CAPÍTULO 7.
LA LIBRERÍA QT.

Para la programación tanto de los Waspmotes, como de la GUI utilizada para monitorizar los datos provenientes de los mismos, WaspMonitor, se ha utilizado el lenguaje de programación C/C++. Sin embargo, en la programación de la interfaz gráfica de usuario y de toda la parte que corresponde al PC (lectura desde el puerto USB, etc.), se ha empleado una librería C++ denominada Qt y otras derivadas de ésta, como son Qextserialport o Qwt.

Lo que diferencia a Qt de una librería C++ cualquiera es que añade muchísimas funcionalidades a C++, cambiándolo de tal forma, que prácticamente crea un nuevo lenguaje de programación. Además, facilita la tarea de programar en C++, que en casos como en la programación de entornos gráficos, puede ser bastante ardua.

No obstante, es importante destacar que Qt no deja de ser C++, es decir, siempre se pueden usar librerías estándar o cualquier otra libería y la sintaxis de C++ normal y corriente, por lo cual, es muy versátil.

Por otra parte, existen *bindings* de Qt para que programadores de otros lenguajes de programación puedan utilizar las librerías sin tener que dejar de usar su lenguaje habitual. Ejemplos de estos *bindings* son Qt Jambi (Java), PyQt (Python), PHP-Qt (PHP) o Qyoto (C#), entre otros muchos.

Otro punto clave de Qt es que se considera una biblioteca "multiplataforma", ya que permite programar el mismo código y utilizarse para crear programas para Linux, Windows, Mac OS, etc., permitiendo realizar lo mismo que Java, pero siendo mucho más eficiente al no haber "máquina virtual" de por medio, sino código compilado expresamente para cada máquina.

Puesto que Qt se creó y creció como software libre (aunque en la actualidad hay disponible una versión comercial alternativa), existen muchísimas comunidades de programadores que crean librerías extraordinariamente útiles, como son el caso de Qextserialport o Qwt, y que al ser de código abierto, se pueden utilizar libremente y de manera gratuita.

Qt fue desarrollada inicialmente por la empresa noruega Trolltech (fundada por Haavard Nord y Eirik Chambe-Eng, dos estudiantes del Instituto Noruego de Tecnología) y posteriormente comprada por Nokia en 2008. Está disponible en tres tipos de licencia, GNU LGPL, GNU GPL y Propietaria (Nokia) y se ha utilizado para desarrollar software de todos los ámbitos como por ejemplo: Skype, Mathematica, Google Earth, Adobe Photoshop Album, etc.

En la fecha de realización del proyecto, la biblioteca va por la versión Qt 4.8.

Por todo lo expuesto, y mucho más, en el presente proyecto se ha dedicado un capítulo en el que se explican de manera breve las características generales de esta potente herramienta de programación.

7.1. El modelo de objetos Qt.

Qt se basa en el modelo de objetos Qt. Esta arquitectura es la que hace que Qt sea tan potente y fácil de usar. Los dos pilares de Qt son la clase *QObject* (objeto Qt) y la herramienta MOC (compilador de metaobjetos).

La programación mediante el modelo de objetos Qt se fundamente en derivar las clases nuevas que se creen de objetos *QObject*. Al hacer esto, se heredan una serie de propiedades que caracterizan a Qt (y lo hacen especial frente al C++ estándar):

- Gestión simple de la memoria.
- Signals y slots.
- Propiedades.
- Auto-conocimiento.

No hay que olvidar, sin embargo, que Qt no deja de ser C++ estándar con macros, por lo que cualquier aplicación "programada en Qt", puede tener código en C++ estándar o incluso en C.

De estas características, se comentarán en los siguientes apartados las dos primeras, por ser las empleadas principalmente en el proyecto y en general en cualquier aplicación escrita en Qt.

7.1.1. Gestión simple de la memoria.

Cuando se crea una instancia de una clase derivada de un *QObject* es posible pasarle al constructor un puntero al objeto padre. Esta es la base de la gestión simple de memoria.

Al borrar un objeto padre, se borran todos sus objetos hijos asociados. Esto quiere decir que una clase derivada de *QObject* puede crear instancias de objetos "*hijos-de-QObject*" pasándole el puntero *this* como padre sin preocuparse de la destrucción de estos hijos.

Esto es una gran ventaja frente al C++ estándar, ya que en C++ estándar cuando se crea un objeto siempre hay que tener cuidado de destruir los objetos uno a uno y de liberar memoria con la sentencia *delete*.

Figura 7.1: Ejemplo de QWidget padre con hijos.

Un ejemplo de aplicación podría ser por ejemplo crear un *QWidget* ventana (objeto padre) con dos botones *QPushButton* y una línea de texto editable *QLineEdit* (hijos de esta ventana). Esto se puede ver en la figura 6.1. *QWidget* es una clase derivada de *QObject* que permite representar objetos gráficos como ventanas, botones, etc. Por tanto, a su vez, *QPushButton* y *QLineEdit* derivan de QWidget.

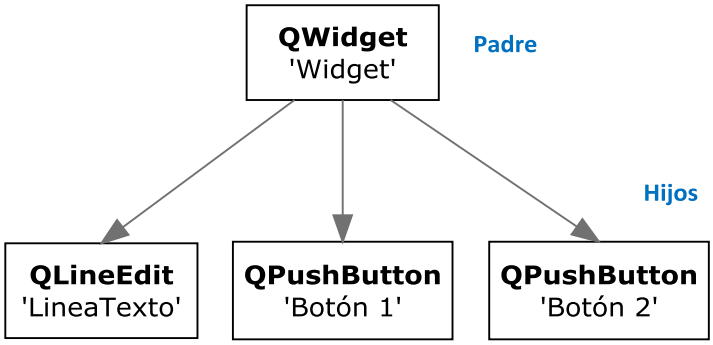

Figura 7.2: Jerarquía de memoria del QWidget de la figura 6.1.

Al destruir la ventana se destruirían los hijos liberando la memoria correspondiente de manera transparente para el programador. La jerarquía de memoria se puede ver en la figura 6.2. Cabe destacar que de cada *QObject* también podrían crearse objetos hijos, aumentando el árbol de memoria. Estos objetos "nietos" del padre original se eliminarían al eliminarse su objeto padre.

Nótese que la terminología de hijos y padres no debe confundirse con la utilizada en los procesos, por ejemplo. Cuando se habla de padres e hijos, nos estamos refiriendo a objetos Qt.

7.1.2. Signals y slots.

Las *signals* y los *slots* (señales y ranuras, en castellano) son lo que hace que los diferentes componentes de Qt sean tan reutilizables.

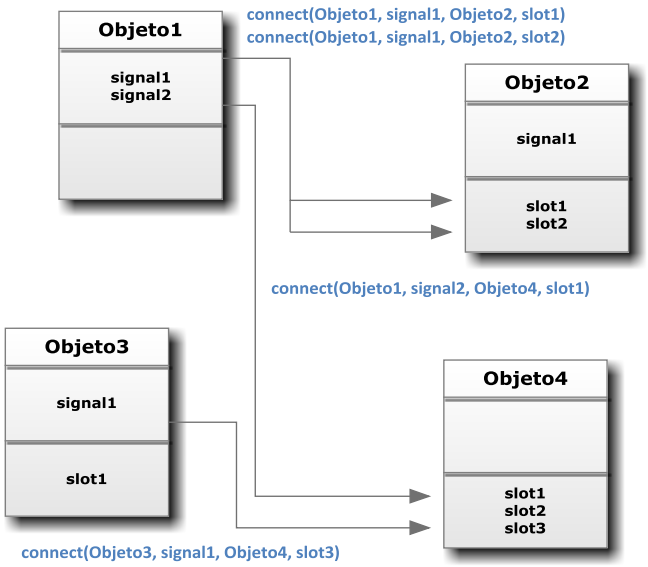

Figura 7.3: Conexiones entre signals y slots de diferentes objetos.

Proporcionan un mecanismo a través del cual es posible exponer interfaces que pueden ser interconectadas libremente (figura 6.3). Para realizar la interconexión se usa el método "*connect*".

Un ejemplo de una *signal* podría ser pulsar un botón *QPushButton* y la *signal* que se emite cuando se pulsa, *clicked(),* y un *slot* podría ser por ejemplo, escribir un texto en un *QLineEdit*.

La ventaja principal de las *signals* y de los *slots* es que el *QObject* que envía la *signal* no tiene que saber nada del *QObject* receptor. Este tipo de programación se llama en inglés "*loose coupling*" y permite integrar muchos componentes en el programa de manera sencilla y minimizando la probabilidad de fallo.

Para poder usar las *signals* y los *slots* cada clase tiene que declararse en un fichero de cabecera y la implementación se sitúa en un archivo '.cpp' por separado. Este archivo de cabecera se pasa entonces a través de una herramienta conocida como el MOC (compilador de metaobjetos). El MOC produce un '.cpp' que contiene el código que permite que funcionen las s*ignals* y los *slots* y otras muchas características de Qt. Para realizar este proceso de compilación que puede parecer complejo, se utiliza la herramienta de qmake de Qt que lo realiza de manera automática. De estas herramientas se hablará en el apartado 6.2.

En la figura 6.4 se puede ver el flujo desde que se crea una clase derivada de *QObject* hasta que se compila:

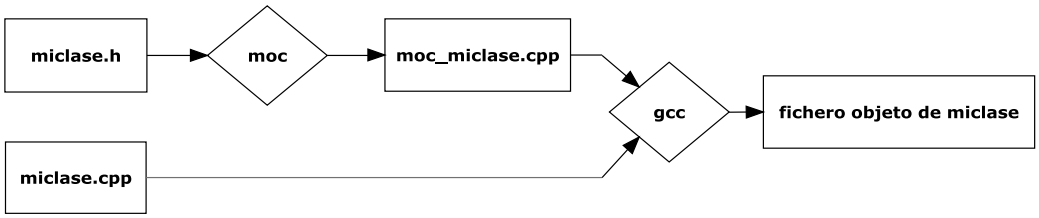

Figura 7.4: El flujo del MOC.

Aun así, como ya se mencionó en la introducción de este capítulo, una aplicación Qt sigue siendo C++ al 100%. Por tanto, las s*ignals* y los *slots* son simplemente palabras reservadas o *keywords* que reemplaza el preprocesador convirtiéndolas en código C++ real.

Los *slots* se implementan como cualquier método miembro de la clase mientras que las *signals* las implementa el MOC. Cada objeto tiene una lista de sus conexiones (qué *slots* se activan con qué *signals*) y sus *slots* (cuáles se usan para construir la tabla de conexiones en el método *connect*). Las declaraciones de estas tablas están escondidas en la macro *Q_OBJECT*. Esto se puede ver en el código de la figura 6.5.

```
class MiObjeto : public QObject
{
  Q_OBJECT
public:

  MiObjeto( QObject *parent=0) : QObject( parent)
  {
    // . . .
  }

public slots:
  void miSlot( void )
  {
    // . . .
  }
signals:
  void miSignal();
};
```

Figura 7.5: Código fuente genérico de un QObject (objeto Qt).

7.2. Las herramientas de Qt.

Las herramientas principales que permiten funcionar a Qt son las siguientes:

- Variables de entorno.
- Qt Creator.
- Qmake.
- UIC.
- MOC.

7.2.1. Variables de entorno.

Para poder usar Qt es necesario que se configuren tres variables: *QTDIR*, *PATH* y *QMAKESPEC*.

- *QTDIR*. Variable que indica el directorio que contiene la distribución de Qt, en el caso actual la 4.8. para permitir la compilación de Qt.
- *PATH*. Variable que se usa para poder invocar a las herramientas Qt desde la línea de comandos.
- *QMAKESPEC*. Variable que contiene la ruta del directorio mkspecs, necesario para compilar nuestro programa.

Si se usa Qt Creator, estas variables se configuran de manera automática en la instalación, sin embargo es posible configurarlas a mano si se desea compilar externamente al IDE.

7.2.2. Qt Creator.

Es un IDE (Entorno Integrado de Desarrollo) multiplataforma creado inicialmente por Trolltech y mantenido en la actualidad por Nokia. Requiere la versión de las bibliotecas Qt 4.x en adelante. Integra un editor de texto, un depurador y Qt Designer.

Qt Designer es una herramienta muy potente (antes era un programa independiente), utilizada en varias partes de este proyecto, que permite crear "widgets" o elementos gráficos de manera visual, facilitando en gran medida la creación de código para la parte de la GUI. Tras diseñar de manera gráfica un widget, se genera un archivo con extensión '.ui', que no es más que un archivo XML que posteriormente se traduce a una clase en C++ mediante el UIC.

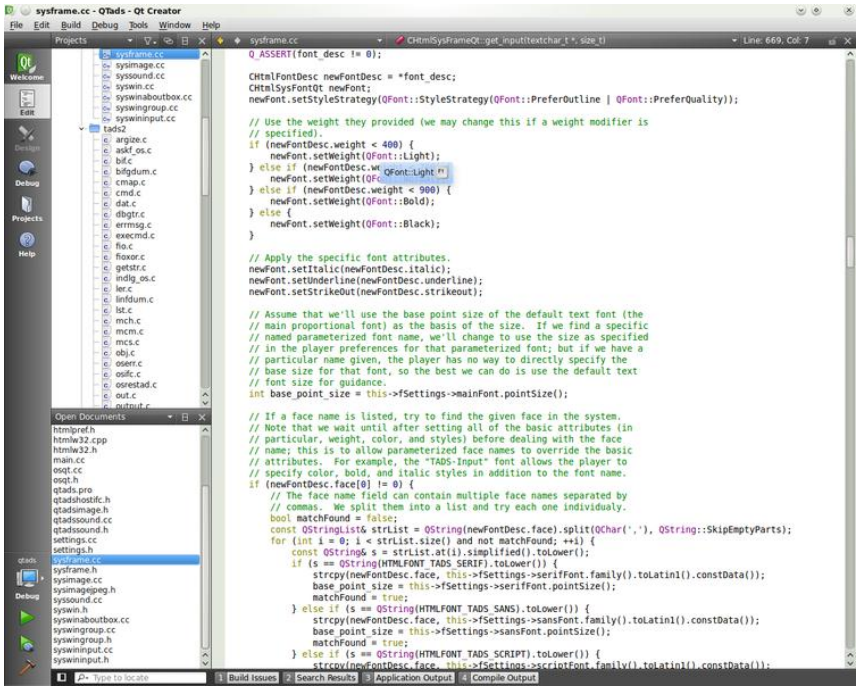

Figura 7.6: Qt Creator en su versión Linux.

7.2.3. Qmake.

Puesto que las aplicaciones hechas con Qt dependen de las librerías Qt, sus propias librerías y en algunos casos, librerías de terceros (como es el caso de Qwt o Qextserialport) y por otra parte, intenta ser totalmente portable, los "makefiles" pueden convertirse en ficheros tremendamente complejos. Por si fuera poco, no sólo se compila código fuente en C++, sino que el MOC introduce un paso intermedio adicional de precompilación y encima, los archivos '.ui' (Interfaces de Usuario o widgets) pueden compilarse a partir de archivos XML a clases C++.

Qmake tiene el objetivo de simplificar todo este proceso de generación de código, que con la herramienta "make" clásica, sería tremendamente complejo.

Afortunadamente, si se utiliza el IDE Qt Creator, se generan unos archivos '.pro' (archivos de descripción de proyecto) que Qmake interpreta y permite generar todo el código para compilarlo posteriormente con gcc con pulsar sólo un botón.

7.2.4. UIC.

UIC (User Interface Compiler) o compilador de interfaces de usuario traduce los ficheros '.ui' generados por Qt Designer (dentro de Qt Creator) a clases C++. Qmake configura esta herramienta de manera automática.

Para poder utilizar una clase proveniente de un archivo '.ui' en el código Qt, los desarrolladores recomiendan especialmente un método denominado "método de herencia múltiple", que consiste en crear una clase que herede de QWidget (lo habitual) y además herede de Ui::NombreForm (donde NombreForm será el nombre del archivo '.ui' también llamado *form*). Con esto se consiguen dos cosas, por una parte se puede acceder a todos los componentes del *form* desde el ámbito de la subclase y por otra parte, se pueden usar *signals* y *slots* de forma habitual. Para que todo esto funcione hay que añadir con un #include "ui_nombreform.h" (el cual será el archivo de cabecera correspondiente al fichero '.ui' generado por el UIC). Este método tan simple nos permitirá usar código hecho "a mano" y código generado por el UIC a partir de Qt Designer.

7.2.5. MOC.

MOC (Meta Object Compiler) o compilador de metaobjetos crea metaobjetos que describen las clases haciendo uso de las *signals* y los *slots*.

7.3. Clases de Qt.

El número de clases de Qt es numerosísimo, por tanto, en este apartado tan sólo se van a describir algunas de las utilizadas en el proyecto.

- ***QObject***: Es la clase más importante de todas, ya que describe el objeto básico Qt, que permite usar la gestión simple de memorias o las *signals* y *slots*.
- ***QWidget***: Es la clase que permite crear el elemento básico de interfaz de usuario.
  - ***QMainWindow***: Widget de tipo MainWindow (ventana principal de programa).
  - ***QDialogBox***: Widget de tipo cuadro de diálogo estándar.
  - ***QLabel***: Widget que permite mostrar etiquetas de texto o imágenes.
  - ***QLineEdit***: Widget que muestra una línea de texto.
  - ***QTextEdit***: Similar a QLineEdit pero permitiendo mostrar texto, incluso en formato HTML.
  - ***QPushButton***: Widget que genera un botón.
  - ***QFrame***: Widget de tipo marco.
  - ***QMessageBox***: Widget de tipo Message Box.
- ***QLayout***: Es la clase que gestiona la geometría de los widgets.
  - ***QBoxLayout***: Alinea los widgets de manera vertical u horizontal.
- ***QThread***: Clase que gestiona hilos de ejecución independientes del SO que se esté usando.
- ***QTimer***: Clase que proporciona una interfaz de alto nivel para los timers.

7.4. Librerías de enlazado dinámico necesarias para ejecutar una aplicación Qt.

La mayoría de las aplicaciones Qt, se compilan con enlazado dinámico. Es decir, son necesarios unos archivos '.dll' en caso de Windows, o '.lib' en caso de POSIX para poderse ejecutar.

En el caso de Windows, para poder ejecutar una aplicación Qt son necesarias las siguientes '.dll' en el mismo directorio o en el directorio "system32":

- ***QtCore4.dll***: Librería correspondiente al núcleo de Qt 4.
- ***QtGui4***: Librería correspondiente a la parte gráfica de Qt (GUI).
- ***Libgcc_s_dw2-1.dll*** y ***mingwm10.dll***: Librerías correspondientes al compilador.

Otra opción es realizar un enlazado estático y se engloba todo en un archivo, reduciendo un poco el tamaño, pero se ha descartado el proceso por no considerarse necesario.

Por otra parte, no hay que olvidar que en el caso de utilizar otras librerías dinámicas externas, también hay que incluir las correspondientes '.dll', como son el caso de Qwt o QextSerialPort.

7.5. Qwt.

Qwt (Qt Widgets for Technical Applications) es una librería que contiene componentes para desarrollo de interfaces gráficas de usuario para aplicaciones técnicas. Es ampliamente utilizado en el ámbito de la automatización y la ingeniería en general.

En el proyecto se ha utilizado la versión Qwt 6.0.1, compatible con versiones de Qt superiores a la 4.4. Qwt se distribuye con "Licencia Qwt, versión 1.0", derivada de la LGPL.

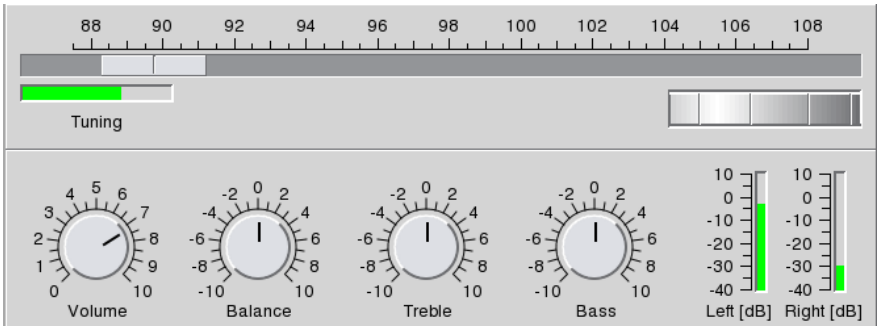

Figura 7.7: Radio implementada con diales, sliders y termos Qwt.

Contiene una lista de *widgets* Qt entre los que se incluyen elementos de control y de monitorización:

- **Gráficas en 2D**: histogramas, espectrogramas, Bodes, etc.
- **Manejo de escalas**.
- **Sliders** (barras deslizantes).
- **Diales y ruedas.**

- **Brújulas.**
- **Termómetros.**

En la página oficial de Qwt en Sourceforge[27], se dispone de documentación con todas las clases que compone la librería y en foros como Qt Centre [28] donde existen partes dedicadas a esta librería y a otras muchas relacionadas con Qt, se puede solicitar ayuda a la comunidad de programadores de software libre.

En este proyecto, las dos clases bases de Qwt que se han utilizado son dos:

- ***QwtThermo***: Widget que sirve marca valores en una barra en un intervalo determinado (similar a un termómetro). Permite configurar la escala, el rango de valores y si el layout (disposición del widget) es vertical u horizontal. Por otra parte, la parte gráfica también es configurable (color, anchura de la barra, etc.)
- ***QwtPlot:*** Widget que sirve para dibujar gráficas en 2D. Se pueden añadir curvas 'QwtPlotCurve', marcadores 'QwtPlotMarker', rejillas 'QwtPlotGrid', etc.

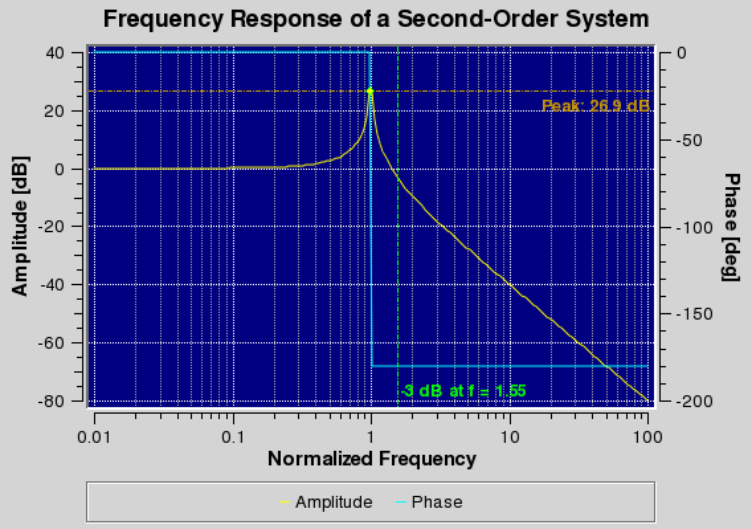

Figura 7.8: Ejemplo de QwtPlot con un diagrama de Bode.

Es importante destacar, que estas son las clases base, pero después son necesarias muchas otras clases para poder configurar el widget según la necesidad del programador. Por poner un ejemplo, un QwtPlot es configurable en gran medida, cambiándole etiquetas, curvas, fondos, escalas, etc, como se puede ver en el ejemplo de gráfica de un Bode de un sistema de segundo orden en la figura 6.8.

7.6. QextSerialPort.

QextSerialPort es la segunda librería externa derivada de Qt utilizada en el proyecto aparte de Qwt. Según la definen sus creadores [29] es una "librería multiplataforma para puerto serie". Se distribuye con licencia MIT, la cual es muy similar a la licencia BSD. Por tanto es software libre.

Las dos clases básicas de la librería son:

- ***QextSerialPort***: Encapsula un puerto serie tanto para sistemas POSIX (Unix, Linux, Mac OS, etc.) como para sistemas Windows.
- ***QextSerialEnumerator***: Enumera los puertos que están disponibles actualmente en el sistema.

En la figura 6.9 se puede ver el diagrama de herencia de la clase *QextSerialPort*. Por un lado se observa que existe herencia múltiple donde, uno de los "padres" correspondería al dispositivo de sistema POSIX y otro, al dispositivo de sistema Windows. No hay que olvidar que la lectura en un SO y en otro, se hace de manera diferente. Gracias al uso de Qt y QextSerialPort, se puede programar de manera transparente y multiplataforma.

QIODevice es una clase base de la librería Qt que engloba todos los dispositivos de Entrada/Salida, como pueden ser por ejemplo los archivos, los sockets TCP, o en este caso, el puerto serie. Es el pilar de QextSerialPort y es por ello por lo que se considera una librería derivada de Qt.

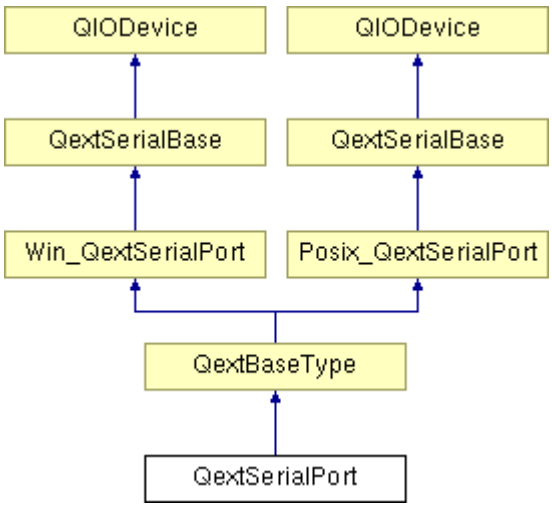

Figura 7.9: Diagrama de herencia de QextSerialPort.

En el proyecto, el "puerto serie" es en realidad un puerto serie virtual, implementado a través de un USB, un chip FDDi y unos drivers que sirven para que el puerto USB se comporte virtualmente como un puerto serie RS-232.

QextSerialPort permite programar la lectura y escritura del puerto serie por *polling* o por eventos. En el proyecto se probaron ambos métodos, pero se terminó optando por la programación por eventos, ya que el *polling*, aun realizándolo en un hilo separado de ejecución, hacía consumir entre un 30% y un 40% de una CPU de doble núcleo y 3 GHz. Con el método por eventos, cuando llegan o se escriben, se produce un evento, liberando la CPU el resto del tiempo.

Por otra parte, dispone de funciones muy útiles, para configurar el puerto, la lectura y la escritura, de manera mucho más cómoda y portable, que programándolo de forma directa.