Kart électrique

version 1.0

Revue Finale

Étudiant : GARCIA Tom (IR)

Table des matières

Le projet Kart

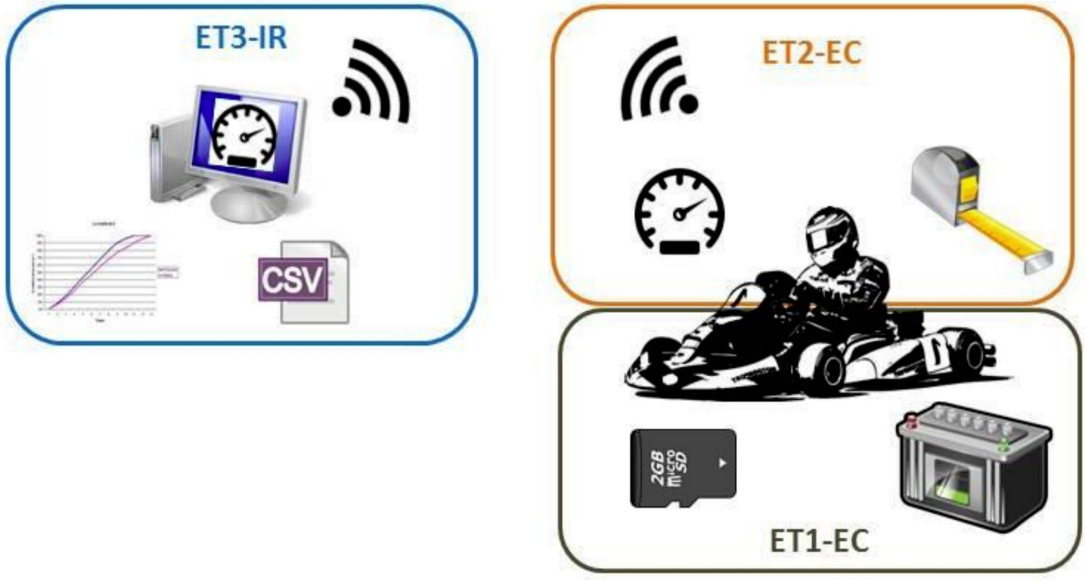
Présentation

Il s'agit de **réaliser la conception complète de la chaîne de télémétrie des essais en course d'un kart électrique**.

L'opérateur doit pouvoir **démarrer un essai** (une campagne d'acquisition de données télémétriques) d'un kart en indiquant son **code opérateur** et en précisant l'**identifiant du kart** et son modèle. L'essai doit être horodaté (date et heure de début et de fin). Une détection du kart à superviser doit être réalisée avant de démarrer la collecte des données télémétriques.

L'opérateur doit pouvoir ensuite **suivre l'affichage en temps réel des données télémétriques** collectées. Lorsqu'il mettra fin à l'acquisition, il pourra alors **exploiter les données sous forme de graphiques** et éventuellement les **sauvegarder dans le format CSV**. Ceci lui permettra soit de les recharger dans le logiciel soit de les importer dans un tableur par exemple.

Les parties embarquées sur le kart seront développées sur un micro-contrôleur par des étudiants EC. Ces données seront captées, mises en forme et transmises à un ordinateur distant. Le support de transmission sans fil des données sera défini par ces étudiants.



Le projet Kart

Les **données de télémétrie** seront :

- la tension (en Volts) de la batterie ;
- le courant (en Ampères) de la batterie ;
- le pourcentage de charge de la batterie ;
- la température moteur (en degrés Celsius) ;
- la vitesse (en km/h)

L'application logicielle devra assurer :

- de superviser l'essai d'un kart
- d'acquérir les données de télémétrie par transmission sans fil
- d'afficher en « temps réel » les données reçues périodiquement
- d'exploiter les données de télémétrie sous forme de graphes
- d'exporter les données de télémétrie dans le format CSV

Contraintes économiques et techniques

Le système sera maquetté et conservé par l'établissement.

Le budget matériel sera donc à la charge de l'établissement.

Le prix des batteries du kart n'est donc pas pris en charge car supérieur à 1000 euros d'autant que le Kart ne peut pas rouler dans l'espace qui nous est attribué. Un partenariat avec la section BTS électrotechnique est possible afin de disposer d'un jeu de batteries.

L'environnement au campus n'est pas adéquat pour permettre au kart d'effectuer un essai.

Exigences

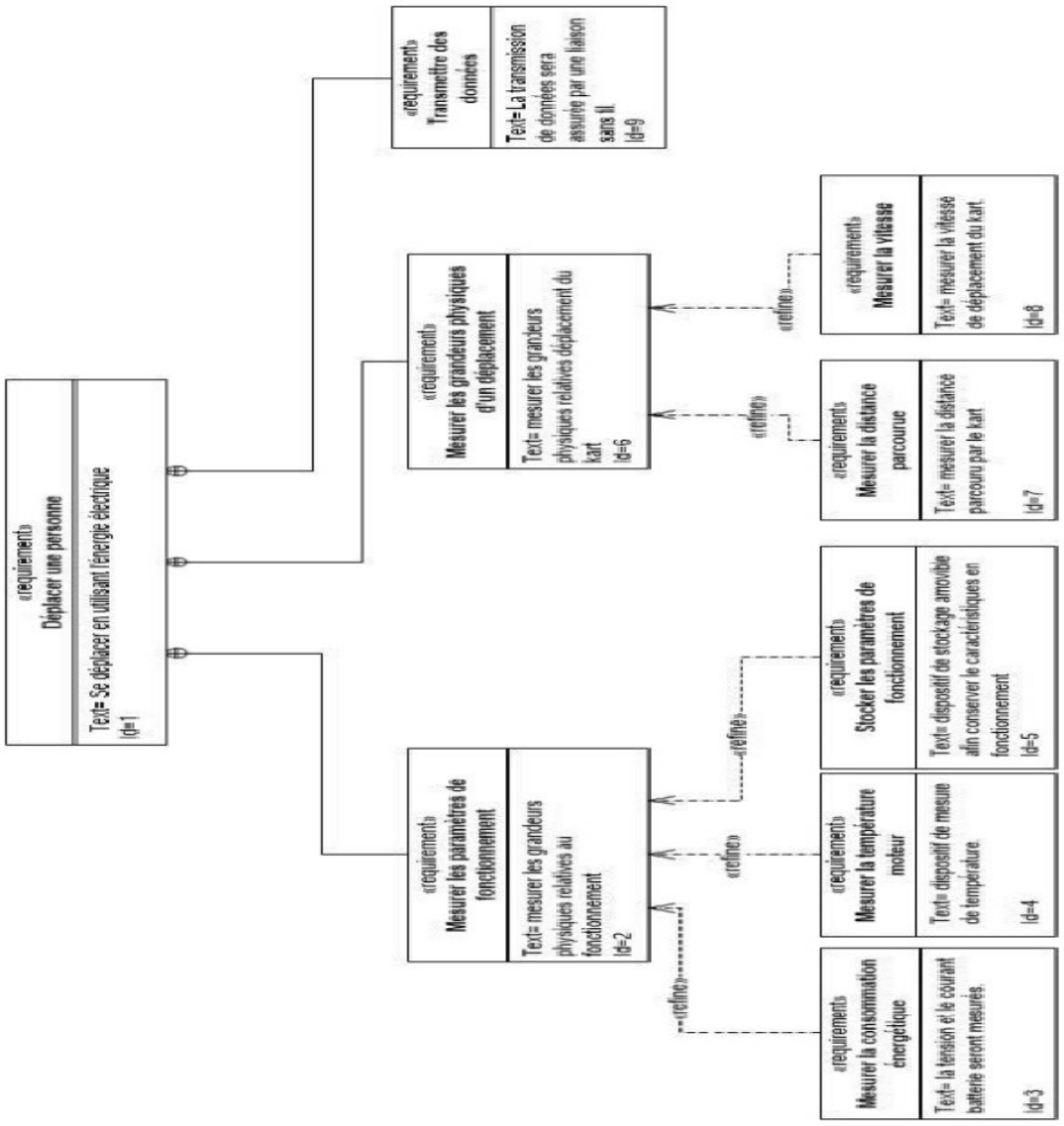


Diagramme des exigences

Les exigences qualités du développement pour ma partie (IR) sont :

- Le développement se fera de manière itérative et incrémentale (méthodologie UP) ;
- La modélisation du système devra être réalisée avec le langage de modélisation UML 2 ;
- L'architecture du logiciel sera «orientée objet» et multitâche si besoin ;
- Le langage de programmation à utiliser est le C++ ;
- L'IHM doit être développée avec l'API Qt de Qt Development Frameworks, filiale de Digia ;
- L'implémentation des structures de données doit privilégier les structures de données de la STL ou équivalent dans Qt ;
- Le codage doit respecter le standard de codage C++ en cours dans la section ;
- La chaîne de production des exécutables doit être réalisée avec un gestionnaire de type make/qmake et le compilateur GNU g++ ;
- Le gestionnaire de gestion de versions utilisé sera subversion ;
- La documentation du code sera générée à partir de doxygen ;
- La suite bureautique libre LibreOffice ou OpenOffice sera utilisée pour tous les documents "papier" et les diaporamas ;
- Les données de l'application seront exclusivement gérées par des fichiers XML.

Ressource	**Version**
OS	GNU Linux (Ubuntu 12.04.5 LTS)
EDI	Qt Creator 2.4.1, ATMEL Studio V7,
Compilateur	GNU g++/gcc version 4.6.3
Débuguer	GNU gdb 7.4
Fabrication	QMake 2.01a et GNU make 3.81
API GUI	Qt 4.8.1et Qwt version 5 ou 6
UML	bouml 4.23
Tests	CppUnit 1.12.1
Versions	subversion (client svn 1.6.17)
Documentation	Doxygen version 1.7.6.1 et pandoc 1.9.1.1
Gantt	Planner (version 0.14.5) ou gantter

Étude et réalisation

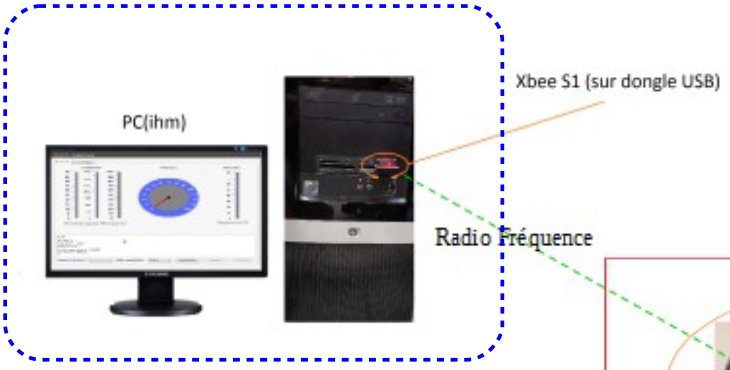
Objectifs

L'étudiant **Garcia Tom** doit concevoir l'application qui permet de **superviser les essais d'un kart**. Il doit mettre en œuvre un type de communication (défini par les étudiants EC) permettant au poste de supervision de recevoir les données des karts.

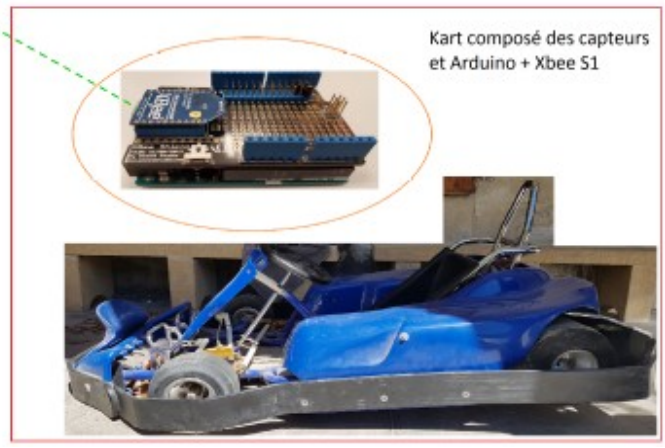
Je devais donc :

- exploiter des données de télémétrie
- collecter des données de l'essai
- visualiser les données de télémétrie en temps réel
- visualiser les graphiques
- exporter les données de l'essai

Configuration d'exploitation



SUPERVISION
Tom Garcia (étudiant IR)



KART
Étudiants EC

Déploiement

Pour la partie SUPERVISION, le système est construit autour d'un ordinateur de type PC sur un système d'exploitation GNU/Linux (Ubuntu 12.04) afin d'exécuter l'application **IHMKart**. Celle-ci utilise une bibliothèque **qwt** (*Qt Widgets for Technical Applications*) pour l'affichage des *widgets* techniques ainsi que les courbes.

La communication avec le Kart est possible grâce à 2 modules Xbee. Côté PC, le module Xbee se connectera par un *dongle* USB.

Le calculateur embarqué est un Arduino.

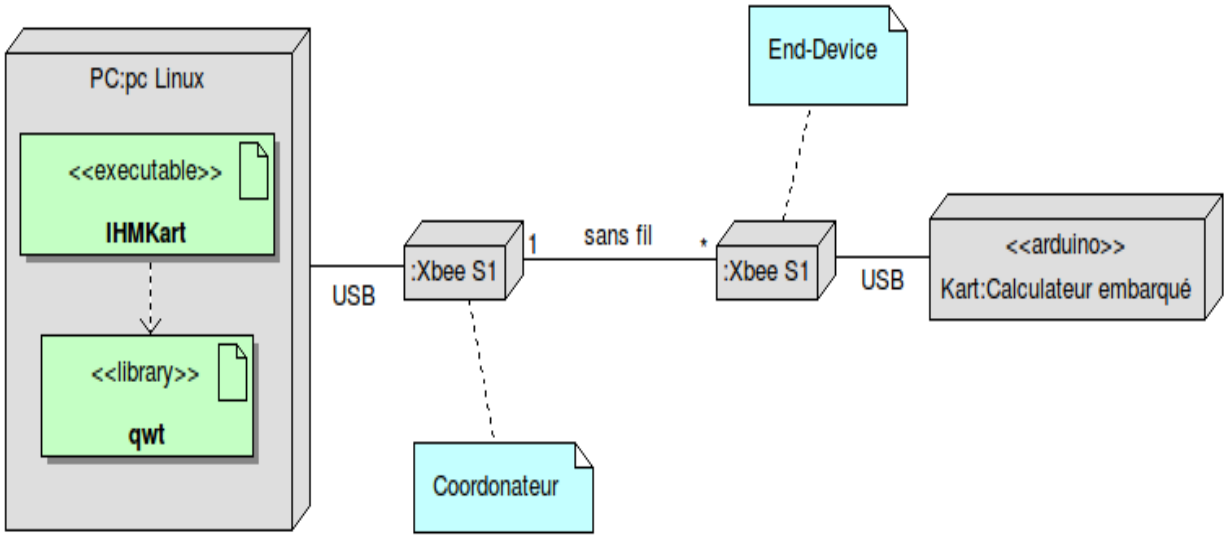


Diagramme de déploiement

L'application **IHMKart** gérera ses paramètres de configuration dans des fichiers XML :

- `operateurs.xml` : contiendra la liste et les codes des opérateurs habilités à faire des essais
- `karts.xml` : contiendra la liste et les paramètres des karts reconnus par l'application

Les cas d'utilisation

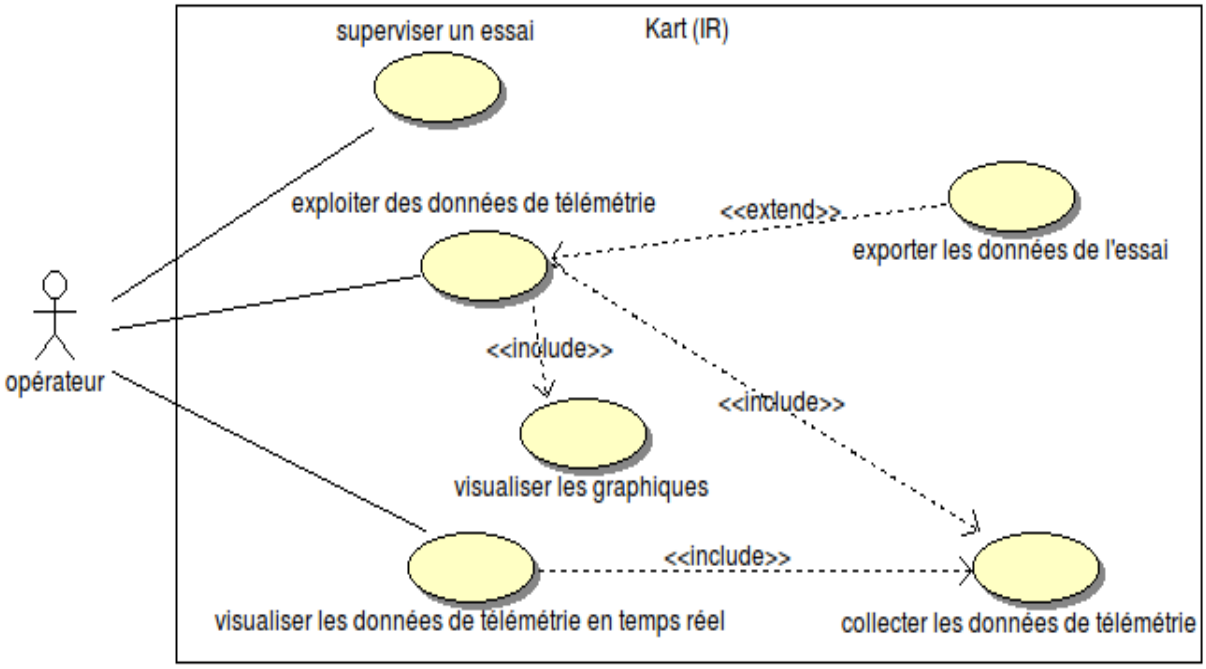
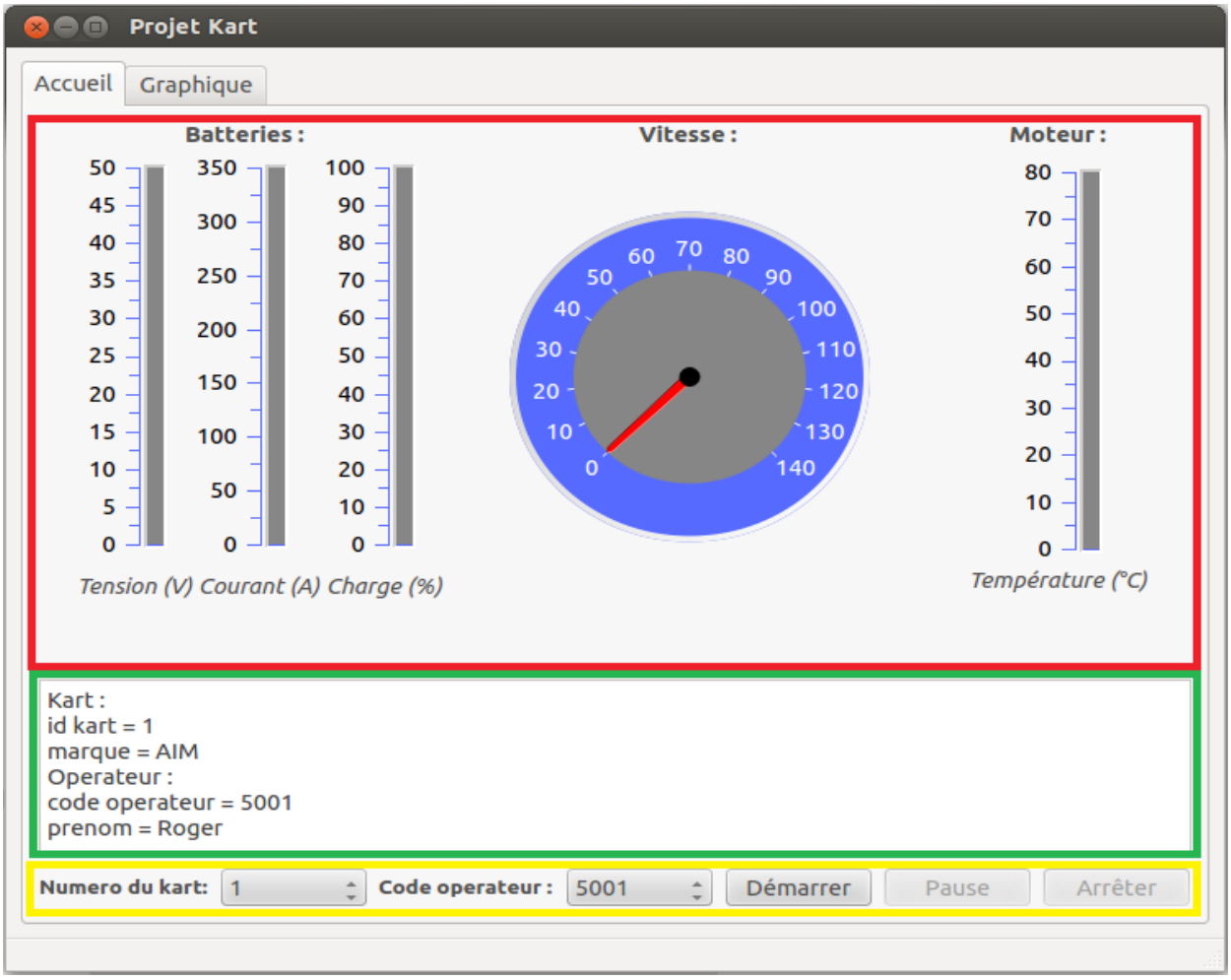


Diagramme des cas d'utilisation

L'application est composée de deux onglets : **Accueil** et **Graphique**.

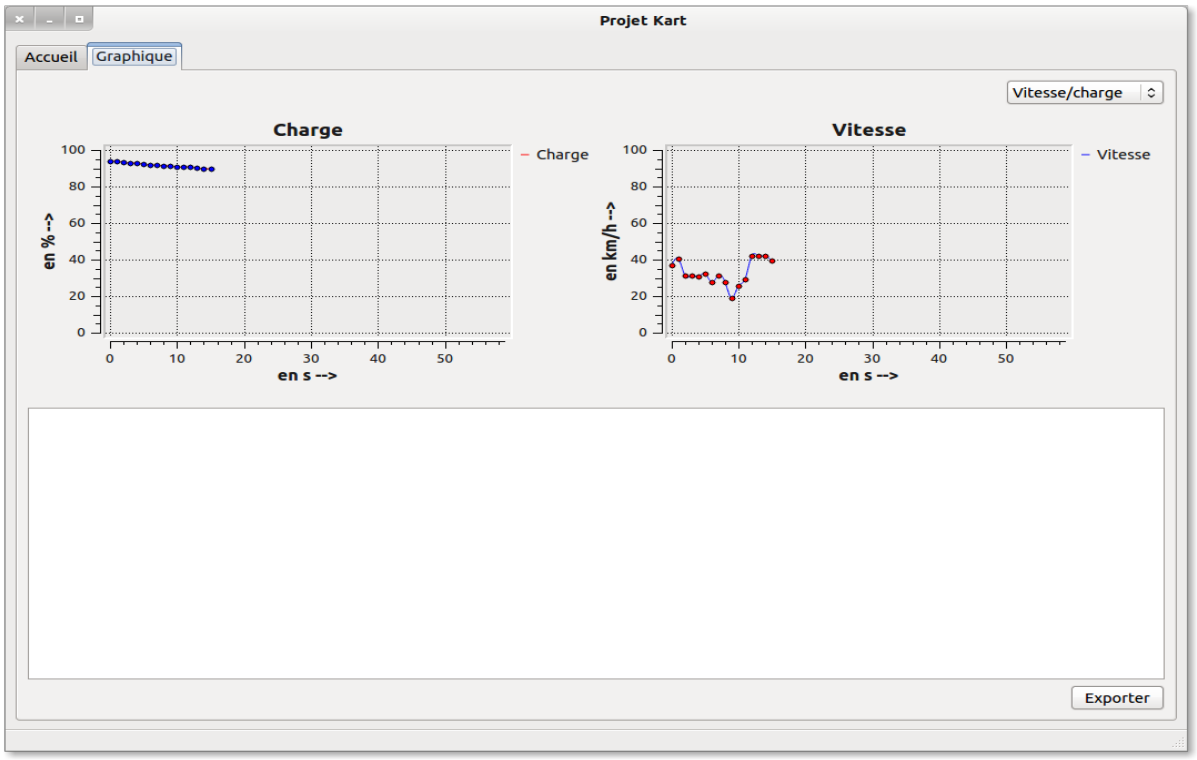
Dans l'onglet **Accueil**, l'ihm est décomposée en trois parties :

- Superviser un essai
- Les données des fichier XML
- Visualiser les données de télémétrie en temps réel



L'onglet Accueil

Et l'onglet **Graphique** :



Mise en œuvre du matériel

Présentation

Les modules Xbee permettent la communication de petites radios, à consommation réduite pour les réseaux à dimension personnel. Plus précisément des WPAN (*Wireless Personnel Area Networks*).

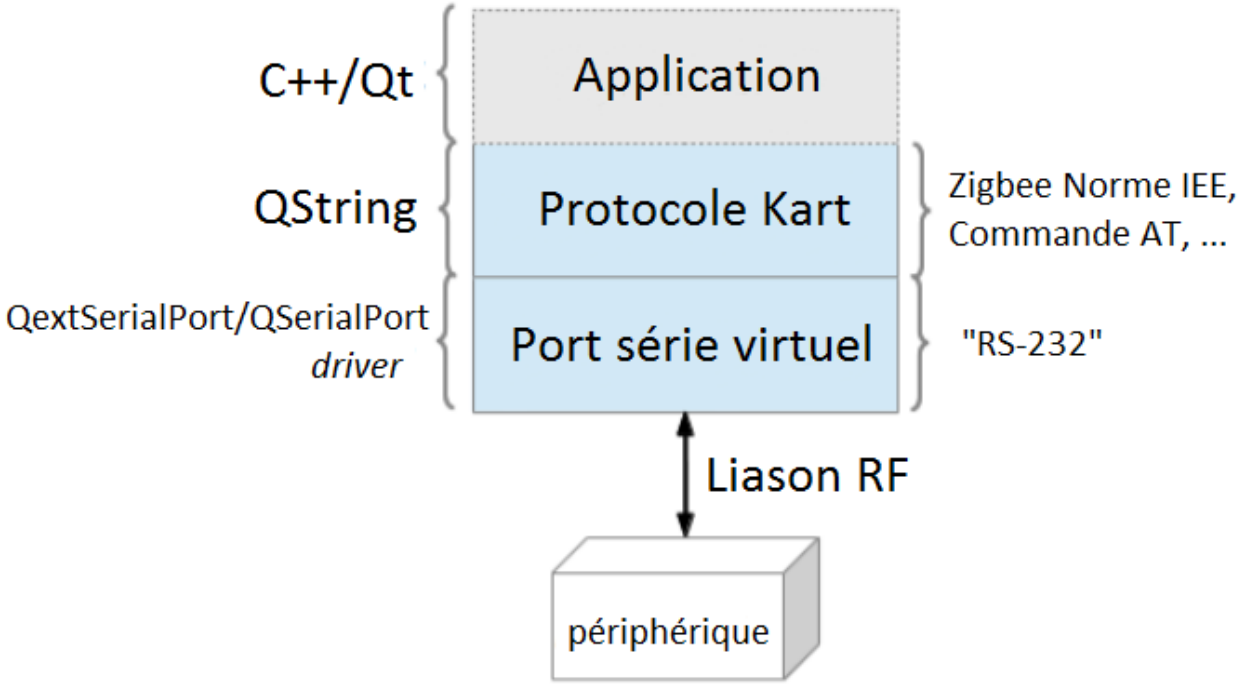
Ces modules suivent un protocole Zigbee basé sur la norme IEE 802.15.4.

Côté PC sous GNU/Linux, le module Xbee sera exploité comme un port série virtuel. Un protocole Kart (sous forme

Les modules Xbee S1 (sur un dongle)

de chaînes de caractères) est défini pour permettre l'échange de données entre le PC et le calculateur embarqué du kart.

Le module Xbee se configure à l'aide de commandes AT.



Configuration des modules Xbee

La configuration des modules est différent selon le système d'exploitation. Pour ce projet nous travaillons avec un environnement Linux (Ubuntu) donc la configuration se fera donc avec ***cutecom*** (un émulateur de terminal série). La configuration des modules sous Windows est fournie en annexe.

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 pilote de périphérique xbee du module choisi dans ce projet un 'Xbee S1'.

Avant de brancher le module xbee, on sauvegarde la configuration actuelle :

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

On branche ensuite le module xbee et on sauvegarde les modifications de la configuration système :

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

Maintenant on analyse la prise en charge du 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`

Le pilote de périphérique est le module `ftdi_sio` et il est accessible par le fichier de périphérique `/dev/ttyUSB0`.

On utilise les commandes AT, ces commandes sont utilisées pour communiquer avec les périphériques (ici « Xbee S1 ») 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	2525	Adresse du réseau
ATCH	C	Permet de changer le channel
ATMY	Simulateur :5000 PC (ihm) : 5001	Adresse du module dans le réseau
ATAP	0	Permet de changer le mode de fonctionnement
ATDH	0	pour utiliser des adresse sur 16 bits
ATDL	Simulateur : 5001 PC (ihm): 5000 ou Broadcast: FFFF	Adresse du destinataire dans le réseau
ATWR	-	Écrit la nouvelle configuration dans la mémoire flash du module
ATCN	-	Sort du mode configuration

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 charriot » (*CR line End*).

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

Pour communiquer les deux modules Xbee doivent avoir le même **PAN ID** (commande « ATID ») et le même **canal** (commande « ATCH »).

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)	2525
ATCH(CR line End)	C

On remarque donc que nous avons un PAN ID = 2525 et canal = C (Par défaut le canal est C).

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 faut savoir qu'il existe 3 modes de communication : « TRANSPARENT », « COMMANDE » et « API » :

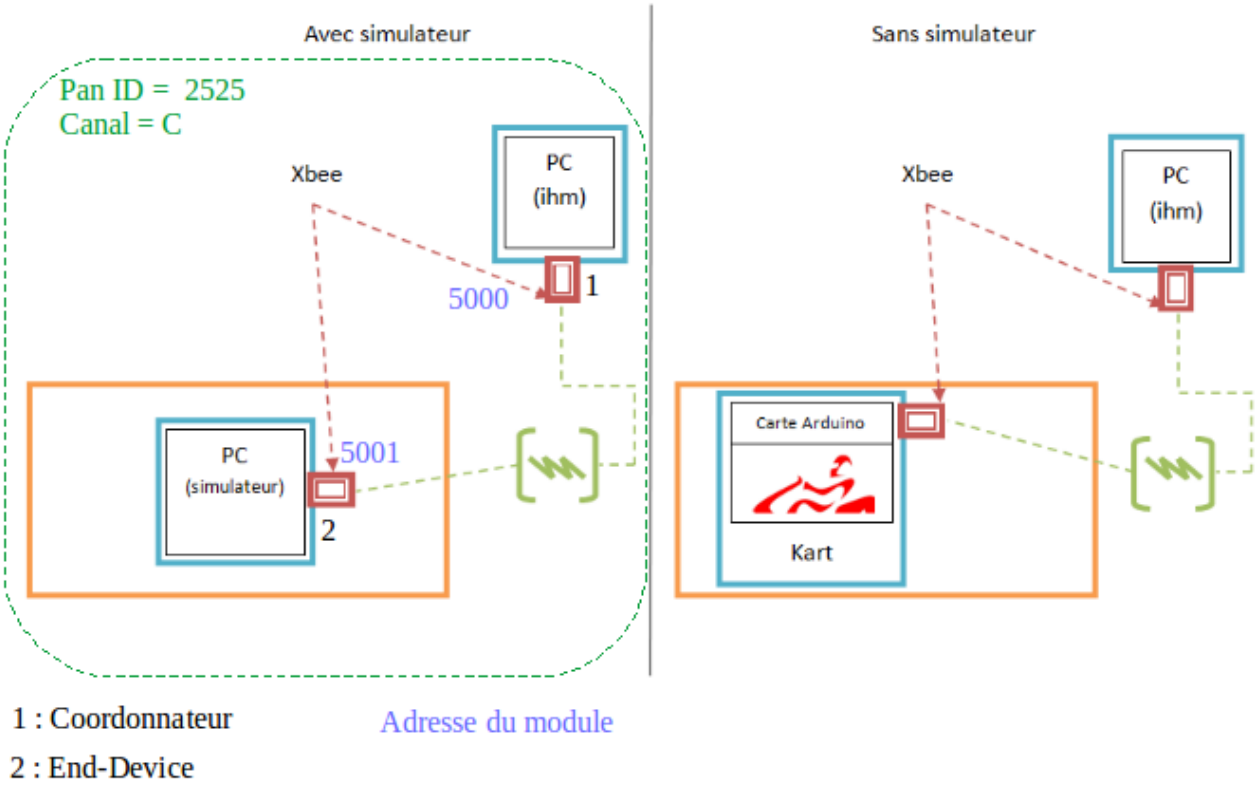
- Le mode Transparent est le mode par défaut, il est aussi plus simple d'utilisation. Car on émet et reçoit directement sur le port série.
- Le mode commande sert uniquement à configurer les modules par les commandes AT.
- Le mode API permet de programmer la communication au niveau trame du protocole 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 ».

Les autres commandes AT sont utiles pour configurer les adresses sources et destination du module.

+++	OK
ATID	2525
ATCH	C
ATMY	5000
ATAP	0
ATDH	0
ATDL	5001
ATWR	OK
ATCN	OK

Afin de pouvoir développer et valider ma partie, on m'a fourni un simulateur de kart.



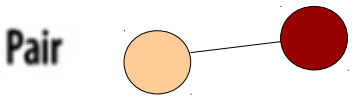
Ce schéma permet d'avoir une vision du système avec et sans l'utilisation du simulateur. Le simulateur simule un kart. Il permet de valider la partie PC supervision avant d'effectuer une validation sur site avec les parties des étudiants EC responsables du kart. Le simulateur a donc été utilisé pendant ma phase de développement pour un travail autonome.

Le **Coordonnateur** : est le responsable du réseau, il gère les nœuds au sein du WPAN

Le **End-Device** : peut être considéré comme un capteur (ici il représente un kart sur lequel des capteurs sont installés)

Dans ce projet le module installé sur le PC est le **coordonnateur** et il peut avoir plusieurs karts donc plusieurs **end-device.** Donc un réseau de type étoile.

Dans la situation avec le simulateur. Le simulateur est donc un kart, le réseau xbee est une **simple paire (point à point)**, car on communique avec un seul kart simultanément.

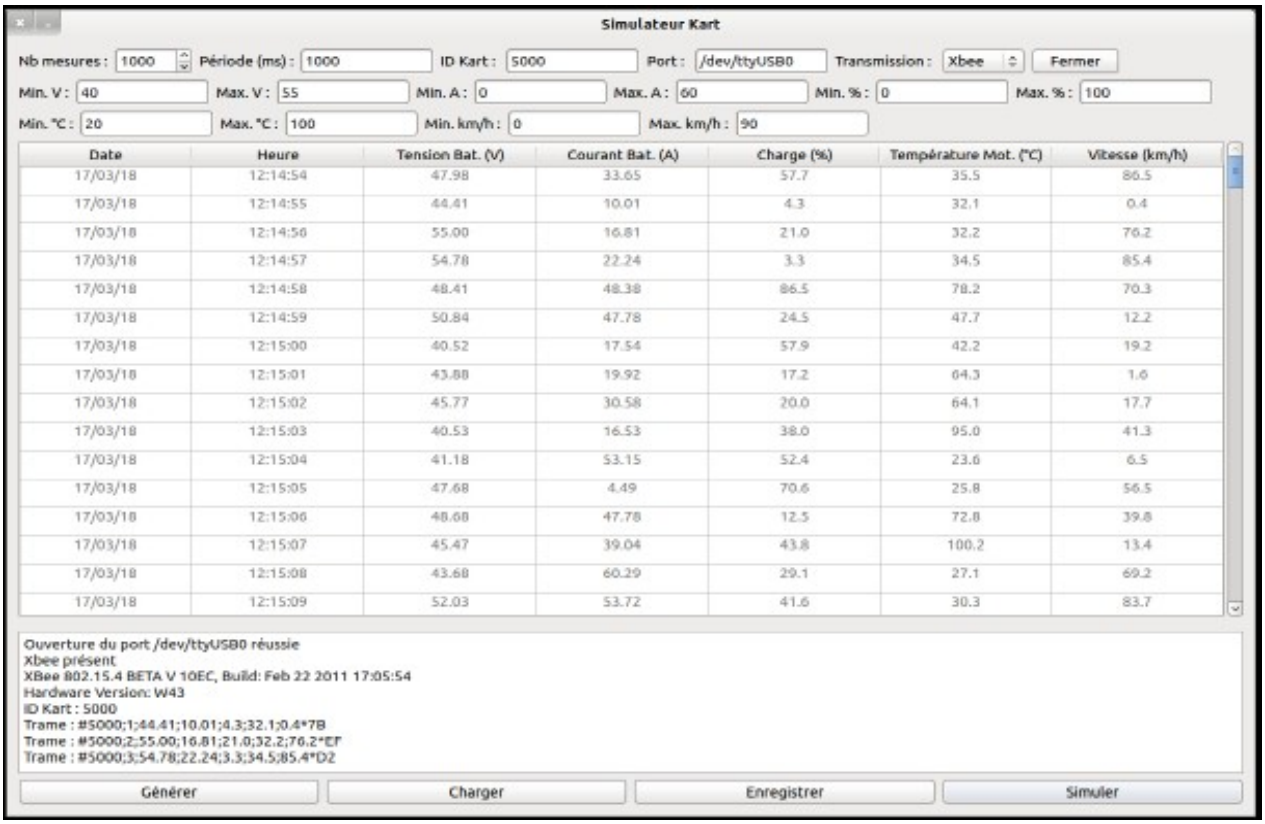


De plus la communication est réalisé dans un seul sens (**Simplex**) :

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

Sur le simulateur, il faut choisir :

- le port du module Xbee : ici le fichier « `/dev/ttyUSB0` »
- le type de transmission : « xbee »
- les valeurs minimum et maximum (si besoin) de chaque donnée télémétrique



Le simulateur de kart

Le protocole de communication

Pour communiquer entre le PC supervision et le kart, il a fallu établir un protocole de communication spécifique. Les données envoyées sont celles des capteurs présents sur le kart.

Les données télémétriques sont :

- Tension de la batterie (en volts)
- Courant de la batterie (en Ampères)
- Charge de la batterie (en pourcentage)
- Température du moteur (en degrés Celsius)

- Vitesse (en Km/h)

La trame possédera :

- un caractère pour délimiter le début de la trame, on utilise le « # »
- un séparateur pour séparer les données, on utilise le « ; »
- un symbole pour marquer le début du **checksum** (Le *checksum* permet de vérifier la validité de la trame reçue), on utilise le « * »

Il a fallu ajouter deux données pour connaître à qui appartiennent les données reçues :

- ID_TRAME : qui permet de connaître l'identifiant de la trame.
- ID_KART : qui permet de connaître l'identifiant du kart.

Donc la trame Kart est définie de cette manière :

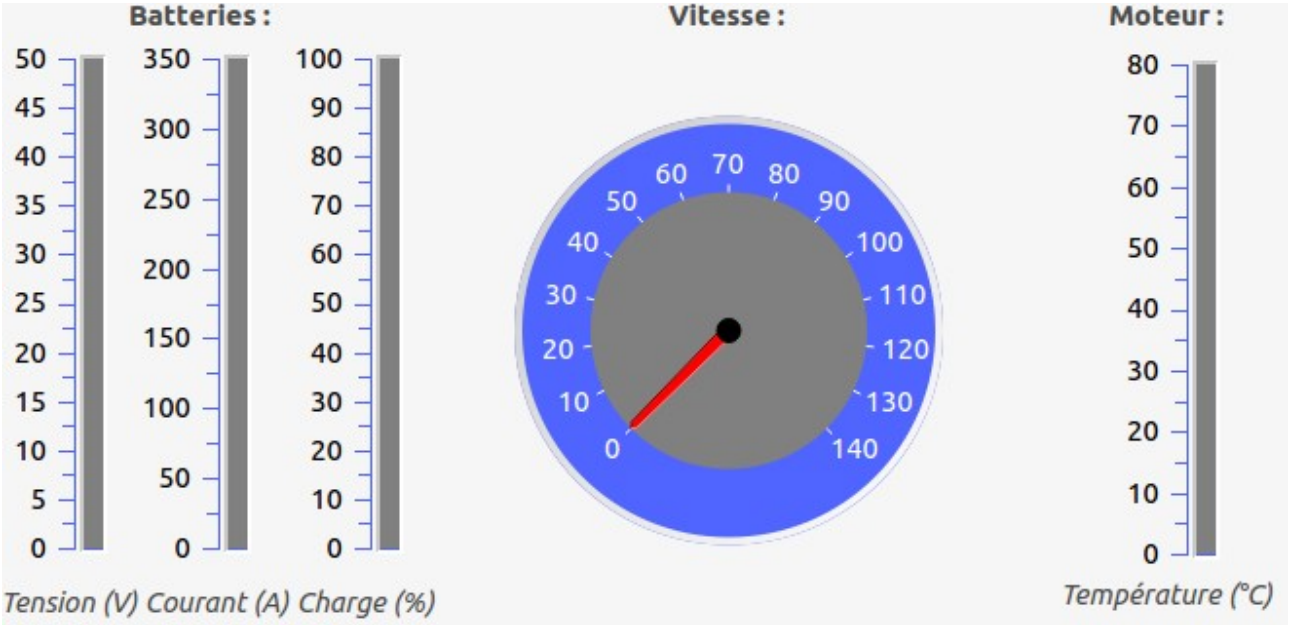
ID_KART ; ID_TRAME ; TENSION ; COURANT ; CHARGE ;

TEMPERATURE_MOTEUR ; VITESSE * CHECKSUM

Un exemple d'une trame :

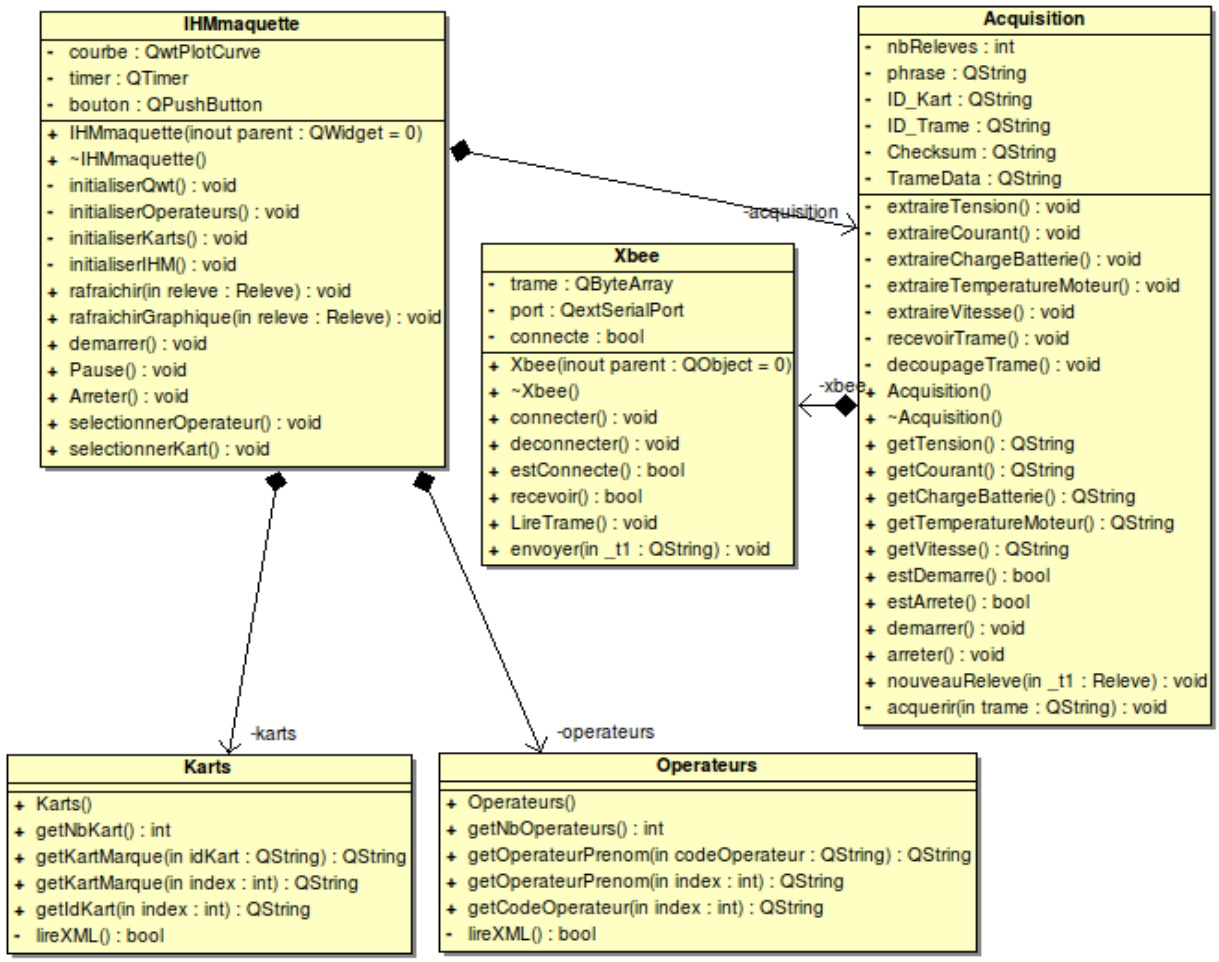
#101;1;48,2;12,5;77;63,1;25,63*1C

Ensuite ces valeurs sont affichées sur l'ihm :



Conception logicielle

Diagramme de classes



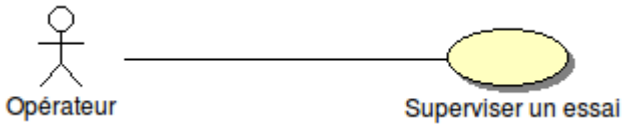
La classe Xbee permet de communiquer avec les karts : elle ouvre le port et reçoit les trames.

La classe Acquisition récupère les trames, les découpe, extrait les données et les conservent dans une structure (un relevé de mesures).

La classe IHMmaquette affiche les données télémétriques dans des **widgets**.

Les classes Karts et Opérateurs permettent de lire et de récupérer les valeurs des fichiers XML respectifs.

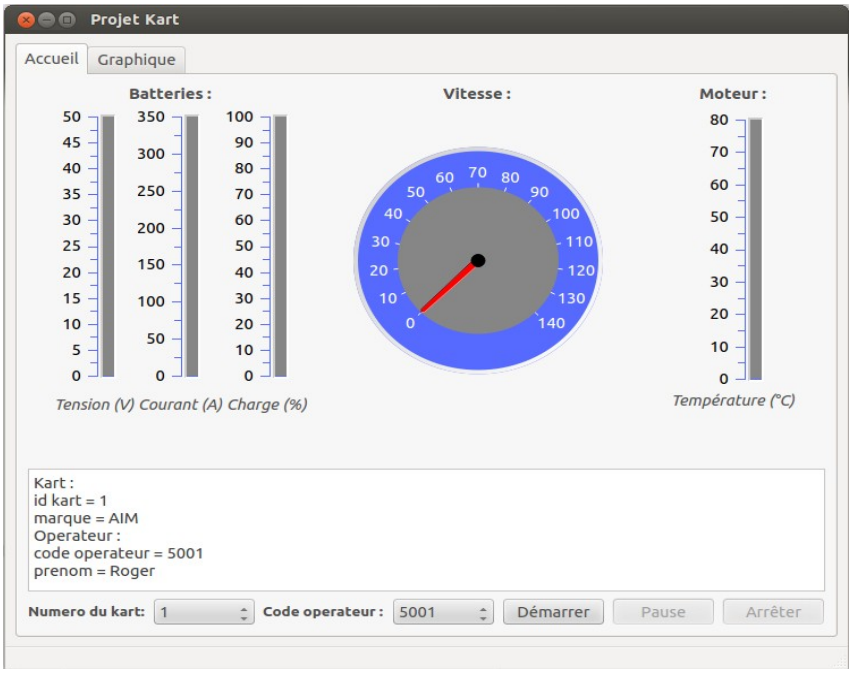
Cas d'utilisation « Superviser un essai »

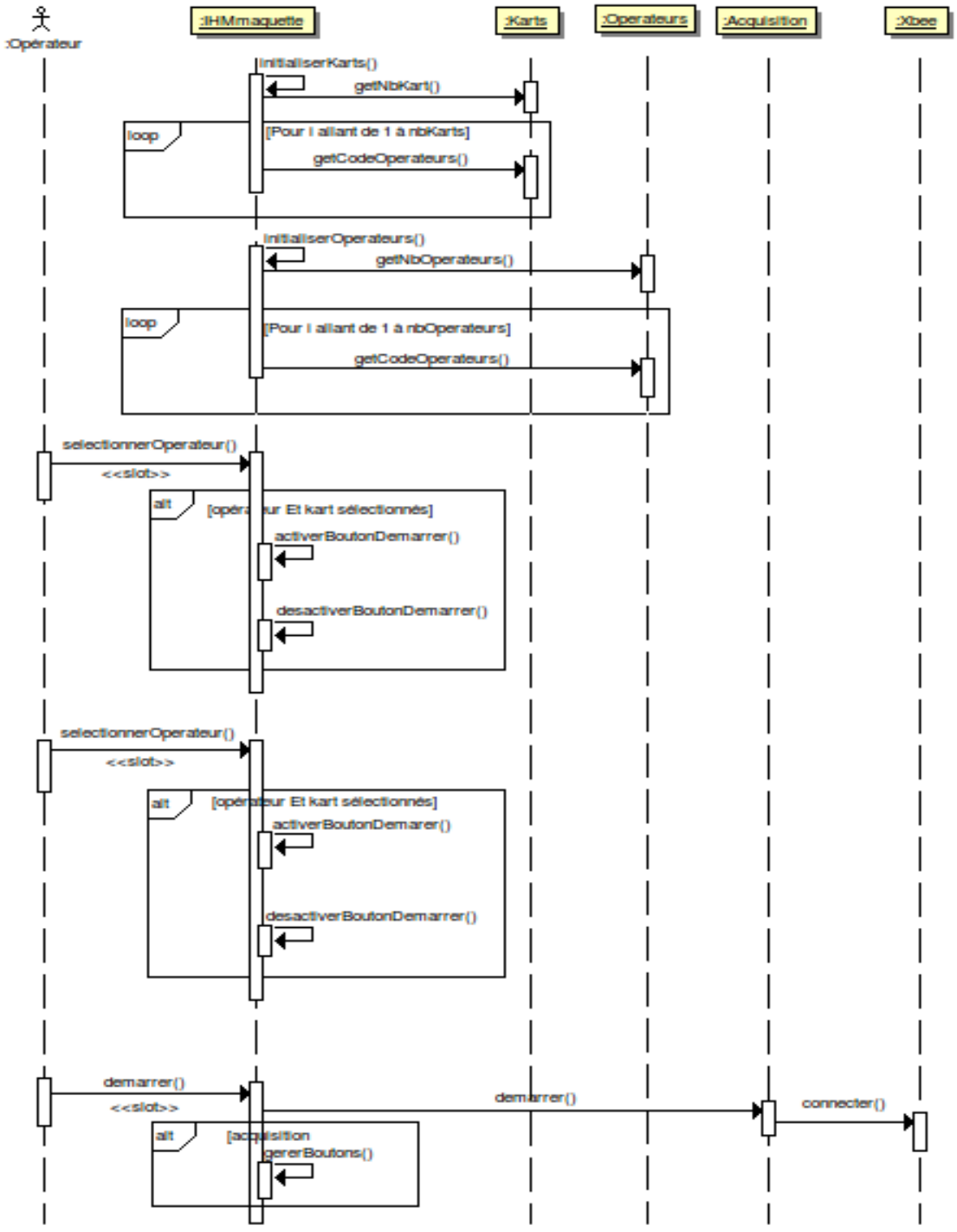


L'opérateur doit pouvoir **démarrer** l'essai (une campagne d'acquisition de données télémétriques) d'un kart en indiquant son **code opérateur** et en précisant l'**identifiant du kart** et son modèle. Il pourra éventuellement pendant l'essai **mettre en pause** l'acquisition des données. À la fin de la **supervision d'un essai**, il pourra **arrêter** l'acquisition.

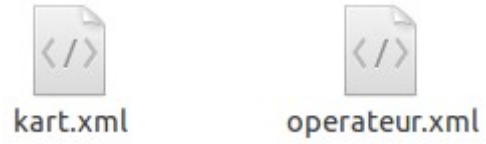
Scénario « Démarrer un essai »

Deux listes déroulantes permettent à l'opérateur de choisir le numéro du kart et son code d'opérateur pour l'essai.




:Opérateur
:IHMmaquette
:Karts
:Operateurs
:Acquisition
:Xbee
initialiserKarts()
getNbKart()
loop
[Pour i allant de 1 à nbKarts]
getCodeOperateurs()
initialiserOperateurs()
getNbOperateurs()
loop
[Pour i allant de 1 à nbOperateurs]
getCodeOperateurs()
selectionnerOperateur()
<<slot>>
alt
activerBoutonDemarrer()
desactiverBoutonDemarrer()
selectionnerOperateur()
<<slot>>
alt
activerBoutonDemarer()
desactiverBoutonDemarrer()
demarrer()
<<slot>>
demarrer()
connecter()
alt
gererBoutons()

On commence par récupérer la liste des identifiants des karts et des codes opérateurs. Ces informations sont stockées dans des fichiers XML. Un fichier pour les numéros de karts et un autre pour les codes opérateurs.



Les deux fichiers « `kart.xml` » et « `operateur.xml` » sont enregistrés dans un format standard qui permet à l'application d'échanger et de distribuer des données.

XML est un langage de balisage extensible, un document xml obéit aux règles syntaxiques et peut répondre à une DTD (*Document Type Definition*) ce qui n'est pas le cas ici.

De plus ce langage permet une universalité et une portabilité car en caractère ASCII et dispose d'un encodage UNICODE.

Dans ce projet il est utilisé comme un répertoire pour lister les karts et les opérateurs, qui utilisent ce dispositif.

Les documents Xml sont structurés en cinq parties :

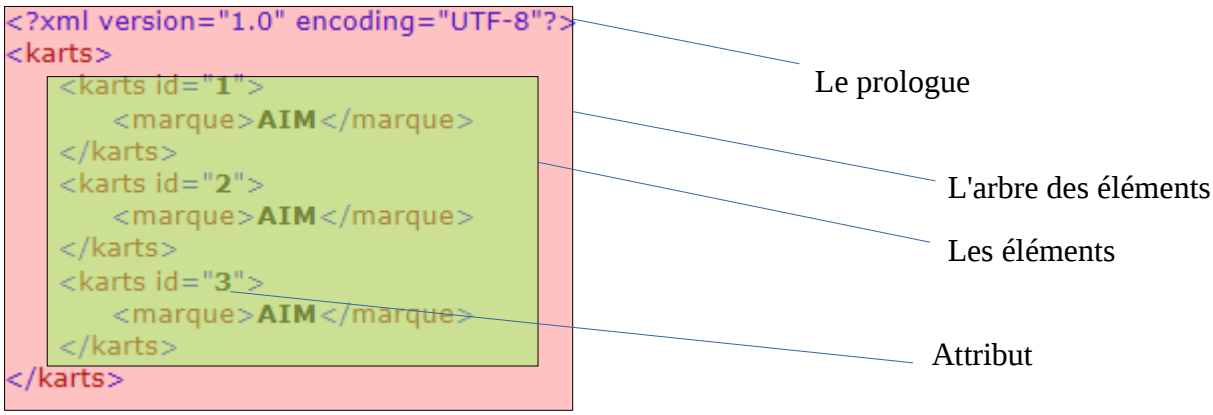
- le prologue : indique la version XML, le jeu de caractères et la présence ou pas d'une DTD (non présent dans ce projet)
- les instructions de traitement : instructions relatives à des applications qui traiteront le document comme une feuille de style (non présent dans ce projet)
- Commentaires : qui permet de commenter le document (exemple : « liste des numéro de karts)
- l'arbre des éléments : le contenu du document. Il considéré comme l'élément racine qui peut contenir d'autres éléments.
- Les éléments : qui sont aussi appelés des ***nœuds*** ou *node* sont définis par des balises :
  - <element> : balise d'ouverture
  - </element> : balise de fermeture
  - <element/> : balise vide

Un élément peut posséder des **attributs**.

Exemple : `<kart id="1"><marque>AIM</marque></kart>`

L'élément « **kart** » possède un attribut « **id** » avec pour valeur « **1** ».

Description du fichier « `kart.xml` »:



Les informations récupérés des fichiers Xml sont affichés dans la partie « les données des fichiers XML »

```
Kart :
id kart = 1
marque = AIM
Operateur :
code operateur = 5001
prenom = Roger
```

Pour stocker les données d'un élément « kart », j'ai défini une structure :

```
  typedef struct
{
    QString  id; /**< l'identifiant du kart (QString) */
    QString  marque; /**< la marque du kart (QString) */

} Kart;
```

Pour conserver l'ensemble des karts dans la classe Karts, j'utilise une liste :

```
  QList<Kart> karts;
```

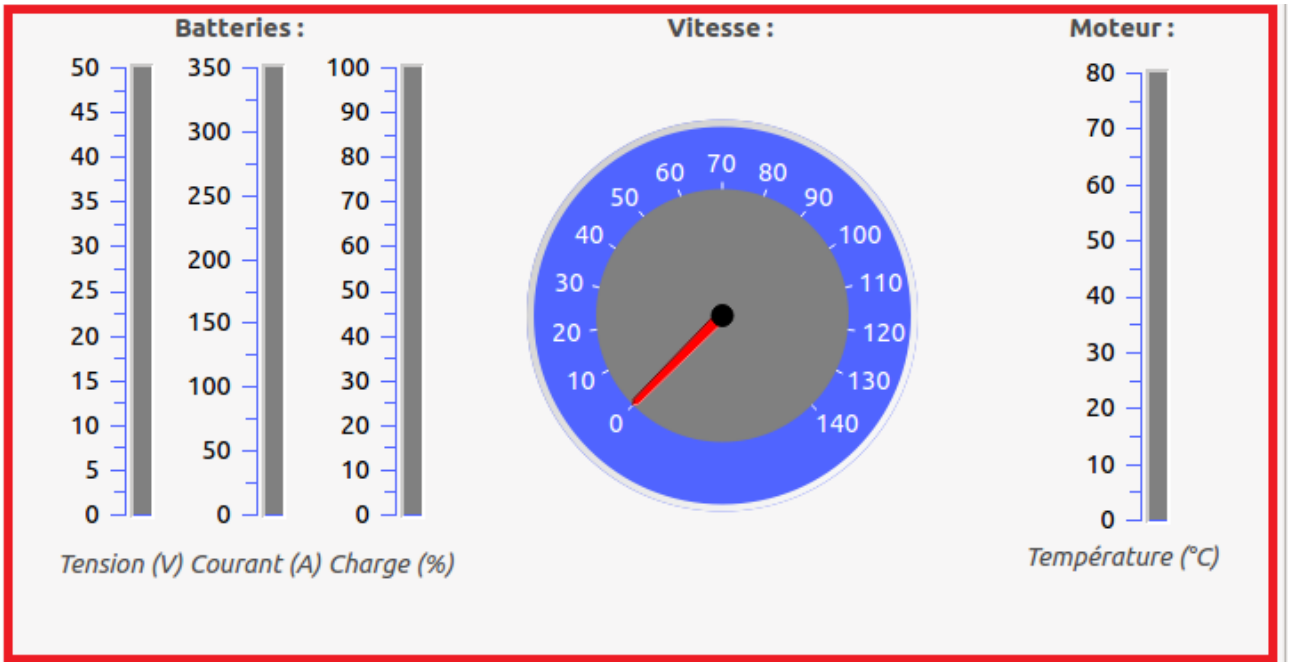
C'est la méthode lireXML() de la classe Karts qui réalise l'extraction des données du fichier pour remplir la liste `karts` :

```
  bool Karts::lireXML()
{
    QFile fichierXML("kart.xml");
    if(!(fichierXML.open(QIODevice::ReadOnly)))
    {
        QMessageBox::critical(0,"Erreur",QString::fromUtf8("Le fichier kart.xml
n'a pas pu être ouvert !"));
        return false;
    }
    else
    {
        QDomDocument documentXML;

        documentXML.setContent(&fichierXML, false);
        QDomElement racine = documentXML.documentElement(); // <karts>
        if(racine.isNull())
        {
            qDebug() << "<MaFenetre::lireXML()> erreur racine !";
            return false;
        }

        QDomNode noeudKart = racine.firstChild();
        if(noeudKart.isNull())
        {
            qDebug() << "<MaFenetre::lireXML()> erreur racine vide !";
            return false;
        }

        while(!noeudKart.isNull())
        {
            QDomElement elementKart = noeudKart.toElement(); // <kart>
            if(elementKart.isNull())
            {
```

```
                qDebug() << "<MaFenetre::lireXML()> erreur element kart !";
                break;
            }

            QDomNode noeudMarque = elementKart.firstChild();
            if(!noeudMarque.isNull())
            {
                QDomElement elementMarque = noeudMarque.toElement(); // <marque>

                kart.id = elementKart.attribute("id"); // attribut id
                kart.marque = elementMarque.text(); // élément marque
                karts.push_back(kart); // ajoute le kart à la liste

            }
            else qDebug() << "<MaFenetre::lireXML()> erreur noeud marque !";

            noeudKart = noeudKart.nextSibling();
        }

        fichierXML.close();
    }

    return true;
}
```

Cas d'utilisation « Visualiser les données de télémétrie en temps réel »

L'opérateur doit pouvoir **visualiser** un essai (une campagne d'acquisition de données télémétriques) d'un kart. Pour cela il doit remplir les conditions du cas **« superviser un essai »** (être connecté)**.** Ces données seront affichées en temps réel sur des **Widgets QWT**. Plus précisément sur un **QwtThermo** et un **QwtDial**. Les deux **widgets** héritent de **QWidget** (La classe de base de tous les objets graphiques dans Qt).



Les données qui concernent la batterie et le moteur sont affichées sur des **QwtThermo**.

Et la vitesse sur un **QwtDial**.

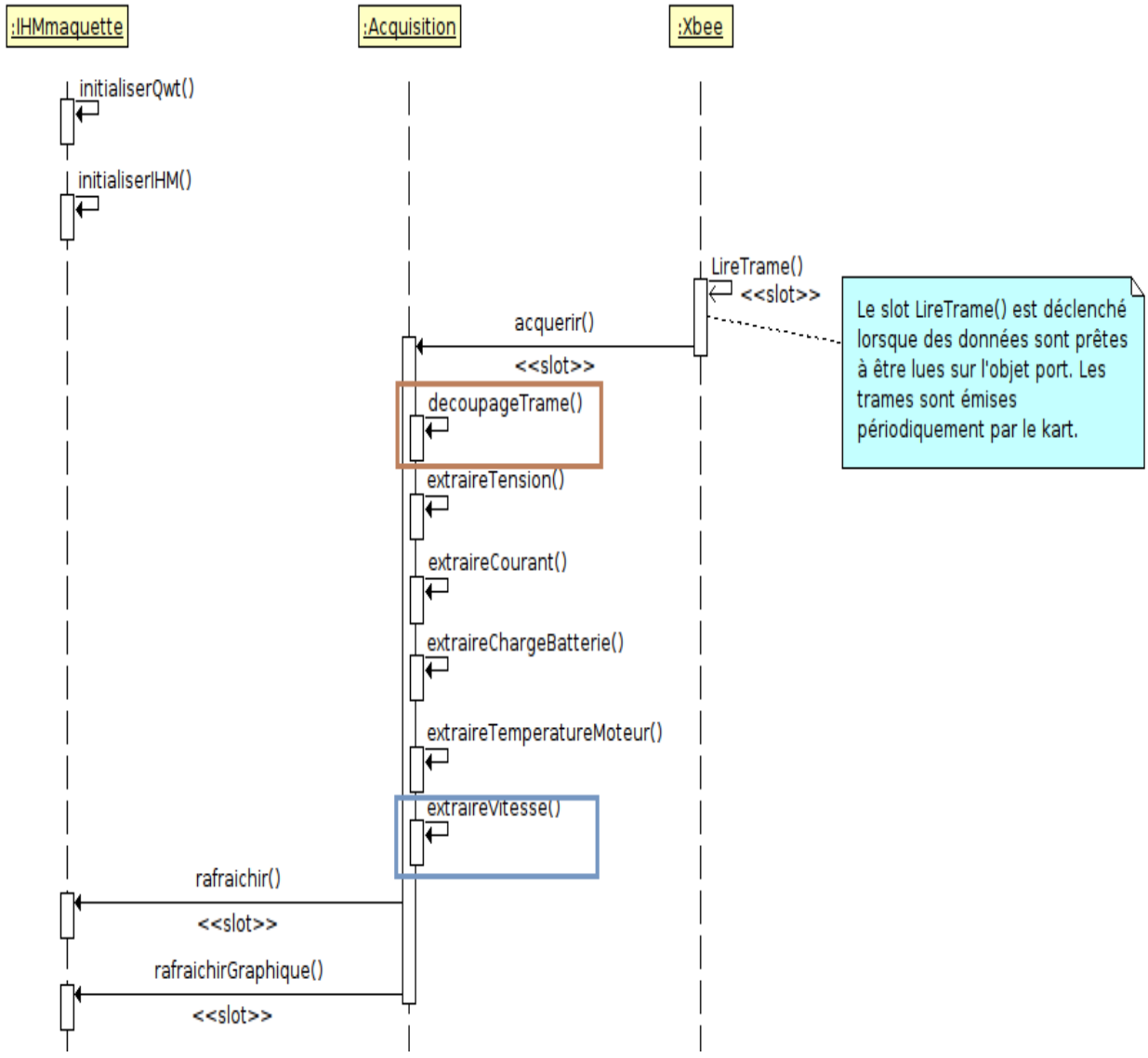
QWT (***Q**t **W**idgets for **T**echnical Applications*) est un jeu de widgets pour Qt utiles pour des programmes techniques.

La méthode initialiserQwt() de la classe IHMmaquette a pour rôle de personnaliser l'affichage des widgets QWT. Par exemple pour le QwtDial :

```
  void IHMmaquette::initialiserQwt()
{
    // Vitesse
    // L'origine
    ihm->qwtVitesse->setOrigin(135.0);
    ihm->qwtVitesse->setScaleArc(0.0, 270.0);
    ihm->qwtVitesse->setScale(0, 10, 10);
    ihm->qwtVitesse->scaleDraw()->setSpacing(4);
    ihm->qwtVitesse->setRange(0,140,10);
    ihm->qwtVitesse->setReadOnly(true);
```

```
    // Aiguille
    QwtDialSimpleNeedle *needle = new QwtDialSimpleNeedle(
QwtDialSimpleNeedle::Arrow, true, Qt::red, QColor(Qt::black).light(70)); /**<
TODO */
    ihm->qwtVitesse->setNeedle(needle);
    ihm->qwtVitesse->setScaleComponents( QwtAbstractScaleDraw::Ticks |
QwtAbstractScaleDraw::Labels );
    ihm->qwtVitesse->setScaleTicks(0, 0, 4);

    // modifier les couleurs QwtVitesse
    QPalette p2 = ihm->qwtVitesse->palette();
    p2.setColor(QPalette::Base, QColor(80, 100, 255));
    p2.setColor(QPalette::WindowText, QColor(128, 128, 128));
    p2.setColor(QPalette::Text, QColor(255, 255, 255));
    ihm->qwtVitesse->setPalette(p2);

    // etc ... pour les autres widgets
}
```

On suppose ici que nous sommes déjà connecté donc que le scénario « Démarrer un essai » du cas d'utilisation « Superviser un essai » a été réalisé.



La méthode decoupageTrame() de la classe Acquisition a pour rôle de segmenter et de découper la trame. Elle permet d'enlever le caractère de début de trame « # », la partie qui concerne le *checksum* et de conserver les champs ID_Kart et Id_Trame.

```
//TRAME
void Acquisition::decoupageTrame()
{
    phrase.remove(0,1); //pour enlever le "#"
    ID_Kart = phrase.section(';',0,0);//left(3);
    qDebug() << Q_FUNC_INFO << "ID_Kart" << ID_Kart << endl;
    phrase.remove(0,5);

    ID_Trame = phrase.section(';',0,0);
    qDebug() << Q_FUNC_INFO << "ID_Trame" << ID_Trame << endl;
    //phrase.remove(0,2);

    Checksum = phrase.right(2);
    qDebug() << Checksum << endl;
    phrase.remove(strlen(phrase.toStdString().c_str())-
3,strlen(phrase.toStdString().c_str()));//26 28

    TrameData = phrase;
    qDebug() << TrameData << endl;
}
```

En rouge les champs enlevés lors du découpage et en orange les champs conservés.

En bleu le reste des champs conservés dans la TrameData.

ID_KART ; ID_TRAME ; TENSION ; COURANT ; CHARGE ;

TEMPERATURE_MOTEUR ; VITESSE * CHECKSUM

La méthode extraireVitesse() de la classe Acquisition a pour rôle de récupérer le champ « vitesse ». Pour cela il faut préciser une section : une section correspond à une valeur entre deux délimiteurs, exemple «;VITESSE; » = une section. Elle conserve la valeur extraite de la section dans l'attribut vitesse qui se trouve dans la structure releve.

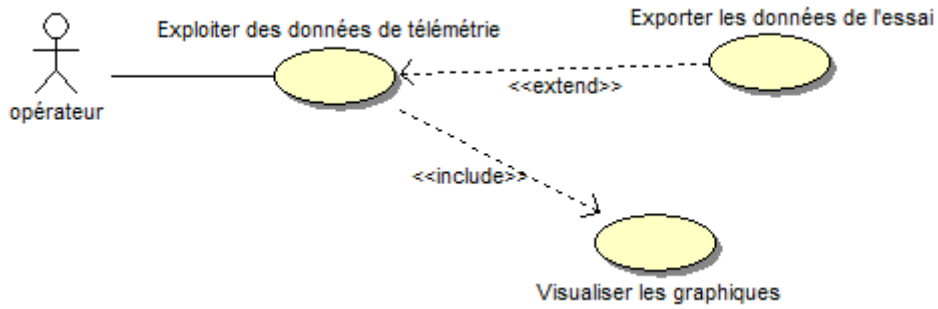
```
void Acquisition::extraireVitesse()
{
    //découpe la trame avec le délimiteur

    releve.vitesse = TrameData.section(';', 5, 5);
    qDebug() << releve.vitesse << endl << TrameData << endl;
}
```

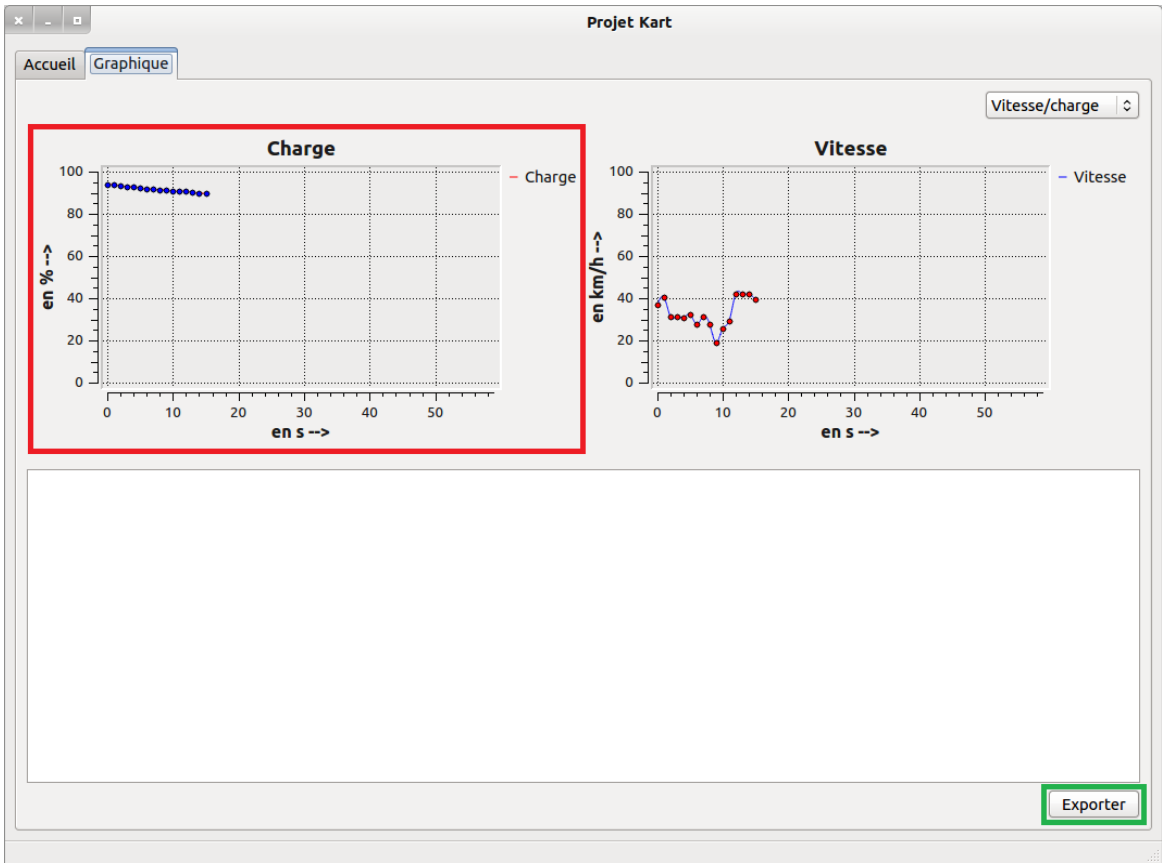
L'onglet Graphique de l'ihm

L'onglet graphique de l'ihm est quand à lui composé de deux parties.

- Visualiser les graphiques.

- Exporter les données de l'essai.(sa réalisation est optionnelle)





Pour le scénario « Visualiser les graphiques ».

L'opérateur doit pouvoir **visualiser** un essai (une campagne d'acquisition de données télémétriques) d'un kart. Pour cela il doit remplir les conditions du cas **« superviser un essai »** (être connecté)**.** Ces données seront affichées en temps réel sur un **Widgets QWT**, plus précisément sur un **QwtPlotCurve**. Ce **widgets** hérite de **QWidget** (La classe de base de tous les objets graphiques dans Qt), Il permet de concevoir des graphiques et faire des tracés.

La méthode rafraichirGraphique(Releve Releve) de la classe IHMmaquette a pour rôle d'afficher les valeurs présente dans la structures releve sur le graphique correspondant :

```
void IHMmaquette::rafraichirGraphique(Releve releve)
{
    QTime maintenant = QTime::currentTime();
    int seconde = maintenant.second();
```

```
    if(seconde == 0)
    {
    //on efface les tracés précédents
        charge.x.clear();
        charge.y.clear();
        courbeCharge->setSamples(charge.x, charge.y);
        ihm->qwtGraphiqueCharge->replot();
    }
    // on ajoute les mesures
    charge.x.push_back((double)seconde);
    charge.y.push_back(releve.chargeBatterie.toDouble());
    // on trace les points x,y
    courbeCharge->setSamples(charge.x, charge.y);
    ihm->qwtGraphiqueCharge->replot();
```

Ceci représente le code de la partie graphique (charge uniquement).

Pour le scénario « Exporter les données de l'essai ».

L'opérateur peut s'il le souhaite, exporter les données de l'essai dans un format CSV. Pour pouvoir sauvegarder toutes les informations d'un kart pendant un essai. Les données seront enregistrées sous forme de tableur.

Exemple de fichier CSV, à travers une application qui lit ces fichiers :

Tension (V)	Courant (A)	Charge (%)	Température (°C)	Vitesse (km/h)
12,8	63	97	20	8,7
14	79	97	22	12,2
16,2	82	96,7	23	15,4
18,5	87	96,7	23	18,6
19,6	92	96,5	25	21,9
24	101	96,5	26	27,3
26,5	113	96,3	28	33,4
29	123	96,3	31	38
32	127	96	37	44

Avec un éditeur de texte :

Il faut choisir un séparateur. On utilisera le séparateur ‘;’ et on protégera les données avec des guillemets ‘"’

Tension ; Courant ; Charge ; Temperature ; Vitesse

"12,8" ; "63" ; "97" ; "20" ; "8,7"

"14" ; "79" ; "97" ; "22" ; "12,2"

Et on fait de même pour tout le fichier(ici seul les deux premières lignes et la légende ont été retranscrites)

Dans les fichiers CSV on retrouvera les différentes données de la structure releve.

```
typedef struct
{
    /** @todo : ajouter un horodatage */
    QString tension; /**< la tension en Volts */
    QString courant; /**< le courant en Ampères */
    QString chargeBatterie; /**< la charge de la batterie en % */
    QString temperatureMoteur; /**< la température du moteur en °C */
    QString vitesse; /**< la vitesse en km/h */
} Releve;
```

De plus cette structure nous permet de manipuler ces mesures dans notre programme.

Test de validation : Recette

Description	OUI	NON
La supervision d'un essaipermet de démarrer latélémétrie	✓	
La récupération des donnéestélémétriques est opérationnelle	✓	
L'affichage des donnéestélémétriques en temps réel estfonctionnel	✓	
L'affichage des donnéestélémétriques sous forme degraphiques est fonctionnel	✓	
L'exportation des donnéestélémétriques au format CSVest possible		✗

Glossaire

- Données télémétriques : des informations qui sont envoyées à distance par des procédés optiques, acoustiques où radio.
- CSV : (*Comma Separated Values*) est un fichier tableur,contenant des données sur chaque ligne séparée par un caractère de séparation (« , » , « ; » où une tabulation).
- API :(*Application Programing Interface*)une interface de programmation applicative est un ensemble normalisé de classes, de méthodes ou de fonctions par lesquelles un logiciel (ici Qt creator) offre des services et une bibliothèque.
- Subversion : (en abrégé svn) est un logiciel de gestion de versions. Le principe est d'avoir un dépôt centralisé et unique. Subversion fonctionne sur le mode *client-serveur*.
- Doxygen : est un générateur de documentation sous licence libre capable de produire une documentation logicielle à partir du coce source d'un programme. Pour cela il utilise une syntaxe de différents commentaires.
- QWidget : QWidget est la classe de base de tous les objets d'interfaces utilisateurs. Un widget est l'atome de l'interface utilisateur : il peut recevoir un événement (souris, clavier où système…) et dispose d'un représentation graphique à l'écran.
- QwtThermo : est un widget qui permet de faire des thermomètres.
- QwtDial : est un widget qui permet de faire des horloges, des compteurs de vitesse...
- QString : est un type de Qt , il représente le type string en C++
- QList : est un conteneur de Qt, il représente le conteneur list en C++
- Dongle : est un appareil qui peut être branché directement sur un port USB de notre ordinateur, il fonctionne comme une passerelle entre l'ordinateur et le module Xbee
- Arduino : est une carte électronique composé d'un micro-contrôleur, qui peut être programmé pour effectuer des taches dans certains domaines (domotique, robotique ,système embarqué...)
- IEE 802.15.4 : est un protocole de communication défini par l'IEEE. 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 et du faible débit des dispositifs utilise ce protocole. Comme le protocole de communication Zigbee.

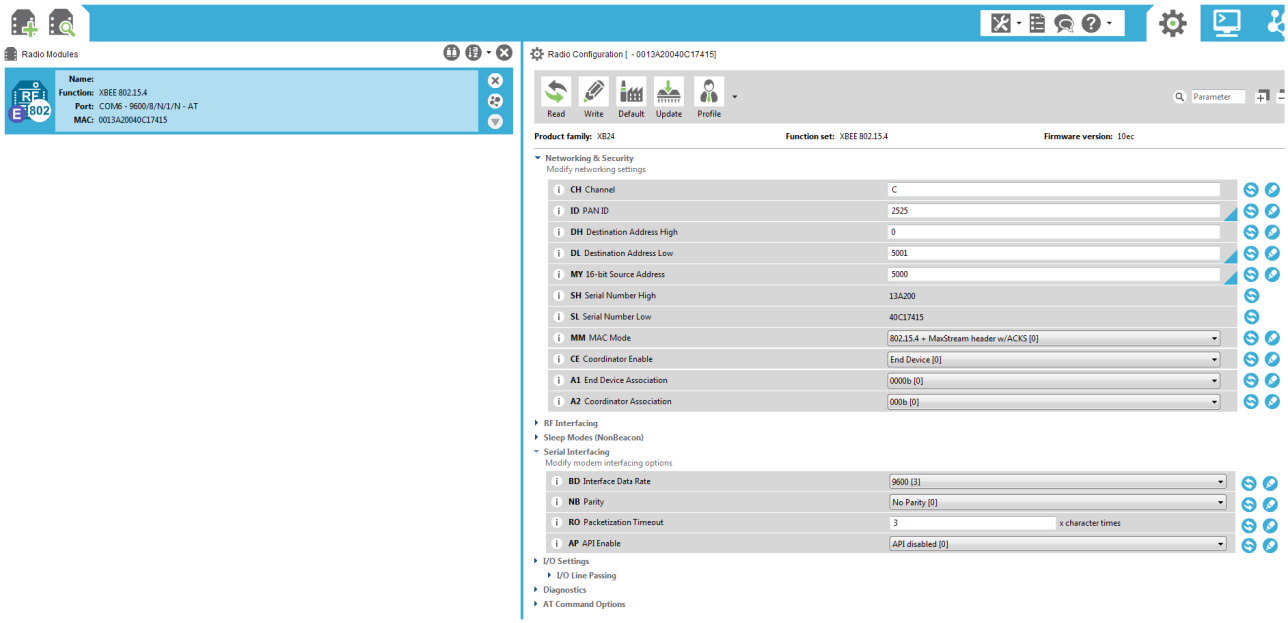
- Zigbee : est un protocole de haut niveau permettant la communication de petites radios, à consommation réduite, basée sur la norme IEEE 802.15.4 pour les réseaux à dimension personnelle (WPAN)
- RS-232 : est une norme standardisant une voie de communication de type série, il est communément appelé « port série ».
- Trame : est un paquet d'information véhiculé au travers d'un support physique.
- DTD : (document type définition) est soit un fichier, soit une partie d'un document SGML ou XML, qui décrit la grammaire du document. Liste des éléments (ou balises), des attributs, leur contenu et leur agencement.
- UNICODE : est un standard informatique qui permet des échanges de textes dans différentes langues, à un niveau mondial. Il permet de donner à tout caractères de n'importe quel système d'écriture un nom et un identifiant numérique.
- Noeuds : en XML il existe trois différents nœuds. Les nœuds de document, d'élément et de texte :
    - Le nœud de document est la racine du document Xml, Il est abrégé avec la barre oblique « / »
    - Le nœud d'élément est désigné par un nom qualifié au sein d'un espace de noms (<espace:élément/>), Un élément peut contenir des attributs et la plupart des autres nœuds : texte, éléments (à l'exception du nœud de document).
    - Le nœud de texte n'a pas d'enfants, il est toujours contenu dans un élément. (Exemple : un nœud de texte, un nœud d'élément et un nœud de texte)

Annexe 1 : Configuration des modules Xbee sous Windows avec XCTU

Pour configurer nos modules sur le système d'exploitation Windows, nous utiliseront XCTU.

XCTU : est un logiciel avec une interface graphique qui nous permettra de paramétrer nos modules sans écrire les commandes AT. Toutes les informations de vos modules xbee sont enregistrés sous forme de formulaire que vous pouvez modifier au clavier .

Voici à quoi ressemble le logiciel, nous retrouvons les mêmes informations (canal, Pan Id...).



Annexe 2 : Caractéristiques du modules Xbee

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

Worldwide Acceptance

FCC Approval (USA) Refer to Appendix A [p65] for FCC Requirements. Systems that contain XBee®/XBee-PRO® RF Modules inherit Digi Certifications.

ISM (Industrial, Scientific & Medical) **2.4 GHz frequency band**

Manufactured under **ISO 9001:2000** registered standards

XBee®/XBee-PRO® RF Modules are optimized for use in the United States, Canada, Australia, Japan, and Europe. Contact Digi for complete list of government agency approvals.

Annexe 3 : Commandes AT du module Xbee

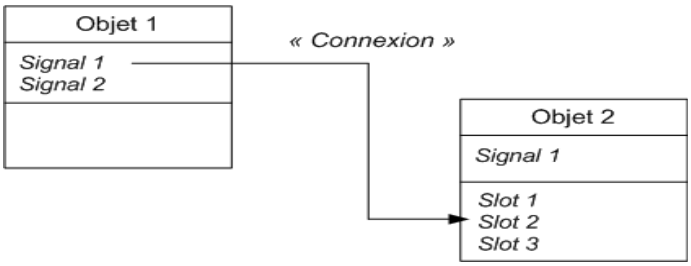
AT Command	Command Category	Name and Description	Parameter Range	Default
CH	Networking {Addressing}	**Channel**. Set/Read the channel number used for transmitting and receiving data between RF modules (uses 802.15.4 protocol channel numbers).	0x0B - 0x1A (XBee) 0x0C - 0x17 (XBee-PRO)	0x0C (12d)
ID	Networking {Addressing}	**PAN ID**. Set/Read the PAN (Personal Area Network) ID. Use 0xFFFF to broadcast messages to all PANs.	0 - 0xFFFF	0x3332 (13106d)
DH	Networking {Addressing}	**Destination Address High**. Set/Read the upper 32 bits of the 64-bit destination address. When combined with DL, it defines the destination address used for transmission. To transmit using a 16-bit address, set DH parameter to zero and DL less than 0xFFFF. 0x000000000000FFFF is the broadcast address for the PAN.	0 - 0xFFFFFFFF	0
DL	Networking {Addressing}	**Destination Address Low**. Set/Read the lower 32 bits of the 64-bit destination address. When combined with DH, DL defines the destination address used for transmission. To transmit using a 16-bit address, set DH parameter to zero and DL less than 0xFFFF. 0x000000000000FFFF is the broadcast address for the PAN.	0 - 0xFFFFFFFF	0
MY	Networking {Addressing}	**16-bit Source Address.** Set/Read the RF module 16-bit source address. Set MY = 0xFFFF to disable reception of packets with 16-bit addresses. 64-bit source address (serial number) and broadcast address (0x000000000000FFFF) is always enabled.	0 - 0xFFFF	0
SH	Networking {Addressing}	**Serial Number High**. Read high 32 bits of the RF module's unique IEEE 64-bit address. 64-bit source address is always enabled.	0 - 0xFFFFFFFF [read-only]	Factory-set
SL	Networking {Addressing}	**Serial Number Low**. Read low 32 bits of the RF module's unique IEEE 64-bit address. 64-bit source address is always enabled.	0 - 0xFFFFFFFF [read-only]	Factory-set

Annexe 4 : Les Signal/Slot sur QT

Tout d'abord sur Qt creator on peut programmer des évènements, on dispose pour cela de signaux et de slots. Un slot correspond à la fonction qui est appelée lorsqu'un événement se produit. On dit qu'un signal appelle un slot. (Le slot est une méthode d'une classe.

Il faut lier un signal avec un slot Qt appelle cela une « connexion »

-Le Signal clicked() qui s'execute sur le BoutonDemarer, appelle la méthode demarrer().




```
connect(ihm->BoutonDemarer, SIGNAL(clicked()),this,SLOT(demarrer()));
```