Lycée du Grésivaudan Meylan
BTS Systèmes Numériques
option Informatique et Réseaux
formation en temps plein

Projet informatique 2019

Dossier technique pour l'épreuve E6-2

Surveillance de Qualité de l'Air

S.Q.A

Dossier de conception

Partie personnelle

Arthur CÔTE

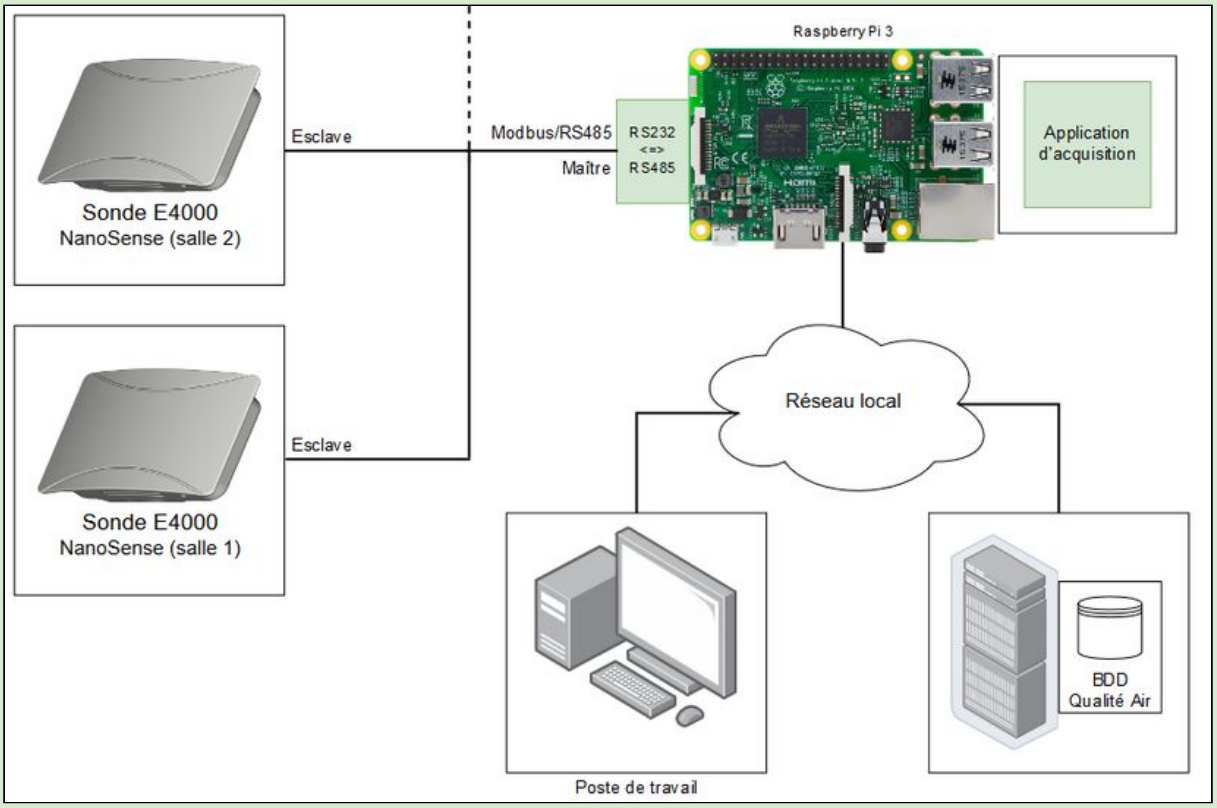
Sommaire:

A. Introduction

L'application d'acquisition consiste en l'envois périodique d'une requête modbus vers chacune des sondes installées et de la gestion des données reçues ainsi que du stockage de ces dernières dans la base de données.

B. Environnement de conception

1. Environnement matériel



- La Raspberry PI est un nano-ordinateur à faible consommation de courant électrique, elle convient donc à l'exécution en continue de l'application d'acquisition étant donné que l'application ne nécessite pas de hautes performances.

- L'adaptateur RS232 / RS485 permet la liaison entre la Raspberry et les sondes

- La sonde E400 est destinée à réguler le renouvellement de l'air en intérieur. Elle permet de relever des mesures de CO2, de COV, d'humidité (absolue et relative), de température et de température ressentie (fonction de l'humidité). Elle sera utilisée dans son mode de communication de base : Modbus.

- Modbus est un protocole maître-esclaves de communication développé par Modicon systems (racheté par Schneider). Il s'agit d'une technique de communication entre des systèmes informatiques, il a été conçu au départ pour des automates programmables. Dans un réseau ModBus standard il y a un maître et jusqu'à 247 esclaves. Le protocole Modbus peut être implémenté sur une liaison série asynchrone de type RS-485 ou RS-422 ou TTY (boucle de courant), avec des débits et sur des distances variables. On parle alors de Modbus Série. Ou bien sur TCP/IP sur Ethernet. On parle alors de Modbus TCP/IP.

2. Environnement logiciel

Plateformes : Windows 7, Raspbian
Outils de développement : PhpMyAdmin, Eclipse
Langages : JAVA, SQL
Bibliothèques : JLibModbus, JSSC, MySQL JDBC
Utilitaires : FileZilla, Putty, CAS Modbus Scanner
Protocole : Modbus

- **Le protocole Modbus :**

La mise en œuvre du protocole Modbus sur une liaison RS485 permet de construire un réseau maître/esclaves peu coûteux avec un débit de transmission allant de 1200 à 1,5 kbits/s. Une trame Modbus comprend :

- L'adresse comprise entre 0 et 247 de l'esclave concerné.
- Le numéro de fonction compris entre 1 et 255 que doit exécuter l'esclave (écriture, lecture). Dans une réponse normale l'esclave renvoie le même code de fonction que celui qu'il a reçu. Dans une réponse d'erreur (exception) l'esclave renvoie un code construit à partir du code reçu en mettant le MSB à 1.
- Les données fournies. S'il n'y pas d'erreur le champ données de la réponse contient les données demandées. S'il y a une erreur le champ données contient un code d'exception que l'application côté maître peut utiliser pour déterminer quelle action doit être entrepris.
- Le code de vérification d'erreur appelé contrôle de redondance cyclique sur 16 bits ou CRC16 calculé par le maître et envoyé à l'esclave. Le checksum est re-calculé par l'esclave et comparé à la valeur transmise par le maître. S'il y a une différence l'esclave ne construit pas la réponse.

Modbus offre 19 fonctions différentes. Elles se caractérisent par un code fonction sur un octet (exprimé en hexadécimal). Tous les équipements ne supportent pas tous les codes fonction.

- **JLibModbus :**

Il s'agit de l'implémentation du protocole Modbus en Java. Elle supporte plusieurs librairies de communication (jssc, rxtx, purejavacomm, java comm api). Elle implémente la quasi-totalité du protocole Modbus v1.1b.

- **JSSC :**

Java Simple Serial Connector (JSSC) permet la mise en oeuvre de la connexion série. Elle sera donc utilisé en parallèle avec JLibModbus afin d'employer le protocole Modbus.

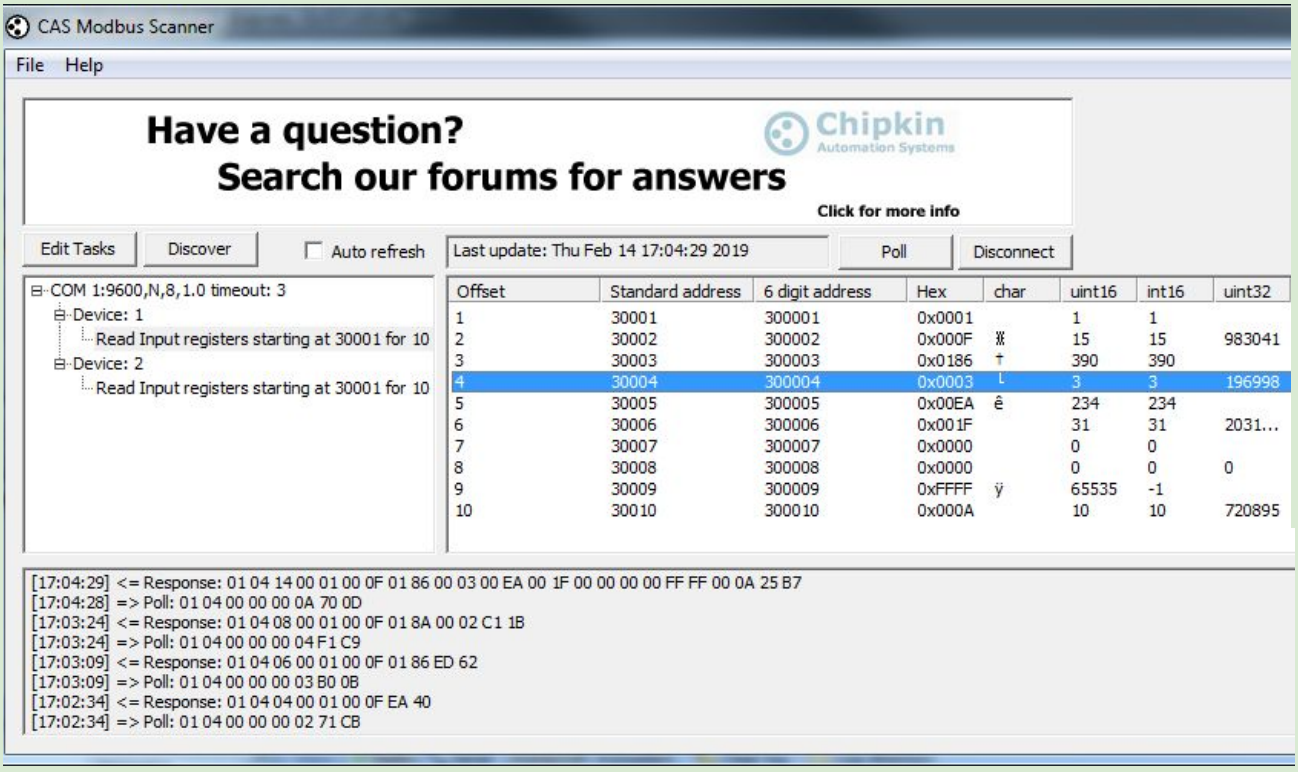
- **MySQL JDBC:**

Cette librairie permet la connexion aux bases de données ainsi que l'exécution de requêtes SQL. Elle sera utilisé pour la manipulation des données de la base de données.

3. Mise en oeuvre des matériels et des bibliothèques

Test préliminaire :

Afin de faire une première mise en oeuvre du protocole Modbus et de vérifier le bon fonctionnement des sondes et plus généralement du matériel, l'outil CAS Modbus Scanner permet de récupérer un premier jeu de valeurs.



Il faut ici entrer les paramètres de la connexion série (port COM, débit, bits de stop, parité...) puis enregistrer tous les appareils esclaves à l'aide de leur adresse. Ensuite il faut créer un requête avec un numéro de fonction, le décalage et la taille. On peut alors envoyer les requête et analyser la réponse.

Mise en oeuvre de JLibModbus :

JLibModbus rend possible l'utilisation du protocole Modbus en Java. Il est donc primordial la mettre en oeuvre étant donné que la communication avec les sondes est à la base de l'application.

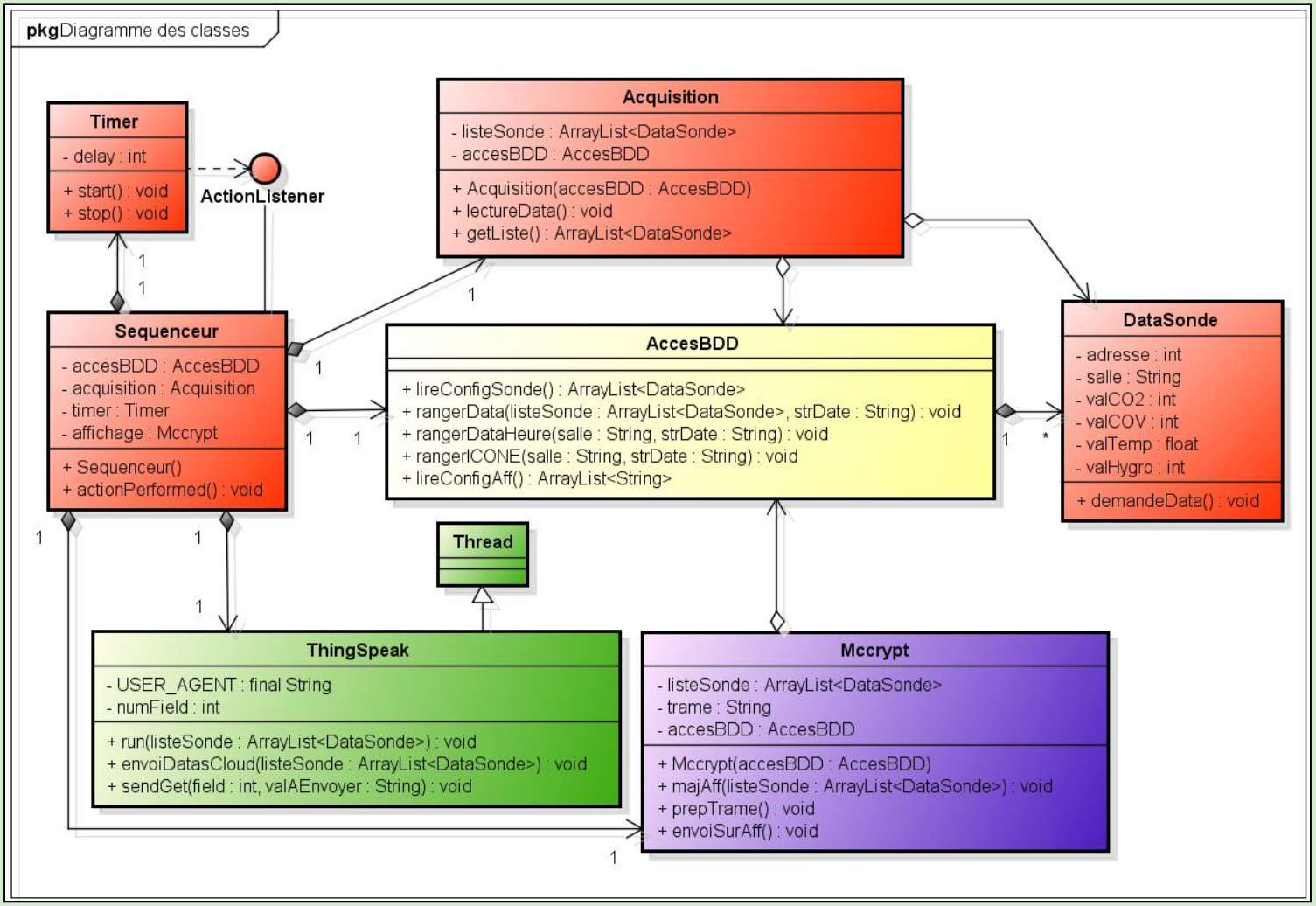
```
//preparation de la connexion
SerialParameters sp = new SerialParameters();
int[] registerValues = null;
try {
    String[] dev_list = SerialPortList.getPortNames();
    // si il y a au moins un port serie
    if (dev_list.length > 0) {
        sp.setDevice(dev_list[1]);
        sp.setBaudRate(SerialPort.BaudRate.BAUD_RATE_9600);
        sp.setDataBits(8);
        sp.setParity(SerialPort.Parity.NONE);
        sp.setStopBits(1);

        SerialUtils.setSerialPortFactory(new SerialPortFactoryJSSC());
        ModbusMaster m = ModbusMasterFactory.createModbusMasterRTU(sp);
        m.connect();

        int slaveID = adresse;
        int offset = 0;
        int quantity = 6;

        try {
            //demade des donnees
            registerValues = m.readHoldingRegisters(slaveID, offset, quantity);

            //rangement des donnees dans les attributs
            valCO2 = registerValues[2];
            valCOV = registerValues[3];
            valTemp = ((float)registerValues[4]/10);
            valHygro = registerValues[5];
```

Il faut tout d'abord créer un objet de type *SerialParameters* et y ajouter le port COM ainsi que les paramètres de la connexion série qui conviennent au convertisseur RS232 / RS485.
Il faut ensuite créer un objet de la classe *ModbusMaster* et lancer la connexion.
Il suffit alors d'exécuter la méthode *readHoldingRegisters* sur l'objet précédemment créé avec comme paramètres l'adresse de la sonde, où commence la requête et la taille de cette dernière. Cette méthode renvois la réponse de cette requête sous la forme d'un tableau d'entiers.On peut alors manipuler les données.
La connexion doit être terminée à l'aide de la méthode *disconnect* de la classe *ModbusMaster*.

C. Développement de l'application

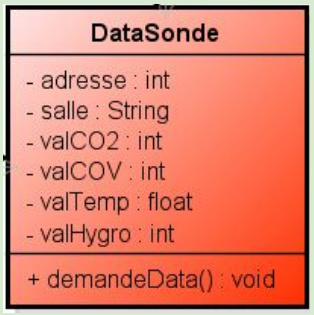
1. Diagrammes de classes et diagrammes dynamiques

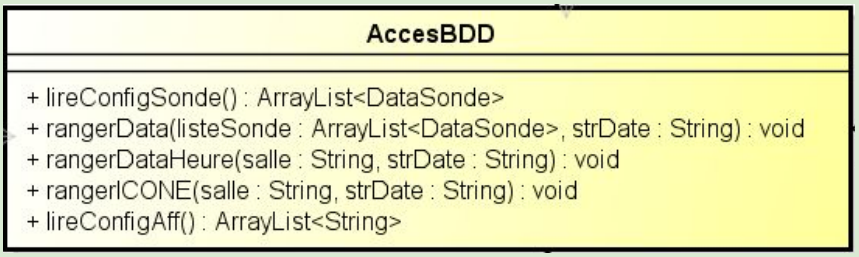
1.1. Diagramme des classes



1.2. Rôle des différentes classes

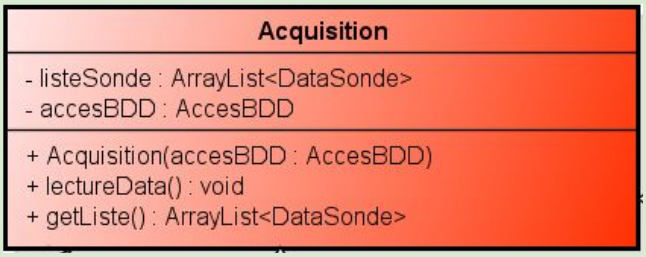
La classe *DataSonde* possède en attributs l'ensemble des données correspondant à une sonde ainsi que la méthode *demandeData* qui permet d'interroger la sonde en question et de remplir les valeurs de la qualité de l'air.





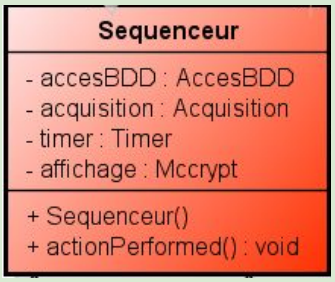
La classe *AccesBDD* contient les méthodes chargées des connexions à la base de données. Les méthodes concernant ma partie de l'application sont:

- *lireConfigSonde* qui lit la table *parametre_sonde* et renvois un tableau de DataSonde avec une DataSonde pour chaque ligne de la table.
- *rangerData* qui permet de ranger les données dans la BDD.
- *rangerDataHeure* qui permet de ranger la moyenne des valeurs sur la dernière heure.
- *rangerICONE* qui sert à calculer et ranger les valeurs d'icône sur la journée. L'ICONE est un indice de confinement de l'air pour les salles de classe, fondé sur la mesure de la concentration en dioxyde de carbone (CO2).



La classe *Acquisition* manipule le tableau de *DataSonde* et se charge d'appeler les méthodes d'*AccesBDD* et de *DataSonde* nécessaires.

La classe *Sequenceur* créé les objets nécessaires et est aussi chargée de la gestion du temps: elle est abonnée au *Timer*. À intervalle régulier, la méthode *actionPerformed* appelle les méthodes *lectureData* et *majAff*.

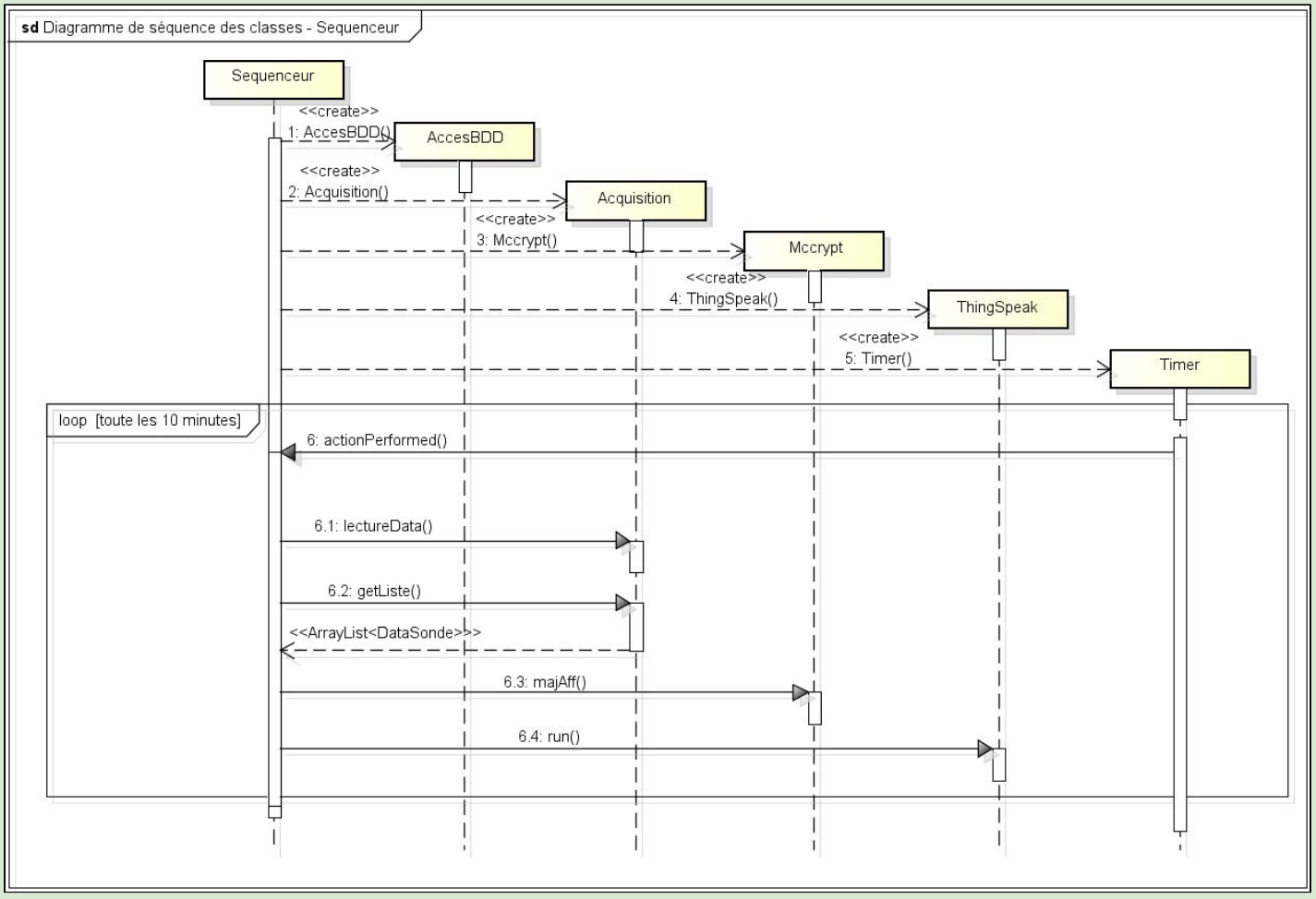


1.3. Justification des liens

La classe *Sequenceur* est reliée à *Acquisition et AccesBDD* par une composition car elle est chargée de leur création. *Acquisition* possède une référence sur *AccesBDD* afin de la manipuler.

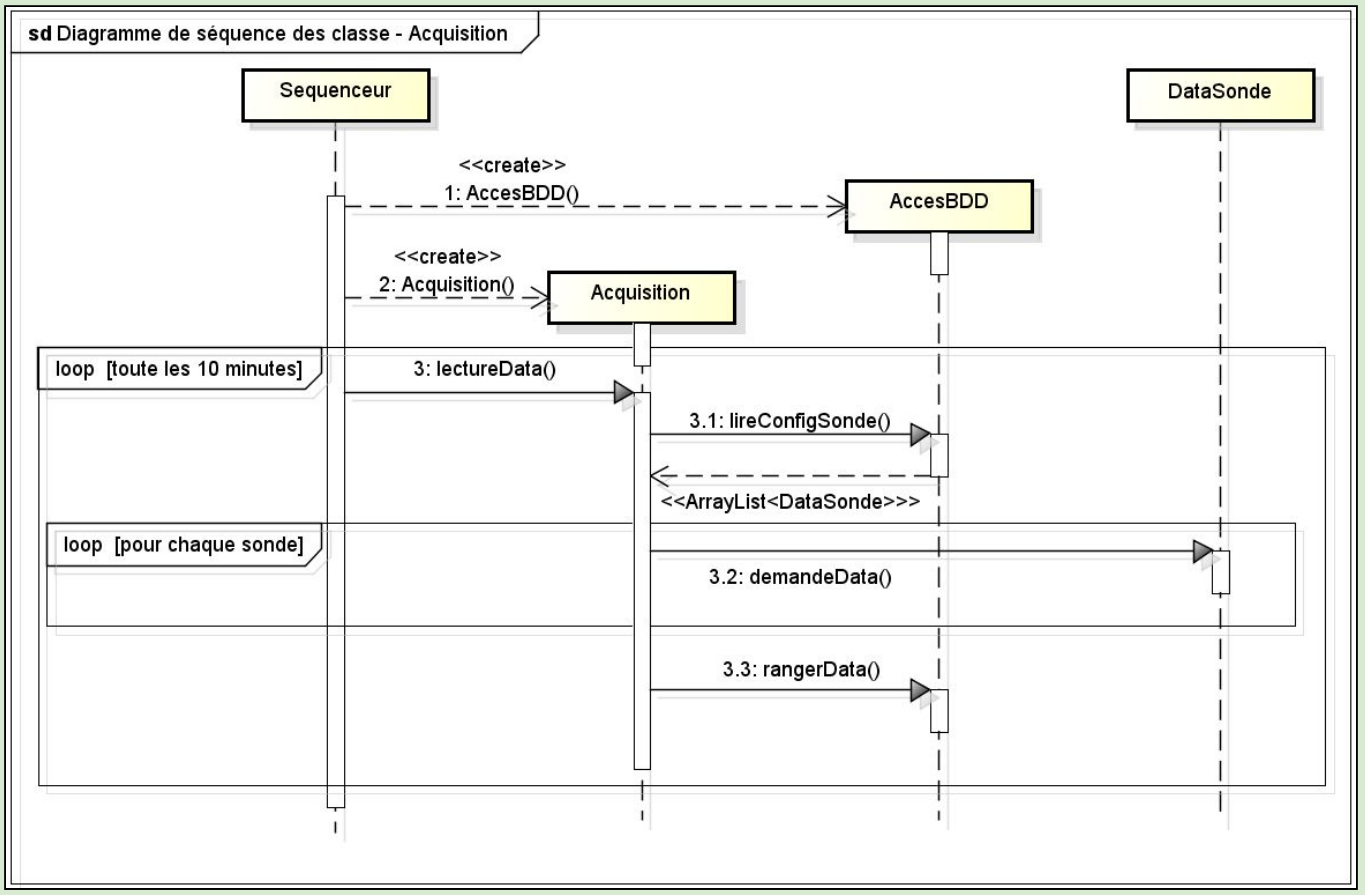
1.4. Présentation de la vue dynamique

Fonctionnement de la classe Sequenceur



La classe *Sequenceur* crée l'ensemble des autres classes, avec *AccesBDD* en premier afin de transmettre sa référence à *Acquisition* et *Mccrypt*.
Toutes les 10 minutes, le *Timer* appelle la méthode *actionPerformed()* qui va appeler la méthode *lectureData()* d'*Acquisition* afin de mettre à jour le jeu de donnée et l'envoyer à l'afficheur LED et la BDD. Elle va ensuite récupérer le tableau de *DataSonde* et le transmettre à *Mccrypt* grâce à *majAff()* et à *ThingSpeak* avec la méthode *run().*

Fonctionnement de la classe Acquisition



La méthode *lectureData()* consiste en la lecture de la configuration des sondes avec *lireConfigSonde()* d'*AccesBDD* puis le remplissage des données des *DataSonde* avec *demandeData()* et enfin le rangement des données sur la BDD avec *rangerData()*.

1.5. Présentation et justification des éléments de conception remarquables

Afin d'éviter tout problème lié à une différence de date et/ou de temps entre la raspberry et le serveur contenant la BDD, à chaque rangement de données dans la BDD, on récupère d'abord la valeur de la date de la raspberry (avec un objet de type *LocalDateTime*) qu'on rangera avec les données dans la requête SQL.

En vue que l'application s'adapte aux modification apportées aux sondes (nombre de sondes, adresses attribuées et les noms des salles assigné) et ainsi éviter quelconques intervention, la méthode *lireConfigSonde()* viens lire la table *parametre_sonde* afin de remplir un tableau de *DataSonde* avec un champ pour chaque ligne de la table.

Code source de *lireConfigSonde()*

```
public ArrayList<DataSonde> lireConfigSonde(){

    ArrayList<DataSonde> listeSonde = new ArrayList<DataSonde>();
    try {
        //connexion a la BDD 'sqa' en tant que 'pi'
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://172.16.125.140/sqa?useSSL=false";
        Connection conn = DriverManager.getConnection(url,"pi","raspberry");

        //preparation et envois de la requete SQL
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery("SELECT * FROM parametre_sonde");

        //on parcours la reponse, pour chaque ligne, on ajoute une nouvelle sonde dans la listeSonde
        try {
            while ( rs.next() ) {
                listeSonde.add(new DataSonde(rs.getInt("ADRESSE_SONDE"), rs.getString("NOM_SALLE")));
            }
        } catch (SQLException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }

        //fermeture de la connexion
        conn.close();

    } catch (Exception e) {
        System.err.println("Got an exception! ");
        System.err.println(e.getMessage());
    }

    return listeSonde;
}
```

Pour calculer et ranger la moyenne des données sur la dernière heure, la méthode *rangerDataHeure()* sélectionne dans la table *datas* la moyenne des données correspondant à une classe où le champ *DATES* est supérieur à la date actuelle moins une heure:

ResultSet rs = st.executeQuery("SELECT AVG(CO2), AVG(COV), AVG(TEMP), AVG(HUM) FROM datas WHERE SALLE=\""+salle+"\"AND DATES >= \""+strDate+"\"");

Il suffit ensuite de ranger ces données moyennes dans la table *datas_heures*:

st.executeUpdate("INSERT INTO datas_heures (ID, DATES, CO2, COV, TEMP, HUM, SALLE) VALUES (NULL, CURRENT_TIMESTAMP, '"+avgCO2+"', '"+avgCOV+"', '"+avgTemp+"', '"+avgHygro+"', '"+salle+"');");

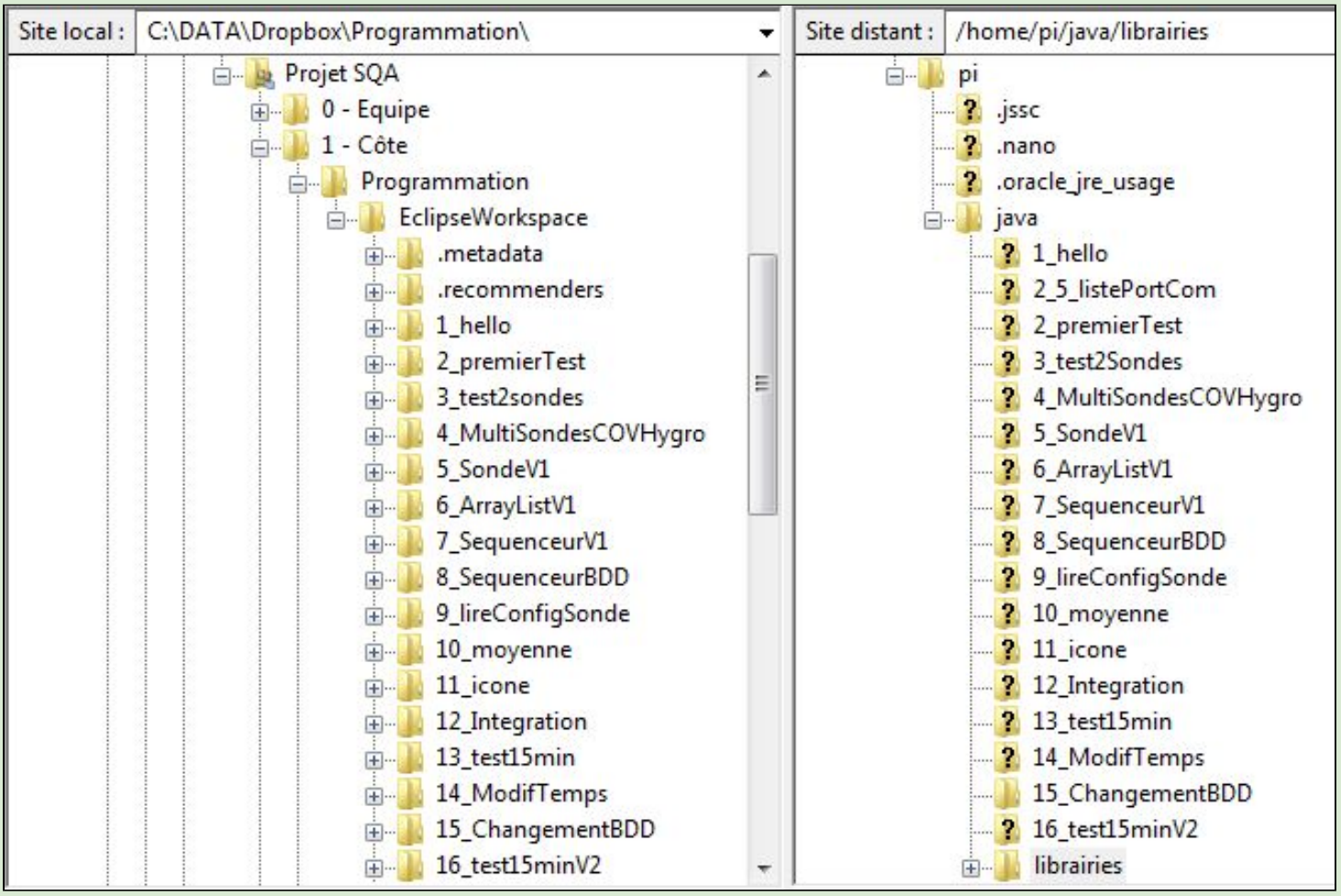
2. Tests unitaires

2.1. Les différentes étapes de la conception

Méthode de développement incrémentale:

1. Finalisation des branchements de la maquette.
2. Test préliminaires avec CAS Modbus Scanner.
3. Première mise en oeuvre de JLibModbus.
4. Récupération des données de 2 sondes.
5. Création de la classe *DataSonde*.
6. Création et prise en main du tableau de *DataSonde*.
7. Création de la classe *Sequenceur*.
8. Création de la classe *Acquisition*.
9. Création de la classe *AccesBDD*.
10. Intégration, améliorations et tests.

Arborescence du projet



2.2. Résultats du programme de test unitaire le plus abouti

```
pi@raspberrypi:~/java/14_ModifTemps $ java -classpath .:classes:/home/pi/java/librairies/j
libModbus/'*' Main
Debut du timer


Il est: 2019-05-28 10:25:22

Lecture de la configuration des sondes
SQL: SELECT * FROM parametre_sonde

Remplissage de listeSonde
SQL : INSERT INTO datas (ID, DATES, CO2, COV, TEMP, HUM, SALLE) VALUES (NULL, "2019-05-28
10:25:22", '578', '27', '22.3', '53', 'CP');
SQL : INSERT INTO datas (ID, DATES, CO2, COV, TEMP, HUM, SALLE) VALUES (NULL, "2019-05-28
10:25:22", '534', '40', '22.2', '54', 'CE1');


Il est: 2019-05-28 10:25:37

Lecture de la configuration des sondes
SQL: SELECT * FROM parametre_sonde

Remplissage de listeSonde
SQL : INSERT INTO datas (ID, DATES, CO2, COV, TEMP, HUM, SALLE) VALUES (NULL, "2019-05-28
10:25:37", '566', '25', '23.0', '53', 'CP');
SQL : INSERT INTO datas (ID, DATES, CO2, COV, TEMP, HUM, SALLE) VALUES (NULL, "2019-05-28
10:25:37", '534', '40', '22.4', '54', 'CE1');


Il est: 2019-05-28 10:26:22

Lecture de la configuration des sondes
SQL: SELECT * FROM parametre_sonde

Une heure est passee
SQL : SELECT AVG(CO2), AVG(COV), AVG(TEMP), AVG(HUM) FROM datas WHERE SALLE="CP"AND DATES
>= "2019-05-28 10:25:22"
INSERT INTO datas_heures (ID, DATES, CO2, COV, TEMP, HUM, SALLE) VALUES (NULL, CURRENT_TIM
ESTAMP, '570', '25', '22', '53', 'CP');
SQL : SELECT AVG(CO2), AVG(COV), AVG(TEMP), AVG(HUM) FROM datas WHERE SALLE="CE1"AND DATES
 >= "2019-05-28 10:25:22"
INSERT INTO datas_heures (ID, DATES, CO2, COV, TEMP, HUM, SALLE) VALUES (NULL, CURRENT_TIM
ESTAMP, '532', '39', '22', '54', 'CE1');
```

2.3. Etude critique de la couverture des tests.

En ce qui concerne la couverture des tests, elle n'est pas suffisante pour une application professionnelle. Nous avons testé la modification des adresses des sondes, la suppression et l'ajout de sondes. Le principal défaut concernant la couverture des tests est l'absence de gestion des logs. En effet le déroulement du projet étant sur une période trop restreinte, je n'ai pu le mettre en place dans les délais nécessaire.

D. Intégration progressive

1. Justification des étapes d'intégration.

La première étape de l'intégration fût l'intégration de l'application de pilotage de l'afficheur à LED. Grâce au concept objet ainsi qu'au travail d'analyse effectué au préalable, l'intégration de cette application ne nécessita que de légères modifications (tel que l'adresse de la BDD par exemple).
La deuxième étape consistait en l'intégration de l'application cloud. Similairement à la première étape, cette intégration se déroula sans difficultés remarquables.

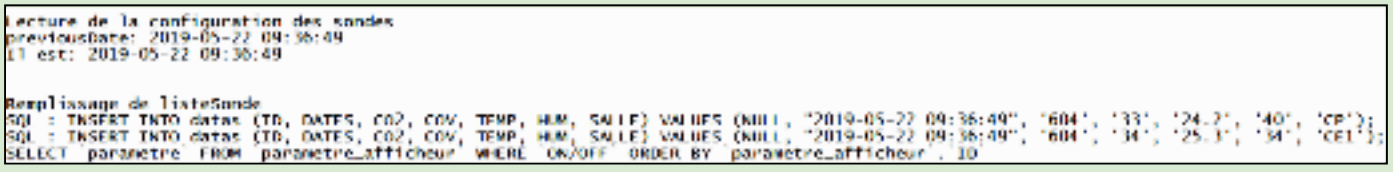
Grâce au concept objet, l'intégration consista donc simplement en l'import d'une classe, sa création et l'appel d'une de ses méthodes:

- *majAff()* de *Mccrypt* pour l'application de pilotage de l'afficheur à LED.
- *start()* de *ThingSpeak* pour l'application cloud. Cette application est donc exécuté dans un thread à part car son exécution nécessite 20 secondes par sondes à cause des limitations de ThingSpeak.

2. Tests d'intégration

Afin d'effectuer le premier test d'intégration, il faut vérifier si les données de l'afficheur LED sont bien mise à jour à chaque nouveau jeu de donnée ainsi que leur validité.
Le second test d'intégration est fortement similaire au premier. En effet, seul la plateforme sur laquelle les données sont envoyées est changé. Ici elle est en ligne et non physique.

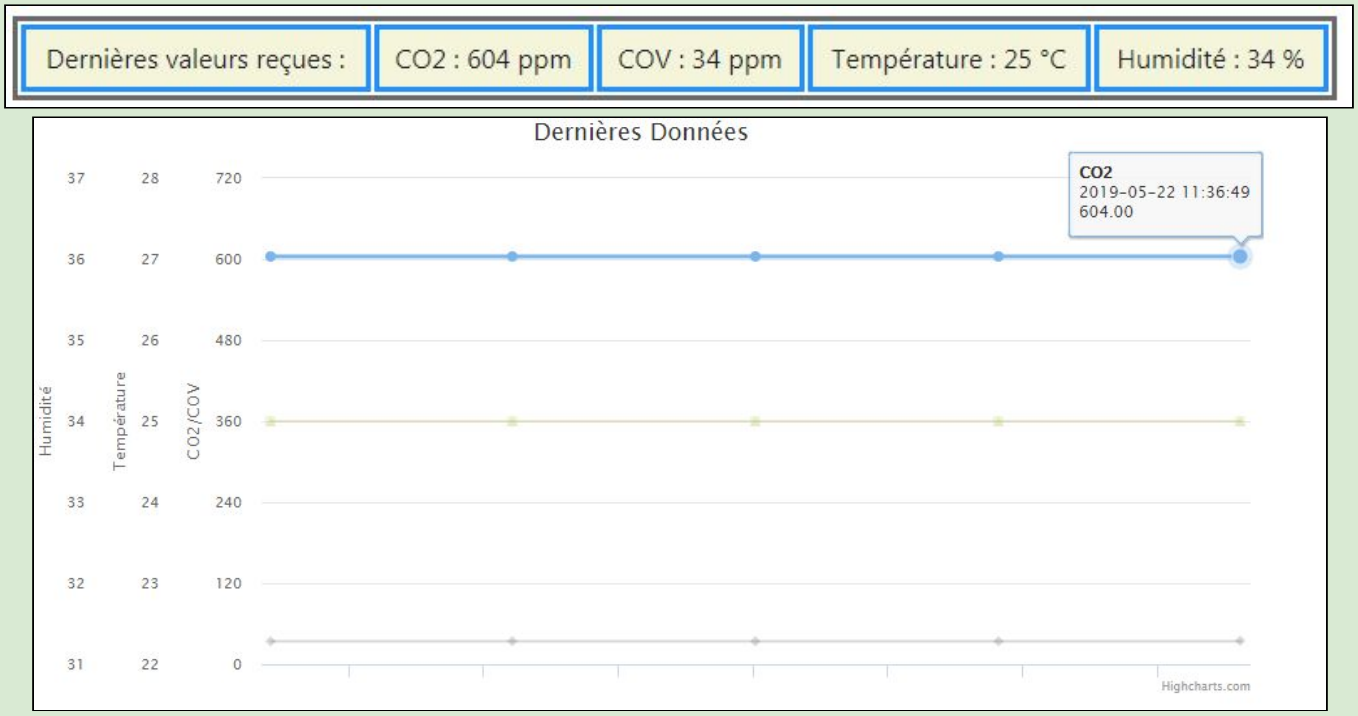
Valeurs affichées sur l'application d'acquisition:



Valeurs rangés dans BDD:

2019-05-22 09:36:49	604	33	24	40	CP
2019-05-22 09:36:49	604	34	25	34	CE1

Valeurs affichées sur le site WEB:



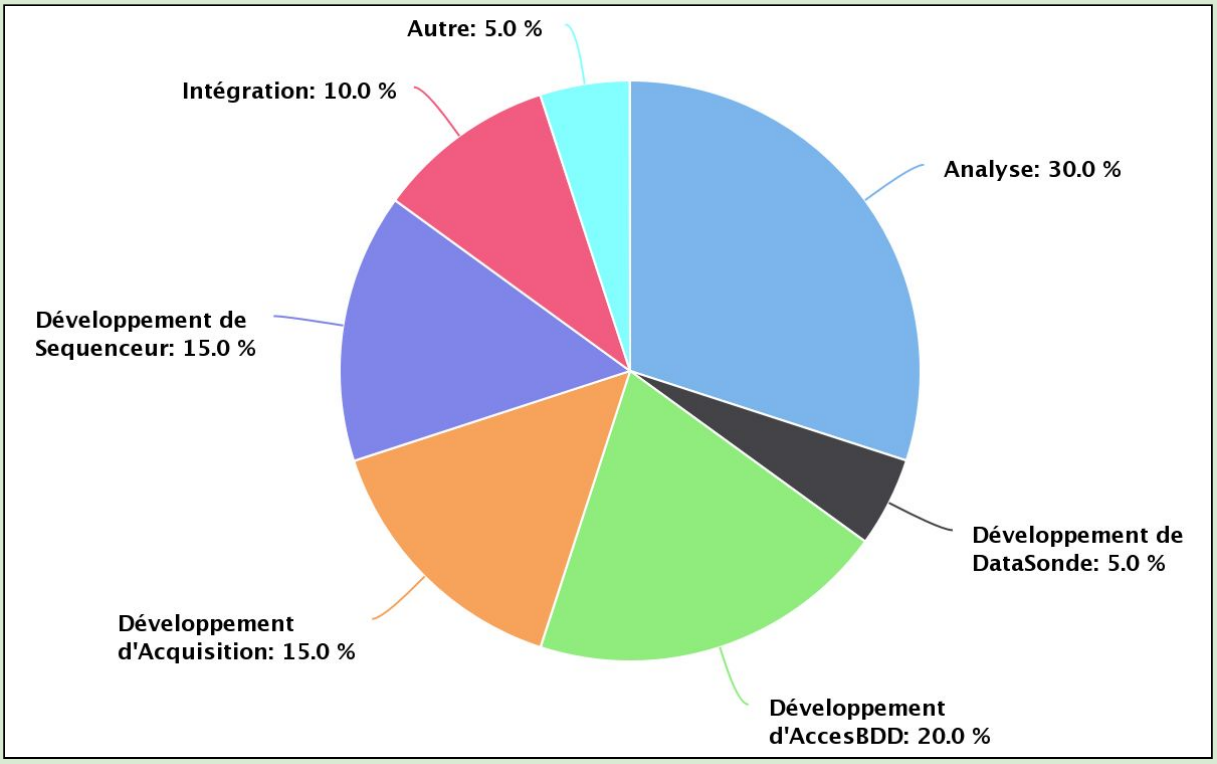
Valeurs affichées sur l'afficheur LED:

CP CO2:604ppm COV:33ppm Temp:24.2°C Humid:40%
CE1 CO2:604ppm COV:34ppm Temp:25.3°C Humid:34%

Valeurs affichées sur l'application cloud ThingSpeak:

E. Conclusion

Répartition du temps:



En conclusion, l'analyse, malgré avoir nécessité beaucoup de temps, s'est révélée fortement utile notamment lors de la phase d'intégration. Ce projet m'a apporté des connaissance à la fois techniques (Java, prise en main d'un librairie…) et aussi pédagogique (il s'agissait de mon premier projet tutoré). J'ai aussi eu l'occasion de me rendre contre de l'importance du travail d'équipe.