CAPÍTULO 8.
WASPMONITOR.

Waspmonitor es la GUI (Graphical User's Interface) o Interfaz Gráfica de Usuario, desarrollada en el proyecto con el fin de monitorizar los mensaje recibidos desde los Waspmotes en la zona bajo estudio.

Este programa se ha programado íntegramente en C++, aunque parte de la interfaz gráfica se ha diseñado con Qt Designer (parte del IDE Qt Creator comentado en el capítulo 7). Se ha hecho uso de las librerías Qt, para el núcleo del programa y la interfaz gráfica general, Qwt, para las gráficas de monitorización y Qextserialport, para la parte de comunicación entre el ordenador y el Waspmote Gateway. Se ha desarrollado para sistemas Windows, aunque debido a la naturaleza de Qt, es fácilmente portable a sistemas POSIX (Linux, Mac OS, etc.)

Este capítulo se divide en dos partes: en la primera se describe el funcionamiento del programa y en la segunda parte se explica cómo se ha programado.

8.1. Descripción del programa.

La ventana principal de Waspmonitor se puede ver en la figura 7.1.

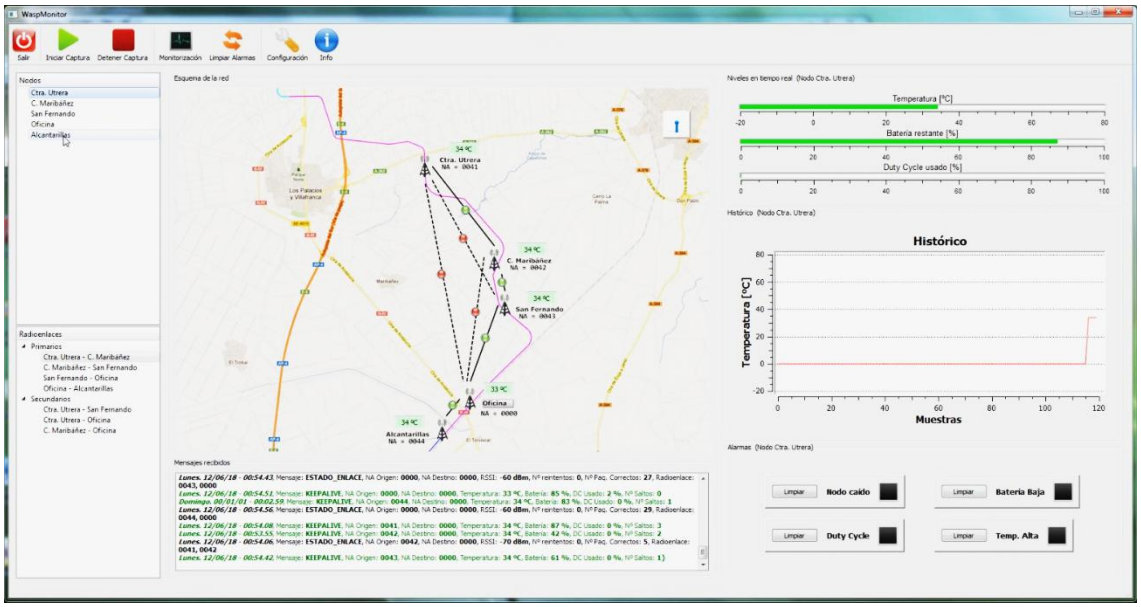

Figura 8.1: Ventana principal de Waspmonitor.

Las partes de esta ventana son las siguientes:

- Barra de herramientas.
- Lista de nodos y radioenlaces.
- Mapa del canal con los nodos, radioenlaces y las alarmas.
- Barras de medida.

- Histórico de medidas.
- Panel de alarmas.
- Ventana de log de mensajes recibidos.
- Barra de Estado.

8.1.1. Barra de herramientas.

La barra de herramientas se puede ver en la figura 7.2:

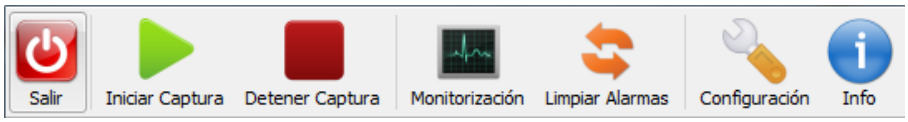

Figura 8.2: Barra de herramientas de Waspmonitor.

La función de los botones se describe en la siguiente tabla:

	**Salir.** Sale del programa, deteniendo la captura de tramas en caso de que esté en curso y cerrando el puerto serie.
	**Iniciar Captura.** Abre el puerto serie e inicia la captura de tramas desde el Waspmote Gateway.
	**Detener Captura.** Detiene la captura de tramas desde el Waspmote Gateway.
	**Monitorización.** Muestra la ventana principal de monitorización de Waspmonitor (figura 7.1)
	**Limpiar Alarmas.** Limpia los iconos de alarma del mapa y de los paneles de alarma de los distintos nodos y radioenlaces.
	**Configuración.** Abre la ventana de configuración de Waspmonitor.
	**Info.** Muestra la ayuda de Waspmonitor.

Tabla 8.1: Botones de la Barra de Herramientas.

8.1.2. Lista de nodos y radioenlaces.

Al pulsar sobre alguno de los ítems de estas listas que se ven en la figura 7.3, cambia el panel de monitorización al nodo o radioenlace determinado que sea. En concreto, cambian las barras de medida, el histórico de medidas y el panel de alarmas.

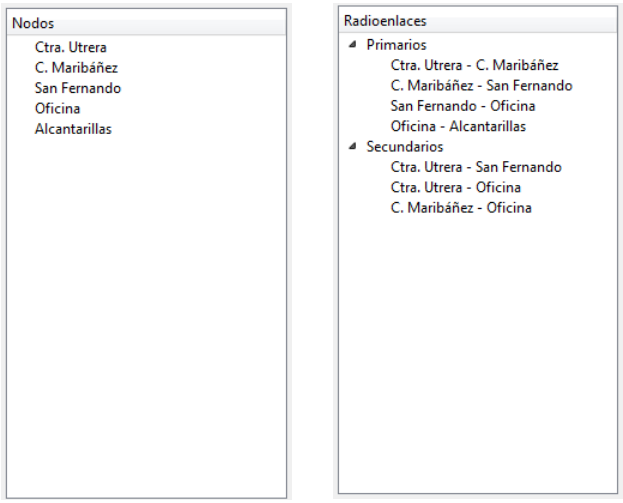

Figura 8.3: Lista de nodos y radioenlaces.

8.1.3. Mapa del canal.

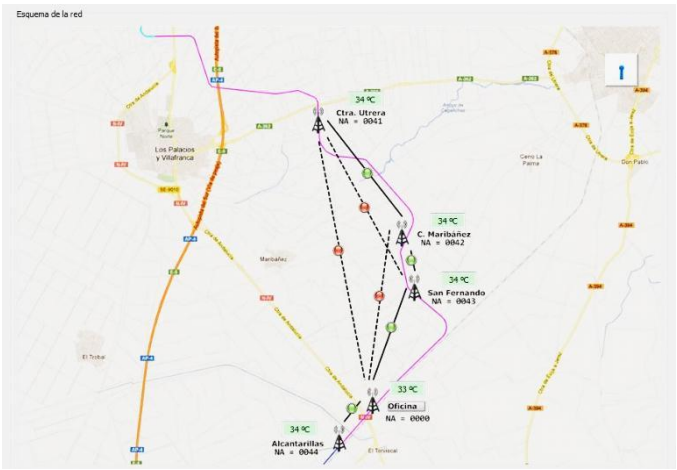

Figura 8.4: Mapa del canal.

En el mapa del canal se muestran diversos iconos y etiquetas que sirven para tener una visión global del canal y de la red e identificar de manera clara el estado de los nodos y radioenlaces y ver si ha saltado alguna alarma.

Los diferentes iconos que pueden aparecer en el mapa son los siguientes:

	**Iconos de estado de radioenlace.** Pueden tener 3 estados: - **Rojo**: Radioenlace caído. - **Amarillo**: Deshabilitada la monitorización del radioenlace. - **Verde**: Radioenlace activo.
	**Iconos de alarma.** Están asociados a los nodos e indican: - **Batería Baja.** - **Duty Cycle alto.** - **Temperatura alta.**

Tabla 8.2: Iconos del mapa.

Por otra parte, las etiquetas de temperatura cambian de color de igual forma que lo hacen los iconos de estado de los radioenlaces, tomando los colores rojo, amarillo y verde, cando están caídos, deshabilitados o activos, respectivamente.

Se muestra también un pequeño icono animado con forma de antena en la esquina superior derecha del mapa, indicando que se están capturando tramas en el Gateway.

8.1.4. Ventana de log.

En la ventana de log (figura 7.5), se muestran los mensajes de Keepalive, Estado de Enlace y Alarma, recibidos desde los nodos.

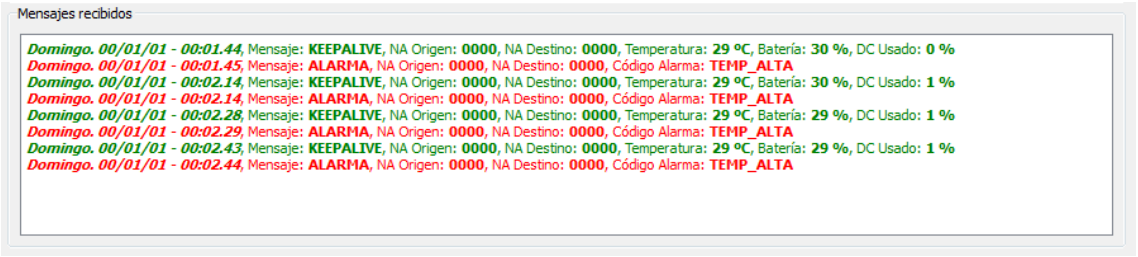

Figura 8.5: Ventana de log.

8.1.5. Barras de medida.

Hay dos tipos de barra de medida, una para los nodos (figura 7.6) y otra para los radioenlaces (figura 7.7). En el primer caso se muestran los valores de temperatura obtenida por el sensor de temperatura del RTC, el nivel de batería restante en el Waspmote y el Duty Cycle usado por el xBee. Las barras cambian de color, cuando se superan diferentes umbrales, como se puede observar en las figuras.

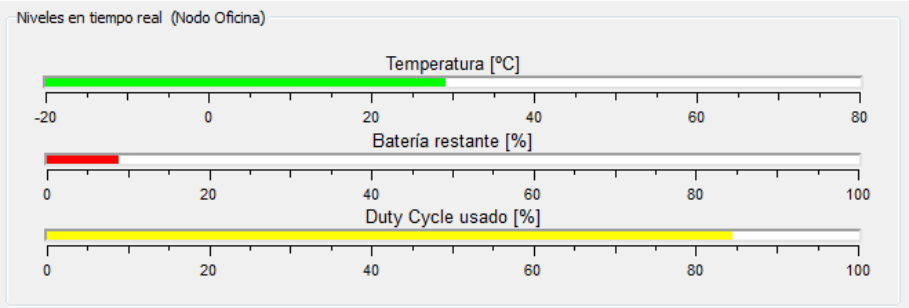

Figura 8.6: Barras de medida (Nodo).

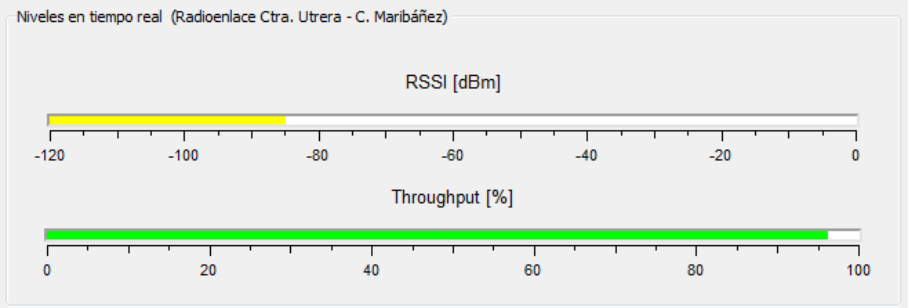

Figura 8.7: Barras de medida (Radioenlace).

8.1.6. Históricos de medidas.

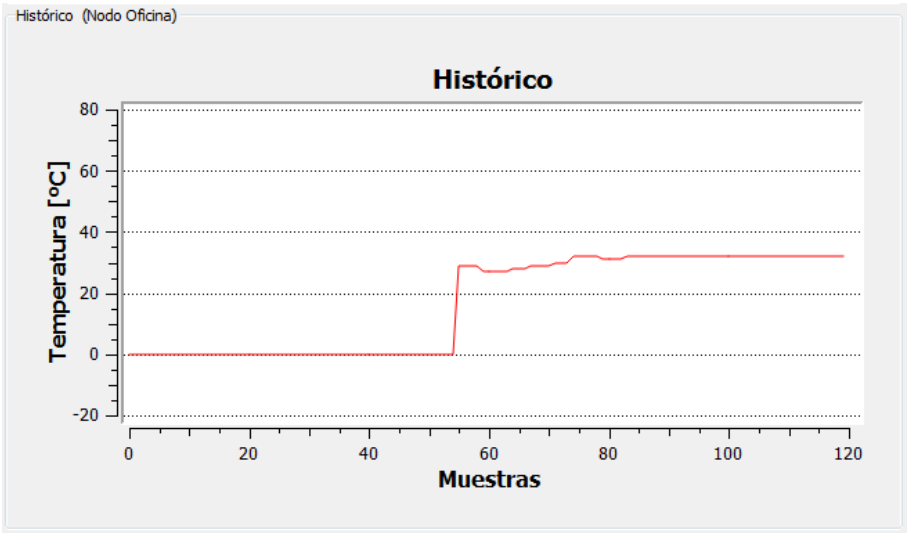

Figura 8.8: Históricos de medidas (Nodo).

Waspmonitor tiene dos históricos de medidas en los que se representan las últimas 120 muestras tomadas cada 30 segundos. Al igual que en el caso de las barras de medida, existe uno para nodos (figura 7.8) y uno para radioenlaces. En el caso de los nodos, los valores representados son las temperaturas medidas por el RTC y en el caso de los radioenlaces, los RSSI.

8.1.7. Paneles de alarma.

En el caso de los paneles de alarma, Waspmonitor también tiene paneles distintos para el caso de los nodos (figura 7.9) y de los radioenlaces (figura 7.10).

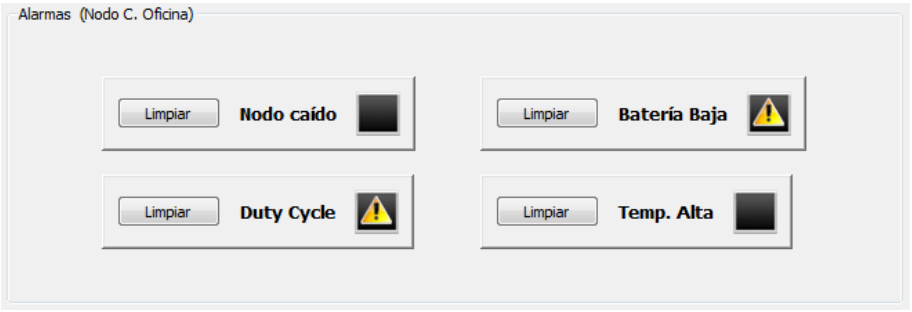

Figura 8.9: Panel de alarmas (Nodo).

Figura 8.10: Panel de alarmas (Radioenlace).

Las alarmas de los paneles de alarma son las siguientes:

- **Nodo caído**: Se indica cuando se recibe un mensaje de Alarma por Caída de Enlace. Al caer un enlace, el nodo de destino se supone que está caído. También se produce cuando se sobrepasa un tiempo de 90 segundos sin recibir ningún mensaje de Keepalive desde dicho nodo.
- **Batería Baja**: Se indica cuando se recibe un mensaje de Alarma por Batería Baja.
- **Duty Cycle**: Se indica cuando se recibe un mensaje de Alarma por Duty Cycle alto.
- **Temperatura alta**: Se indica cuando se recibe un mensaje de Alarma por Temperatura alta.
- **Radioenlace caído**: Se indica cuando se recibe un mensaje de Alarma por Radioenlace Caído. También se activa esta alarma cuando transcurren 90 segundos sin recibir ningún mensaje de Keepalive desde el nodo origen del radioenlace (se activará esta alarma en todos los radioenlaces asociados al nodo caído).

8.1.8. Configuración de Waspmonitor.

Al pulsar en el botón con un icono de llave inglesa se accede al menú de configuración de Waspmonitor. En la figura 7.11 se puede ver la ventana.

Figura 8.11: Ventana de configuración de Waspmonitor.

Antes de empezar a capturar ningún dato, se recomienda configurar Waspmonitor correctamente. En especial es fundamental seleccionar el puerto COM adecuado para el Waspmote Gateway. En caso de elegir uno erróneo, Waspmonitor indicará un error de apertura de puerto. También se recomienda deshabilitar la monitorización de los nodos que no se vayan a emplear, para evitar que salten alarmas falsas.

Las opciones de configuración son las siguientes:

- **Habilitación/deshabilitación de monitorización de nodos.**
  Deshabilitar la monitorización implica que Waspmonitor no hará saltar las alarmas si pasan 90 segundos sin recibir un mensaje de Keepalive desde uno de los nodos.
  Por defecto están habilitados todos los nodos.
- **Habiltación/deshabilitación de logs.**
  Waspmonitor puede crear archivos de log en formato .CSV (Comma-Separated Values) y HTML donde se guarden todos los mensajes recibidos desde los nodos.
  Por defecto están habilitados los logs en .CSV y en .HTML.
- **Selección de puerto COM para el Gateway.**
  Para que Waspmonitor pueda comunicarse con el Waspmote Gateway es necesario elegir el puerto serie COM adecuado.

Por defecto, el puerto serie elegido es COM1, pero es necesario elegir correctamente el puerto antes de iniciar la monitorización, ya que el puerto COM cambia a menudo al conectar y desconectar el Waspmote Gateway.

8.1.9. Cuadros de diálogo.

Waspmonitor dispone de diversos cuadros de diálogo que informan de errores, dan advertencias, etc. sobre eventos que ocurran durante la ejecución del programa. En la figura 7.12 se muestra como ejemplo el cuadro de diálogo que aparece cuando se produce un error crítico al intentar abrir el puerto serie e iniciar la captura de mensajes.

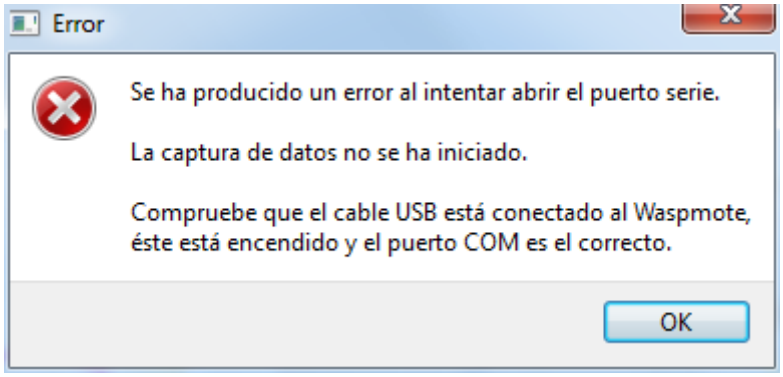

Figura 8.12: Cuadro de diálogo de Error.

Los diferentes cuadros de diálogo que pueden aparecer en Waspmonitor son estos:

- **Advertencias**:
  - **Proceso de captura ya iniciado**: Aparece si se pulsa más de una vez el botón *Iniciar Captura* ya que si la captura ya está en curso, no se puede volver a iniciar, hay que detenerla antes con el botón *Detener Captura*.
  - **Salir del programa:** Al pulsar el botón de Salir o la X de la ventana, aparece este cuadro de diálogo. Se detiene el proceso de captura y se cierra el puerto serie.
  - **No se puede cambiar el puerto mientras la captura está en curso**. Aparece al intentar cambiar el puerto COM en la ventana de configuración cuando la captura está iniciada.
  - **No se puede abrir el archivo de log CSV o HTML**. Aparecen si hay algún tipo de problema al abrir los archivos de log por ejemplo, porque el disco duro esté lleno.
- **Errores críticos**:
  - **Error al intentar abrir el puerto serie**. Este cuadro de diálogo se muestra si el puerto serie elegido en la ventana de configuración es incorrecto, si el Waspmote no está conectado o está conectado pero apagado. También puede aparecer si existe algún otro tipo de error al intentar abrir el puerto serie.

8.1.10. Barra de Estado.

La Barra de Estado es la zona inferior de la ventana principal de Waspmonitor en la que se muestran mensajes de carácter informativo cuando suceden distintos eventos o cambios en el programa.

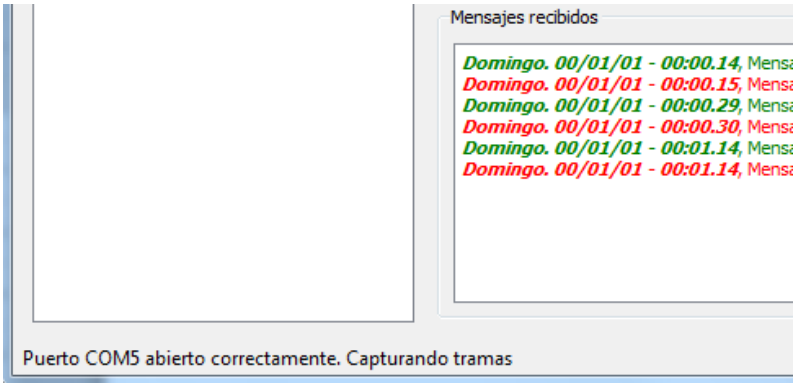

Figura 8.13: Barra de estado mostrando un mensaje informativo.

Los diferentes mensajes que se muestran son:

- **Mensajes informativos**. Puerto abierto/cerrado, captura iniciada, etc.
- **Mensajes de error**. Error al abrir el puerto, etc.
- **Mensajes de cambio de configuración**. Habilitación/deshabilitación de monitorización de nodos, etc.

8.1.11. Ayuda de Waspmonitor.

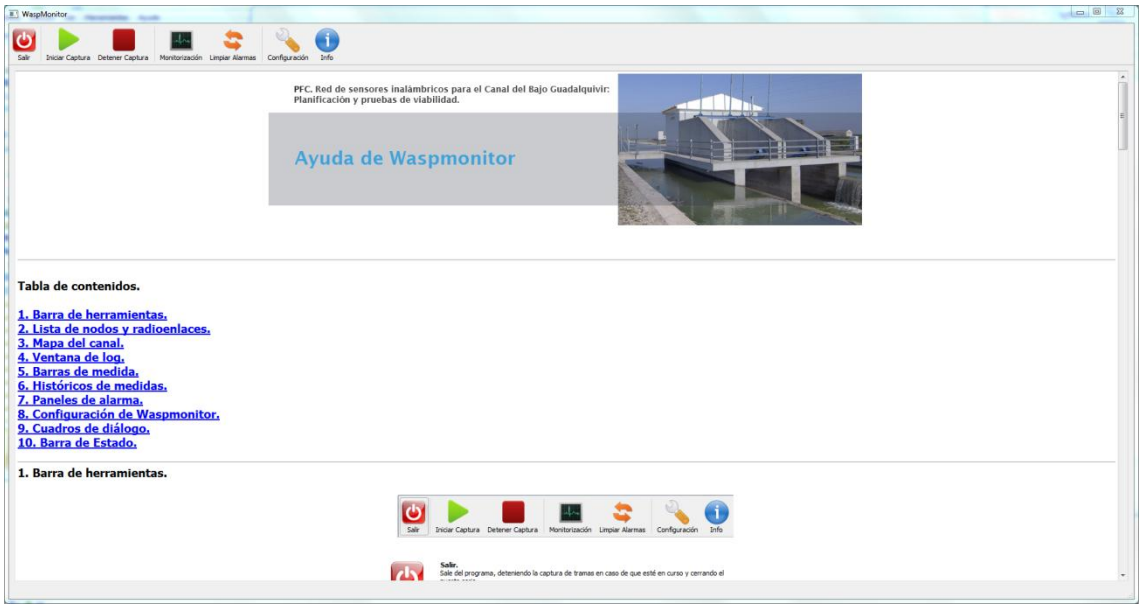

Figura 8.14: Ayuda de Waspmonitor.

En esta ventana se muestra la ayuda de Waspmonitor que es, más o menos, lo que está descrito en este apartado.

8.2. Programación de Waspmonitor.

En este apartado se explica cómo se ha programado Waspmonitor haciendo uso de las librerías Qt, Qwt y Qextserialport, así como su editor visual Qt Designer.

Waspmonitor cumple las siguientes tareas:

- Servir como interfaz gráfica (GUI) para las pruebas de campo en el canal.
- Servir como interfaz entre el PC y el Waspmote Gateway para poder recibir los mensajes de los sensores.
- Monitorizar y guardar los datos recibidos de los Waspmotes.

8.2.1. Diagrama de clases C++.

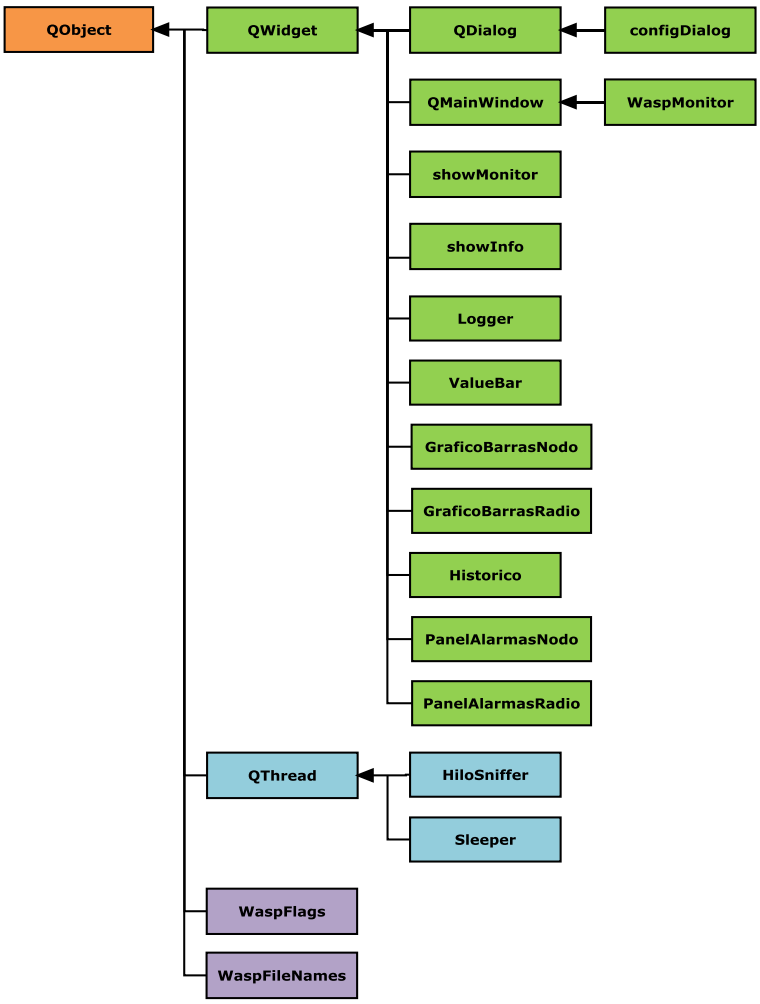

Figura 8.15: Diagrama de clases de Waspmonitor y herencia.

En la figura 7.15 se puede ver el diagrama de herencia de clases C++ utilizado en Waspmonitor. Como ya se introdujo en el apartado 6.3., donde se explicaban las clases de Qt, la clase base de todas es QObject. A la hora de implementar Waspmonitor se ha seguido de manera estricta el modelo Qt, para hacer un código lo más homogéneo posible. Es por ello que no se ha creado ninguna clase C++ "pura", si no que todas derivan de QObject, para que se mantuvieran las propiedades de Qt (manejo de signals y slots, gestión simple de memoria, etc.).

Básicamente hay tres grupos de clases:

- **Derivadas de QWidget**: Implementan la parte gráfica (GUI) del programa.
- **Derivadas de QThread**: Implementan la comunicación con el puerto serie en un hilo separado y gestión de hilos.
- **Derivadas directamente de QObject**: Flags del programa y Nombres de archivos.

8.2.2. Diagrama jerárquico de QObjects.

Además de la herencia clásica entre clases de C++, Qt añade una jerarquía extra en la que ciertos QObjects son hijos de otros QObjects. El hecho de que un QObject sea hijo de otro, implica por ejemplo que cuando su padre libere la memoria, él también liberará memoria y por otra parte, se permitirá la comunicación con signals y slots entre ellos, lo que facilita increíblemente la programación.

En la figura 7.16 se muestra un diagrama jerárquico en el que se muestra a jerarquía de los QObjects de forma simplificada. Se han omitido una gran cantidad de QObjects "secundarios", como los QLayouts o muchos QFrames, que se usan para colocar los Widgets de manera ordenada dentro de la ventana principal. Por tanto, se han dejado los imprescindibles, ya que lo que se pretende es que se tenga una visión global del programa.

El QObject padre de todos los demás es Waspmonitor, que es una clase que deriva de QMainWindow, como se pudo observar en la figura 7.15.

Los 3 primeros hijos de este QObject, son los componentes de dicha QMainWindow:

- **QStackedWidget**: Una "pila de widgets" entre la Barra de Herramientas y la Barra de Estado donde se muestra o la ventana de monitorización, englobada en el QObject "showMonitor", o bien la ayuda de Waspmonitor, de la cual se hace cargo "showInfo".
- **QToolBar**: La barra de herramientas, cuyos botones están representados por QObjects del tipo QAction.
- **QStatusBar**: La barra de estado.

Estos QObjects son el "esqueleto" de la QMainWindow y por tanto de la GUI, y se puede entender fácilmente observando la figura 7.17.

Figura 8.16: Diagrama general de QObjects de Waspmonitor.

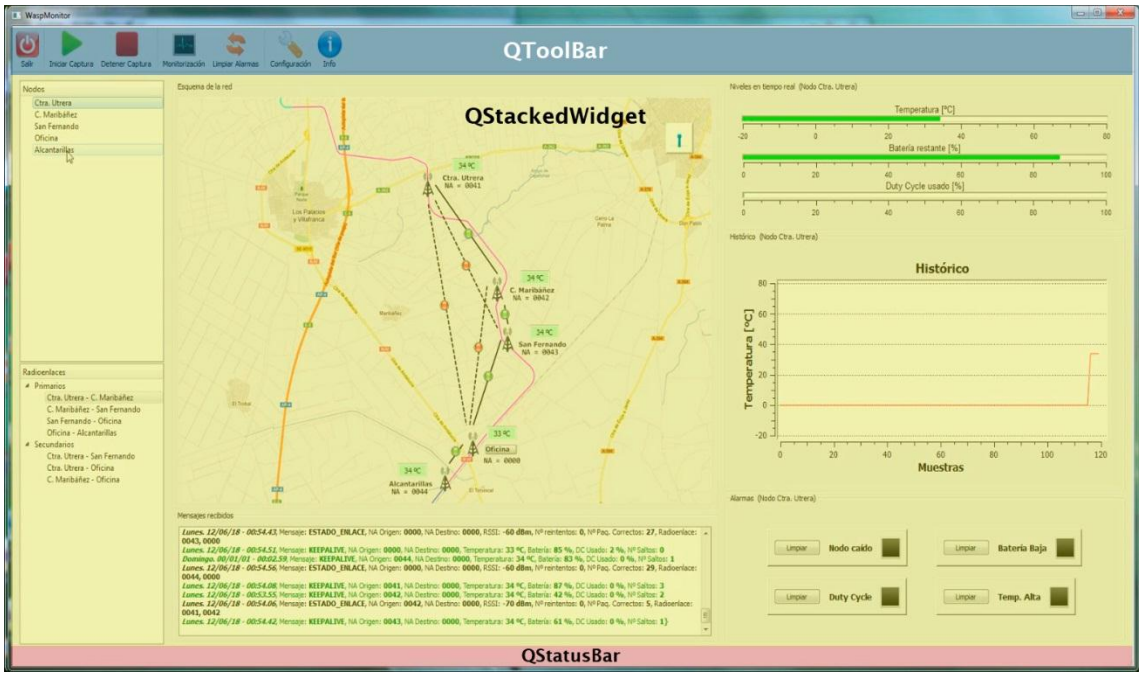

Figura 8.17: QObjects principales de la QMainwindow de Waspmonitor.

En el mismo nivel de jerarquía, los siguientes hijos son:

- **HiloSniffer**: Hilo que se encarga de configurar correctamente el puerto serie, recibir los datos y guardarlos para posterior monitorización y almacenamiento.
- **QTimer y QTime**: Son temporizadores que se encargan de hacer saltar las alarmas cuando caen los nodos que están situados justo al lado del central y no se pueden recibir mensajes de Caída de Enlace.
- **QString y QStringlist**: Son dos clases que se usan para almacenar y tratar el mensaje recibido desde el puerto serie, previamente tratado por HiloSniffer.
- **WaspFlags y WaspFlagsNames**: Son clases auxiliares para el manejo de banderas y nombres de archivo.
- **configDialog**: Ventana de configuración de Waspmonitor.

En los siguientes apartados se explica de manera más detallada cómo se han programado las diferentes partes de Waspmonitor.

8.2.3. Función main.

La función principal, o main(), lo que hace es mostrar en un principio la “Splash Screen” o pantalla de inicio del programa, crear un objeto de clase “Waspmonitor” y mostrarlo, del que serán hijos todos los demás, como se ha visto en la figura 7.16., y ajustar la resolución del objeto Waspmonitor según la resolución de pantalla del escritorio. Finalmente se libera memoria del objeto (y con esta orden, se libera la memoria del resto).

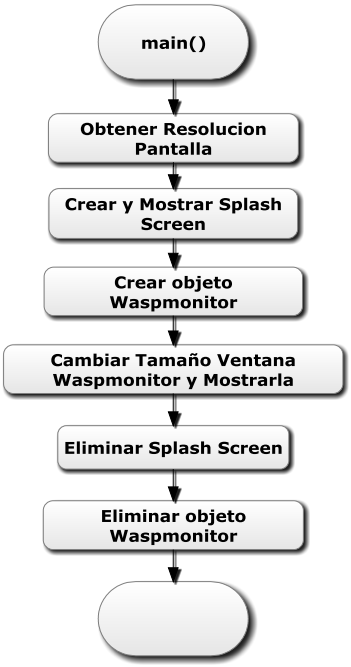

Figura 8.18: Diagrama de flujo de la función 'main' de Waspmonitor.

8.2.4. Programación de la GUI.

Al crear el objeto de tipo Waspmonitor, se invoca el constructor del mismo, realizándose las llamadas a las funciones de la figura 7.19.

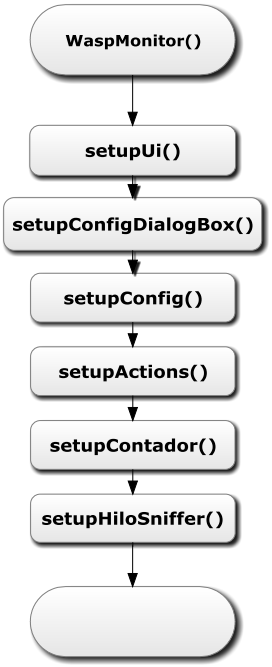

Figura 8.19: Constructor de Waspmonitor.

La tarea que realiza cada una de dichas funciones se detalla a continuación:

- **setupUi()**: Configura toda la parte gráfica (User's Interface) generada mediante el UIC a partir de los forms creados con Qt Designer.
- **setupConfigDialogBox()**: Configura la ventana de configuración de Waspmonitor.
- **setupConfig()**: Configura los flags y los nombres de los archivos.
- **setupActions()**: Realiza las conexiones de signals y slots para los botones de la Barra de Herramientas y los botones de los Paneles de Alarma.
- **setupContador()**: Se configura el contador general que comprueba cada 30 segundos el estado de los nodos, emitiendo un signal que recibe un slot llamado "checkEstadoNodos()". Una vez dentro de dicho slot se comprueba que cada contador particular asociado a cada nodo no ha sobrepasado los 2 minutos (no han pasado 90 segundos sin recibir un Keepalive).
- **setupHiloSniffer()**: Crea el objeto que se dedicará a capturar del puerto serie e invoca a su constructor. Por otra parte, conecta la signal "mensajeListo()" que se produce cuando hay un mensaje completo leído desde el puerto serie con el slot "updateOutput()", que sirve para actualizar las gráficas y el log.

Los slots de Waspmonitor son los siguientes:

- **mostrarMonitorizacion():** Hace visible el widget "showMonitor".
- **mostrarInfo():** Hace visible el widget "showInfo".
- **mostrarConfig():** Hace visible el widget "showConfig".
- **checkEstadoNodos():** Comprueba si los nodos están vivos y en caso negativo, activa las alarmas correspondientes.
- **salir():** Sale de la aplicación en respuesta a una signal emitida por el botón de Salir de la barra de herramientas.
- **iniciarCaptura():** Inicia el hilo de captura de datos a través del puerto serie y actualiza los nombres de los archivos de log.
- **detenerCaptura():** Detiene el hilo de captura.
- **limpiarAlarmas():** Limpia los iconos de alarma tanto del mapa como del panel de alarmas.
- **updateOutput():** Actualiza la salida del hilo de captura "hiloSniffer". Cuando se recibe un mensaje completo, se monitoriza y se guarda en los logs, en caso de que esté activado que se archiven los logs. Se hablará más de él en el apartado 7.2.4.1.
- **elegirPuerto(QString):** Elige el puerto serie COM que se usará para capturar datos a través del Waspmote Gateway.
- **Slots de habilitación/deshabilitación de monitorización de nodos:** Habilitan/deshabilitan la monitorización de cualquiera de los nodos, excepto del nodo de la Oficina (ya que no tiene sentido deshabilitar el nodo en el que se reciben todos los mensajes).
- **Slots de habilitación/deshabilitación de monitorización de logs:** Habilitan/deshabilitan el almacenamiento en logs en .CSV o en .HTML.

- **Slots de limpieza de iconos de alarma:** Limpian los iconos de alarma. Son llamados desde el slot limpiarAlarmas().

8.2.4.1. updateOutput().

Este es el slot más importante probablemente de toda la clase Waspmonitor pues es la función encargada de procesar el mensaje, y actualizar la monitorización y recopilación de datos.

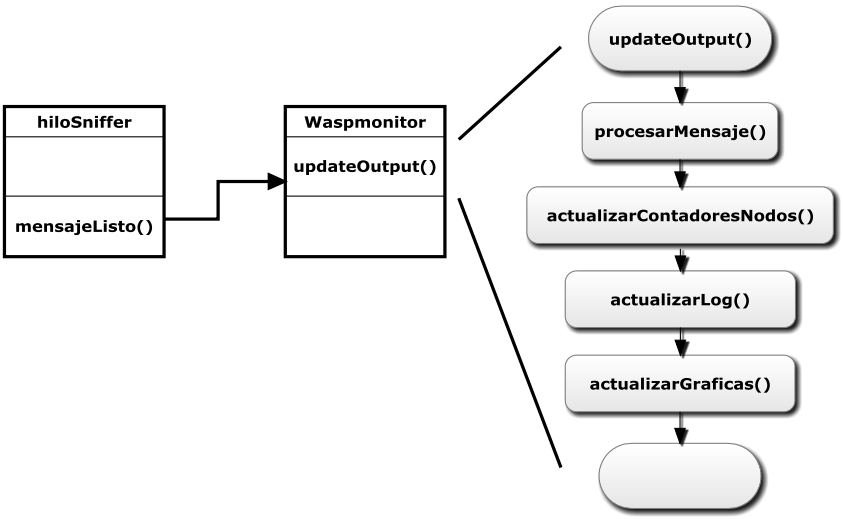

Figura 8.20: Emisión de signal por parte de hiloSniffer y captura de la misma por el slot updateOutput de Waspmonitor.

En la figura 7.20. se ve el flujo completo desde que se emite la signal desde el objeto hilosSniffer hasta que es captada por el slot updateOutput() del objeto Waspmonitor.

Una vez que se ejecuta el slot, se realizan las siguientes acciones:

- **procesarMensaje()**: Esta función procesa el mensaje válido recibido en formato QString y hace lo siguiente: Crea una QStringList con los campos del mensaje separados por las comas (que se usará posteriormente en otras funciones) y tras esto, traduce los días de la semana al castellano.
- **actualizarContadoresNodos():** Actualiza los contadores de estado de los nodos. Si se recibe un Keepalive de un determinado nodo, se reinicia el contador.
- **actualizarLog():** Esta función actualiza la ventana de Log (dando formato a los mensajes según del tipo que sean) y crea los archivos de log tanto Html como CSV, dependiendo de si los flags de configuración están activados.
- **actualizarGraficas():** Actualiza las gráficas de monitorización, las etiquetas de temperatura, los iconos de estado y de alarma y los paneles de alarma a partir del mensaje nuevo de datos llegado desde hiloSniffer. En concreto, se llaman a las siguientes funciones:
    - **actualizarEtiquetas():** Actualiza las etiquetas de temperatura del mapa.

- **actualizarBarras():** Actualiza los Iconos de estado del mapa.
- **actualizarHistoricos():** Actualiza las gráficas de históricos.
- **actualizarPanelesAlarma():** Actualiza los paneles de alarma.
- **actualizarIconosAlarma():** Actualiza los iconos de alarma del mapa.

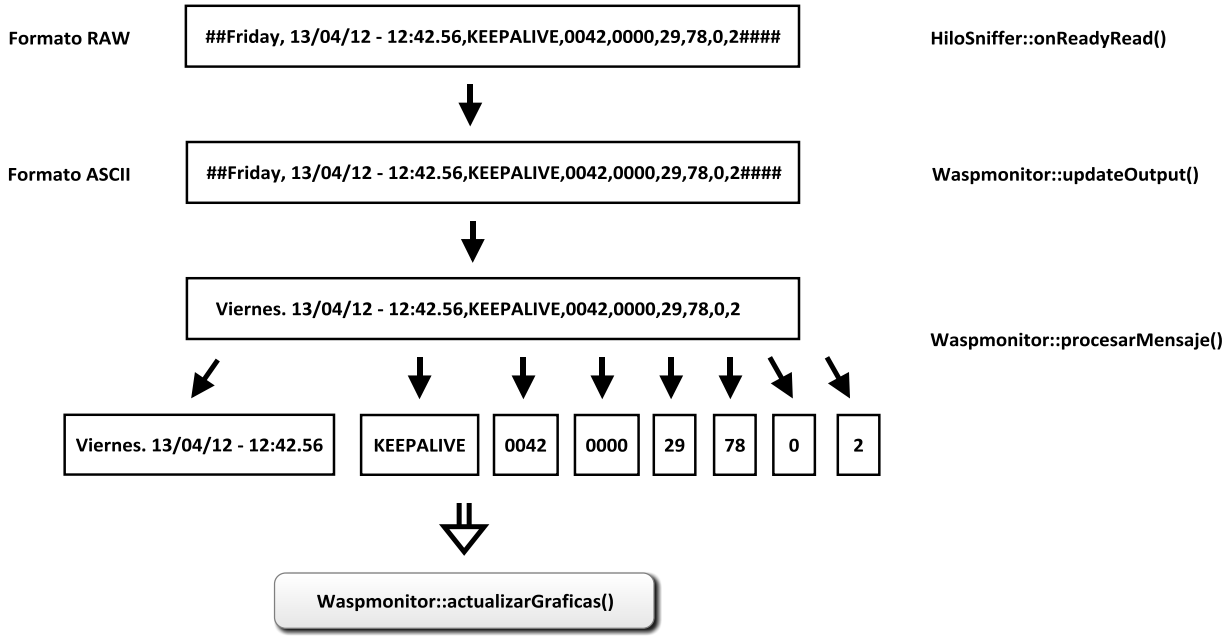

Figura 8.21: Proceso de extracción del mensaje y de los campos del mismo para actualización de las gráficas.

8.2.4.2. showMonitor.

showMonitor es la clase que se encarga de generar la parte de monitorización de Waspmonitor. Parte de la clase se ha desarrollado mediante forms con Qt Designer y otra parte se ha programado directamente en C++.

showMonitor se compone de una serie de objetos que representan los widgets de la interfaz gráfica, como son por ejemplo las etiquetas de temperatura, los leds de estado del mapa, los iconos de alarma del mapa, etc. y de funciones que sirven para configurar las distintas. La clase, también dispone de una serie de slots, que se conectan a las signals que emiten las acciones de las listas de nodos y radioenlaces (acción pulsar) y permiten que se cambie el panel de monitorización que se muestra. Por último, como todas las clases generadas mediante forms, tiene un puntero "ui" a los elementos generados mediante el UIC, para acceder a dichos objetos.

Al crear un objeto de tipo showMonitor dentro del constructor de Waspmonitor, se invoca a su vez al constructor del primero y éste llama a las funciones que crean la interfaz gráfica:

- **setMap()**: Añade el mapa de un tamaño u otro segun la resolución de la pantalla.
- **setTempTags()**: Añade las etiquetas de temperatura en la posición correspondiente del mapa según la resolución de pantalla.
- **setMapIcons()**: Añade los iconos de estado y alarma en el mapa.
- **setGrafBarNodos()** y **setGrafBarRadio()**: Configuran los widgets de Grafica de Barras de los Nodos y de los Radioenlaces, respectivamente.
- **setHistoricos():** Configura los widgets de Históricos de temperatura y RSSI.
- **setPanelAlarmasNodos()** y **setPanelAlarmasRadio()**: Configura los widgets de Panel de Alarmas de los Nodos.

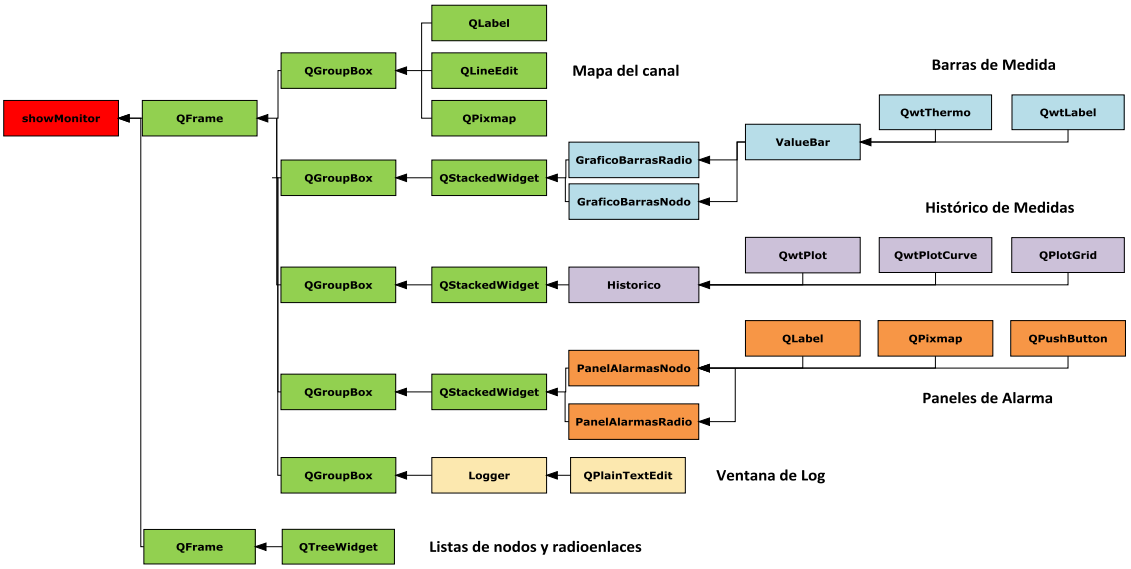

Figura 8.22: Diagrama simplicado de QObjects de showMonitor.

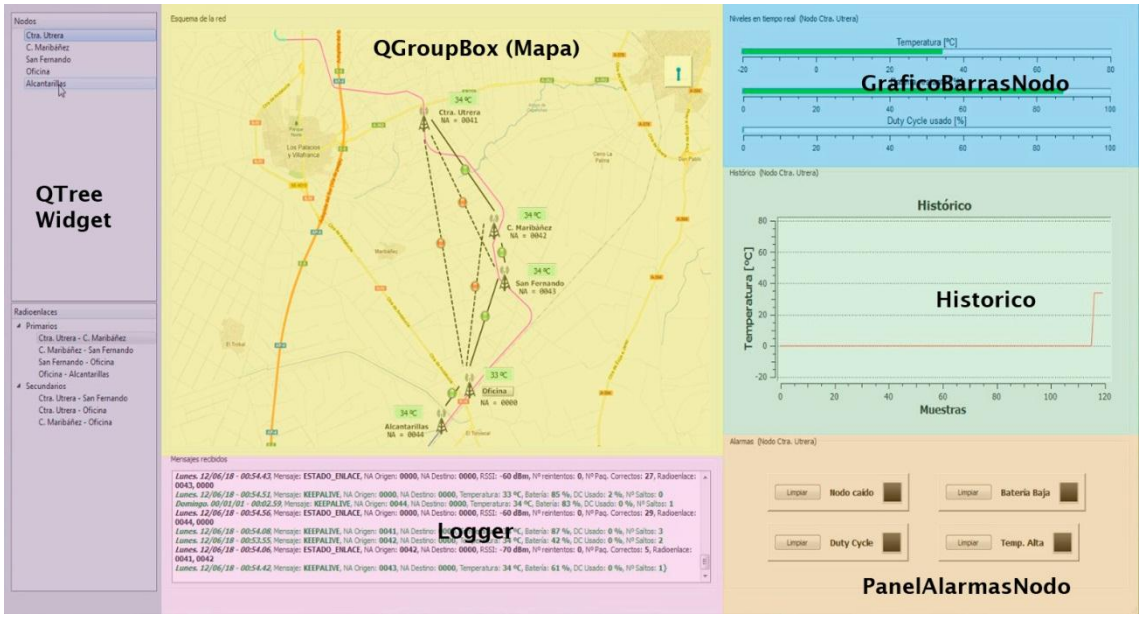

Figura 8.23: QObjects principales de showInfo.

A continuación se van a explicar las clases que se han programado con las librerías Qwt:

1. **GraficoBarrasNodo y GraficoBarrasRadio.**

   Estas dos clases, contienen por un lado QWidgets de tipo "Valuebar", como se puede ver en la figura 7.22. y por otro, una función para actualizar cada tipo de barra, temperatura, batería, DC, para la clase GraficoBarrasNodo y RSSI o Throughput, para la clase GraficoBarrasRadio. En estas funciones se actualizan los valores de las valuebar y se cambian los colores, según el valor que sea. La actualización de estas gráficas se produce mediante la llamada a la función Waspmonitor::actualizarGraficas(), cuando se recibe un mensaje nuevo (ver figura 7.20).

   La clase Valuebar está diseñada con la librería Qwt y básicamente se compone de un widget Qwt de tipo QwtThermo y un QLabel para etiquetar la barra.

2. **Históricos.**

   Los widgets Qwt utilizados han sido los necesarios para el renderizado de la gráfica, esto es, QwtPlot, QwtPlotCurve y QwtPlotGrid. Se ha utilizado antialiasing en el renderizado para suavizar la curva y se ha configurado la rejilla para que aparezcan sólo las líneas horizontales. Al igual que en el caso de los gráficos de barras, en los históricos, la actualización se produce mediante una función interna de la clase, en este caso, actualizarGrafica(). Esta función se llama desde Waspmonitor::actualizarGraficas() y lo que realiza es un evento Qwt de replot(), que produce que se vuelva a redibujar la gráfica, desplazándola una muestra hacia la izquierda. Para el resto de etiquetas de las gráficas, etc. se han usado objetos Qt estándar, como QStrings, etc.

3. **PanelAlarmasNodo y PanelAlarmasRadio.**

   Si bien, estas dos clases no han sido programadas usando la librería Qwt, se han incluido en esta sección, puesto que forman parte de la zona de "monitorización" de la clase "showMonitor". Se han programado directamente en Qt, sin usar los formularios ni Qt Designer, puesto que daba problemas la implementación visual. Los widgets fundamentales de estas clases se pueden ver en el diagrama simplificado de la figura 7.20.

8.2.4.3. configDialog.

Esta clase ha sido generada completamente por el UIC, ya que la ventana de de configuración 'configDialog' se diseñó con Qt Designer.

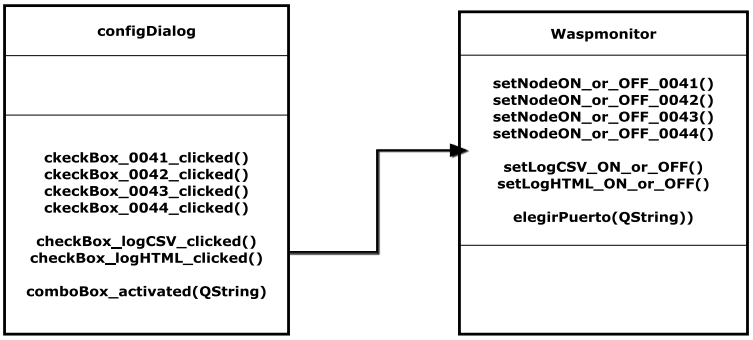

Figura 8.24: Signals y slots relacionadas con la ventana de configuración de Waspmonitor.

La clave de esta clase, más allá de los QObjects que la forman, que simplemente generan la interfaz gráfica, son las signals de configDialog y los slots que capturan dichas signals ejecutando funciones en el objeto Waspmonitor. Para ver los widgets fundamentales que componen esta clase se puede ver el diagrama simplificado general de QObjetcts de Waspmonitor en la figura 7.16.

Signals de configDialog.

- **checkBox_0041_clicked(), checkBox_0042_clicked(), checkBox_0043_clicked() y checkBox_0044_clicked()**: Son signals que se emiten cuando se pulsa un checkbox para habilitar o deshabilitar la monitorización de un determinado nodo. Dichas signals son capturadas por slots del objeto Waspmonitor, setNodeON_or_OFF_004x(), que se encargan de realizar las acciones pertinentes para la habilitación/deshabilitación.
- **checkBox_logCSV_clicked() y checkBox_logHMTL_clicked()**: Son signals que se emiten al pulsar un checkbox encargado de habilitar o deshabilitar el almacenamiento en logs en formato .CSV o .HTML. y son capturadas por los slots setLogCSV_ON_or_OFF() y setLogHTML_ON_or_OFF() del objeto Waspmonitor, que se encargan de realizar las acciones que procedan para la habilitación/deshabilitación.
- **comboBox_activated(QString)**: Es una signal que se emite al seleccionar el puerto COM adecuado de la comboBox de la ventana de configuración y es capturada por el slot elegirPuerto(QString) de Waspmonitor.

8.2.5. Descripción del hilo de captura (hiloSniffer).

El hilo de captura se inicia cuando se llama al slot 'Waspmonitor::iniciarCaptura()' (Se pulsa el botón de play rojo de la Barra de Herramientas) y se detiene cuando se llama al slot 'Waspmonitor::detenerCaptura()'.

La función run() del hilo, que sería el equivalente a un main() en un proceso, hace lo que se siguiente (figura 7.20):

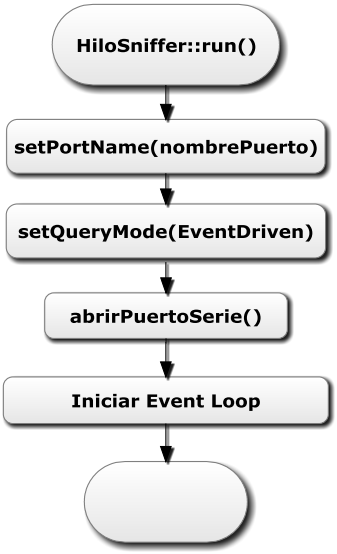

Figura 8.25: Función run() del hilo de captura.

Los pasos serían los siguientes:

- **setPortName()**: Desde la ventana de configuración, representada por la clase 'configDialog', se elige el puerto y con esta función, de la librería Qextserialport, se selecciona el puerto serie COM adecuado. Puesto que la selección se realiza al iniciar la captura, si se intenta cambiar el puerto COM durante la captura, aparecerá un cuadro de diálogo de advertencia diciendo que no se permite cambiar, si previamente no se ha detenido.
- **setQueryMode()**: Qextserialport permite dos métodos de funcionamiento para leer y escribir en el puerto serie. El primero es el método de 'polling' y el segundo el 'event driven'. Es decir, o sondeando continuamente el puerto, o mediante eventos. En un principio se intentó el método de polling, pero consumía una cantidad excesiva de CPU (cerca del 50% en un Intel E5700 Core2Duo) y por tanto, pese a su simplicidad, se descartó. El método elegido es por eventos. Este se basa en iniciar un "bucle de eventos" en el cual el hilo espera que se produzca algún evento como por ejemplo 'readyRead' (listo para leer), para llamar a un slot que procese los datos recibidos, y se consume CPU sólo en dicho momento.
- **abrirPuertoSerie()**: Esta función realiza las siguientes acciones:
  - Abre el puerto serie mediante la función open() de la librería Qextserialport.
  - Configura el puerto serie con los parámetros:
    - *Tasa de baudios*: 38400 baudios.
    - *Bits de datos*: 8 bits.
    - *Bits de stop*: 1 bit.
    - *Paridad*: Ninguna.
    - *Control de flujo*: Hardware.
  - Se conecta la signal 'readyRead()', que produce el objeto cuando hay datos recibidos por el puerto serie, con el slot 'onReadyRead()', del mismo objeto,

que se encarga de procesar los datos recibidos y estructurarlos como un mensaje “inteligible” por Waspmonitor.

- **Iniciar el Event Loop**: Con la función exec() de Qextserialport.

8.2.5.1. Slot onReadyRead().

El slot ‘onReadyRead()’ que se ha nombrado en la función ‘abrirPuertoSerie()’ es muy importante, ya que realiza el tratamiento de los datos recibidos que se leen por ráfagas y que contienen caracteres “inteligibles” y valores hexadecimales que no se pueden mostrar y forman parte de la cabecera API de la trama enviada via radio.

El pseudocódigo del slot sería el siguiente:

```
HiloSnifer::onReadyRead()
{
     bytesAvailable();   // Se comprueban los bytes que hay para leer

     read(bytes);    // Se leen los bytes.

     bytesReceived = bytesReceived.append(bytes)   // Los bytes leídos se van añadiendo a una

                                                   // cadena auxiliar de bytes recibidos

     auxHexArray = bytesReceived.toHex();  // Se convierte la cadena a hexadecimal.

  if (auxHexArray.contains(0x23232323)    // Si contiene la secuencia de final ‘####’
  {
      quitarCabecerasAPI();  // Se quitan todos los datos que forman parte de la cabecera API

      bytesReceived.clear(); // Se borra la cadena una vez que se guarda el mensaje.

  }
}
```

Figura 8.26: Pseudocódigo del slot 'onReadyRead()'.

En el pseudocódigo se puede ver que hay un momento en el que se convierten los datos de una QString (cadena Qt estándar) a una cadena hexadecimal. Esto se hace así puesto que la cadena contendrá caracteres que no son imprimibles (bytes RAW) y al intentar operar con ellos, las funciones de librería de manejo de QStrings, daban error.

En la figura 7.22. se puede ver un ejemplo con un mensaje de Keepalive en el que se visualizar el proceso de guardar el mensaje en formato RAW en la QString ‘bytes’(con las cabeceras API en RAW en color azul claro), cómo se va añadiendo a la QString ‘bytesReceived’ y cómo se extrae finalmente el mensaje válido, una vez que se recibe la trama completa (gracias al delimitador de fin de trama ‘####’).

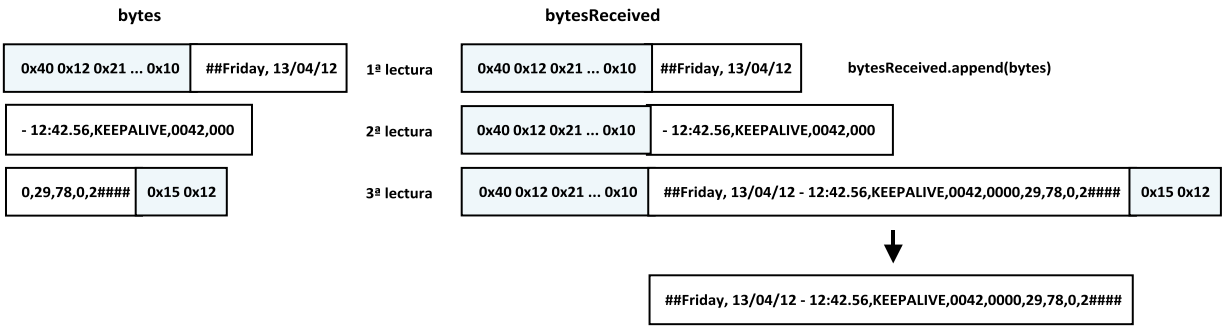

Figura 8.27: Extracción de mensaje mediante 'onReadyRead()'.

8.2.6. Timers.

En Waspmonitor, se han utilizado dos contadores, uno general (QTimer) que cada 30 segundos llama al slot checkEstadoNodos(), que se encarga de comprobar que los nodos siguen vivos, haciendo uso a su vez de un contador para cada nodo (QTime). Waspmonitor supone que un nodo está caído si pasados 90 segundos, no se ha recibido nada, y hace saltar las alarmas de nodo caído y todos sus radioenlaces asociados, caídos.

La diferencia entre el QTimer y el QTime es que el QTimer lanza una signal periódica (30 segundos) en este caso, y el QTime es un simple “reloj” que cuenta el tiempo que ha transcurrido.