état d’une fenêtre, ajout d’un évènement sous la forme *WindowsEventData* dans une pile d’évènements (\_*WindowsEventToSend*)
* Toute les secondes, un évènement Revit *RevitPeriodicEvent* scrute la pile d’évènement, et par évènement présent dans la pile
  + Retrouve l’objet REVIT associé
  + Cherche la propriété *« Vent Opening Angle »*
  + Modifie la valeur de cette propriété (créé une *transaction* de type *SetParameter*):
    - A 90 si la fenêtre est ouverte
    - A 0 si la fenêtre est fermée

# Scripts python

## Généralités linux/python

### Accès

Se connecter via ssh sur l’adresse ip 10.192.12.5 avec les informations suivantes :

**Login** : ssegard

**Password** : cesi

Au cas où, le mot de passe root est cesi

### Lancement d’un script

Les scripts sont disponibles dans le répertoire /home/ssegard/python

Pour lancer un script, il faut taper la commande suivante :

*nohup python script.py*

nohup permet de garder le script en arrière plan lors de la fermeture d’une session

### Vérification d’un script

Taper la commande suivante :

Ps-ef | grep NomScript (exemple : ps -ef | grep modbus )

ssegard 2207 2187 0 sept.04 pts/4 00:07:07 python modbus.py

Pour le tuer : killall pid (exemple : killall 2207 )

## Script TempObject.py

Ce script se connecte :

* Aux capteurs de temperatures (POE) des 4 salles
  + Une classe Internal Capteur instancié 4 fois
* A un webService (<https://www.prevision-meteo.ch/services/json/nanterre>)
  + Une classe MeteoData instancié une fois

Toutes les minutes, le script fait appel respectivement à

* Internal.ComputeMeteoData pour relever les températures en interne
* External. ComputeMeteoData pour relever les températures au webService

Il sauvegarde ces données dans la base de données au démarrage, toutes les minutes et toutes les heures

## Script modbus.py

Ce script se connecte à l’automate modbus (10.192.16.31). Il est principalement constitué des 3 fonctions suivantes :

*getCapteurFenetreData()* : récupère les données relatives au fenêtre et les traite

*getMeteoData()* : récupère les données relatives à la station météo

*RecordData()* : enregistre des données dans la base de données

Toute les secondes, ces 3 fonctions sont appelées successivement, et les données sont enregistrées par minute et par heure. Pour les fenêtres, il y a une petite comparaison par valeur permettant de créer un enregistrement dans la table FenetreEvents

Pour toute questions relative à cet automate modbus, se refererer au document « Table d'échange ModbusTCP Wago »

## Script vmc.py

Ce script se connecte à l’automate modbus (10.192.16.1) gérant la vmc. Il est principalement constitué des 2 fonctions suivantes :

*GetVmcData()*: récèupere les informations nécessaire aux adresses suivantes

|  |
| --- |
| Modbus address Default value Function Description Value  1 Actual/Setpoint Outdoor temperature, (Only read) 283  3 Actual/Setpoint "0=Stopped,1=Starting up,2=Starting reduced speed,3=Starting full speed ,4=Starting normal run,5=Normal run,6=Support control heating,7=Support control cooli  ng,8=CO2 run  9=Night cooling,10=Full speed stop,,11=Stopping fan,12=Fire mode,13=Recirculation run,14=Defrosting"5  7 Supply,Extract and Room temperatures Supply air temperature 281  9 Supply,Extract and Room temperatures Extract air temp 267  13 SAF/EAF Pressure and Flow Supply air fan pressure 962  14 SAF/EAF Pressure and Flow Extract air fan pressure 2285  15 SAF/EAF Pressure and Flow Supply air fan flow. Scale factor = 1 991  16 SAF/EAF Pressure and Flow Extract air fan flow Scale factor = 1 1527  122 SAF/EAF Pressure and Flow Control signal supply air fan 64  123 SAF/EAF Pressure and Flow Control signal extract air fan 55  331 Actual/Setpoint Total calculated Power consumtion (kW) 39 |

Il y a un décalage de 1 entre l’adresse et le registre modbus, pour lire la Extract AirTemp, il faut lire à l’adresse 8

Ensuite, il les traites (par exemple, une temperature est exprimé en 1/10 de dégré, il faut donc diviser par 10 pour avoir une temperature en °C (par exemple, le système pour la température d’extraction (Extract air Temp) retourne 267 pour une temperature à 26,7

*RecordData()* : enregistre des données dans la base de données

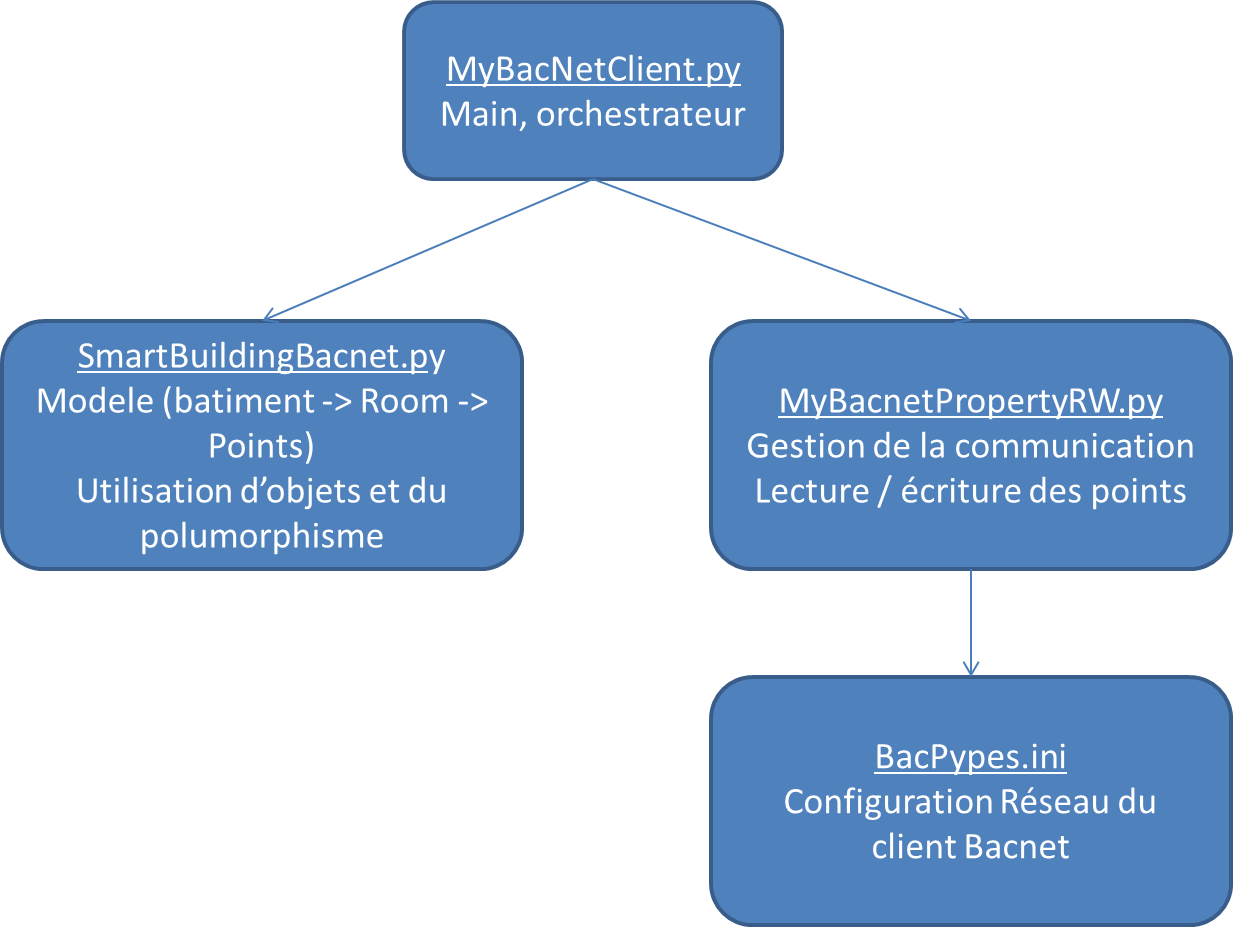
Toutes les minutes, les données sont prélevés, traités, et enregistréesIl existe des centaines d’autres données, référencées dans le document *VIM Corrigo\_E\_ventilation\_signals.xml*

## Script MyBacnetClient.py

Ce script gère la communication avec le bacnet, c’est-à-dire les données et contrôle du système de chauffage

Le protocole Bacnet est un protocole objet assez compliqué, le script hérite de cette complexité

Ce script est constitué de 3 fichiers architecturé de la facon suivante:



MyBacNetClient.py est le fichier main

Au démarrage, il initie la connexion avec *MyBacnetPropertyRW* (fonction *Init()* ), et initialise le modèle du SmartBuilding

Ensuite, chaque minute,

* Il traite les consignes :
  + Il va vérifier dans la base de données si une nouvelle consigne est arrivée
  + Il appelle SmartBuildingBacnet.py pour transformer cette consigne en points Bacnet (via la fonction *BacNet.GetSetPointsBuildingValues* pour le bâtiment ou *BacNet.GetRoomSetPointsBuildingValues* pour une salle)
  + Il envoie ces points via MyBacnetPropertyRW.py (via la fonction *MyBacnetPropertyRW.WriteValues*)
* Il lit les points :
  + Il lit l’état des points via *MyBacnetPropertyRW.py* (fonction *MyBacnetPropertyRW.ReadValues* )
  + Il appelle SmartBuildingBacnet.py pour transformer ces points en valeur comprehensible (methode *BacNet.UpdateValues )*
  + Il écrit ces données dans la base de données

SmartBuildingBacnet contient la modélisation du batiment.

Il contient plusieurs types d’objet :

Classe SmartBuilding : classe qui s’occupe du batiment. Cette classe possède des Room

Classe Room : classe qui s’occupe des salles, cette classe a un nom, un numéro de groupe (c’est-à-dire un numéro de télécommande BACNET, servant à identifier la télécommande et le bloc chaud/froid – se reporter à la documentation MITSUBISHI)  
Elle a également un certain nombre de points représentant chacun des données (ex : Température de Consigne, Mode de fonctionnement).

Ces points ont des types spécifiques (entier, flottant, … ) mais héritent tous de la classe BacNetPoint).

Un BacNetPoint représente un point et une valeur. Il a un nom (exemple : On Off Setup), un identifiant de point (exemple : 4), et un type. Là encore se reporter à la documentation technique de Mitsubishi.

Par exemple :



Devient :

self.OperationalModeSetup = multiStateOutput("Operational Mode Setup",GroupId,5,["","Cool","Heat","Fan","Auto","Dry","Setback"])

Les valeurs sous forme de texte, notamment pour les signaux binaires ou multi-états, sont stockées au niveau des classes filles