Université de Franche-Comté
UFR Sciences et techniques

MASTER M2 M2E2-ESE
Année 2015

Mahmoud ADDOUCHE
Systèmes Embarquées

Thème :
Système Client-Multiserveur pour la commande d'une chambre Thermique

Étudiant :

TORRES ZETINO Juan Carlos
MOUSSA Rachid

Besançon le 31 Décembre 2015

Contenu

1 Introduction 3
2 Système de Mesure client et serveur 4
2.1 Interface Graphique Client –Serveur 4
3 Logiciel de commande de l'interface Client –Serveur 5
3.1 Programmation de Client 5
3.1.1 Fonctionne pour récupérer la Température de la Chambre 5
3.1.2 Fonctionne pour Modifier la Température de la Chambre 6
3.1.3 Fonction pour récupérer l'état de la valve 7
3.2 Programme du Serveur (Chambre thermique) 7
3.2.1 Tableau de paramètres Serveur 7
3.2.2 Exécution d'instructions Utilisateur 8
4 Système de Mesure 1 Client et 3 Serveurs 9
4.1 Interface Graphique pour 3 Serveurs 9
4.2 Programmation en mode Client -3Serveurs 9
4.2.1 Fonctionne de Sélecteur de Chambre 10
4.2.2 Adaptation des fonctionne du Clients pour 3 serveurs 11
5 Vérification de Système Central de température 11
5.1 Vérification fonction Réglage de température de la chambre 11
5.2 Vérification de fonctionne Récupération de Température pour différent Serveurs 12
5.3 Vérification Fonctionne d'état de valve pour 3 Serveurs 12
5.4 Test d'Auto-vérification de syntaxe de commandes 13
6 Conclusion 13

1 Introduction

Le présente TP vise à réaliser une application virtuel de commande d'une chambre thermique avec l' interpréteur de commandes **Python version3.0** en utilisant l'environnement de travail eric6 avec liaison à la plateforme Graphique orientée Object QTDesigner , le premier sert à créer un projet basée sur Python 3.0 et il offre différents outils pour le développement du code ,vérification des erreurs , arrangement de fichier et ses dépendances et possibilité d'exécuter le code en séquence .

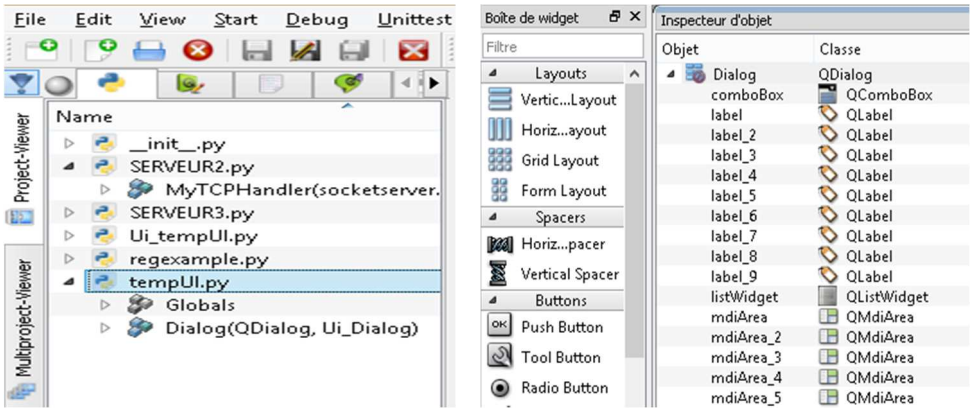


Fig1 : *Outils de développement eric6 ainsi que la plateforme graphique QT Designer*

Le deuxième à utiliser sera **QTDesigner** qui est une plateforme qui contient une série de modules graphiques qui peuvent être utilisées pour créer des boutons , radio buttons , text frames , widgets et ils peuvent être reliés avec un projet en python basé sur l'IDE eric6 afin de générer partie du code pour les différents Objects graphiques présentent sur l'application.

Dans un premier temps nous allons implémenter un système client-mono serveur afin de connaitre le fonctionnement des différents classes et modules de la librairie PyQT5 sur python3 , ensuite nous allons générer un nouvel projet afin de réaliser le système complète pour 3 serveurs et un commande centralisé, les deux application seront vérifiées pour valider son bon fonctionnement ainsi que la communication Ethernet sur la PC .

2 Système de Mesure client et serveur

Dans cette partie nous allons réaliser une communication client- serveur dans laquelle nous allons commander de valves de ventilation pour réguler la température, d'autre part le client de l'application sera représenté par une central de commande individuelle tandis que la chambre thermique sera représenté par le Serveur externe.

2.1 Interface Graphique Client –Serveur

Dans un premier temps nous allons réaliser l'interface pour commander une seul chambre thermique depuis la central Client, pour cela nous utilisons l'environnement QT designer qui permet de dessiner et personnaliser les différents widgets, listes, boutons, combo boxes, etc. pour créer une interface utilisateur interactive.

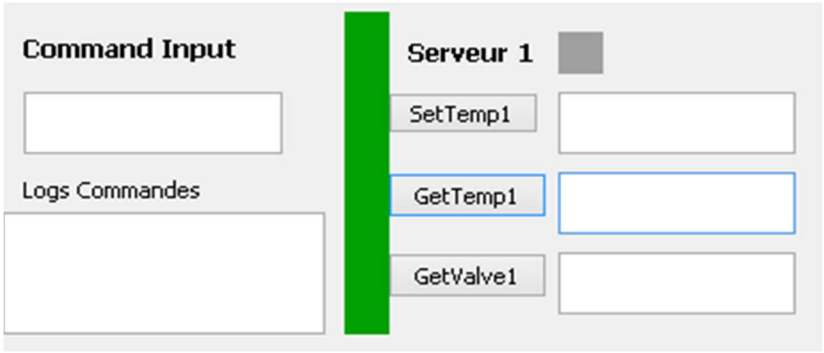


Fig2 : *Interface Graphique Client – Serveur réalisée dans QT designer*

Dans la suite nous expliquerons les fonctionnalités de différents paramètres représentées sur l'interface utilisateur :

- **Command Input :** c'est une QTextEdit Object qui permet d'insérer les différentes instructions pour commander la chambre thermique (serveur1).
- **Logs Commandes :** c'est une QTextEdit Object en mode lecture qui permet de visualiser la réponse du serveur selon les différentes commandes envoyées .
- **Bouton SetTemp1**:c'est une QPush Button Object qui permet d'envoyer et valider la commande au serveur pour **spécifier la température de consigne** de la chambre, il sera associé avec une QTextEdit pour afficher la réponse du serveur
- **Bouton GetTemp1**:c'est une QPush Button Object qui permet de valider la commande et **récupérer la température** de la chambre, il sera associé avec une QTextEdit pour afficher la réponse du serveur
- **Bouton GetValve1**: c'est une QPush Button Object qui permet de valider la commande et **récupérer l'état de la valve** de la chambre, il sera associé avec une QTextEdit pour afficher la réponse du serveur
- **Indicateur Serveur 1**: c'est une QMdiArea Object qui fonctionne comme LED et permet de visualiser si la chambre est active.

3 Logiciel de commande de l'interface Client –Serveur

Afin de réaliser l'application avec fonctionnalités interactives nous commençons pour importer l'UI que nous venons de réaliser sur QT Designer et de exécuter la compilation de formes pour notre projet en Python 3.0, ensuite nous devons générer une classe Dialog qui contient la définition des tous les propriétés des objets présentent sur l'UI , nous pourrons ensuite appeler les différents méthodes pour configurer les objets graphiques selon nos besoin , finalement il faudra sélectionner les évènement qui seront associées aux objet graphiques pour déclencher une action , par exemple mesure de température , modification de variables selon la choix de l'utilisateur .

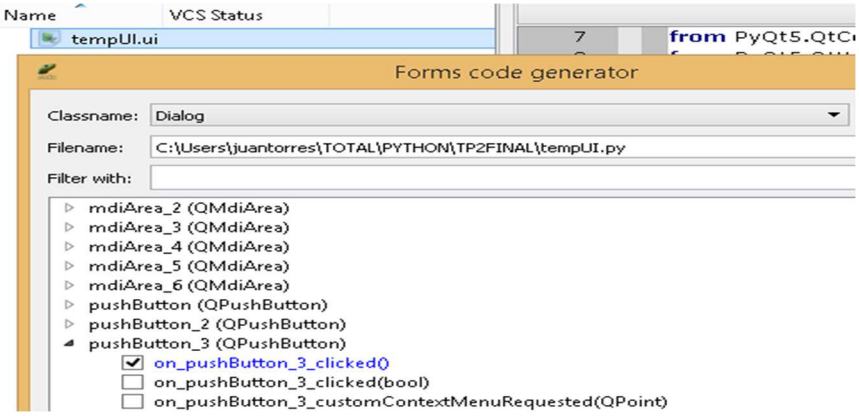


Fig2 : Génération de classe principale Dialog et ses évènements PyqtSlot dépendants

3.1 Programmation de Client

Pour programmer le Client nous allons commencer pour définir les différents actions que l'utilisateur pourra exécuter afin de surveiller la chambre thermique de façon efficace, ainsi nous aurons 3 fonctionnes principales à programmer qui seront :

- **Récupération de la température dans chambre thermique**
- **Modification de température dans chambre thermique**
- **Récupération état de la valve de ventilation**

Chaque fonctionne sera associé à un évènement pyQSlot dans lequel nous ajouterons le code respective pour établir une communication Ethernet avec le serveur distante, ainsi que pour modifier les propriétés du objet graphique si nécessaire en agissant sur le paramètre d'intérêt , un exemple de évènement sera le **def on_pushButton_2_clicked(self):** qui sera exécuté quand l'utilisateur appuie sur le bouton2, Dan la suite nous expliquons les code plus en détail .

3.1.1 Fonctionne pour récupérer la Température de la Chambre

Cette méthode permet de récupérer l'instruction tapé par l'utilisateur en format string , et de la comparer avec une instruction valide en mémoire qui sera « **gettemp sys output »** ,de cette façon si la commande est valide nous allons envoyer l'instruction au serveur qui saura l' interpréter pour nous donner la température de la chambre ,le résultat sera affiché dans le bloc textEdit. Dans le cas

que le commande tapé soit erroné, le système affichera un message d'erreur **« Erreur commande »** comme nous voyons dans le code ci-dessous.

```
def on_pushButton_clicked(self):
    #Method pour Obtenir Temperature 1
    getTemp1=self.textEdit_7.toPlainText()
    #getTemp1="gettemp sys output"
   #Instruction a envoyer="gettemp sys output"
    if(self.selector==1):
        if((getTemp1=="gettemp sys output")and(self.selector==1)):
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            try:
                sock.connect((HOST, self.PORT))
                sock.sendall(getTemp1.encode())
                # Receive data from the server and shut down
                temp1Real = sock.recv(self.RPORT).decode()
                self.textEdit.append(temp1Real)
                self.textEdit_2.append("**Requet getTemp recu **")
            finally:
                sock.close()
        else:
            self.textEdit_2.append("Erreur Commande")
```

Fig3: *implémentation en Python de la fonction pour obtenir la température de la chambre*

3.1.2 Fonctionne pour Modifier la Température de la Chambre

La fonction suivante permet de modifier la valeur de température consigne stockée dans la mémoire du Serveur 1, d'abord nous validons la syntaxe du message qui dois être égal à **« settemp** », ensuite nous envoyons l'instruction spécifié dans la variable **« settemp1 »,**

Le serveur interprétera cette commande qui contiendra la valeur en string de la nouvelle température consigne , ensuite après que le serveur a fait son traitement le client va récupérer la variable **tempConsigne** dans un bloc textEdit4 pour pouvoir le visualiser

```
def on_pushButton_2_clicked(self):
    #Method pour Regler(SET) la  Temperature 1 Consigne
    settemp1=self.textEdit_7.toPlainText()
    #settemp1="settemp 28 input"
   #Instruction a envoyer="settemp 40 input"
    m2=re.findall(r'\w{2,8}',settemp1)
    if(self.selector==1):
        if((m2[0]=="settemp")and(self.selector==1)):
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            try:
                sock.connect((HOST, self.PORT))
                sock.sendall(settemp1.encode())
                # Receive data from the server and shut down
                tempConsigne = sock.recv(self.RPORT).decode()
                self.textEdit_4.append(tempConsigne+" °C")
                self.textEdit_2.append("**Requet setTemp "+tempConsigne+" reussi**")
            finally:
                sock.close()
        else:
            self.textEdit_2.append("Erreur Commande")
```

Fig4 *: implémentation en Python de la fonction pour Modifier la température de la chambre*

3.1.3 Fonction pour récupérer l'état de la valve

La fonctionne suivante permet de récupérer l'état de la valve de contrôle , quand la température de la chambre soit inférieur à la température consigne ,ainsi nous allons ouvrir la valve pour envoyer l'aire chaud pour rétablir la température , le traitement est complètement automatisé par le serveur qui décidera son état selon la valeur de température consigne spécifié par l'utilisateur.

```
def on_pushButton_3_clicked(self):
    #Method pour Obtenir l'etat de la valve 1
    getValve1=self.textEdit_7.toPlainText()
    #getValve1="getvalve sys output"
   #Instruction a envoyer="getvalve sys output"
    if(self.selector==1):
        if((getValve1=="getvalve sys output")and(self.selector==1)):
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            try:
                sock.connect((HOST, self.PORT))
                sock.sendall(getValve1.encode())
                # Receive data from the server and shut down
                etatValve1 = sock.recv(self.RPORT).decode()
                self.textEdit_3.append(etatValve1)
                self.textEdit_2.append("**Requet getValve recu **")
            finally:
                sock.close()
        else:
            self.textEdit_2.append("Erreur Commande")
```

Fig5 : *implémentation en Python de la fonction état de Valve*

Dans le code nous allons comparer l'instruction **«getvalve sys output**» qui permet au serveur d' envoyer l'état de son valve après traitement. Nous récupérons cette valeur dans la variable etatValve1 qui sera affiché pour son visualisation dans le bloc textEdit3.

3.2 Programme du Serveur (Chambre thermique)

Afin de interpréter les différents commandes envoyées par l'utilisateur nous allons coder en python un Serveur qui permettra d'émuler la chambre thermique , il aura les fonctionnalités d'écriture et lecture ainsi que la comparaison de valeurs pour modifier sa réponse notamment dans le cas que valve de chauffage doive changer son état selon sa température consigne .

3.2.1 Tableau de paramètres Serveur

Afin de différencier les commandes envoyées par l'utilisateur sur le **Port 1024** nous allons utiliser la classe **re** acronyme d'**expression réguler(regex)** qui permet de séparer les mot d'un message pour pouvoir les stocké dans un tableau. L'analyse de différents mots contenus dans le message permettra au Serveur de choisir l'instruction à exécuter par exemple capture de la température, ou bien vérification de l'état de la valve.

```
def handle(self):
    self.data1 = self.request.recv(1024).strip()
    requetS=self.data1.decode()
    m=re.findall(r'\w{2,8}', requetS)#diviser les mot en groups de [2 a 8] caracteres
    requet=m[0]#recuperer la mot 0 depuis la liste  pour comparer dans la condition IF
```

Fig6 : *implémentation du regex dans le message utilisateur*

3.2.2 Exécution d'instructions Utilisateur

Dans cette étape nous allons exécuter les différents instructions de la chambre thermique, nous allons définir la variable global état Valve qui servira pour enregistre en mémoire l'état de la valve chaque fois que le serveur soit interrogé , ensuite nous définirons la **variable temp1Real** qu'indique la valeur actuel de température dans la chambre.

```
global etatValve1
temp1Real=35
if(requet=="gettemp"):
    self.request.sendall(str(temp1Real).encode())
elif(requet=="settemp"):
    temp1Consigne=int(m[1])
    if(temp1Real<temp1Consigne):
        etatValve1="OFF"
    else:
        etatValve1="ON"
    self.request.sendall(m[1].encode())
elif(requet=="getvalve"):
        self.request.sendall(etatValve1.encode())
else:
    message="errer de syntax  tapez le bonne Commande"
    self.request.sendall(message.encode())
```

Fig7 *: bloc if pour l'analyse des instructions*

Comme nous voyons sur l'image le système utilise un bloc if else pour identifier les 3 types de commandes possibles :**« gettemp », « settemp »** et **« getvalve »** , dans le cas de **settemp** , il y aura un bloc if additionnel pour changer l'état de la valve selon la comparaison de la valeur de température consigne et la valeur réel dans la chambre .

4 Système de Mesure 1 Client et 3 Serveurs

Maintenant nous allons épandre le système à 3 Serveurs, nous commencerons pour ajouter le code nécessaire dans le fichier client qui permettra de surveiller et initialiser les échanges de paramètres de température, état de valve et température consigne entre les différentes chambres de façon centralisée.

Ensuite nous allons générer 3 nouveaux fichiers Serveurs qui vont recevoir les requêtes depuis le Client Centralisé pour communiquer les données physiques des chambres thermiques.

4.1 Interface Graphique pour 3 Serveurs

Dans cette étape nous allons étendre le système de mesure de température à 3 chambres thermiques commandées de manière centralisé, dans la suite nous présentons l'interface graphique adaptée pour cette configuration réalisée sur QT Designer.

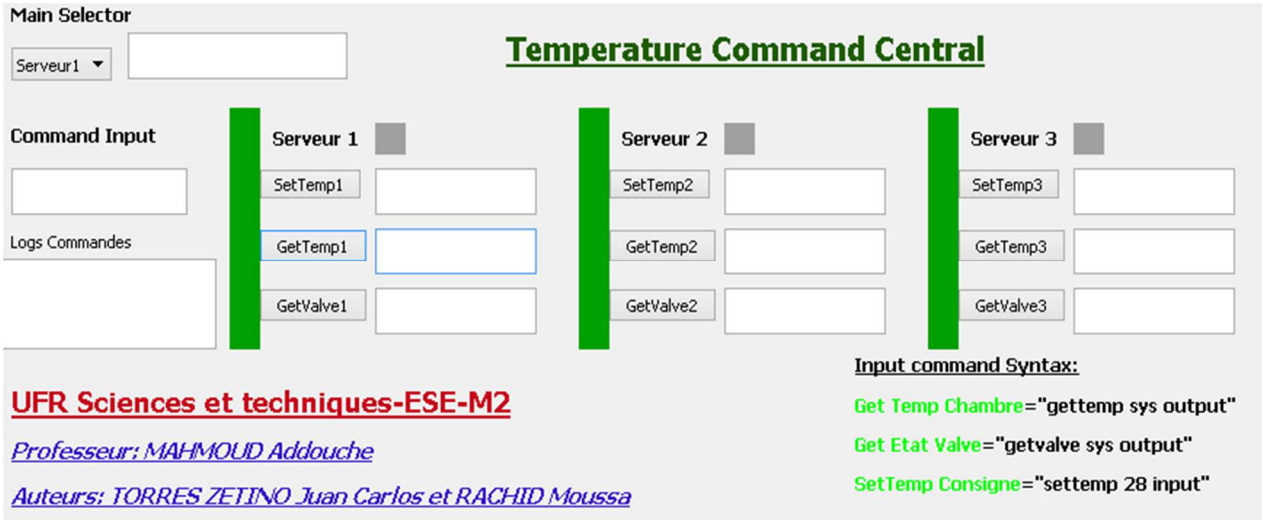


***Fig 8**: Système centralisé de température pour 3 Chambres thermique*

- **Main Selector :** c'est une Qcombobox qui permet de choisir le port de communication avec le serveur dans notre cas nous aurons PORT 1024 ,1025 et 1026
- **Bouton SetTempX**,GetTempX et GetValveX : ces boutons aurons un index x correspondant au serveur choisi par l'utilisateur , il permet d'envoyer de commandes de manière précis
- **Indicateur Serveur x**: ce sont des leds représentées par QMdiArea objects qui permettent d'indiquer si le serveur est active dans un moment déterminé

4.2 Programmation en mode Client -3Serveurs

Les modifications à effectuer sur le programme client seront les suivantes :

4.2.1 Fonctionne de Sélecteur de Chambre

Le code suivant permet de choisir le serveur à commander par le client, d'abord nous utilisons un combo box avec 3 ports pour établir la communication avec le serveur de manière précis.

Nous utilisons la méthode **comboBox.itemText** qui permet de récupérer la valeur en format string correspondant à l'option choisi par l'operateur, cette valeur sera stockée dans la variable C2, ensuite selon la valeur « Serveur1 » , « Serveur2 » et « Serveur3 » nous configurons le RPORT Récepteur 1024 , PORT 9999 et l'index sélecteur .

```
def on_comboBox_activated(self, p0):
    C2=self.comboBox.itemText(self.comboBox.currentIndex())
    brush1 = QtGui.QBrush(QtGui.QColor(160, 0, 0))
    brush2 = QtGui.QBrush(QtGui.QColor(160,160, 160))
    if(C2=="Serveur1"):
        self.mdiArea.setBackground(brush1)
        self.mdiArea_2.setBackground(brush2)
        self.mdiArea_3.setBackground(brush2)
        self.RPORT=1024
        self.PORT=9999
        self.selector=1
    elif(C2=="Serveur2"):
        self.mdiArea.setBackground(brush2)
        self.mdiArea_2.setBackground(brush1)
        self.mdiArea_3.setBackground(brush2)
        self.RPORT=1025
        self.PORT=9998
        self.selector=2
    elif(C2=="Serveur3"):
        self.mdiArea.setBackground(brush2)
        self.mdiArea_2.setBackground(brush2)
        self.mdiArea_3.setBackground(brush1)
        self.RPORT=1026
        self.PORT=9997
        self.selector=3
    self.textEdit_5.append("port"+str(self.RPORT))
```

Fig9 : *Code additionnel pour réaliser la sélection de Serveur (chambre Thermique)*

Ensuite nous allons utiliser le bloc if pour exécuter l'option depuis la combobox choisi par l'utilisateur, les variables globales du PORT de communication ainsi que l'index du serveur seront modifiées pour permettre au client de prend la commande sur le Serveur active .

Au même temps nous allons modifier les **objets mdiArea** qui serviront pour changer la couleur des leds selon l'activation où désactivation de Serveurs.

4.2.2 Adaptation des fonctionne du Clients pour 3 serveurs

Afin d'inclure les différents fonctions de requête pour tous les serveurs, nous allons récréer des fonctions auxiliaires

- **Récupération de la température dans chambre thermique**
- **Modification de température dans chambre thermique**
- **Récupération état de la valve de ventilation**

Chaque nouvelle fonction aura une condition if dans lequel la variable **selector changera selon le Serveur spécifié par l'utilisateur, ensuite nous exécuterons la requête uniquement s'il correspond au serveur active, sinon nous afficherons un message d'erreur « erreur commande » pour indiquer à l'utilisateur que sa requête n'a pas été pris .**

```
#*********************************************SERVEUR 2*****************************
    @pyqtSlot()
    def on_pushButton_4_clicked(self):
    #Method pour Obtenir l'etat de la valve 1
        getValve1=self.textEdit_7.toPlainText()
        #getValve1="getvalve sys output"
        #Instruction a envoyer="getvalve sys output"
        if(self.selector==2):
            if((getValve1=="getvalve sys output")and (self.selector==2)):
                sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                try:
                    sock.connect((HOST, self.PORT))
                    sock.sendall(getValve1.encode())
                    # Receive data from the server and shut down
                    etatValve1 = sock.recv(self.RPORT).decode()
                    self.textEdit_10.append(etatValve1)
                    self.textEdit_2.append("***Requet getValve recu ***")
                finally:
                    sock.close()
            else:
                self.textEdit_2.append("Erreur Commande")
```

```
#*********************************************SERVEUR 3*****************************
    @pyqtSlot()
    def on_pushButton_7_clicked(self):
    #Method pour Obtenir l'etat de la valve 1
        getValve1=self.textEdit_7.toPlainText()
        #getValve1="getvalve sys output"
        #Instruction a envoyer="getvalve sys output"
        if(self.selector==3):
            if((getValve1=="getvalve sys output")and (self.selector==3)):
                sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                try:
                    sock.connect((HOST, self.PORT))
                    sock.sendall(getValve1.encode())
                    # Receive data from the server and shut down
                    etatValve1 = sock.recv(self.RPORT).decode()
                    self.textEdit_11.append(etatValve1)
                    self.textEdit_2.append("***Requet getValve recu ***")
                finally:
                    sock.close()
            else:
                self.textEdit_2.append("Erreur Commande")
```

***Fig10 : implémentation** de fonctionnes spécifiques pour la commande de chaque serveur*

5 Vérification de Système Central de température

Les tests que nous allons présenter servirons pour analyser et valider le bon fonctionnement des différents types de requêtes envoyées vers les serveurs, en autre nous allons vérifier qu'on puisse échanger entre les 3 chambres thermiques depuis le client central et prévenir avec des messages d'alerte lors de l'application de commandes non définis .

5.1 Vérification fonction Réglage de température de la chambre

Le première étape consiste à sélectionner le serveur à activer sur le combobox de l'UI , ensuite on devra spécifier la valeur de température du serveur 2 à écrire , pour cela nous allons taper la commande **« settemp 37 input »** sur la ligne de commande ,ensuite on devra appuyer sur **SetTemp2** qui permettra de régler la température , finalement pour vérifier la bonne exécution de côté du serveur nous allons vérifier les commandes Logs Commandes .

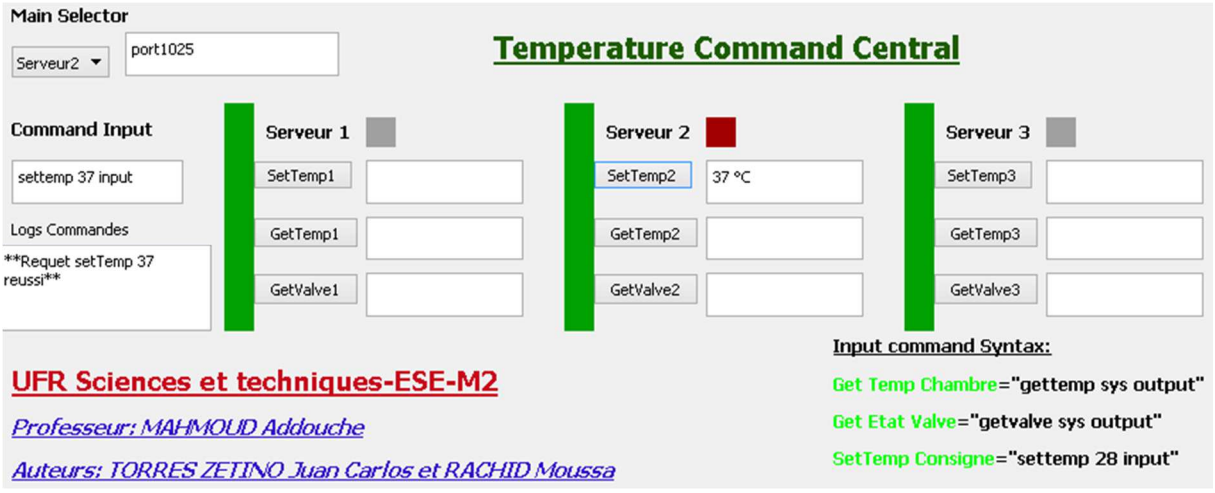


Fig11 : *Système de mesure à 3 Serveurs pour récupérer la température*

5.2 Vérification de fonctionne Récupération de Température pour différent Serveurs

En changeant le serveur actif dans la combobox nous allons taper l'instruction **« gettemp sys output »**dans la ligne de commande, ensuite nous allons appuyer sur **GetTempX** qui permet de récupérer la température de la chambre dans le bloc de texte de côté, nous pouvons finalement vérifier aussi l'exécution de commande dans la fenêtre Logs Commandes.

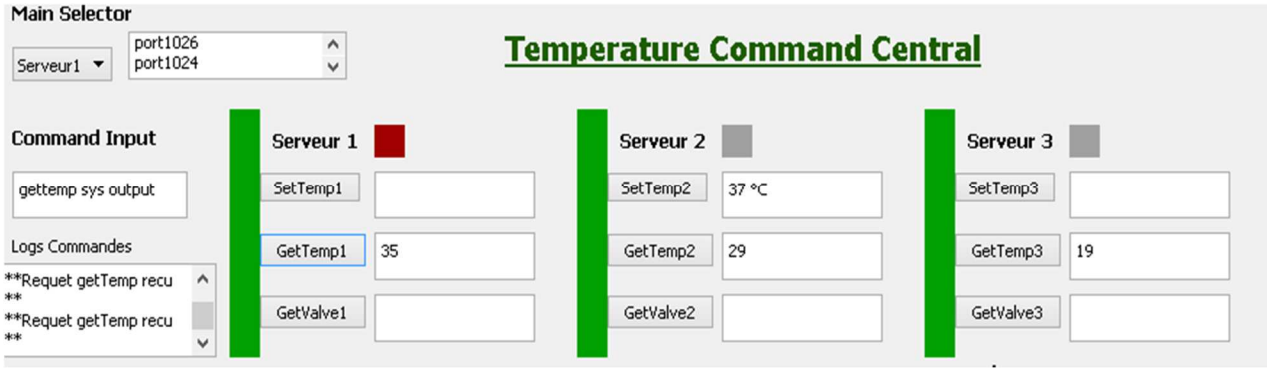


Fig12 : *test de récupération de la température Serveur 1*

5.3 Vérification Fonctionne d'état de valve pour 3 Serveurs

Pour connaitre l'état de la valve dans la chambre thermique 1 nous avons implémenté une commande SCPI qui est définis par la syntaxe « **getvalve sys output »** qui permettra d'interroger le serveur en envoyant une requête de l'état de la valve, si la température consigne est inférieur à la température dans la chambre alors la valve sera ON pour laisser passer de l'aire et refroidir le volume.

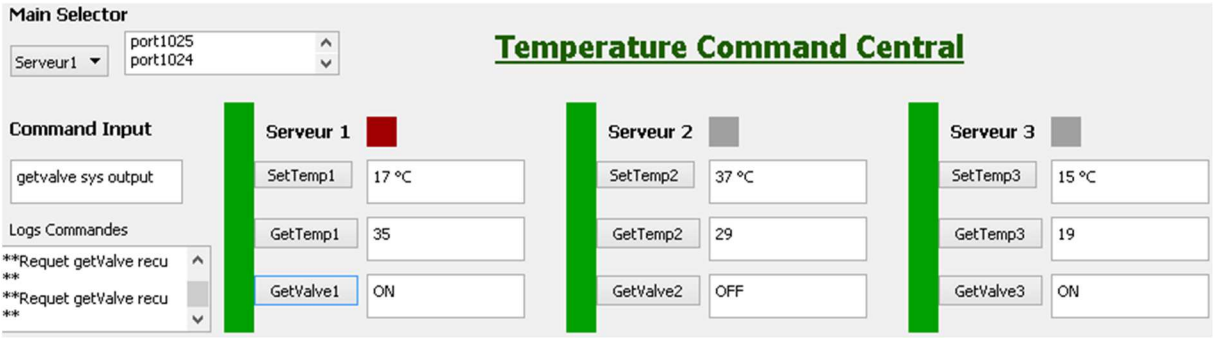
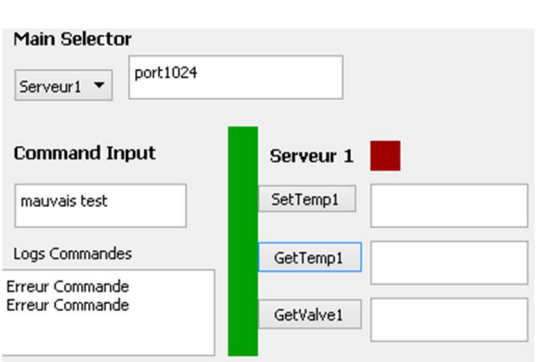
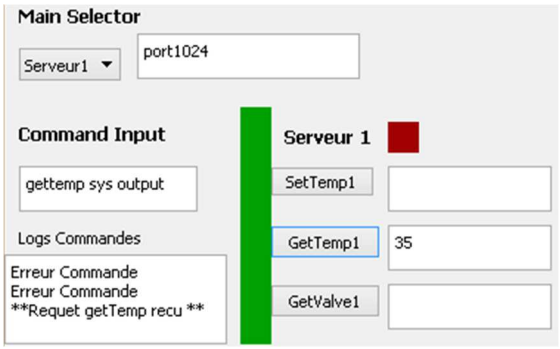


***Fig13** : état de la valve en ON pour décrémenter la température jusqu'à la valeur consigne*

5.4 Test d'Auto-vérification de syntaxe de commandes

Nous avons la possibilité de différencier entre les commandes **valides SCPI** avant l'envoie au serveur grâce à une condition if ajouté dans chaque sous-fonctionne , pour cela nous allons spécifier une syntaxe diffèrent à celle définis dans le dictionnaire de commandes, ainsi quand l'utilisateur envoie un mauvais commande le système affichera une alerte.





Ainsi, si nous envoyons de syntaxes différents le système répondra avec un message d'erreur pour indiquer à l'operateur la cause du problème, ensuite nous afficherons le message dans **le bloc de text Logs Command**, pour obtenir la valeur désiré du serveur il restera seulement à taper une commande valide reconnu par le serveur par exemple **«getemp sys output»** et nous aurons la réponse affiché sur le bloc de texte associe à la chambre d'intérêt.

6 Conclusion

Le présente TP nous a permis de comprendre l'utilisation du langage Python pour la réalisation d'une interface Graphique qui sera commandé par communication Ethernet Client-Serveur , nous avons utilisé des outils de dessin disponibles sur QT Designer qui offrent une environnement orientée Object afin d'implémenter notre application .

Python offre les avantages suivantes , une exécution immédiat sans compilation , une programmation orienté Object et une gestion automatique de la mémoire par rapport aux langages en C classique , en plus nous avons eu la possibilité d'utiliser de différents librairies pour la manipulation de chaînes , gestion client –serveur d'internet , interface graphique , etc. .