POLYTECH
CLERMONT-FERRAND

ZEBULLE
RELAXATION SENSORIELLE

Note d’application

MISE EN PLACE D'UNE COMMUNICATION BLUETOOTH SOUS ANDROID/QT

Genty Damien GE5A Option Robotique et puissance

PROJET : SPECIFICATION D’UN SYSTEME DE RELAXATION | P15AB09

Table des matières

Table des figures 2
Introduction 3
L'interface du périphérique Bluetooth RN42 et du microcontrôleur PIC18F27J53 5
Configuration de la communication 5
L'écriture et la lecture 6
La communication Bluetooth sous QT Android 7
Les classes nécessaires au fonctionnement de la communication Bluetooth 7
La classe QBluetoothSocket 7
La classe QBluetoothDeviceDiscoveryAgent 7
La classe QBluetoothDeviceInfo 7
La classe Btport 7
Les variables de la classe Btport 8
Les fonctions gérant la connexion du Bluetooth 8
Le scan des périphériques Bluetooth allumés 8
Fonction de connexion à un périphérique donné 9
L'écriture et la lecture 10
Conclusion 12

Table des figures

Figure 1 : Schéma bloc de la communication Bluetooth entre la tablette et le microcontrôleur 3
Figure 2 : Exemple d'une relation entre un signal et un slot 4
Figure 3 : Contenu du registre TXSTAX (registre de transmission de données)..... 5
Figure 4 : Registre RXSTAX (registre de réception de données)..... 6
Figure 5 : Registre de configuration de la vitesse de transmission 6
Figure 6 : Commande nécessaire à l'utilisation des classes de Bluetooth 7
Figure 7 : Utilisation de la classe QBluetoothDeviceDiscoveryAgent 8
Figure 8 : Connexion du socket au périphérique Bluetooth désigné 9
Figure 9 : Interconnexion de la communication Bluetooth 11

Introduction

Le Bluetooth est un protocole de communication permettant l'échange de données de manière non filaire et dans un rayon de 100 mètres environ. Dans ce projet, il est utilisé pour faire dialoguer une application pour tablette Android développée sous QT Creator 5.5 et un microcontrôleur PIC18F27J53 de la marque Microchip.

Le microcontrôleur utilise son interface de communication série pour se relier au périphérique Bluetooth RN42 de la marque Microchip. Le système complet peut donc être représenté sous forme de schéma bloc de la façon suivante :

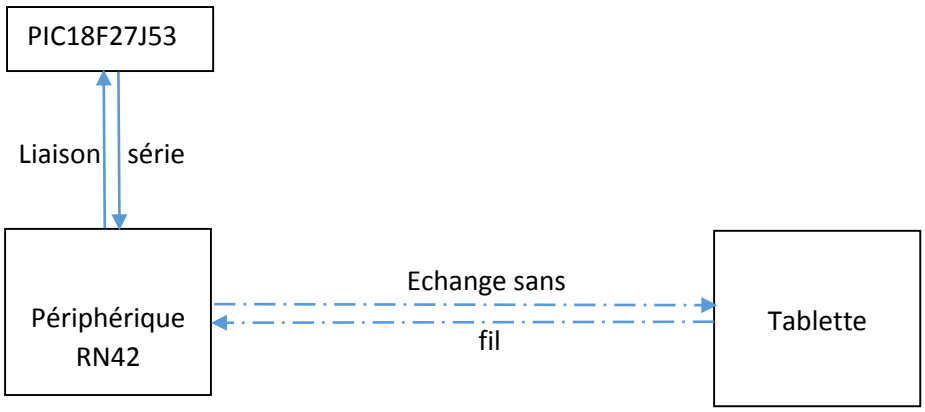

Figure 1 : Schéma bloc de la communication Bluetooth entre la tablette et le microcontrôleur

L'environnement de développement QT Creator est un outil permettant la création d'interface graphique. Cet environnement se découpe en plusieurs modules qui facilitent la création des interfaces. Un de ces modules, QT Android, permet quant à lui d'exporter les applications crées sous QT Creator pour les utiliser sous des téléphones ou tablettes utilisant le système d'exploitation Android.

La particularité de cet IDE (Integrated Development Environment) réside en son langage de programmation. En effet, une majorité d'IDE utilisent le Java pour développer sous Android. QT Creator, quant à lui, utilise principalement le C++. La communication Bluetooth développée ici est donc totalement réalisée en C++. L'autre particularité de QT réside en l'utilisation des signaux et des slots. La relation signal/slot est la suivante :

Un signal correspond un évènement, un slot va correspondre à la cause d'un évènement. Signaux et slots peuvent être reliés via des connects. La figure suivante par exemple représente la relation liant le changement d'une valeur d'un slider vertical, MusicSlider, avec la mise à jour du contenu d'une spinbox, MusicSpinBox et inversement :

```
connect(ui->MusicSpinBox, SIGNAL(valueChanged(int)), ui->MusicSlider, SLOT(setValue(int)));
connect(ui->MusicSlider, SIGNAL(valueChanged(int)), ui->MusicSpinBox, SLOT(setValue(int)));
```

Figure 2 : Exemple d'une relation entre un signal et un slot

A travers cette note d'application, différents points seront abordés :

En premier lieu, une explication rapide de la liaison série mise en place au niveau du microcontrôleur sera réalisée. Suite à cette partie, la création de la communication Bluetooth sous QT Android sera effectuée en débutant par une explication générale des différentes classes utilisées et en poursuivant sur la mise en place totale du Bluetooth au travers de l'application.

L'interface du périphérique Bluetooth RN42 et du microcontrôleur PIC18F27J53

Il s'agit ici d'introduire rapidement comment le périphérique RN42 peut être utilisé avec un microcontrôleur. Le but de cette partie est donc d'avoir une compréhension globale de la communication Bluetooth au niveau d'un microcontrôleur et son périphérique Bluetooth.

Configuration de la communication

Comme dit précédemment, le microcontrôleur communique via une liaison série avec le périphérique Bluetooth RN42. Cette liaison est réalisée à l'aide d'un des modules UART (Universal Asynchronous Receiver Transmitter) du microcontrôleur. Pour réaliser cette liaison série il est donc obligatoire que le microcontrôleur soit constitué d'au minimum un module UART.

Ce type de module nécessite l'utilisation de deux broches appelées Rx et Tx représentant respectivement la broche de réception et de transmission de données. Pour les utiliser il est nécessaire de configurer le microcontrôleur en positionnant la broche Rx en entrée et Tx en sortie.

L'UART du microcontrôleur a été configuré pour fonctionner en mode asynchrone. Cette configuration permet un échange de données qui ne soit pas géré par un signal d'horloge.

L'utilisation de l'UART est gérée à l'aide des fonctions contenues dans la librairie <plib/usart.h> fournie par Microchip. L'utilisation de ces fonctions nécessitent, en fonction du microcontrôleur utilisé, différentes utilisations de la directive #define :

- #define MASKS_AND_USE permettant l'utilisation des masques pour les fonctions de configurations de l'UART,

- #define EAUSART_Vx qui permet de définir la version des fonctions à utiliser. Ces versions dépendent du microcontrôleur utilisé. Dans notre cas, utilisant le PIC18F27J53, il faut utiliser le #define EAUSART_V11.

La configuration de l'UART dépend de 3 registres :

Le premier permet de configurer l'UART pour la transmission de données en fixant en autre le mode de communication (synchrone ou asynchrone), le nombre de bits utilisé pour une transmission (8 ou 9) et la type de transmission (rapide ou lent).

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

Figure 3 : Contenu du registre TXSTAX (registre de transmission de données)

Le second registre configure l'UART pour la réception des données arrivant du périphérique Bluetooth.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Figure 4 : Registre RXSTAX (registre de réception de données)

Le dernier configurae la vitesse de transmission des données.

R/W-0	R-1	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDEN
bit 7							bit 0

Figure 5 : Registre de configuration de la vitesse de transmission

La configuration de ces différents registres peut notamment s'effectuer grâce aux fonctions *OpenXUSART* et *baudXUSART* pour que la liaison UART fonctionne en asynchrone et à 115200 bauds. (Le X représente ici le numéro de l'UART utilisée, si le microcontrôleur ne possède qu'une seule UART les fonctions s'écrivent alors sans numéro)

L'écriture et la lecture

La lecture se fait à l'aide de la fonction *ReadXUSART().* Il est nécessaire de vérifier que le bit RCXIF, indiquant qu'une réception est déjà en cours, soit passé à la valeur 1 avant d'utiliser la fonction de lecture.

L'écriture, quant à elle, s'effectue en utilisant la fonction *WriteXUSART(char c)*. Cette fonction va permettre d'envoyer à la liaison série le caractère c qui sera ensuite envoyé par le périphérique Bluetooth. Avant de pouvoir renvoyer un nouveau caractère il faudra attendre que le bit TRMT du registre TXSTA soit passé à 1, cela indiquant que la trame de transmission est terminée.

La communication Bluetooth sous QT Android

Les classes nécessaires au fonctionnement de la communication Bluetooth

Dans l'application développée une classe *Btport* est créée pour instancier toute la communication Bluetooth. Celle-ci utilise différentes classes, présentées par la suite, et qui, pour être utilisées, doivent être ajoutées à l'intérieur du fichier des propriétés de l'application grâce à la ligne :

```
QT       += core gui bluetooth
```

Figure 6 : Commande nécessaire à l'utilisation des classes de Bluetooth

Les classes utilisées pour développer la communication Bluetooth sont les suivantes :

La classe QBluetoothSocket

Cette classe permet d'autoriser ou de rejeter une connexion du périphérique Bluetooth à un autre périphérique. Elle créée un socket supportant deux protocoles de communication : le L2CAP, non supportée par les systèmes Android et BlackBerry, et le RFMCOMM permettant de simuler une liaison série de type RS-232 et supportée par les systèmes Android. Il s'agit du protocole mis en place sur l'application.

La classe QBluetoothDeviceDiscoveryAgent

Cette classe est utilisée pour la découverte de périphériques Bluetooth. Pour utiliser correctement cette classe, il est nécessaire d'en créer une instance et de la connecter aux signaux *deviceDiscovered()* et *finished()* avant de débuter une recherche à l'aide de la méthode *start()*.

La classe QBluetoothDeviceInfo

Cette classe permet d'accéder au contenu des informations d'un périphérique Bluetooth particulier et de les enregistrer dans différentes variables.

La classe Btport

Comme dit précédemment il s'agit de la classe utilisée dans l'application pour permettre toute la communication Bluetooth.

Son constructeur, *Btport(_infos *info)* permet d'initialiser toutes les variables utiles à la communication. La fonction appelle la variable *info qui permet à l'application de savoir si une

connexion automatique du Bluetooth a été réalisée avec le périphérique RN42 relié au microcontrôleur et pour enregistrer les informations d'identification du périphérique.

Les variables de la classe Btport

Les principales variables de la classe Btport sont booléennes et permettent de connaitre l'état actuel de la connexion.

- *bool bBtScanRunning* : Permet de savoir si une recherche de périphériques est en cours.
- *bool bBtScanFinished* : Indique si une recherche de périphériques est terminée.
- *bool bBtConnected* : Permet de savoir si le périphérique est connecté.
- *bool bBtConnectionPending* : Indique si une tentative de connexion est en cours.
- *bool bBtBusySending* : Indique si le périphérique est en train d'envoyer des données.

- *QBluetoothDeviceDiscoveryAgent * BtDiscoveryAgent* : Instance de la classe *QBluetoothDeviceDiscoveryAgent* permettant de récolter les informations d'un périphérique Bluetooth.
- *QBluetoothSocket * BtSocket* : Instance de la classe *QBluetoothSocket* permettant un transfert de données via une communication Bluetooth.

Ces informations sont par la suite utiles pour effectuer les différentes fonctions d'écriture, de lecture ou d'erreur par exemple.

Durant la construction de la classe toutes les variables booléennes sont initialisées à *false*.

Les fonctions gérant la connexion du Bluetooth

Le scan des périphériques Bluetooth allumés

Cette fonction utilise le booléen *bBtScanRunning* permettant de savoir si un scan est déjà en cours. Si ce n'est pas le cas, la variable est mise à jour pour indiquer qu'une recherche débute. Une instance de la classe *QBluetoothDeviceDiscoveryAgent* et le code de la figure n° 7 sont alors utilisés pour obtenir les informations des différents périphériques découverts par le scan.

```
BtDiscoveryAgent = new QBluetoothDeviceDiscoveryAgent();
connect(BtDiscoveryAgent,SIGNAL(deviceDiscovered(QBluetoothDeviceInfo)),this,SLOT(BtDiscoveryAgentDeviceDiscovered(QBluetoothDeviceInfo)));
connect(BtDiscoveryAgent,SIGNAL(finished()),this,SLOT(BtDiscoveryAgentFinished()));
BtDiscoveryAgent->start();
```

Figure 7 : Utilisation de la classe QBluetoothDeviceDiscoveryAgent

Les lignes de connexion permettent ici de relier les signaux *deviceDiscovered(QBluetoothDeviceInfo)* et *finished()* aux slots *BtDiscoveryAgentDeviceDiscovered(QBluetoothDeviceInfo)* et *BtDiscoveryAgentFinished()*.

– Le slot BtDiscoveryAgentDeviceDiscovered(QBluetoothDeviceInfo) :

Ce slot va permettre la récupération des informations d'un périphérique Bluetooth. En l'occurrence, on s'attache ici à récupérer le nom du périphérique et son adresse.
Une fois ces informations récupérées, le signal *BtNewClient* peut être émis manuellement à l'aide de la commande *emit* pour indiquer qu'un nouveau périphérique Bluetooth a été découvert et pour que celui-ci s'affiche comme dans l'application à l'intérieur d'une page de débug.

- Le slot BtDiscoveryAgentFinished() :

Ce slot permet de mettre à jour les variables booléennes *bBtScanRunning* et *bBtScanFinished*. Le slot va aussi émettre le signal *BtScanFinished()* pour indiquer sur la page de débug que la recherche est terminée.

Fonction de connexion à un périphérique donné

La fonction prend en entrée une variable *QString name* qui va permettre de réaliser la connexion Bluetooth entre la tablette et le périphérique désigné par *name*.

Un premier if sur le booléen *bBtConnectionPending* est utilisé pour savoir si une tentative de connexion est déjà en cours de. Si c'est le cas, la fonction effectue un return pour ne pas empêcher la tentative déjà en cours.

Un second if sur *bBtConnected* est effectué pour savoir si une connexion a été réalisée. Si c'est le cas, la méthode *close()* de *BtSocket* va fermer toute connexion liée à *BtSocket*. Une déconnexion de tous les signaux et slots est aussi réalisée en utilisant la méthode *disconnect(BtSocket, 0,0,0)*. Le but est de réinitialiser le socket pour établir une nouvelle connexion.

Une fois ces deux étapes effectuées l'adresse liée au nom du périphérique est récupérée pour débuter la connexion.

```
bBtConnectionPending=true;
BtSocket = new QBluetoothSocket(QBluetoothServiceInfo::RfcommProtocol);
BtSocket->connectToService(QBluetoothAddress(adresse),QBluetoothUuid(QBluetoothUuid::Rfcomm));
connect(BtSocket,SIGNAL(connected()),this,SLOT(BtConnected()));
connect(BtSocket,SIGNAL(error(QBluetoothSocket::SocketError)),this,SLOT(BtErrorBt(QBluetoothSocket::SocketError)));
qDebug() << "Etat : " << BtSocket->state();
Info->Bt_connection_ok=true;
```

Figure 8 : Connexion du socket au périphérique Bluetooth désigné

Celle-ci débute par la création d'une instance d'un nouveau *QBluetoothSocket* et continue avec un appel à la méthode *connectToService* qui est utilisé pour se connecter au périphérique lié à l'adresse

adresse et en choisissant, comme dit précédemment, le protocole Rfcomm pour permettre la communication Bluetooth.

Les connect permettent ici de lier les signaux *connected()* et *error(QBluetoothSocket ::SocketError)* du socket aux slots *BtConnected()* et *BtErrorBt*(*QBluetoothSocket ::SocketError*).

-Les signaux :

Ici, les signaux sont directement émis par l'instance *BtSocket* lorsqu'une connexion est établie ou lorsqu'une erreur est repérée. Ils n'ont donc pas besoin d'être émis manuellement à l'aide la commande *emit*.

- Les slots :

Le slot *BtConnected*() permet d'indiquer que la connexion est réussie en mettant à jour les booléens concernés. Ce slot va aussi créer la connexion permettant la lecture des données de *BtSocket* avec le slot de lecture *BtDataRead()* et émettre le signal indiquant que la connexion Bluetooth a bien pu être réalisée.

Ici, le slot *BtErrorBt(QBluetoothSocket ::SocketError)* va indiquer que la connexion a échoué en ré-initialisant les booléens, en libérant *BtSocket* et en émettant le signal *BtConnectionFail()* pour afficher dans la page de débug l'échec de connexion.

L'écriture et la lecture

L'envoi de données via une application peut par exemple se faire à l'ouverture d'une page ou lors de l'appui d'un bouton. Dans l'application développée ici, cet envoi se fait au travers d'un signal nommé *SendMessage(Qstring data)*. Ce signal doit être connecté au slot *BtSendMessage(QString msg)* définit au niveau de la classe Btport. La connexion est réalisée à l'aide d'un connect lors de l'ouverture de chaque page permettant la transmission de la chaine *data* par le protocole de communication.

Le slot *BtSendMessage(QString msg)* va permettre de prendre le premier caractère de la chaine *msg* à envoyer si et seulement si aucun autre envoi est en cours et de le transformer au format UTF-8. Ce codage est utilisé pour permettre l'envoi d'un caractère sur un seul octet. En effet, sans cette transformation, les chaines de caractères envoyées pourraient contenir des caractères Unicode qui ne seraient peut-être pas au format ASCII utilisé par le microcontrôleur. Le passage au format UTF-8 permet d'associer à tout caractère de l'Unicode une représentation sous forme d'octet du même caractère compréhensible par le microcontrôleur.
Une fois cette transformation effectuée, le caractère est envoyé sur le socket pour l'envoi seulement si les caractères ne sont pas supérieurs au caractère '1'. Si ce caractère fait parti de la chaine d'envoi il agira comme une temporisation. Cela permet lors de l'envoi de grande chaine de caractères d'être sûr que le flux de données puisse être reçu et traité par le microcontrôleur.

La réception de données est créée à partir du slot *BtDataRead()*. Dans la partie Fonction de connexion à un périphérique donné, il avait été vu que le slot *BtConneted()* permettait la connexion du signal *readyRead()* et du slot *BtDataRead()*. C'est ce second slot qui permet la lecture des données reçues par le périphérique Bluetooth.
En effet, ce slot va lire la chaine de caractères contenue dans le socket lors d'une reception. Une fois cette lecture faite, le signal *BtDataReceived(QString)* va envoyer la chaine de caractère lue au slot

NewData(QString) utilisée dans différentes pages de l'application pour que cette chaine de caractères puisse agir sur un affichage de boutons ou puisse afficher un message d'erreur par exemple.

Finalement, la communication Bluetooth développée sous QT Android et son interaction interne peut être résumée comme ceci, les blocs noirs pour les classes et fonctions, les oranges pour les signaux les bleus pour les slots :

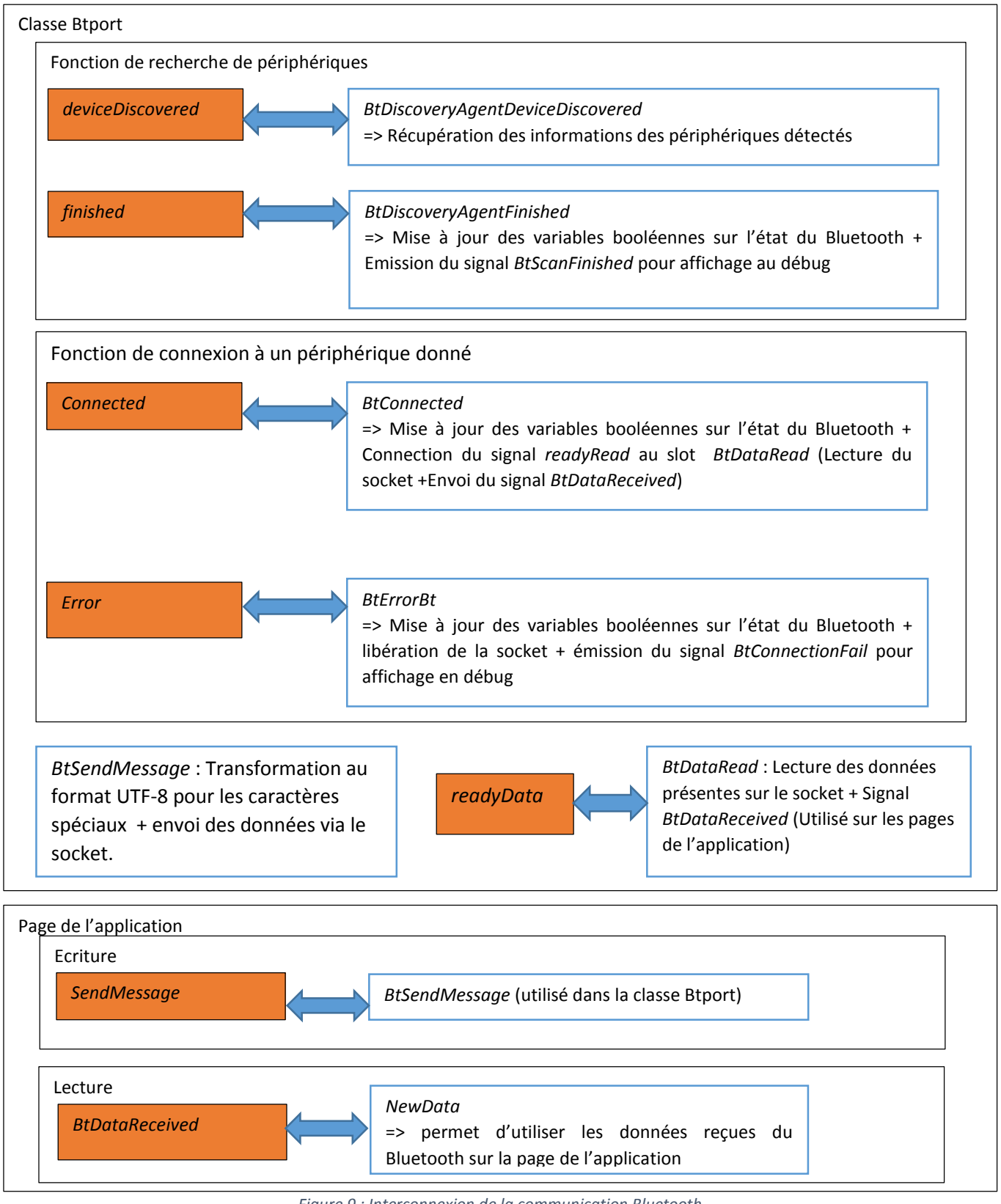

Figure 9 : Interconnexion de la communication Bluetooth

Conclusion

Cette note d'application a permis d'expliquer la génération d'une communication Bluetooth sous QT Android. Celle-ci utilise trois classes principales permettant :

- la détection de périphériques,
- la récupération d'informations les concernant
- et la connexion à un périphérique.

L'utilisation des sockets s'avère utile pour simuler une communication série permettant ensuite à l'application de communiquer via ces sockets. Les signaux et les slots quant à eux permettent entre autre de passer les différentes étapes de la configuration et de gérer l'affichage de l'application lors de la réception ou l'envoi de données.

Le protocole rapidement présenté en première partie et la communication développée sous QT Creator permettent alors d'utiliser une application Android en lien avec un système électronique.