Projet trottinette :

Revue finale

Étudiants : ALCAIS Loïc, LETOCART Nicolas, HOUBAD Samir

Table des matières

Introduction.....3

Présentation générale du système supportant le projet.....4

Planification prévisionnelle et tâches à réaliser.....5

Contraintes matérielles et logicielles.....6

Mise en œuvre du matériel.....11

Présentation.....11

Configuration des modules xBee :.....12

Représentation du système avec et sans simulateur.....15

Protocole de communication :.....18

Cas d’utilisation « Géolocaliser une TTE ».....20

Scénario « Géolocaliser une TTE ».....20

Glossaire.....23

Annexe: QT → Signal/Slot.....24

Annexe : Documentation module xBee Pro.....26

Annexe:Configuration des modules xBee avec Cutecom.....27

Introduction

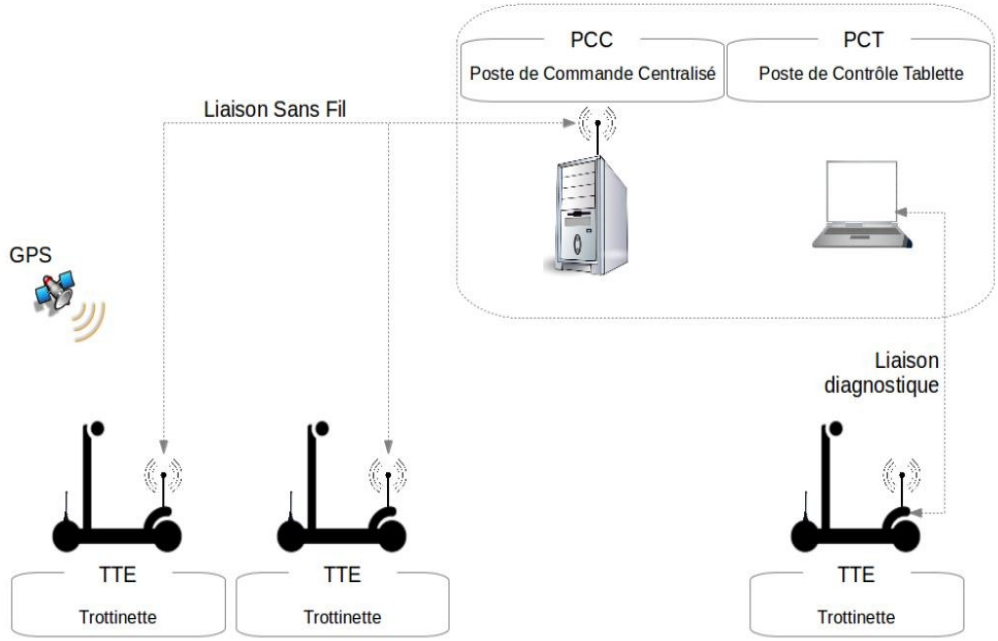
L’exploitant souhaite mettre à disposition des trottinettes tout terrain électrique pour ses clients au sein de l’éco quartier de Beaulieu à Monteux. De plus, pour des raisons de sécurité, des règles de circulation seront prescrites (vitesse, zones réservées, …). D’autre part, l’exploitant souhaite pouvoir arrêter le fonctionnement des trottinettes en cas de sortie de zones.

Présentation générale du système supportant le projet

Pour ce faire nous devrons :

- Géolocaliser les trottinettes en location.
- Détecter les limites de zones.
- Brider la trottinette afin d'immobiliser les trottinettes en cas de sortie de zones, de réguler la vitesse maximale.
- Lire les paramètres de fonctionnement des trottinettes via une prise.

Ainsi, nous récupérerons les données des trottinettes à l'aide de capteurs qui seront transmis au microcontrôleur. Par la suite, via le mode de transmission sans fil Zigbee, les données seront envoyées au poste de contrôle centralisé.



Les tâches à réaliser seront réparties sur 3 étudiants :

2 étudiants EC :

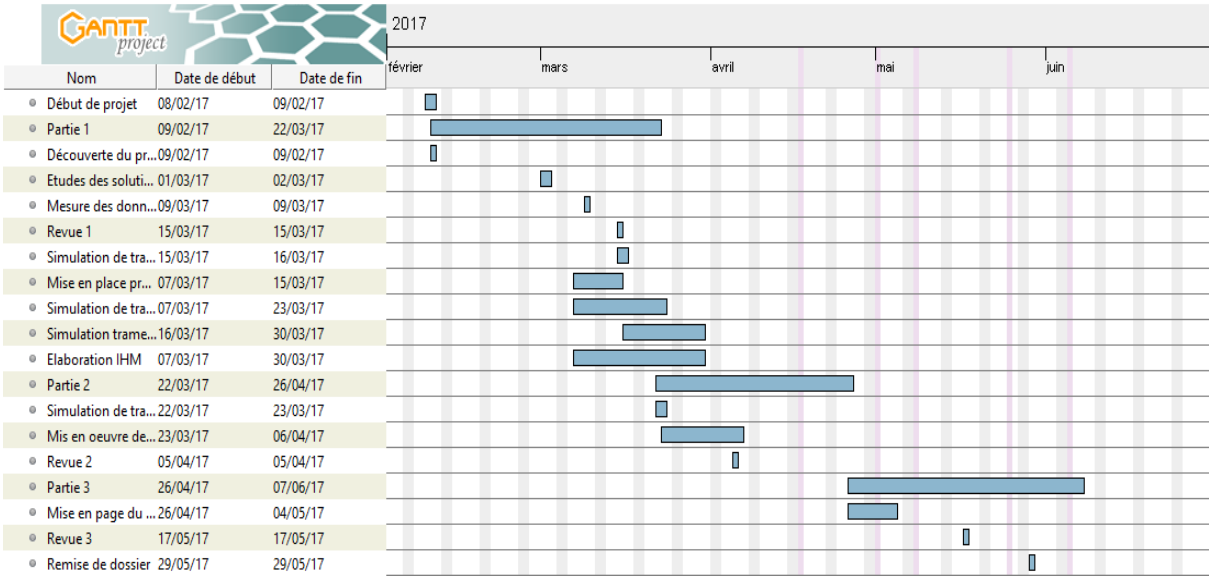
Loïc Alcais devra brider la trottinette, acquérir et stocker les informations issues des capteurs pour ensuite les transmettre au microcontrôleur. À partir de celui-ci les données seront transmises au « PCT » et au « PCC ».

Nicolas Létocart devra géolocaliser les trottinettes, émettre les données (géolocalisation, paramètres de fonctionnement) vers le PCC, réceptionner les ordres de l'exploitant, détecter les limites de zone et enfin pouvoir immobiliser les trottinettes en cas de sortie de zone de circulation.

1 étudiant IR : Samir Houbad devra récupérer les données (localisation, vitesse, charge de la batterie) des trottinettes, envoyer les ordres de supervision, visualiser les données dans une IHM et enfin créer une journalisation des états, données et alarmes.

Planification prévisionnelle et tâches à réaliser

Pour ce faire nous avons établit une planification du temps de travail afin de pouvoir finir le projet dans les temps :



Nom	Date de début	Date de fin
Début de projet	08/02/17	09/02/17
Partie 1	09/02/17	22/03/17
Découverte du pr...	09/02/17	09/02/17
Etudes des soluti...	01/03/17	02/03/17
Mesure des donn...	09/03/17	09/03/17
Revue 1	15/03/17	15/03/17
Simulation de tra...	15/03/17	16/03/17
Mise en place pr...	07/03/17	15/03/17
Simulation de tra...	07/03/17	23/03/17
Simulation trame...	16/03/17	30/03/17
Elaboration IHM	07/03/17	30/03/17
Partie 2	22/03/17	26/04/17
Simulation de tra...	22/03/17	23/03/17
Mis en oeuvre de...	23/03/17	06/04/17
Revue 2	05/04/17	05/04/17
Partie 3	26/04/17	07/06/17
Mise en page du ...	26/04/17	04/05/17
Revue 3	17/05/17	17/05/17
Remise de dossier	29/05/17	29/05/17

Tâches à réaliser de l’étudiant Houbad Samir

- Récupérer les données de localisation et de paramètres de fonctionnement des trottinettes

- Envoyer les ordres de supervision

-Visualiser les données dans une IHM

-Archiver les états, données et alarmes dans une base de données

Contraintes matérielles et logicielles

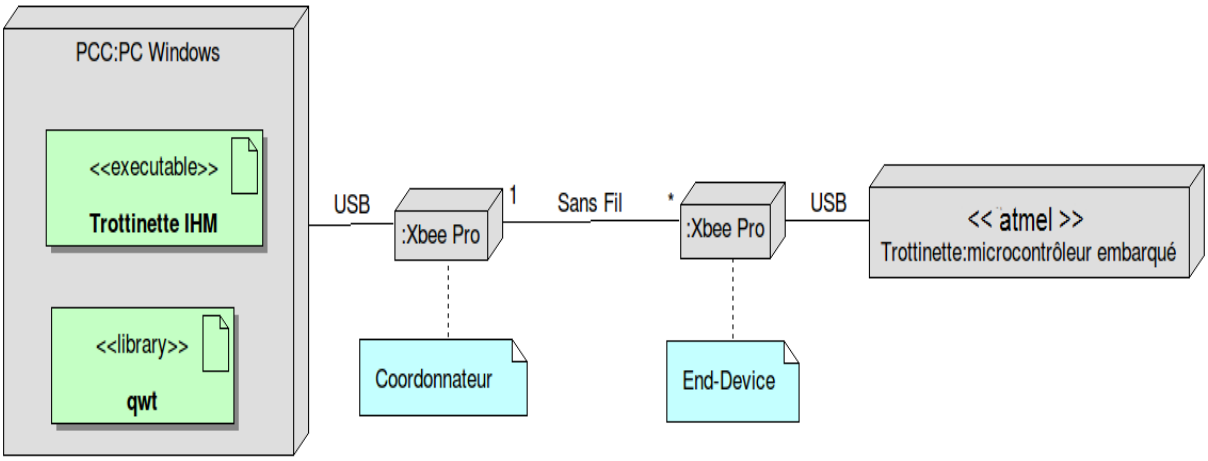
Les ressources matérielles mises à disposition pour mener à bien le projet sont :

Désignation	*Caractéristiques techniques*
SXT 1000 XL	Trottinette Electrique Tout Terrain (Plomb 48V 12Ah) de la marque SXT
C_ATMEL	Carte de développement SAM4S-EK d'ATMEL ou équivalente
PCC	PC HP sous © Microsoft Windows
PCT	PC HP Tablet sous © Microsoft Windows
SANSFIL	Système de communication sans fil
GPS	Adafruit Ultimate GPS Breakout (66 canaux avec mise a jour 10 Hz)
SD	Carte (*Secure Digital*) carte mémoire amovible de stockage de données numériques minimum 1GO
USB-RS232	Adaptateur USB / RS232

Ainsi que les logiciels sont :

Environnement de développement	© ATMEL Studio
Système d'exploitation du PCC et PCT	© Microsoft Windows
Système d'exploitation de développement du PCC et PCT	© Microsoft Windows
Système de gestion de bases de données relationnelles	SQLite3
Gestion et administration de bases de données	sqliteman ou SQLiteManager
Environnement de développement du PCC et PCT	Qt Creator 3.6.1 et Qt Designer
Compilateur du PCC et PCT	MinGW (*Minimalist GNU for Windows*)
API GUI	Qt 5.6
Atelier de génie logiciel	bouml 4.23
Plate-forme de tests unitaires du PCC et PCT	CppUnit 1.12.1
Logiciel de gestion de versions du PCC et PCT	subversion (client TortoiseSVN 1.9.3)
Générateurs de documentation du PCC et PCT	Doxygen version 1.8.11 et pandoc 1.17.0.2
Gestionnaire de projet	Planner (version 0.14.5) ou gantter

Diagramme de déploiement représentant les structures du système ainsi que leurs relations

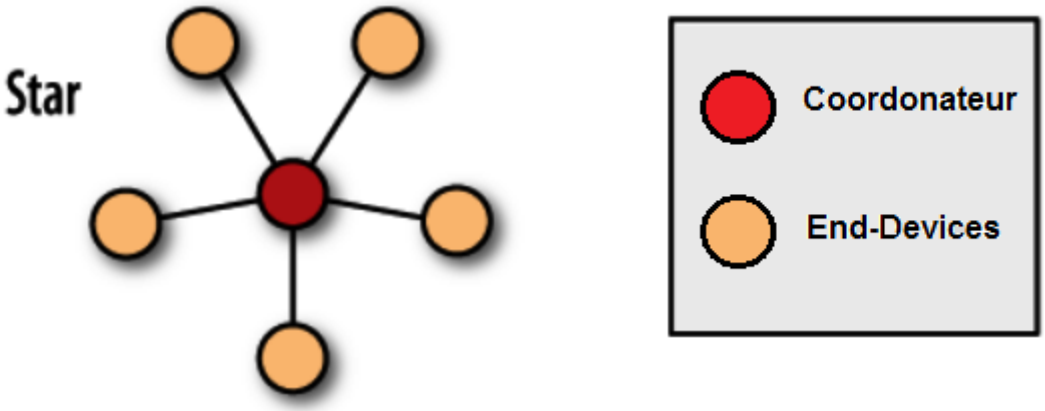


Ainsi au vue du diagramme de déploiement, la partie de Nicolas LETOCART qui repose sur la **carte atmel** est lié à ma partie (PCC).

Les données que nous nous enverrons seront transmises à l'aide des modules xBee Pro S5, 2 types de module sont utilisés :

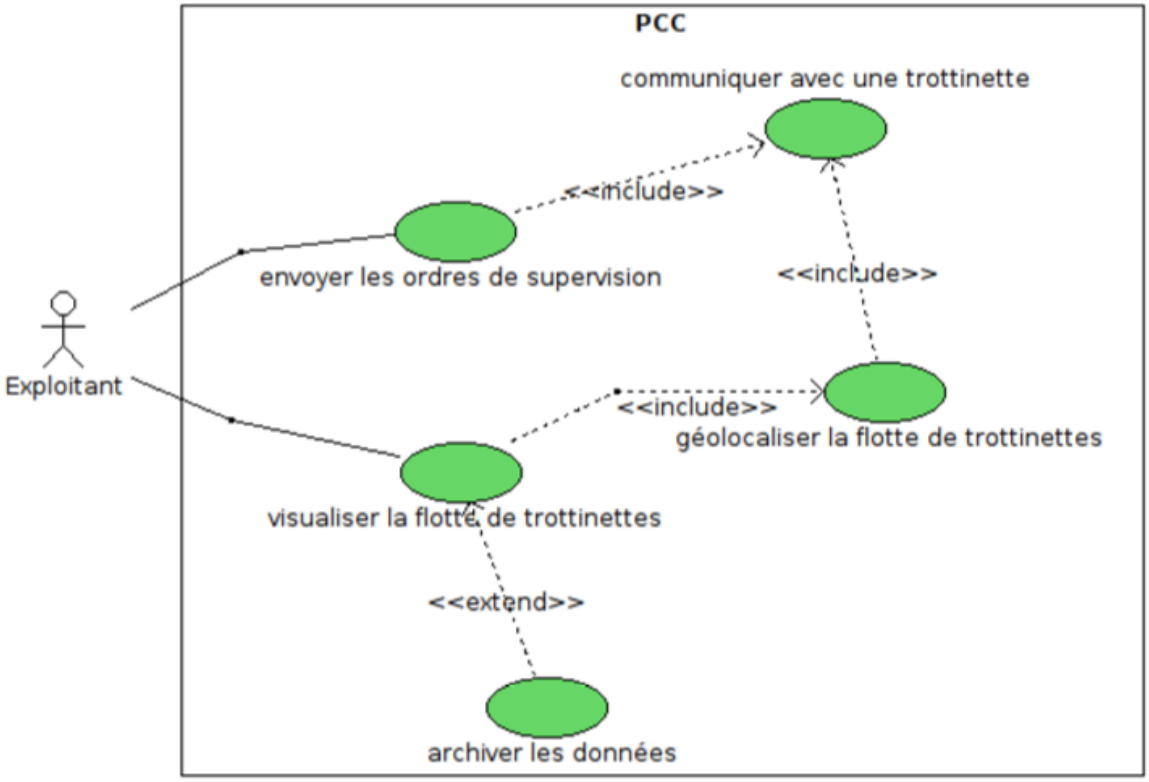
- Le coordonnateur à la fois primordiale et unique au sein d'un même réseau gère l'ensemble du réseau notamment il administre l'adhésion des **nœuds** au réseau.
- Les « End-Devices» sont des modules terminaux comme par exemple des capteurs.

La **topologie physique** est de type **étoile** :



Ainsi ma partie se concentre sur le poste de commande centralisé (c'est à partir de ce poste que l'exploitant de l'éco quartier supervisera sa flotte de trottinettes tout terrain électrique « TTE »).

Diagramme de cas d'utilisation représentant les fonctionnalités du poste de contrôle centralisé « PCC ».



Nous pouvons constater que lorsque la communication se fera avec une trottinette, il sera possible de géolocaliser la flotte de trottinettes et ainsi envoyer les ordres de supervision si nécessaire. De même pour visualiser la flotte de trottinettes dans une IHM, il faut avoir au préalable géolocaliser la flotte ainsi la communication avec une trottinette est primordiale.

Les données spécifiques aux trottinettes (localisation, paramètres de fonctionnement) pourront optionnellement être archivées.

À partir du PCC, via l’IHM l’exploitant pourra donc :

- Visualiser la flotte de trottinettes sur la carte représentant l’éco-quartier de Beaulieu.
- Envoyer les ordres de supervision.
- Visualiser les données de chaque trottinettes.
- Observer la réception des trames provenant des trottinettes.

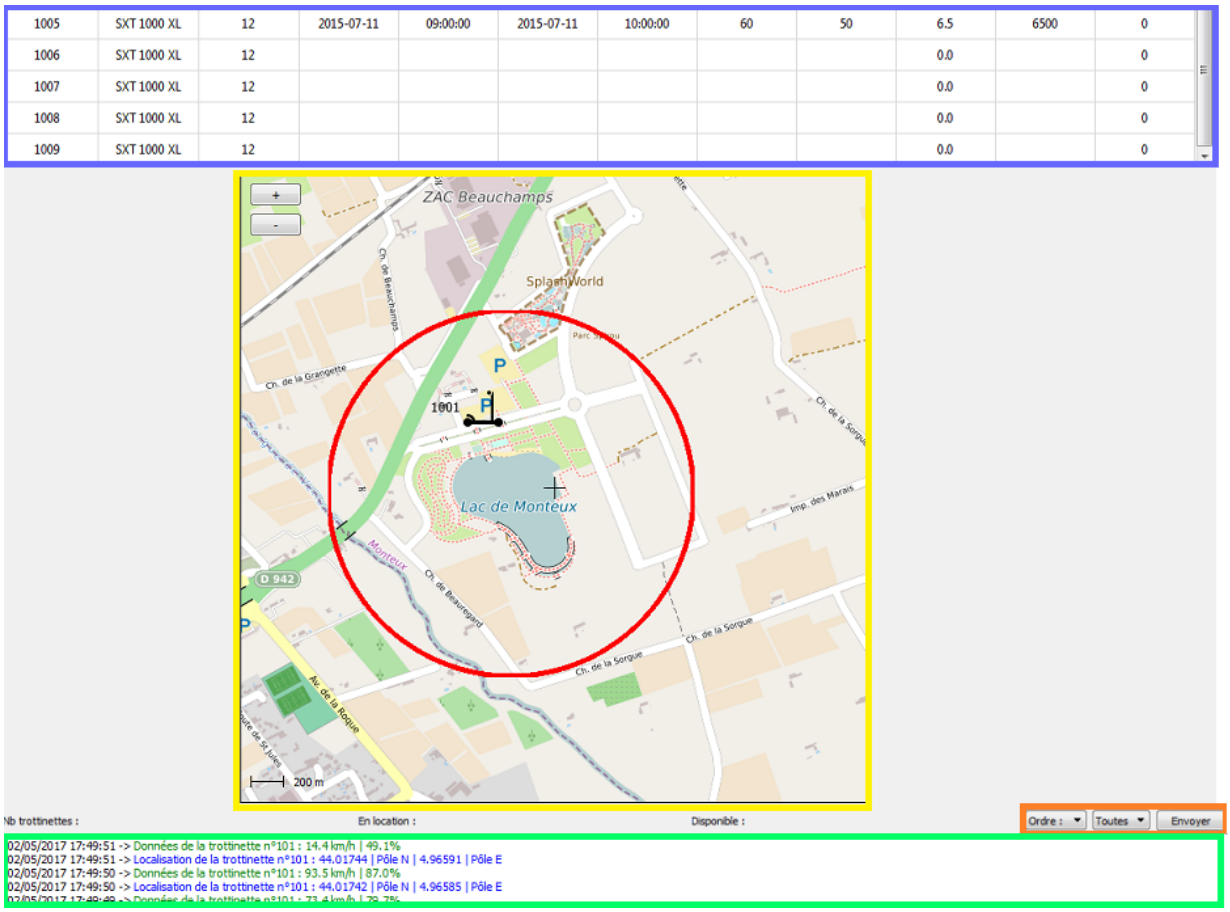
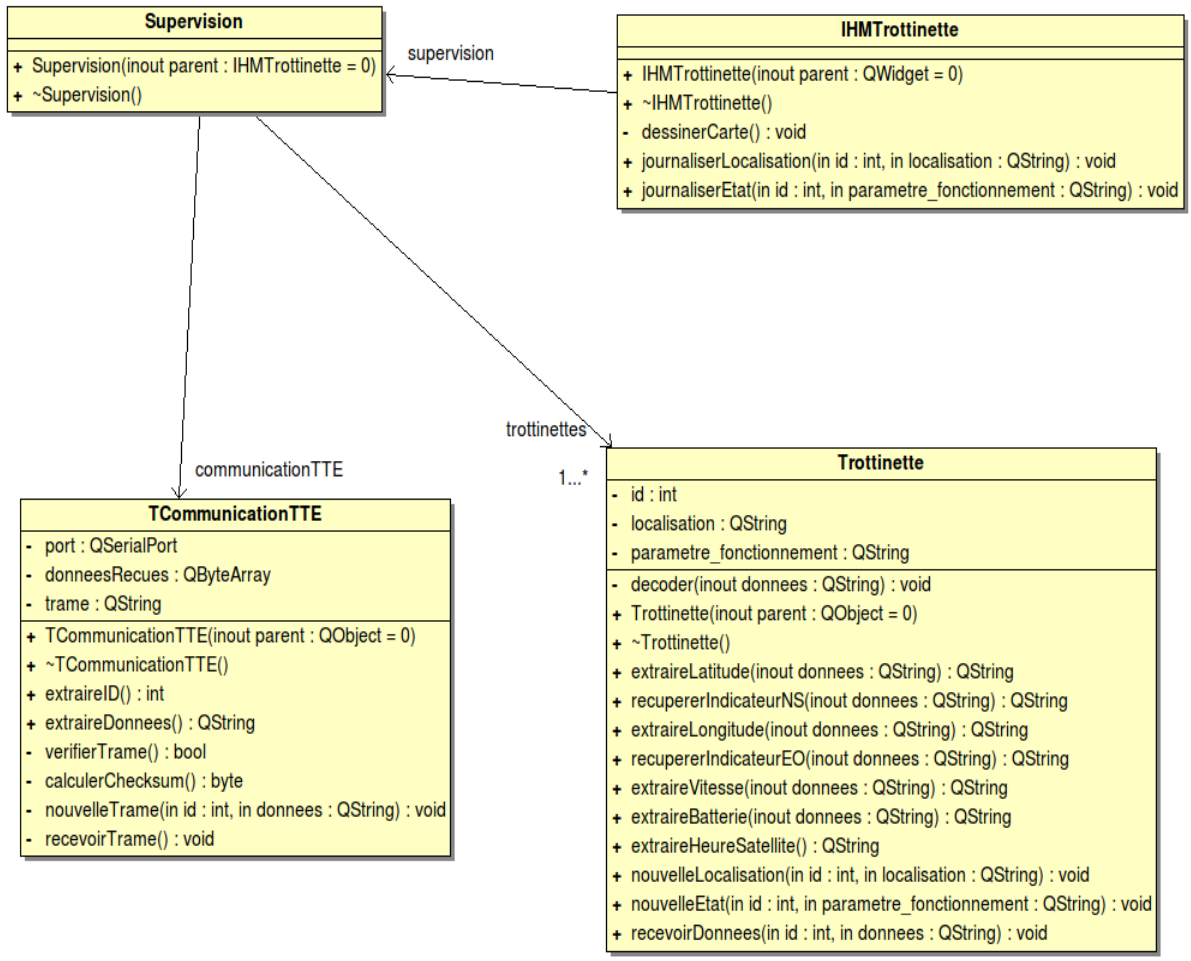


Diagramme des classes



-La classe IHMTrottinette permet d'afficher les données de localisation et de paramètres de fonctionnement ainsi que d'afficher la carte du lac de Beaulieu.

- La classe Supervision envoie des ordres aux trottinettes et transmet la localisation GPS ainsi que les données des trottinettes à l'IHM.

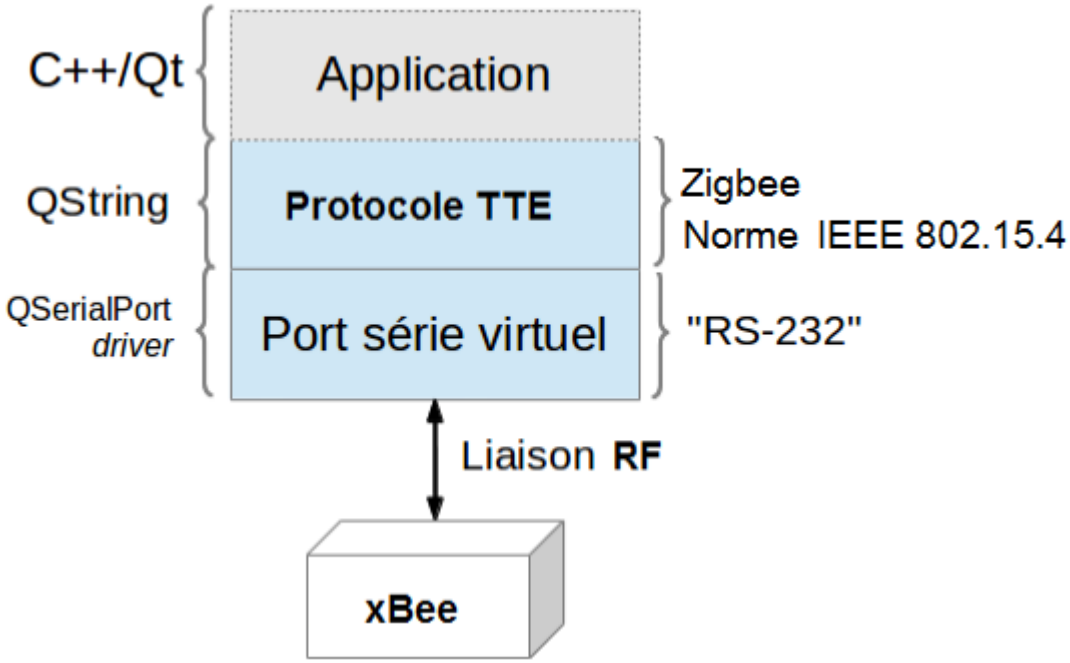
- La classe TcommunicationTTE réceptionne les trames, vérifie si elles sont valides et extrait les données des trames (l'identifiant de la trottinette et ses données).

- La classe Trottinette décode les données des trames afin de distinguer la localisation des trottinettes et leurs paramètres de fonctionnement.

Mise en œuvre du matériel

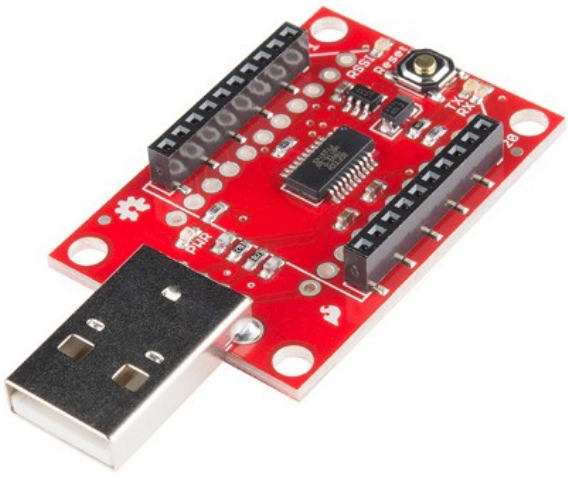
Présentation

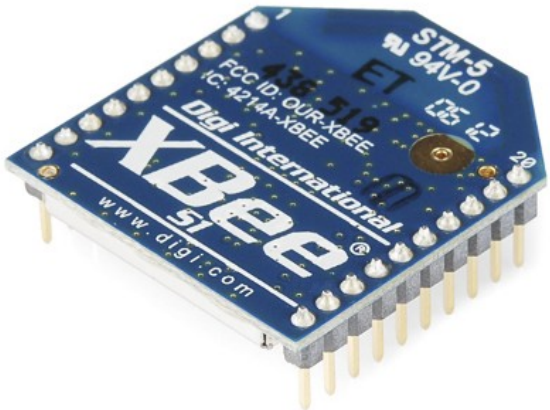
ZigBee est un protocole permettant la communication de petites radios (**voir annexe doc module xBee Pro**), à consommation réduite, basée sur la norme **IEEE 802.15.4** pour les réseaux à dimension personnelle (*Wireless Personal Area Networks* : WPAN).



Le protocole de communication a été établi entre membres du projet. Pour réceptionner et envoyer des trames on utilisera les services de la classe QserialPort (propre à Qt5). Pour ce qu'il s'agit de décoder les trames on utilisera les services de la classe QString de Qt.

La communication entre le microcontrôleur et le PCC se fait donc via les modules radiofréquences xBee. Ces modules sont branchés sur des ports USB via des *dongles :*

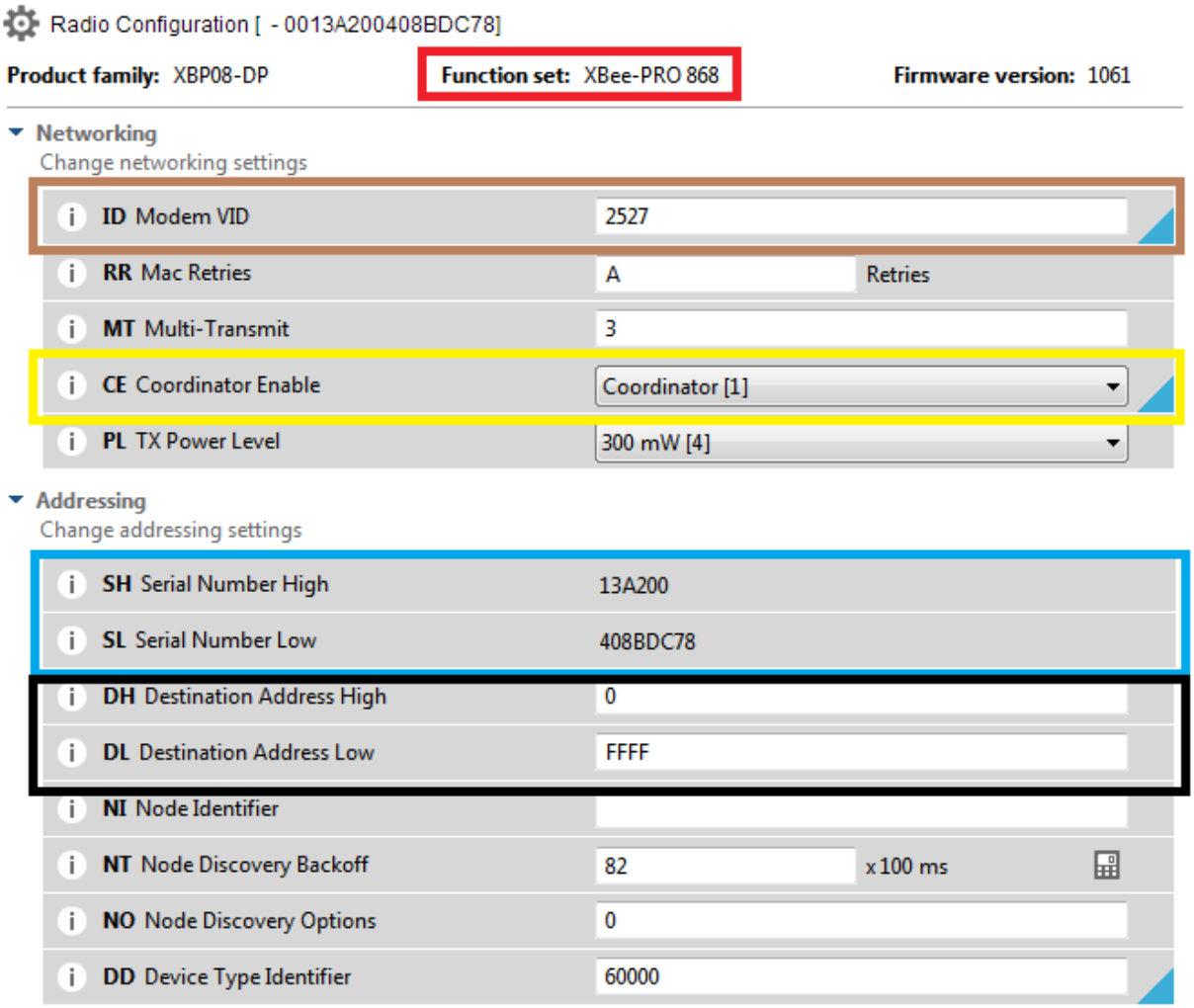




Configuration des modules xBee :

Travaillant sur le système d'exploitation Microsoft Windows, il a fallu installer le logiciel XCTU permettant de modifier la configuration des modules xBee. La configuration des modules sous Linux est fournie en annexe.

Ainsi en premier lieu, il a fallu configurer le module servant de coordonnateur :

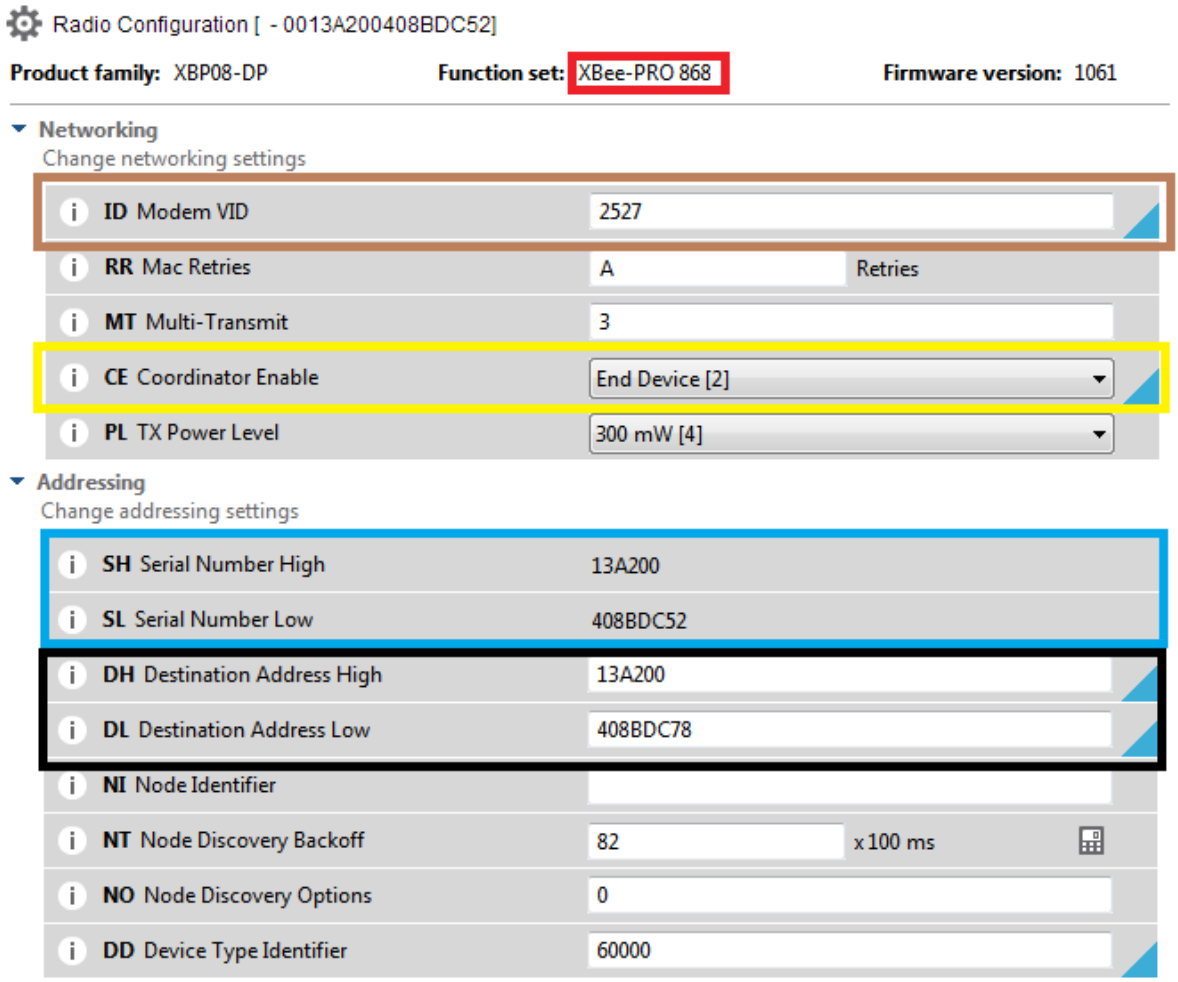


Ainsi le coordonnateur a une adresse de destination en broadcast (DL : FFFF) ce qui lui permet d'atteindre tout les modules xBee qui sont sur le réseau personnel 2527 (paramètre ID identique pour tous).

On retrouve ensuite l'**adresse MAC** de l'appareil formé par le SH et le SL ce qui nous donne :

Adresse MAC : 13A200408BDC78

Puis vient ensuite la configuration du module « End-Device » :

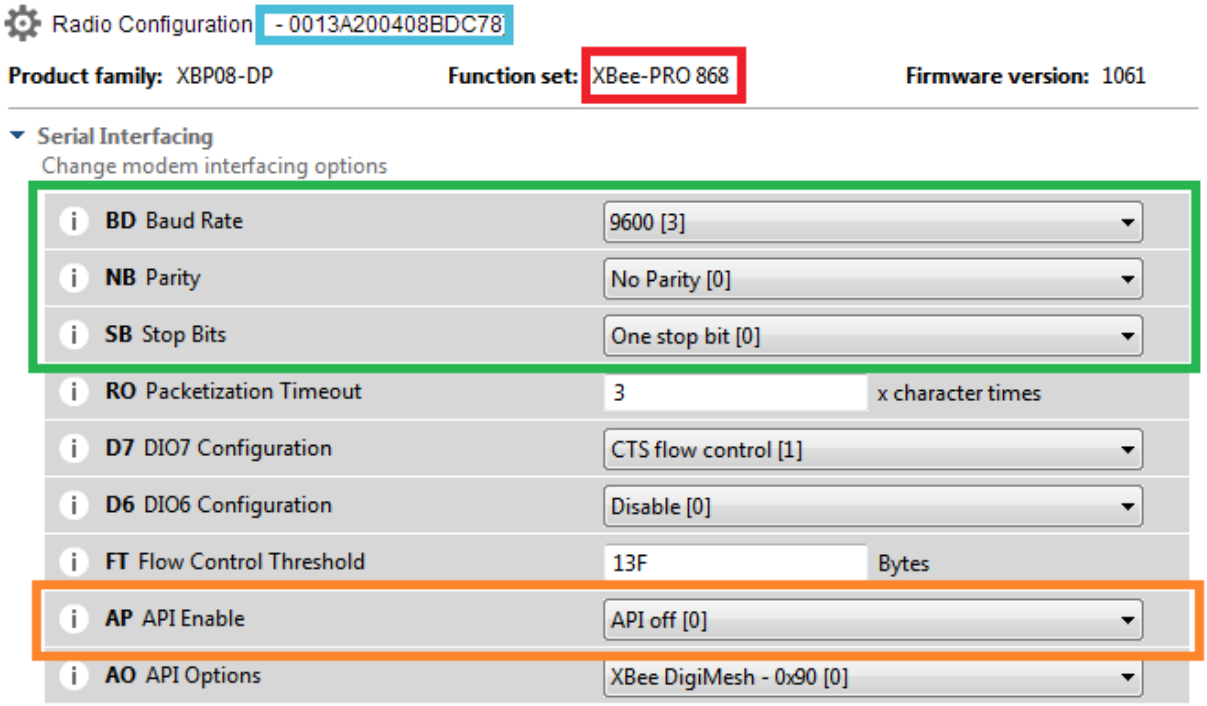


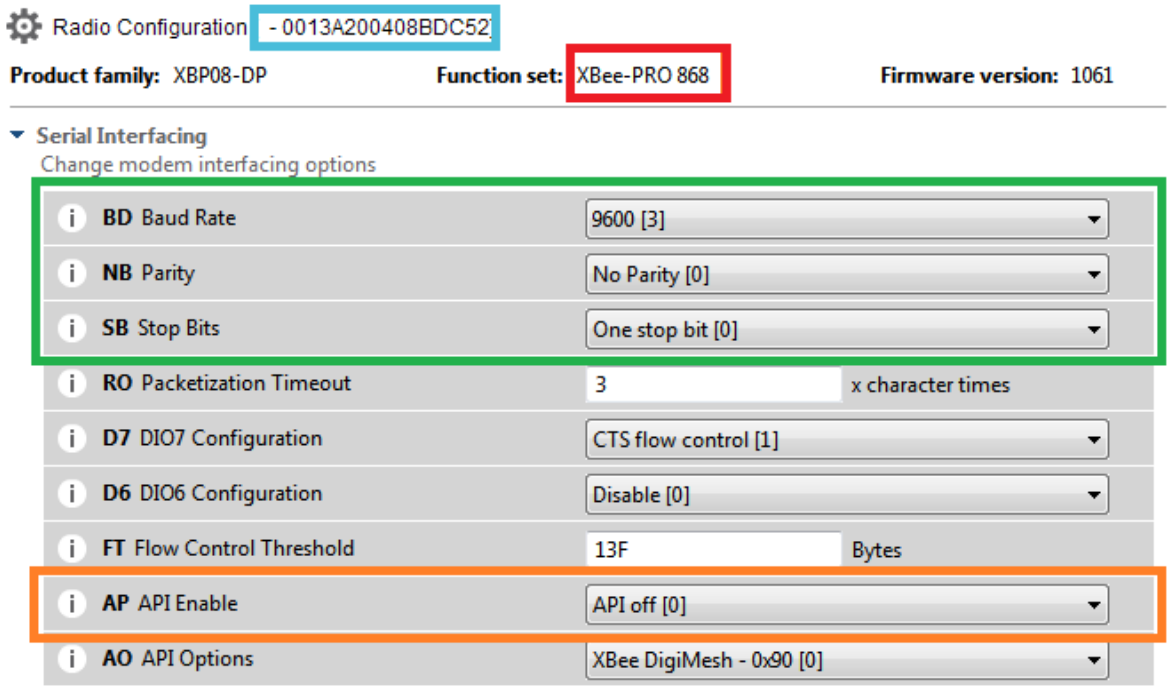
On retrouve bien évidemment le même ID, là où les deux modules diffèrent se situe à l'adresse de destination. En effet à l'inverse du module coordonnateur qui doit communiquer avec plusieurs modules, les modules « End-device » doivent communiquer uniquement avec le coordonnateur ainsi il suffit de mettre l'**adresse MAC** du coordonnateur (SH et SL) dans les champs d'adresses de destination (DH et DL) du « End-Device ».

L'**adresse MAC** de l'appareil est :

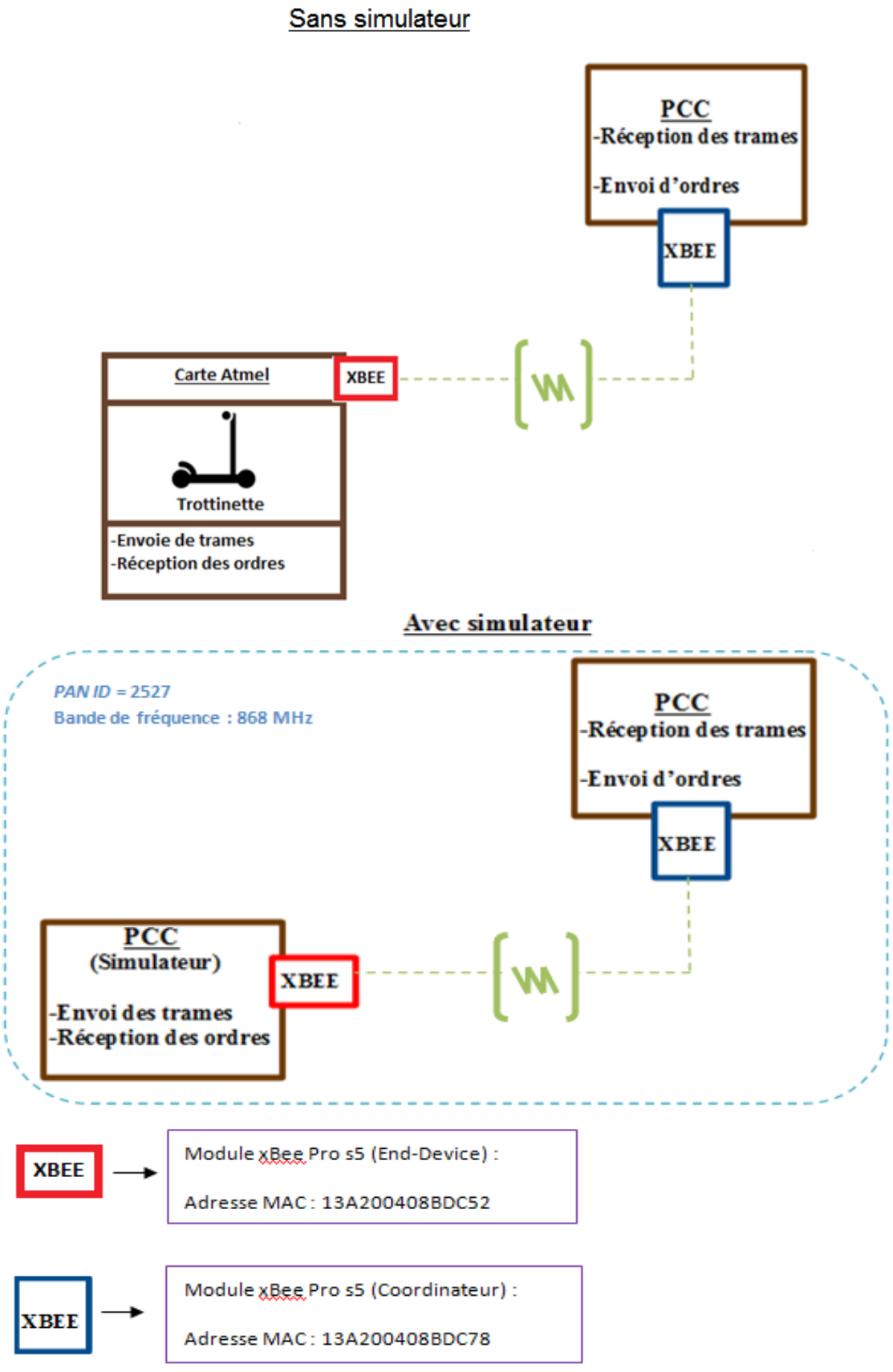
Adresse MAC : 13A200408BDC52

Puis on retrouve respectivement les paramètres par défaut des 2 modules xBee :



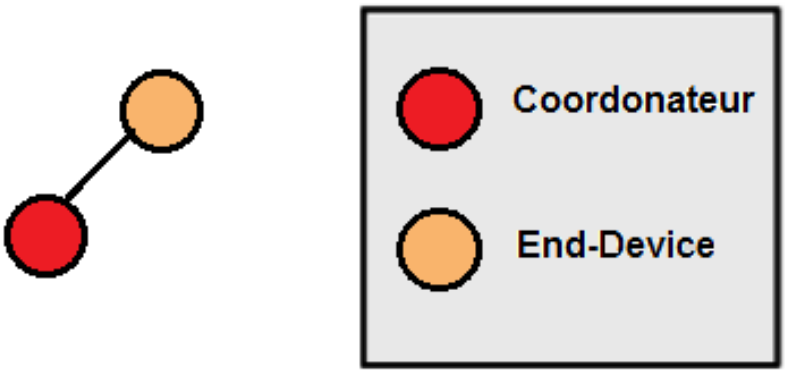


Représentation du système avec et sans simulateur



Le système avec simulateur permet de valider la partie Observer la réception des trames provenant des trottinettes. Le simulateur a donc été utilisé pendant ma phase de développement pour un travail autonome.

Ainsi il fut possible de tester la transmission des données entre un coordonnateur et un « End-Device », la **topologie physique** utilisée pour les tests est donc de type point à point (pair) :

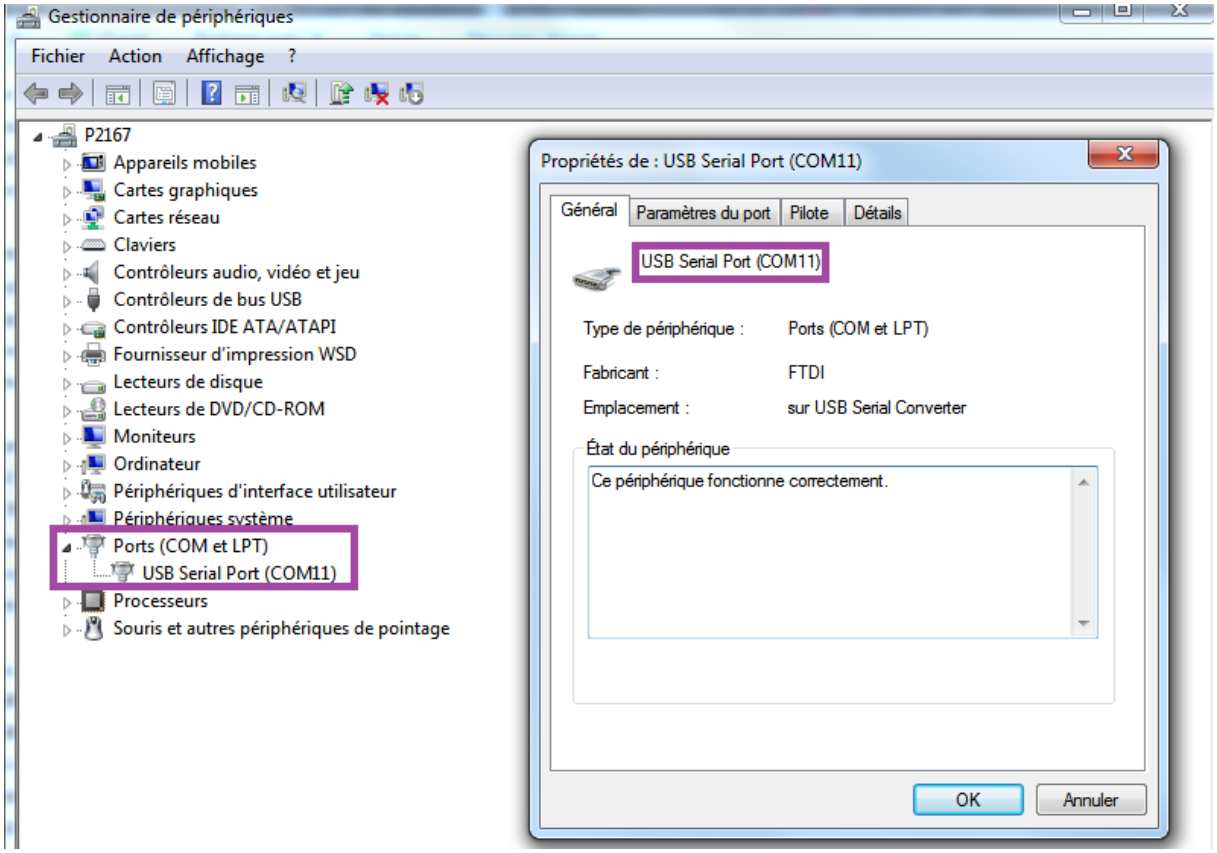


Les modules xBee doivent à la fois réceptionner des données et émettre des ordres, on dit que ce sont des *transceivers***.**

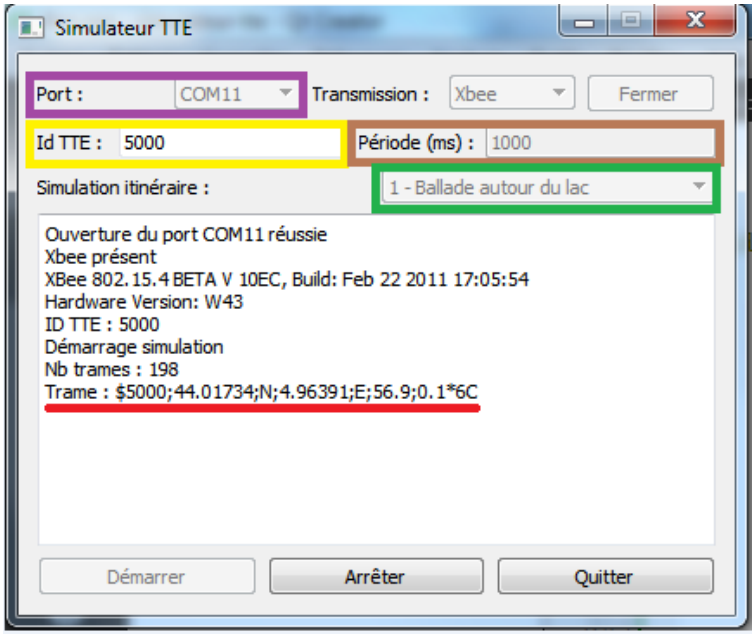
De plus la communication entre les deux modules xBee s'effectue en **Half-Duplex** (les informations sont envoyées dans les deux sens mais pas en même temps) :

Le simulateur est alors lancé sur une autre machine (PC simulateur). Il envoi par conséquent des trames au PCC, grâce à une communication Zigbee.

Le PC simulateur doit détecter le module xBee branché, il est donc nécessaire de vérifier la détection et le port qu'occupe le module via le gestionnaire de périphérique :



Puis vient l'ouverture du simulateur :



Il faut en premier lieu choisir le port qu'occupe le module. Puis il faut choisir l'identifiant d'une TTE, indiquer la période d'envoi de trames (ici 1 seconde) et enfin l'itinéraire à simuler.

Protocole de communication :

La trame TTE → PCC

Le protocole de communication TTE est indépendant du protocole de transmission (ici sans fil) car nous pourrions l'utiliser avec une liaison série RS232 par exemple.

Le xBee utilise le protocole **IEEE 802.15.4** pour la couche 2 du **modèle OSI**. Le modèle à couche sera :

Couche	Protocole
Application	TTE
Présentation	
Session	
Transport	
Réseau	
Liaison	802.15.4
Physique	802.15.4

La trame a un format fixe composée de 7 champs. Les champs sont séparés par des « ; ».

$ID_TTE;LATITUDE;ORIENTATION_LAT;LONGITUDE;ORIENTATION_LONG;VITESSE;CHARGE*CHECKSUM\r

Le $ est un délimiteur indiquant le début de la trame.

- Numéro d'identification de la trottinette.
- La latitude de la trottinette en degrés décimaux.
- Indicateur de latitude ('N'=nord ou 'S'=sud).
- La longitude de la trottinette en degrés décimaux.
- Indicateur de longitude ('E'=est ou 'W'=ouest).
- La vitesse de déplacement de la trottinette en km/h.
- La charge de la batterie en %.

Le symbole « * » est un délimiteur signifiant le début du checksum.

On retrouve ensuite Le « CHECKSUM » qui est une somme de contrôle codée sur 2 caractères (calculé en hexadécimal à partir d'un XOR).

Puis vient enfin le délimiteur «\r » qui marque la fin de la trame.Voici l'exemple d'une trame valide :

"$101;49,1645;N;1,2311;W;25,3;82*0E\r" Ainsi les trames sont envoyées périodiquement toutes les secondes par une trottinette (TTE).

La trame PCC → TTE

La trame a un format fixe composée de 2 champs. Les champs sont séparés par des « ; ».

$ID_TTE;ORDRE*CHECKSUM\r

c’est une trame apériodique qui contient un ordre (immobilisation, hors-zone) envoyé par le PCC vers une TTE.

Les champs de la trame ont la signification suivante :

- Le $ est un délimiteur indiquant le début de la trame.

Numéro d’identification de la trottinette.

Code de l’ordre envoyé par le PCC :

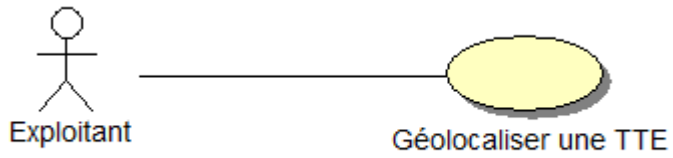
- «1 » : Immobilisation
- « 2 » : Hors-zone

- Le symbole « * » est un délimiteur signifiant le début du checksum.

- On retrouve ensuite Le « CHECKSUM » qui est une somme de contrôle codée sur 2 caractères (calculé en hexadécimal à partir d’un XOR).

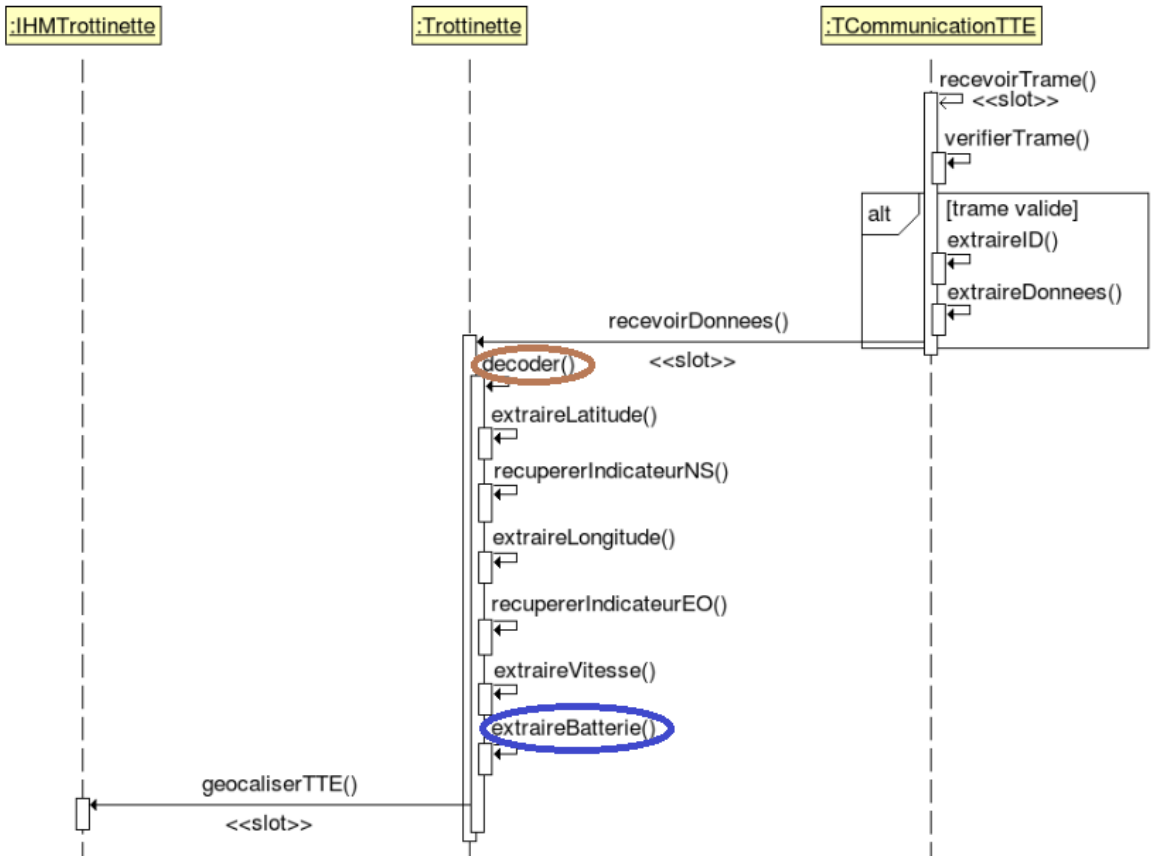
- Puis vient enfin le délimiteur «\r » qui marque la fin de la trame.

Cas d'utilisation « Géolocaliser une TTE »



l'exploitant doit pouvoir géolocaliser une TTE en temps réel (à partir des données de localisation reçues) afin de la visualiser sur une IHM.

Scénario « Géolocaliser une TTE »



Plusieurs étapes sont nécessaire au préalable pour pouvoir géolocaliser une TTE :

La classe **TcommunicationTTE** est le point de départ, elle a pour rôle de :

- Réceptionner les trames provenant des trottinettes.

- Vérifier le format de la trame via une panoplie de tests de validation.

-Lorsque la trame est valide, la classe extrait l'identifiant de la TTE d'une part et les données de l'autre.

Puis la classe **Trottinette** récupère les données de la trame, cette classe a pour rôle de :

- Via la méthode **decoder()**, elle sépare en deux catégories distinctes. D'une part les données de localisation (latitude, longitude, pôles) et de l'autre les paramètres de fonctionnement (vitesse, charge) de la trottinette.

Voici de quelle manière ce présente ces informations :

```
02/05/2017 17:51:23 -> Données de la trottinette n°101 : 20.4 km/h | 37.4%
02/05/2017 17:51:23 -> Localisation de la trottinette n°101 : 44.01382 | Pôle N | 4.9667 | Pôle E
02/05/2017 17:51:20 -> Données de la trottinette n°101 : 4.9 km/h | 84.8%
02/05/2017 17:51:20 -> Localisation de la trottinette n°101 : 44.0138 | Pôle N | 4.96692 | Pôle E
02/05/2017 17:51:19 -> Données de la trottinette n°101 : 44.2 km/h | 8.6%
02/05/2017 17:51:19 -> Localisation de la trottinette n°101 : 44.0138 | Pôle N | 4.96699 | Pôle E
02/05/2017 17:51:18 -> Données de la trottinette n°101 : 63.1 km/h | 8.9%
02/05/2017 17:51:18 -> Localisation de la trottinette n°101 : 44.01381 | Pôle N | 4.96707 | Pôle E
02/05/2017 17:51:16 -> Données de la trottinette n°101 : 100.5 km/h | 16.2%
02/05/2017 17:51:16 -> Localisation de la trottinette n°101 : 44.0138 | Pôle N | 4.96721 | Pôle E
02/05/2017 17:51:15 -> Données de la trottinette n°101 : 33.9 km/h | 17.3%
02/05/2017 17:51:15 -> Localisation de la trottinette n°101 : 44.01381 | Pôle N | 4.96725 | Pôle E
02/05/2017 17:51:14 -> Données de la trottinette n°101 : 93.4 km/h | 63.3%
02/05/2017 17:51:14 -> Localisation de la trottinette n°101 : 44.01383 | Pôle N | 4.9673 | Pôle E
```

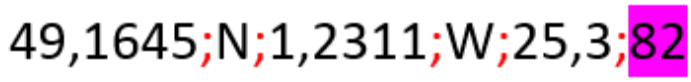
Enfin, la méthode **decoder()** émet ensuite un signal de localisation de la trottinette et un signal d'état de la trottinette (la vitesse et la charge de la trottinette).

```
void Trottinette::decoder(QString &donnees)

{
    QString latitude = extraireLatitude(donnees);
    QString NS = recupererIndicateurNS(donnees);
    QString longitude = extraireLongitude(donnees);
    QString EO = recupererIndicateurEO(donnees);
    QString vitesse = extraireVitesse(donnees);
    QString batterie = extraireBatterie(donnees);

    localisation = latitude + " | Pôle " + NS + " | " + longitude + " | Pôle " + EO;

    parametre_fonctionnement = vitesse + " km/h | " + batterie + "% ";

    emit nouvelleLocalisation(id, localisation);
    emit nouvelleEtat(id,parametre_fonctionnement);
    emit nouvelleLocalisation(QString::number(id), longitude, latitude);

}
```

La méthode extraireBatterie() permet d'extraire la charge de la trottinette. Elle découpe la trame avec le délimiteur « ; » et récupère le sixième champs.

```
QString Trottinette::extraireBatterie(QString &donnees)
{
    QString batterie;
    batterie = donnees.section(';',5,5);
    qDebug() << Q_FUNC_INFO << "[" << __LINE__ << "]" << batterie;
    return batterie;
}
```

Exemple de trame :



La classe **IHMTrottinette** reçoit le **signal** de localisation grâce à la méthode(**slot**) geolocaliserTTE().

Le signal de localisation a été connecté au slot geolocaliserTTE() à partir du constructeur de la classe **Supervision** :

```
Supervision::Supervision(IHMTrottinette *parent) : QObject(parent)
{
    communicationTTE = new TCommunicationTTE(this);
    trottinette = new Trottinette(this);

    connect(trottinette,
    SIGNAL(nouvelleLocalisation(QString,Qstring,Qstring)),parent,
    SLOT(geocaliserTTE(QString,QString,QString)));
}
```

Glossaire

Carte Atmel : Une carte Atmel est un microcontrôleur qui va traiter les informations provenant des capteurs. Un microcontrôleur est un composant électronique, que l’on peut qualifier de condensé de tout ce que contient un ordinateur. Un microcontrôleur intègre entre autres un processeur, une mémoire ROM pour stocker le programme, une mémoire RAM pour stocker les données temporaires et un certain nombre d’entrées/sorties (analogiques et numériques), dont certaines dédiées à des usages particuliers.

Noeuds : Un nœud, en informatique, est un composant qui fait partie d’un réseau. Dans un réseau mailles ZigBee il y a 3 types de nœuds : coordinateur, routeur, et “End Device”.

Topologie physique : Une topologie physique est en fait la structure physique de votre réseau, c'est donc la forme du réseau. Il existe plusieurs topologies physiques : le bus, l'étoile (la plus utilisée), l'anneau, etc.

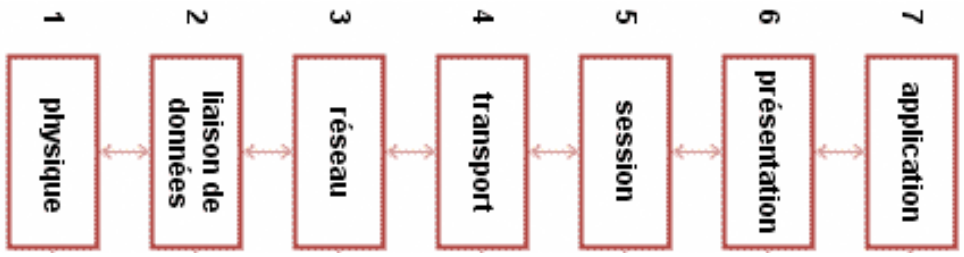
Réseau en étoile : Dans une topologie de réseau en étoile, les équipements du réseau sont reliés à un système matériel central (routeur, commutateur, concentrateur, ...). Celui-ci a pour rôle d'assurer la communication entre les différents équipements du réseau.

Liaison point à point (pair) : Une liaison point à point est une liaison entre deux hôtes uniquement.

Norme IEEE 802.15.4 : Le 802.15.4 est un protocole de communication défini par l’IEEE (Institute of Electrical and Electronics Engineers). Il est destiné aux réseaux sans fil de la famille des LR WPAN (Low Rate Wireless Personal Area Network) du fait de leur faible consommation, de leur faible portée et du faible débit des dispositifs utilisant ce protocole.

Adresse MAC : Chaque matériel réseau reçoit une adresse unique de 12 chiffres en hexadécimal (48 bits), c'est l'adresse Mac appelée aussi physique.Les 6 premiers chiffres désignent le fabricant. L'adresse complète est activée en usine, rendant sa modification impossible sauf dans des cas spécifiques. Cette adresse fait partie de la couche 2 du modèle OSI.

Modèle OSI : Le modèle OSI est une norme qui préconise comment les ordinateurs devraient communiquer entre eux. Le modèle OSI est un modèle en couche, qui ont chacun un rôle défini.



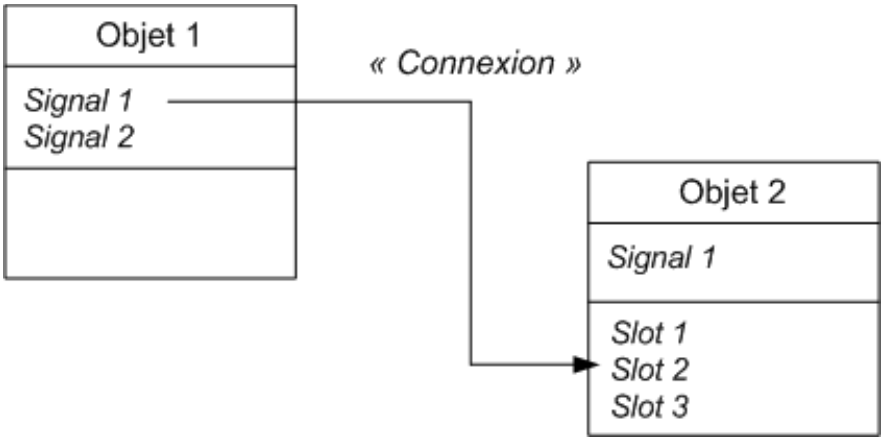
Ainsi le modèle OSI a 7 couches.

Annexe: QT → Signal/Slot

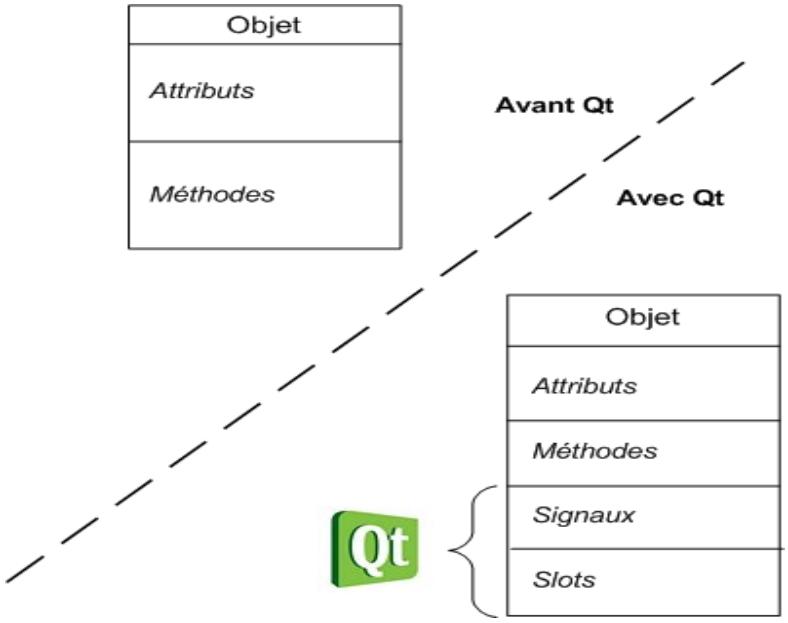
Dans Qt, on parle de signaux et de slots, voici une définition :

- **Un signal** : c'est un message envoyé par un widget lorsqu'un évènement se produit.
- **Un slot** : c'est la fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le signal appelle le slot. Concrètement, un slot est une méthode d'une classe.

Avec Qt, on dit que l'on connecte des signaux et des slots entre eux.



Les signaux et les slots peuvent être considérés comme un nouveau type d'élément au sein des classes, en plus des attributs et méthodes.



Voici un exemple présent dans mon projet Trottinette :

La classe Trottinette émet un signal :

```
signals:
    void nouvelleEtat(int id, QString parametre_fonctionnement);
```

Puis on crée le slot dans la classe IHMTrottinette :

```
public slots:
    void journaliserEtat(int id, QString parametre_fonctionnement);
```

Puis on connecte le signal au slot dans le constructeur de la classe Supervision :

```
Supervision::Supervision(IHMTrottinette *parent) : QObject(parent)
{
    communicationTTE = new TCommunicationTTE(this);
    trottinette = new Trottinette(this);

    connect(trottinette,
    SIGNAL(nouvelleEtat(int,QString)), parent,
    SLOT(journaliserEtat(int,QString)));
}
```

Annexe : Documentation module xBee Pro

1. XBee®/XBee-PRO® RF Modules

The XBee and XBee-PRO RF Modules were engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks. The modules require minimal power and provide reliable delivery of data between devices.

The modules operate within the ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other.

Key Features of the XBee/XBee-PRO Modules

Long Range Data Integrity

XBee

- Indoor/Urban: up to 100' (30 m)
- Outdoor line-of-sight: up to 300' (90 m)
- Transmit Power: 1 mW (0 dBm)
- Receiver Sensitivity: -92 dBm

XBee-PRO

- Indoor/Urban: up to 300' (90 m), 200' (60 m) for International variant
- Outdoor line-of-sight: up to 1 mile (1600 m), 2500' (750 m) for International variant
- Transmit Power: 63mW (18dBm), 10mW (10dBm) for International variant
- Receiver Sensitivity: -100 dBm

RF Data Rate: 250,000 bps

Advanced Networking & Security

Retries and Acknowledgements

DSSS (Direct Sequence Spread Spectrum)

Each direct sequence channels has over 65,000 unique network addresses available

Source/Destination Addressing

Unicast & Broadcast Communications

Point-to-point, point-to-multipoint and peer-to-peer topologies supported

Coordinator/End Device operations

Transparent and API Operations

128-bit Encryption

Low Power

XBee

- TX Peak Current: 45 mA (@3.3 V)
- RX Current: 50 mA (@3.3 V)
- Power-down Current: < 10 µA

XBee-PRO

- TX Peak Current: 250mA (150mA for international variant)
- TX Peak Current (RPSMA module only): 340mA (180mA for international variant)
- RX Current: 55 mA (@3.3 V)
- Power-down Current: < 10 µA

ADC and I/O line support

Analog-to-digital conversion, Digital I/O

I/O Line Passing

Easy-to-Use

No configuration necessary for out-of box RF communications

Free X-CTU Software (Testing and configuration software)

AT and API Command Modes for configuring module parameters

Extensive command set

Small form factor

Annexe:Configuration des modules xBee avec Cutecom

La configuration des modules est différent selon le système d’exploitation. Il est possible de configurer les modules sous l’environnement Linux (Ubuntu) donc la configuration se fera avec ***cutecom*** (un émulateur de terminal série).

Cutecom est une interface graphique fournissant un terminal qui permet de dialoguer avec le matériel physique par une liaison série.

Tout d’abord il faut installer la librairie xBee du module choisie dans mon projet (ici xBee Pro S5).

Prise en charge :

-Avant de brancher le module xBee

```
$ lsusb > lsusb-avant.txt
$ lsmod > lsmod-avant.txt
$ dmesg > dmesg-avant.txt
```

-

Puis on branche le module

```
$ lsusb > lsusb-apres.txt
$ lsmod > lsmod-apres.txt
$ dmesg > dmesg-apres.txt
```

Puis on analyse le module :

```
$ diff lsusb-avant.txt lsusb-apres.txt
Bus 003 Device 091: ID 0403:6015 Future Technology Devices International, Ltd

$ diff dmesg-avant.txt dmesg-apres.txt
usb 3-9.4.2: new full-speed USB device number 91 using xhci_hcd
usb 3-9.4.2: New USB device found, idVendor=0403, idProduct=6015
usb 3-9.4.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 3-9.4.2: Product: FT231X USB UART
usb 3-9.4.2: Manufacturer: FTDI
usb 3-9.4.2: SerialNumber: DA011HYD
ftdi_sio 3-9.4.2:1.0: FTDI USB Serial Device converter detected
usb 3-9.4.2: Detected FT-X
usb 3-9.4.2: Number of endpoints 2
usb 3-9.4.2: Endpoint 1 MaxPacketSize 64
usb 3-9.4.2: Endpoint 2 MaxPacketSize 64
usb 3-9.4.2: Setting MaxPacketSize 64
usb 3-9.4.2: FTDI USB Serial Device converter now attached to ttyUSB0

$ diff lsmod-avant.txt lsmod-apres.txt
ftdi_sio               47922  0
usbserial              37161  1 ftdi_sio

$ modinfo ftdi_sio
...

$ ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 sept. 18 16:24 /dev/ttyUSB0
```

Le module Xbee USB est identifié par les numéros :

- idVendor : `0x0403`
- idProduct : `0x6015`

Puis on utilise les commandes AT, ces commandes sont utilisées pour communiquer avec les périphériques (ici « Xbee Pro S5 ») et le configurer.

Par défaut le module Xbee a les paramètres suivant : 9600 bauds ; 8 bits ; 1 bit de stop et la parité sur None, donc pas de parité.

La liste des commandes AT que nous allons utiliser :

Commande	Valeur	Explication
ATID	2527	Adresse du réseau
ATDH	Coordonnateur : 0 (adresse sur 16 bits) End-Device : 13A200	partie haute de l'adresse
ATDL	Coordonnateur : FFFF (broadcast) End-Device : 408BDC78	adresse de destinataire
ATSH	13A200	numéro de série SH (partie haute)
ATSL	Coordonnateur : 408BDC78 End-Device : 408BDC52	numéro de série SL (partie basse)

Pour utiliser ces commandes, il faut passer en MODE commande. Pour cela il faut saisir une séquence de « +++ » sans délimiteur de fin (*No line End*).

Ensuite on peut donc utiliser les commandes. Il faudra cependant les envoyer avec le délimiteur « retour chariot » (*CR line End*).

Remarque : Le mode commande ne dure que 10 secondes sans rien envoyer.

Pour visualiser les informations de configuration, il faut envoyer les commandes AT suivantes :

Instructions	*Visualisation (sur cutecom)*
+++ (No line End)	OK
ATID (CR line End)	2527
ATSH(CR line End)	*13A200*

Pour modifier une valeur, il faut préciser la commande AT avec un égal (=) et la valeur choisie.

Exemple : +++ (No line End)

ATID=2526 (CR line End)

*L'adresse du PAN ID est maintenant **2526**.*

Il existe 3 modes de communication : « TRANSPARENT », « COMMANDE » et « API » :

- *Le mode Transparent est le mode par défaut, il est aussi plus simple d'utilisation. Car on émet et reçoit directement sur le port série.*
- Le mode Commande sert uniquement à configurer les modules par les commandes AT.
- Le mode API permet de programmer la communication au niveau trame du protocole 802.15.4. Ce mode n'est pas utile pour notre projet.
- On configure donc nos modules en mode 0 (ATAP=0) soit en « mode transparent ».