upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Desarrollo e implementación de una interfaz de usuario en sistemas Linux para equipos de monitorización óptico-electrónicos empleados en infraestructura civil

Memoria trabajo fin de grado realizada por

David Moreno Antón

Director

Javier Urricelqui Polvorinos

Pamplona – 7 de Septiembre de 2018

Agradecimientos

A mi tutor, Javier, quien mostró desde el primer momento confianza plena en mi trabajo. Gracias por tu ayuda ante todos los problemas y dudas que han ido surgiendo durante el proyecto, ayudándome a conseguir el objetivo marcado.

A mis compañeros del grupo por hacerme amenas todas las mañanas con esas llamadas para almorzar. Espero seguir compartiendo más momentos junto a vosotros.

A mi familia, en especial a mi madre y hermano por su cariño, ánimo y apoyo en los momentos más difíciles de estos últimos años. A mi pareja Maider por aguantar el estrés acumulado a finales de semestre y mis rabietas prácticamente diarias. Y sobre todo a alguien especial, alguien que soñó con un hijo graduado y que, aun no estando cerca, estoy seguro de que estará orgulloso de mi. Sin la fuerza de hacer tus sueños posibles esto no hubiese sido posible, papá. Muchas gracias por hacerme sentir valorado.

Resumen

El presente trabajo fin de grado se ha llevado a cabo con el objetivo de desarrollar una aplicación e interfaz gráfico de usuario en sistemas GNU/Linux para el control y funcionamiento de equipos interrogadores empleados para la monitorización de activos presentes en el sector energético y/o de infraestructura civil.

En particular, el grupo de investigación de Comunicaciones Ópticas y Aplicaciones Electrónicas de la Universidad Pública de Navarra (UPNA) trabaja continuamente en el desarrollo de nuevos equipos interrogadores. Este equipamiento permite medir magnitudes físicas, tales como, deformación unitaria, temperatura, entre otras, gracias al uso de la fibra óptica y el fenómeno físico de dispersión Brillouin estimulada. Este fenómeno físico permite medir en cada uno de los puntos de la fibra óptica sin necesidad de añadir ningún sensor adicional en la fibra óptica, únicamente su conexión al equipo interrogador. De esta manera, se podría considerar que este tipo de equipamiento es capaz de generar un sistema nervioso artificial para estructuras, donde la fibra óptica actúa como los “nervios” y los equipos desarrollados representan el “cerebro” donde se procesa la información y se extraen los datos de interés para la monitorización del activo y la generación de alertas que activen determinados protocolos.

En el contexto anteriormente explicado, es necesario el desarrollo de una aplicación e interfaz que permita al usuario final poder interactuar con el equipo interrogador mediante, por ejemplo, la configuración del equipo, la realización de medidas, el análisis y visualización de los datos obtenidos por el equipo interrogador o la configuración de alertas ante determinados eventos. La aplicación desarrollada en este Proyecto Fin de Grado se ha realizado en lenguaje C++ mediante la utilización de GIMP Toolkit (GTK+) como conjunto de herramientas multiplataforma para crear interfaces gráficas de usuario con ayuda de Glade y se ejecuta en un ordenador externo con sistema operativo

Ubuntu. La comunicación entre el ordenador y el equipo interrogador se realiza por conexión serie.

Índice general

Resumen 1

0. Introducción al trabajo fin de grado 1

1. Introducción al desarrollo del GUI 5

1.1. Sistema Operativo 5

1.1.1. Distribuciones GNU/Linux 5

1.2. Lenguajes de programación 9

1.2.1. C++ 9

1.2.2. Python 11

1.2.3. Java 12

1.2.4. Selección del lenguaje de programación 14

1.3. Entorno de desarrollo integrado (IDE) 14

1.3.1. Eclipse 15

1.3.2. Netbeans 16

1.3.3. Selección del IDE 16

1.4. Interfaz gráfica de usuario (GUI) 17

1.4.1. GTK+ 17

1.4.2. Glade 19

2. Desarrollo del GUI 21

2.1. Requisitos del GUI 21

2.2. Descripción de tipos de elementos para el desarrollo de un GUI 25

2.2.1. Descripción XML 25

2.2.2. Descripción elementos GTK+ 27

2.3. Organización del proyecto 45

3. Uso de la aplicación 57

3.1 Barra de herramientas 58

3.2 Explorador de archivos 65

3.3 Panel visualización 67

3.4 MenuBar 68

4. Conclusiones y líneas abiertas 71

4.1. Conclusiones 71

4.2. Líneas abiertas 72

A. Importación del proyecto en IDE 77

B. Configuración de GIT 79

C. Programación 81

Capítulo 0

Introducción al trabajo fin de grado

Motivación del trabajo

Correspondiente al avance de la tecnología, la monitorización de estructuras como aerogeneradores, oleo/gaseoductos o infraestructuras de obra civil como túneles, canales, diques, puentes o presas han adquirido una relevancia muy importante. Estas estructuras necesitan ser examinadas a lo largo de su vida con el fin de mantener no sólo su correcto funcionamiento sino su integridad estructural. En este punto es, donde los sensores de fibra óptica distribuidos adquieren un papel de mayor importancia en la monitorización de la salud estructural de estos activos. Un sensor de fibra óptico distribuido está compuesto por un equipo interrogador al que se conecta la fibra óptica y que, por un fenómeno físico adyacente a la fibra óptica, permite disponer de cientos de puntos de medida a lo largo de la misma. Cada punto de medida en la fibra óptica aporta información de temperatura y/o deformación unitaria.

La programación de aplicaciones para el control y manejo de los datos adquiridos por este tipo de equipamiento es un punto de desarrollo crucial para su uso en la realización de proyectos de monitorización de un determinado activo. La aportación de una aplicación intuitiva para este tipo de equipamientos se podría llegar a resumir en productividad ya que permite el manejo y el orden de todos los recursos e información aportados por el equipo interrogador, facilitando así al usuario poder identificar los puntos clave en cada activo y así poder realizar un correcto mantenimiento del mismo. Además, el control de los datos de un equipo de tales características desde una aplicación hace posible implementar una correcta gestión de alertas que permita activar

determinados protocolos de seguridad, protegiendo de esta manera la integridad de la estructura y de su entorno.

Combinando la necesidad de monitorización de estructuras y la programación de aplicaciones para dicho fin, se está motivando un importante esfuerzo de desarrollo enfocado en la realización de un software amigable e intuitivo para el usuario. Este trabajo fin de grado se engloba precisamente en esta línea de desarrollo del software.

Objetivos del trabajo

El objetivo general de este trabajo fin de grado es el diseño y desarrollo de una aplicación basada en sistemas Linux para el correcto funcionamiento de los equipos interrogadores desarrollados por el grupo de investigación de la Universidad Pública de Navarra y explotados por la empresa Uptech Sensing. Dicha aplicación se basa en un interfaz gráfico de usuario el cual permita el control y manejo presencial de equipos interrogadores.

En base a estas consideraciones se establecieron unas líneas de trabajo las cuales seguir para la consecución del objetivo marcado:

- Familiarización con el funcionamiento de los equipos interrogadores desarrollados en la UPNA y Uptech Sensing.
- Estudio de la programación existente para el control de equipos interrogadores.
- Propuesta de soluciones y mejoras con respecto a la programación existente.
- Selección del entorno de trabajo tales como la configuración de los aspectos necesarios del ordenador (sistema operativo, kernel, liberías etc.) donde se ejecutará la aplicación a desarrollar.
- Estudio del desarrollo y programación de aplicaciones e interfaces gráficas en C++/GTK con Glade.
- Selección del entorno de desarrollo (IDE, Integrated Development Enviroment) y configuración del mismo.
- Programación de la aplicación objeto de este Proyecto Fin de Grado.

Estructura de la memoria

La memoria se encuentra estructurada en 4 capítulos.

Capítulo 1

En el capítulo 1 se hará en primer lugar una introducción al desarrollo de la Interfaz gráfica de usuario (GUI). Comenzando por la distribución y kernel del Sistema Operativo utilizado. Siguiendo con el análisis de diferentes lenguajes de programación proveyendo una descripción general, ventajas e inconvenientes en su uso, aplicaciones escritas usando este lenguaje y el motivo que ha llevado a seleccionar un lenguaje en concreto. A su vez se analizarán diferentes entornos de desarrollo integrado (IDE) y la selección del mismo. Finalmente, se hará una descripción general del conjunto de herramientas multiplataforma usadas para crear dicho GUI. Herramientas tales como GTK+ y Glade.

Capítulo 2

En el capítulo 2 se presentará el desarrollo del GUI en el que se darán a conocer los requisitos necesarios dando una breve descripción del funcionamiento de los interrogadores y las necesidades de configuración y control de los equipos interrogadores desde el GUI. A continuación, se describirán los elementos usados para el desarrollo de la interfaz y la herramienta Glade. Se finalizará hablando sobre la organización del proyecto.

Capítulo 3

En el capítulo 3 se presentará la aplicación desarrollada. Se trata principalmente del desarrollo y la implementación de esta aplicación describiendo su funcionamiento interno, las ventanas y diálogos, el menú y la conexión entre código fuente y Glade.

Capítulo 4

En el capítulo 4, se presentarán las conclusiones generales obtenidas después del desarrollo de la aplicación y se proporcionarán las líneas futuras de trabajo a desarrollar gracias al conocimiento obtenido durante la realización de este trabajo final de grado.

Capítulo 1

Introducción al desarrollo del GUI

Antes de entrar en detalle en los siguientes aspectos, hace falta contextualizar y analizar el proyecto en el que se encuentra el desarrollo de esta aplicación/GUI. El núcleo central de este trabajo reside en la programación de un interfaz gráfico de usuario (del inglés Graphical User Interface, GUI). Su principal uso, en un contexto del proceso de interacción persona-computadora, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computadora, en nuestro caso, el equipo interrogador.

1.1. Sistema Operativo

Windows, Mac OS o distribuciones de Linux son los tres sistemas operativos que, en general, se puede encontrar fácilmente en cualquier ordenador que se encuentre en ejecución. En este Proyecto Fin de Grado, se ha trabajado con una de las distribuciones Linux más conocidas por los usuarios. A continuación, se mostrarán las ventajas de Linux y algunas de las distribuciones más populares.

1.1.1. Distribuciones GNU/Linux

Uno de los puntos por los que es importante empezar hablando son las diferentes categorías de software existente. Existen básicamente dos tipos de software:

propietario y libre de uso bajo licencia [1]. Parece algo obvio que el software privativo o software no libre somete a los usuarios al poder de los desarrolladores del software además de tener que pagar por adquirir licencias de uso. Por el contrario, el movimiento del software libre es aquel que respeta la libertad de los usuarios y la comunidad. Dicho software permite que los usuarios ejerzan el control de sus propias tareas de computación y tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Este es uno de los puntos clave para el uso del sistema GNU/Linux.

En dicho sistema, Linux es el núcleo y está modelado como un sistema operativo tipo Unix [2]. Nadie es dueño de Linux, a diferencia de otros sistemas operativos, con Linux no hay que preocuparse de comprarlo, puesto que ni habrá que pagar para obtener el sistema ni para actualizarlo y cuyos desarrolladores se han comprometido a seguir las pautas para distribuciones de sistemas libres. Esto significa que las distribuciones incluirán y propondrán exclusivamente software libre rechazando así las aplicaciones, plataformas de programación, controladores, firmware, juegos y cualquier otro software que no sea libre. Afortunadamente en Linux además del soporte que pueda ofrecer el desarrollador de la distribución, contaremos con un soporte técnico conformado por los diferentes usuarios de plataformas Linux de casi todas las partes del globo, poniendo en común sus conocimientos y experiencias.

Linux es uno de los sistemas operativos que menos recursos consume [3] y, por tanto, otorga mayor rendimiento. Además, gracias a la gran comunidad de usuarios dispuestos a ayudar, las vulnerabilidades son detectadas y corregidas más rápidamente que en otros sistemas operativos. En el ámbito de la seguridad, es interesante destacar que la mayoría de los virus tienen como objetivo sistemas Windows y MacOS dada la popularidad de estos sistemas [4] tal y como vemos en la Figura 1.1 y Figura 1.2.

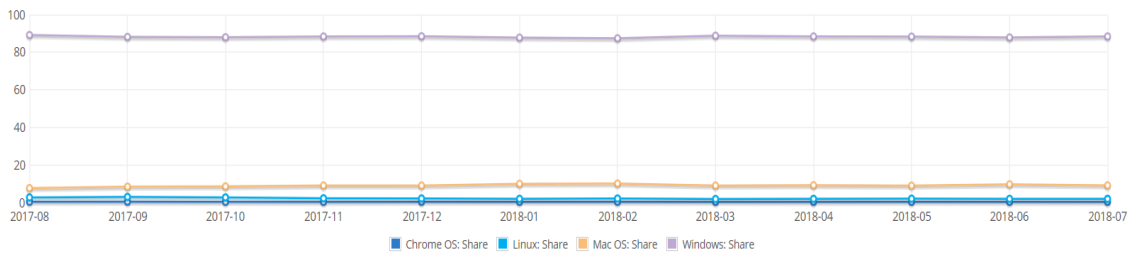


Figura 1.1: Gráfica SO más utilizados actualmente. Eje X: Año-Mes; Eje Y: Porcentaje.

Windows	88.31%
Mac OS	9.02%
Linux	2.20%
Chrome OS	0.31%
Unknown	0.16%
BSD	0.01%

Figura 1.2: Tabla de porcentaje SO más utilizados en la actualidad.

Hay numerosas distribuciones Linux libres [5], de las cuales destacamos las más conocidas a continuación, ya sea por su estabilidad, seguridad o facilidad de uso.

Ubuntu

De todas las distribuciones disponibles esta es una de las más utilizadas gracias a su facilidad de uso, la libertad en la restricción de uso, los lanzamientos regulares y la facilidad en la instalación. Basada en Debian y con un entorno de escritorio Unity, con el cual busca convertirse en una distribución versátil que se pueda utilizar en diferentes dispositivos como ordenadores, móviles y tabletas. Además, ofrece soporte para servidores.

Fedora

Distribución gratuita creada y mantenida por la empresa Red Hat. Tiene tres versiones diferentes para escritorio, servidores y sistemas en la nube, y destaca por su seguridad gracias al sistema SELinux ("Security-Enhanced Linux").

Debian

Debian es muy estable y 100% libre. Además, destaca por su sistema de paquetería .deb y su gestión de paquetes APT. Es una de las distribuciones más importantes de GNU/Linux, ya que en ella se basan gigantes como Ubuntu.

Kali Linux

La distribución avanzada de pruebas de penetración de Linux basada en Debian con una gran cantidad de herramientas para proteger nuestros equipos. Utiliza un kernel personalizado con parches de seguridad y tiene soporte para la arquitectura ARM.

Como se observa, existen diversas distribuciones de Linux sobre las que poder trabajar. En este caso, gracias a la gran popularidad de Ubuntu [6], tal y como se observa en la Figura 1.3, hemos optado por usar una versión de Ubuntu con soporte a actualizaciones durante 4 años, concretamente Ubuntu 16.04.03 LTS lanzada en Agosto del año 2017 con soporte hasta Abril del año 2021. Por motivos que se explicarán posteriormente en el apartado 2.1, la versión de kernel que se instaló fue la 4.4.

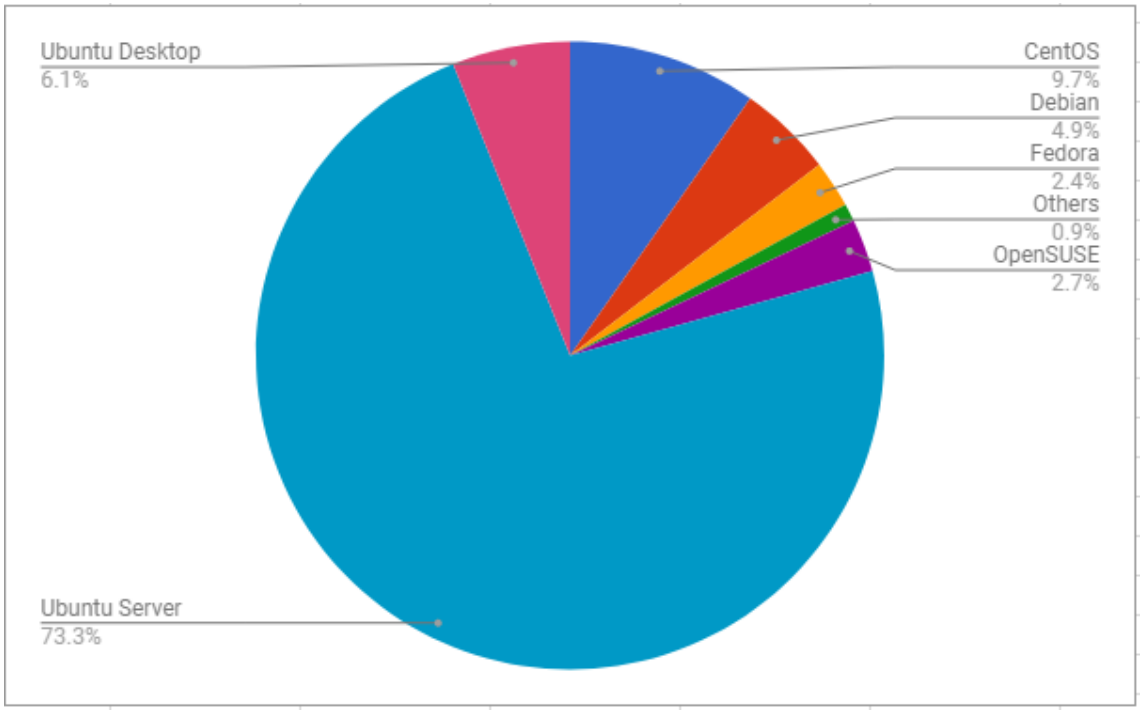


Figura 1.3: Porcentaje de uso en distribuciones Linux.

1.2. Lenguajes de programación

Existen multitud de lenguajes de programación en la actualidad. Cada uno de ellos orientado a determinados tipos de proyectos. En este proyecto se hablará sobre tres lenguajes de programación ampliamente conocidos.

Figura 1.4: Lenguajes de programación. De izquierda a derecha: C++, Java y Python.

1.2.1. C++

Descripción general

C++ es un lenguaje de programación de uso general con un sesgo hacia la programación de sistemas [7]. Soporta abstracción de datos, programación orientada a objetos y programación genérica. Básicamente se creó para extender el lenguaje de programación C añadiéndole la posibilidad de manipulación de objetos.

Ventajas e inconvenientes

Las ventajas de este lenguaje de programación se enumeran a continuación:

- **Compatibilidad:** Esto es debido al gran número de compiladores y bibliotecas existentes. Por este motivo, es mucho más fácil desarrollar grandes proyectos con muchas dependencias ya que existen muchas bibliotecas de código abierto para C++.
- **Compilador**: C++ tiene un compilador para cada sistema operativo importante, por lo que los proyectos que se desarrollan en C++ son increíblemente portátiles.

- **Potencia:** C++ no requiere un tiempo de ejecución especial para su ejecución, se puede crear cualquier tipo de programa, desde la programación de sistemas de bajo nivel hasta las GUI complicadas.
- **Velocidad:** C++ gana mucha velocidad al ser un lenguaje compilado. Los lenguajes no compilados deben interpretarse en tiempo de ejecución, lo que significa que cada acción es un proceso de 2 pasos.

En contraposición, los inconvenientes más importantes son los siguientes:

- **Control:** No hace controles de límites en las matrices, y permite la conversión de tipo incorrecta que naturalmente es un problema que puede ser muy difícil de depurar. Esto le permite al programador llegar a más bajo nivel, pero también le obliga a hacer más trabajo.
- **Flexibilidad:** C++ no tiene mucha flexibilidad para la sintaxis, por lo que puede ser difícil escribir código de forma legible.
- **Memoria:** C++ realiza muy poca administración de memoria, lo que obliga al programador a hacer la mayor parte de ella por sí mismo.

Aplicaciones desarrolladas en C++

Opera es un navegador web creado por la empresa noruega Opera Software mediante el lenguaje C++ [8].Tiene versiones para diferentes sistemas operativos, móviles y tabletas.

Android Su kernel, (en realidad, el mismo que el de Linux pero adaptado para smartphones), cuenta con bibliotecas y drivers de bajo nivel que están programados en C y C++ [9].

Ableton Live es un software rápido, fluido y flexible para la creación y el rendimiento musical [10]. Entre sus características se pueden encontrar efectos, instrumentos, sonidos y todo tipo de características creativas, todo aquello que se necesita para trabajar en el sector musical.

1.2.2. Python

Descripción general

Python es un lenguaje de programación interpretado, orientado a objetos y de alto nivel con semántica dinámica [11]. Sus estructuras de datos integradas de alto nivel, combinadas con el tipado dinámico y el enlace dinámico, lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como también para usarlo como scripting o lenguaje para escribir y administrar programas y códigos. La sintaxis simple y fácil de aprender de Python enfatiza la legibilidad y, por lo tanto, reduce el costo del mantenimiento del programa. Python admite módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización de código.

Ventajas e inconvenientes

Las ventajas de este lenguaje de programación se enumeran a continuación:

- **Fácil aprendizaje:** Uno de los puntos fuertes de este lenguaje de programación es que es fácil de aprender, incluso para un desarrollador novato. Su código es fácil de leer. Además, puede ejecutar muchas funcionalidades complejas con facilidad, gracias a la biblioteca estándar, dando lugar a un desarrollo rápido mediante el uso de menos código.
- **Liviano:** Un equipo con escasos recursos puede llegar a ejecutar Python de manera efectiva.
- **Compatibilidad:** Python tiene compatibilidad con múltiples sistemas y plataformas además de objeto orientado a la programación orientada a objetos. Python tiene una gran cantidad de frameworks que hacen que la programación web sea muy flexible.

En contraposición, los inconvenientes más llamativos son los siguientes:

- **Lento:** Python es un lenguaje lento y no es una buena opción para tareas intensivas en memoria.
- **Desarrollo móvil:** No es un lenguaje muy bueno para el desarrollo móvil.

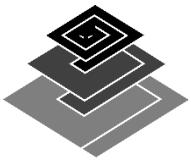
- **Acceso BD y multi-core:** Tiene limitaciones con el acceso a la base de datos y no es bueno para el trabajo multi-procesador/multi-core siendo este un punto muy a su desfavor en cuanto al tipo de aplicación que necesitamos.

Aplicaciones desarrolladas en Python

Calibre es un gestor y organizador de libros electrónicos de código libre que permite además la conversión entre cualquier de los diferentes formatos de archivos de ebooks [12].

FLUMOTION

Flumotion ofrece tecnología de video streaming en directo y bajo demanda simplificando la gestión y el mantenimiento del contenido que ofrecen [13]. Comenzó siendo un proyecto de software libre, pero debido a lo viable que acabó siendo se convirtió en un producto de pago.



Twisted es un framework de red para programación dirigida por eventos escrito en Python y licenciado bajo la licencia MIT [14].

1.2.3. Java

Descripción general

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems [15]. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes.

Ventajas e inconvenientes

Las ventajas de este lenguaje de programación se enumeran a continuación:

- **Facilidad de uso:** De entre todos los lenguajes de programación y todos los métodos para crear aplicaciones, programas de software, páginas webs y demás elementos digitales, destaca Java como uno de los más sencillos de utilizar y de los más fáciles de dominar.
- **Muy extendido:** A su favor hay que decir que es uno de los más extendidos y de los más conocidos por su plataforma y por sus herramientas y sus ventajas para los usuarios, los cuales, siendo conscientes o no, lo utilizan mediante contenido en internet y/o al utilizar ciertas aplicaciones y programas.

Los inconvenientes más llamativos de Java son los siguientes:

- **Rendimiento:** Es un lenguaje interpretado en el que, por su tipología y sus características, cuenta con un rendimiento menor, que hace que los dispositivos y equipos requieran de una mayor potencia y una mayor autonomía.
- **Sintaxis:** La sintaxis de Java resulta algo más engorrosa y complicada que otros lenguajes.
- **Compatibilidad:** Otro punto negativo es que los programas realizados con él no pueden ser ejecutados si no se dispone de la máquina virtual de Java. Aun disponiendo de Java, si está desactivado podríamos no ver ciertos contenidos.

Aplicaciones desarrolladas en Java

Minecraft es un juego de construcción de zona protegida, escrito en Java por Mojang, en el que se puede construir cualquier cosa que se imagine. Es el juego indie más vendido de la historia. [16]

Eclipse es una plataforma que se ha diseñado desde cero para la creación de herramientas integradas de desarrollo web y de aplicaciones [17]. Proporciona un modelo de interfaz de usuario común (UI) para trabajar con herramientas.

Android. Su interfaz de usuario está programado en Java.

1.2.4. Selección del lenguaje de programación

En cuanto a lo que este proyecto respecta, la elección del lenguaje de programación se ha realizado teniendo en cuenta ciertos puntos importantes para el desarrollo y diseño de la interfaz gráfica de usuario. En primer lugar, el equipo de Uptech Sensing ya tenía diversos códigos desarrollados en C++ a la hora de iniciar este trabajo. Además, se buscó la compatibilidad con todos los componentes que se emplean en estos equipamientos. Uno de los componentes más críticos en cuanto a programación se refiere, son las tarjetas de adquisición (Keysight 1071A/5309A y Alazartech ATS860) las cuales, ya tienen sus propias librerías escritas en C++. Por supuesto se tuvo en cuenta el conocimiento previo del lenguaje de programación.

1.3. Entorno de desarrollo integrado (IDE)

Básicamente un entorno de desarrollo integrado es una aplicación cuya finalidad es facilitar la programación de otras aplicaciones [18]. Estos entornos ofrecen una interfaz con los servicios integrales necesarios para que un desarrollador o programador desarrolle su propia aplicación. Entre estos servicios integrales que un IDE posee podemos encontrar un editor de código fuente el cual en la mayoría de IDEs tiene un autocompletado inteligente del código (*IntelliSense*). Un compilador que transforma el código fuente escrito en un formato ejecutable por un ordenador. Un depurador que ayuda a depurar el programa a desarrollar y herramientas de automatización de compilación.

Este tipo de software es de gran ayuda en cuanto al manejo del lenguaje de programación C++ y las librerías utilizadas. Además de aportarnos rapidez y sencillez a la hora de compilar y depurar código, tal y como se verá en el Apéndice B0 más adelante, nos proporcionó un repositorio local (Git) con el que trabajamos durante este tiempo, salvando diariamente el código programado, pudiendo así volver a un estado anterior del programa en cualquier momento.

1.3.1. Eclipse

Eclipse es uno de tantos entornos de desarrollo integrados gratuitos, el cual está basado en Java y fue desarrollado por IBM. El IDE fue supervisado inicialmente por un consorcio de proveedores de software que buscaban crear y fomentar una nueva comunidad que complementara la comunidad de código abierto de Apache.

Hoy en día, Eclipse es administrado por Eclipse Foundation [17], una corporación sin fines de lucro cuyos miembros estratégicos incluyen CA Technologies, IBM, Oracle y SAP. La fundación proporciona servicios de administración de infraestructura y propiedad intelectual (IP) a la comunidad de Eclipse y ayuda a los miembros de la comunidad a promocionar y comercializar productos de software comercial basados en Eclipse.

Los widgets de Eclipse están implementados por una herramienta para Java llamada Standard Widget Toolkit, a diferencia de la mayoría de las aplicaciones Java, que usan las opciones estándar Abstract Window Toolkit (AWT) o Swing. El entorno de desarrollo integrado de Eclipse emplea módulos para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Hoy en día, Eclipse es capaz de usar otros lenguajes de programación como son C/C++ y Python, y trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos.

Por diseño, la plataforma no proporciona una gran cantidad de funcionalidad para el usuario final por sí misma. El valor de la plataforma es lo que fomenta: desarrollo rápido de características integradas basadas en un modelo de plugin.

Finalmente, se debe mencionar que está diseñado para ejecutarse en múltiples sistemas operativos, donde los complementos pueden programarse en las API portátiles de Eclipse y ejecutarse sin cambios en cualquiera de los sistemas operativos admitidos.

1.3.2. Netbeans

NetBeans es un entorno de desarrollo integrado de código abierto (IDE) para desarrollar con Java, PHP, C++ y otros lenguajes de programación, aunque fue desarrollado principalmente para Java. NetBeans está codificado en Java y se ejecuta en la mayoría de los sistemas operativos con una máquina virtual Java (JVM) [19].

NetBeans utiliza componentes, también conocidos como módulos, para habilitar el desarrollo de software. Dichos módulos pueden ser desarrollados independientemente dado que las aplicaciones construidas a partir de estos pueden ser extendidas agregándole nuevos módulos. Estos módulos IDE incluyen NetBeans Profiler, una herramienta de diseño de interfaz gráfica de usuario (GUI) y NetBeans JavaScript Editor.

Entre sus características, destacan: gestión de la interfaz de usuario (menús y barras de herramientas), gestión de configuración de usuario, gestión de almacenamiento, gestión de ventana, marco asistente, librería visual de Netbeans y herramientas de desarrollo integrado. Además, este entorno provee soporte para la creación de aplicaciones orientadas a servicios (SOA), incluyendo herramientas de esquemas XML (*Extensible Markup Language*) y un editor WSDL (*Web Services Description Language)* y permite crear aplicaciones Web con PHP (*Hypertext Preprocessor*).

1.3.3. Selección del IDE

En principio, una diferencia a tener en cuenta es que NetBeans trae por defecto un editor visual con el que podrás crear pantallas de una forma muy sencilla e intuitiva y Eclipse no lo trae (se le puede añadir como plugin) aunque este punto no se tuvo en cuenta ya que teníamos decidido abarcar este tipo de acción mediante GTK+ y Glade.

En cuanto al soporte no se encontró mucha diferencia entre ambos dado que ambos cuentan con soporte multiplataforma, así como soporte para múltiples idiomas como pueden ser C/C++, Java, JavaScript y PHP.

Figura 1.5: Entornos de desarrollo integrados

En este punto nos basamos básicamente en gustos. Eclipse en su versión Oxygen fue el entorno de desarrollo elegido debido a su anterior uso durante el grado y a su facilidad de uso del SVN local [Apéndice B] además de trabajar con distintas perspectivas, con lo que puedes tener distintas configuraciones de tu pantalla de trabajo según tu tipo de proyecto o según la acción que vayas a desarrollar en ese momento (debuggear, sincronizar con SVN, escribir código...). Puedes personalizar cada perspectiva y crear nuevas.

1.4. Interfaz gráfica de usuario (GUI)

La GUI es un tipo de interfaz de usuario que permite a los usuarios interactuar con dispositivos electrónicos a través de iconos gráficos e indicadores visuales. Está formado por imágenes y objetos gráficos, que representan la información y acciones que se encuentran en la interfaz. Su objetivo es el de crear un entorno visual fácil de usar para que fluya la comunicación con el sistema operativo.

1.4.1. GTK+

GTK+ o GIMP Toolkit, es un conjunto de herramientas multiplataforma para crear interfaces gráficas de usuario [20]. Está escrito en C, pero se ha diseñado desde cero para admitir una amplia gama de idiomas, no solo C/C++, sino también Perl, Python, Java, etc. Al ofrecer un conjunto completo de widgets, GTK+ es adecuado para proyectos que van desde pequeñas herramientas únicas hasta completas suites de aplicaciones.

GTK+ además de contar con una API fácil de usar, es software libre y parte del Proyecto GNU. Los términos de la licencia de GTK+ permiten que todos los desarrolladores, incluso aquellos que desarrollan software propietario, lo utilicen sin ningún tipo de licencia o royalties.

En cuanto a la estabilidad, cuenta con el respaldo de una gran comunidad de desarrolladores y tiene mantenedores principales de compañías como Red Hat, Novell, Codethink, Endless Mobile e Intel.

Originalmente, se desarrolló para X Window System, pero ha crecido a lo largo de los años para lograr, a día de hoy, poder usarlo en: GNU / Linux y Unix, Windows y MacOSX.

Widgets

GTK + tiene una colección completa de widgets e interfaces para usar en su aplicación. Algunos de ellos se destacan a continuación:

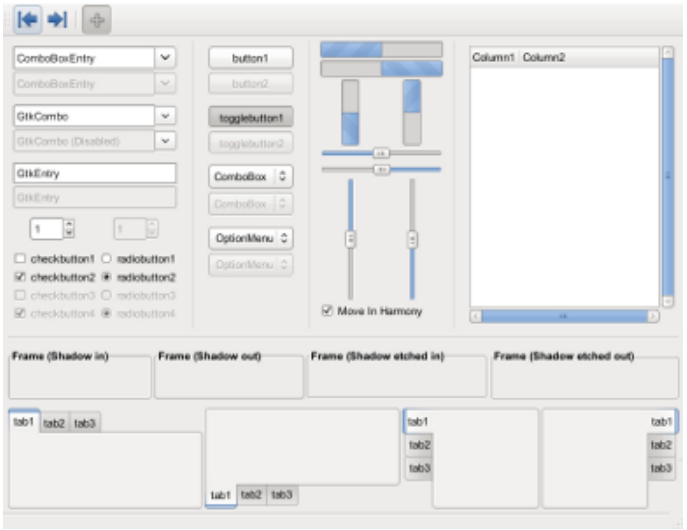


Figura 1.6: Diversos widgets de GTK+

- Windows (ventana normal o diálogos)
- Displays (etiquetas, imágenes, barras de progreso, barras de estado)
- Botones y conmutadores (botones de verificación, botones de radio, botones de alternancia y botones de enlace)
- Escalas numéricas (escalas horizontales o verticales y botones giratorios) e ingreso de datos de texto

- Editor de texto multilínea
- Visor de árbol y lista (con representadores personalizables y separación de modelo/vista)
- Cuadro combinado (con o sin una entrada)
- Menús (con imágenes, botones de radio y elementos de verificación)
- Barras de herramientas (con botones de radio, botones de alternar y botones de menú)
- GtkBuilder (crea su interfaz de usuario desde XML)
- Selectores (selección de color, selector de archivo, selección de fuente)
- Diseños (widget tabulado, widget de tabla, widget de expansión, marcos, separadores y más)
- Icono de estado (área de notificación en Linux, icono de bandeja en Windows)
- Documentos utilizados recientemente (menú, diálogo y administrador)

1.4.2. Glade

Glade es un software gratuito y de código abierto distribuido bajo la Licencia Pública General de GNU. Es una herramienta para el desarrollo de interfaz gráfica de usuario para el kit de herramientas GTK+ y el entorno de escritorio GNOME. Es definida como una herramienta de tipo RAD (del inglés *rapid application development*) [21].

Desde su tercera versión, Glade es independiente del lenguaje de programación, y produce código XML. Estos documentos XML se pueden usar junto con el objeto GtkBuilder, el cual construye una interfaz desde una definición UI XML, para crear una instancia del formulario usando GTK+ [22]. Al usar GtkBuilder, los archivos XML de Glade se pueden usar en numerosos lenguajes de programación, incluidos C, C ++, C #, Java, Perl, Python, entre otros. Dicha clase GtkBuilder permite que las interfaces de usuario se diseñen sin escribir código. La clase describe la interfaz en un archivo de lenguaje de marcado extensible (XML) y luego carga la descripción XML en tiempo de ejecución y crea los objetos automáticamente. La descripción de la interfaz de usuario es independiente del lenguaje de programación que se utiliza.

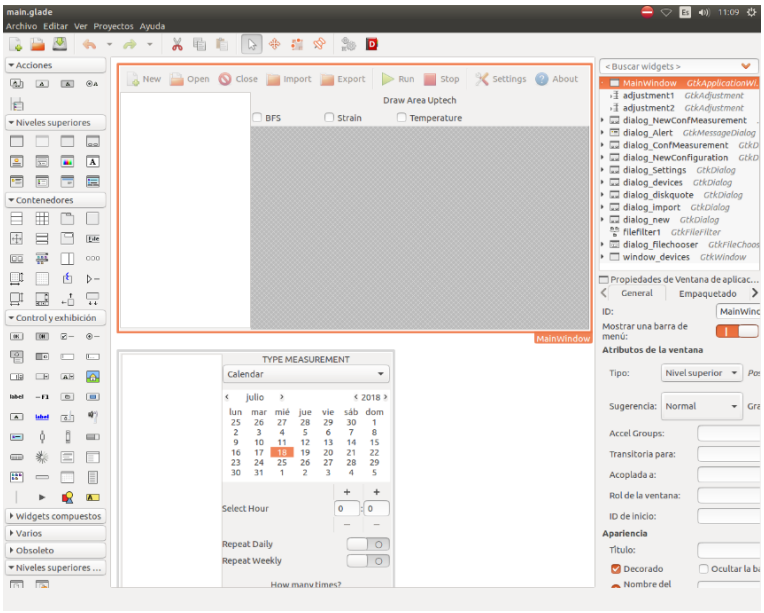


Figura 1.7: Interfaz de Glade.

El interfaz de Glade tiene 5 zonas claras de creación y edición de widgets.

En la zona superior tenemos una paleta con herramientas típicas como puede ser nuevo archivo, abrir archivo, guardar, deshacer y rehacer, copiar, cortar, pegar…

A la izquierda tenemos un explorador dividido en desplegables de diferentes tipos de widgets. El funcionamiento es simple y con solo pulsar los widgets de nivel superior serán añadidos al archivo XML. En los de tipo contenedor, control, varios y demás, seleccionamos el widget que deseemos y lo aplicamos en la zona que queramos de la ventana o diálogo.

La zona principal es la que tenemos en el centro de Glade. En esta zona se irán añadiendo los widgets que vayamos seleccionando y podremos ir poco a poco viendo cómo vamos construyendo nuestro interfaz.

En el lado derecho se encuentran dos zonas claramente diferenciables. La zona superior permite realizar una vista en árbol de los widgets que se están editando y marca su ID. En la zona de debajo, se encuentran las propiedades de estos widgets diferenciando propias y comunes.

Capítulo 2

Desarrollo del GUI

2.1. Requisitos del GUI

Descripción del funcionamiento de los interrogadores

El equipo interrogador permite medir magnitudes físicas, tales como la deformación unitaria y temperatura, gracias al uso de la fibra óptica y el fenómeno físico de dispersión Brillouin estimulada (stimulated Brillouin scattering, SBS). Este fenómeno físico permite medir en cada uno de los puntos de la fibra óptica obteniendo una medida distribuida, es decir, el perfil de temperatura o deformación a lo largo de la fibra óptica.

Dicho equipo, utiliza la topología BOTDA (Brillouin optical time domain analysis), el cual, es el sensor distribuido más usado y evolucionado. Las resoluciones espaciales que se consiguen pueden llegar hasta el centímetro, y las distancias que son capaces de medir llegan a varias decenas de kilómetros con un solo sensor, obteniendo por ejemplo 50.000 puntos de medida en 50 km con una resolución espacial de 1 m.

El principio de funcionamiento básico de los sistemas BOTDA, se basa en la contra propagación de dos ondas en la fibra óptica bajo test tal y como vemos en la Figura 2.1.

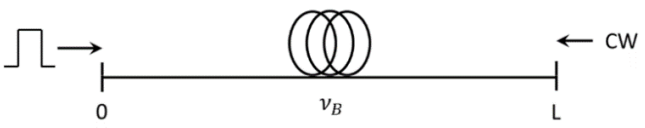


Figura 2.1: Principio de funcionamiento de los sistemas BOTDA.

La onda continua (continuous wave, CW) se propaga con una frecuencia v_S y los pulsos ópticos a una frecuencia v_P tal que la diferencia frecuencial entre ambas ondas sea la correspondiente al desplazamiento en frecuencia Brillouin de la fibra, $v_B = v_P - v_S$. Como consecuencia, la CW será amplificada por SBS con las características del punto de la fibra donde interaccione con el pulso. La señal BOTDA resultante del aumento en potencia de la onda continua, se recibirá como función del tiempo tal y como se observa en la Figura 2.2 Esta función puede traducirse al dominio espacial con la siguiente fórmula:

$$z = \frac{ct}{2n}$$

donde *z* es la distancia donde interactúa el pulso con la CW, *c* es la velocidad de la luz, *n* el índice de refracción de la fibra y el factor 2 es consecuencia del tiempo que tarda al pulso en propagarse hasta el punto de interacción y el tiempo de propagación de la señal BOTDA resultante hasta el receptor.

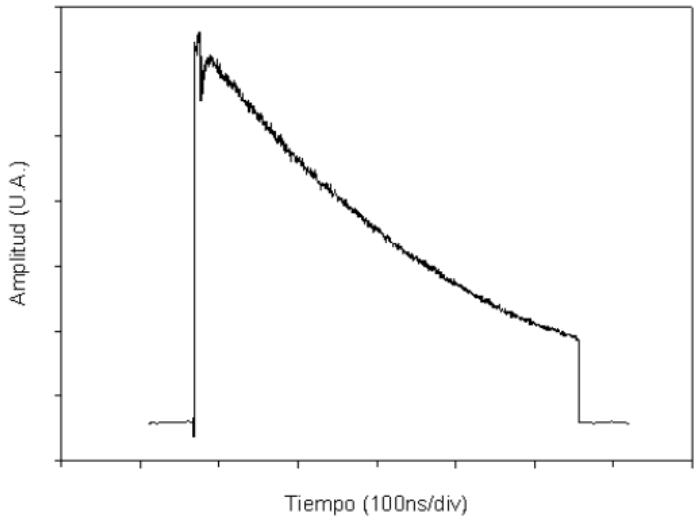


Figura 2.2: BOTDA en ganancia.

La señal BOTDA representada en la Figura 2.2 muestra la interacción Brillouin correspondiente a la frecuencia diferencial dada entre la onda de bombeo (pulsos ópticos) y la Stokes (CW). Si esta diferencia frecuencial se varía, entonces se recibe la

señal correspondiente a otra frecuencia del espectro en ganancia Brillouin. Consecuentemente, mediante variaciones de la frecuencia de una de las dos ondas se puede reconstruir el espectro Brillouin para cada punto de la fibra. El valor de las magnitudes temperatura y deformación unitaria se encuentra inspeccionando el valor del desplazamiento del máximo del espectro Brillouin (BFS).

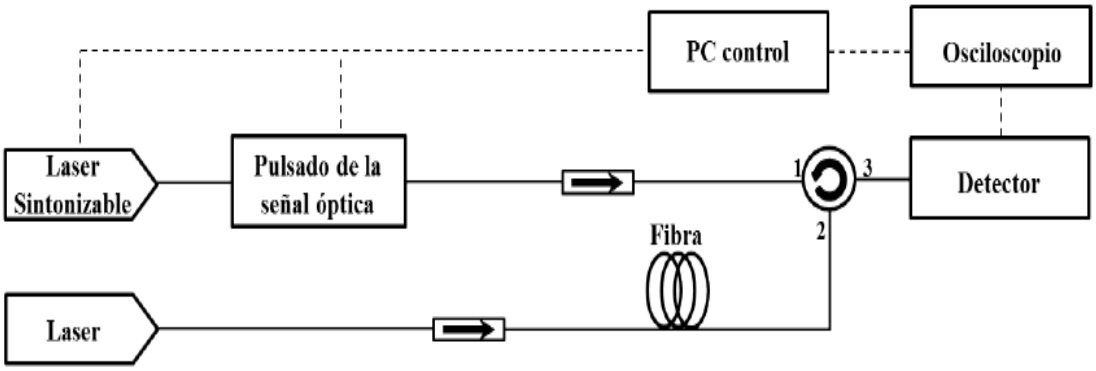


Figura 2.3: Simplificación sistema BOTDA.

Un posible esquema de sensor BOTDA se muestra en la Figura 2.3. En la rama superior se pulsa la señal óptica cuya frecuencia es controlada a través de un láser sintonizable que es el encargado de realizar el barrido en frecuencia. Este barrido en frecuencia tiene como objetivo variar la diferencia frecuencial entre la onda pulsada y la CW que se genera en la rama inferior. La onda continua se propagará a lo largo de la fibra interactuando con los pulsos que transfieren la potencia a la CW y generan la señal BOTDA para cada frecuencia sintonizada. Esta señal se detectará y observará en el osciloscopio. Cada una de las señales con distinta frecuencia se almacenará en un PC donde se reconstruirá el espectro Brillouin. A partir de este punto, el interfaz y el software de post-procesado son los responsables de mostrar al usuario los resultados obtenidos mediante el interrogador, siendo éste el contexto en el que se enmarca el trabajo fin de grado.

Necesidades de configuración y control de los equipos interrogadores desde el GUI

Básicamente existen tres puntos que se deben tener en cuenta en el desarrollo de la interfaz del GUI para este tipo de interrogadores:

1. Sistemas de adquisición. Actualmente se emplean tarjetas de adquisición de diferentes fabricantes para capturar la señal detectada. Estas tarjetas cuentan habitualmente con librerías escritas en C++ para su configuración y manejo. Concretamente, se dispone de tres tipos de tarjetas: Alazartech ATS860, Keysight 1071A y 5309A. El kernel que permite emplear todas las tarjetas bajo un mismo ordenador es el kernel 4.4 (con soporte hasta Abril de 2021).

2. Configuración de características de los interrogadores, por ejemplo, modificar la frecuencia de la onda pulsada, cambio de la duración del pulso, la tasa de repetición de los pulsos, así como el encendido de los diferentes módulos del equipo interrogador se realizarán mediante llamadas a funciones que se ejecutan en el propio equipo interrogador gracias a la conexión serie entre el ordenador y el equipo. Cada uno de estos parámetros será necesario configurarlo desde el interfaz de usuario.

3. Debido a que este tipo de equipamiento permite medir con diferentes configuraciones, por ejemplo, diferentes longitudes de fibra óptica con multitud de resoluciones espaciales, es necesario que el GUI cuente con un sistema que permita la configuración del sensor de medida, así como la configuración de medida, es decir, en qué momento se pretende medir o durante cuánto tiempo.

2.2. Descripción de tipos de elementos para el desarrollo de un GUI

Durante el transcurso del trabajo se han utilizado numerosos widgets de GTK+ para el diseño y desarrollo de la aplicación. Estos widgets nos dan fácil acceso a funciones frecuentemente usadas y nos proveen de información visual interesante para nuestra aplicación. De aquí en adelante nos centraremos es una breve explicación de cada uno de los elementos utilizados para el desarrollo del software, no sin antes, dar a conocer la descripción XML generada automáticamente por Glade.

2.2.1. Descripción XML

Como ya se ha comentado más atrás en este documento, Glade permite distribuir widgets en la pantalla y luego guardar una descripción XML de la distribución. Dejando claro que esta descripción la genera Glade automáticamente conforme se van añadiendo widgets, se recurrirá a una breve explicación de esta descripción XML para introducir tras ello los widgets y la forma en la que se han utilizado.

El código base de esta descripción se basa en las siguientes líneas donde queda descrita la versión XML, la codificación y la librería usada y su versión. Es a partir de aquí donde Glade va añadiendo los widgets conforme son utilizados.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <requires lib="gtk+" version="3.12"/>
</interface>
```

Dentro de las etiquetas <interface> al mismo nivel que <requieres>, se añadirán el objeto principal de nuestra ventana al igual que los diálogos que se vayan implementando. Como se observa en el siguiente fragmento de código más adelante, en el objeto principal de la ventana se puede ver un id que es utilizado para crear una instancia del widget de GTK+ y comunicarnos mediante la programación del código fuente y la descripción XML mediante GtkBuilder

También se tienen diversas propiedades de este objeto que se podrán ir cambiando como ya se ha explicado anteriormente en Glade.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <requires lib="gtk+" version="3.12"/>
  <object class="GtkApplicationWindow" id="MainWindow">
    <property name="can_focus">False</property>
    <property name="window_position">center</property>
  </object>
</interface>
```

Dentro de estos objetos se irán añadiendo sin límite diferentes widgets mediante la etiqueta <child>. Se debe tener en cuenta que ciertos widgets no podrán ser añadidos dentro de otros, es decir, no podrás añadir una ventana dentro de una etiqueta de texto como es obvio. Este último apunte es debido a que es posible modificar el archivo XML a nuestro antojo, pero siempre manteniendo un criterio especifico. La descripción XML resumida de un par de objetos quedaría de la siguiente manera:

```
<child>
    <object class="GtkBox" id="box3">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <child>
            <object class="GtkToolbar" id="toolbar1">
                <property name="visible">True</property>
                <property name="can_focus">False</property>
                <property name="hexpand">True</property>
                <property name="show_arrow">False</property>
                .
                .
                .
            </object>
        </child>
    </object>
</child>
```

Como se ha visto, la descripción que nos aporta Glade es sencilla y muy efectiva ante la compatibilidad de lenguajes de programación. Independientemente del lenguaje en que se está programando, se permite la comunicación con la descripción XML sin ningún problema. Este aspecto es de gran ayuda en caso de que en un futuro se quisiera portar el código de C++ a otro que pudiese ser de mayor interés.

2.2.2. Descripción elementos GTK+

Antes de aportar información acerca de la descripción de los elementos manifestamos el uso de dos objetos de GTK+ totalmente necesarios para nuestra aplicación. Estos dos objetos no se han implementado con la ayuda de Glade sino mediante código fuente en C++. Tras estos dos elementos se continuará con los widgets de GTK+.

GtkApplication

GtkApplication es la clase base de una aplicación GTK+ y su objetivo principal es separar el programa de main() [23]. Maneja la inicialización de GTK+, la gestión de sesiones y administra una lista de ventanas de nivel superior cuyo ciclo de vida se vincula automáticamente al ciclo de vida de la aplicación.

La filosofía de GtkApplication resuelve que a las aplicaciones se les informe qué debe suceder y cuándo debe suceder en respuesta a las acciones del usuario. Con este fin, GtkApplication expone un conjunto de señales a las que una aplicación debe responder, aunque en nuestro caso no se han utilizado todas. Este conjunto de señales son las siguientes:

- startup: Configura la aplicación cuando comienza por primera vez
- shutdown: Realiza tareas de apagado
- activate: Muestra la primera ventana predeterminada de la aplicación. Esto corresponde a la aplicación lanzada por el entorno de escritorio.
- open: abre archivos y los muestra en una nueva ventana.

La utilización de GtkApplication se ha llevado a cabo mediante la definición de la clase principal como vemos en el siguiente fragmento del fichero de cabecera ApplicationScript.h, obviando fragmentos de código de poco interés en cuanto a la explicación.

```
class ApplicationScript : public Gtk::Application
{
protected:
    ApplicationScript();

    void on_startup() override;
    void on_activate() override;
};
```

En el fichero fuente se han aplicado las funciones correspondientes.

```
void ApplicationScript::on_startup(){ … }

void ApplicationScript::on_activate(){ … }
```

GtkBuilder

Un GtkBuilder es un objeto auxiliar que lee las descripciones textuales de una interfaz de usuario y crea una instancia de los objetos descritos [24]. En el caso de este proyecto, en el fichero fuente de GtkApplication se procede de la siguiente manera:

```
m_refBuilder = Gtk::Builder::create();
m_refBuilder->add_from_file("MainWindow/Glade/main.glade");
m_mainWin = nullptr;
m_refBuilder->get_widget_derived("MainWindow", m_mainWin);
add_window(*m_mainWin);
```

Se ha creado un objeto builder donde cargamos los ficheros Glade de las ventanas con add_from_file(). Tras ello se definen los punteros de cada ventana para que carguen los datos correspondientes y para finalizar se añaden las ventanas mediante add_window().

La función get_widget_derived() obtiene un widget cuyos detalles se especifican en el archivo GtkBuilder, pero que implementa su propia clase derivada, es decir, se pasa a las clases propias de cada ventana. Esto permite manejar desde cada fichero fuente de cada ventana su propio archivo XML. La clase del fichero fuente de la ventana debe incluir un constructor como el siguiente:

```
MainWindow::MainWindow(BaseObjectType* cobject, const
Glib::RefPtr<Gtk::Builder>& refGlade) :
    Gtk::ApplicationWindow(cobject),
    builder(refGlade)
{
}
```

Con esto definido, sería tan sencillo como cargar los widgets al builder de la ventana siendo el primer argumento el id del widget en el archivo XML y el segundo argumento el nombre del puntero que hemos definido en el fichero de cabecera.

```
Gtk::Entry *ent_fiberdistance; //Puntero en fichero .h de la ventana

builder->get_widget("ent_fiberdistance", ent_fiberdistance);
```

Tras conocer estos dos principios básicos de toda aplicación GTK+, se continuará con la descripción general de los widgets. Para desarrollar el proyecto, según la disposición de estos widgets en Glade, tenemos diferentes tipos: de nivel superior, contenedores, control y exhibición, widgets compuestos, etc. Cada uno de estos widgets tiene unas propiedades comunes y otras propias de cada widget.

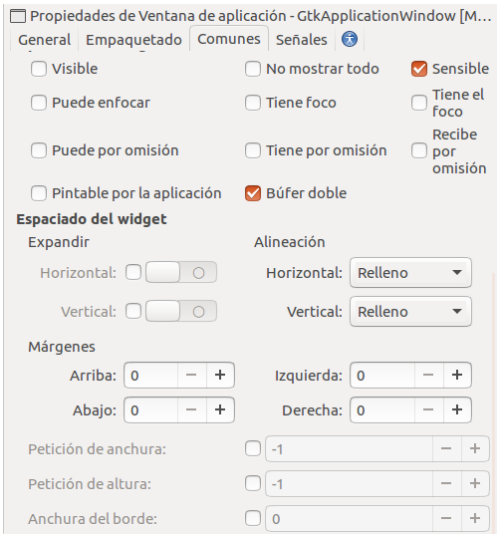


Figura 2.4: Propiedades comunes

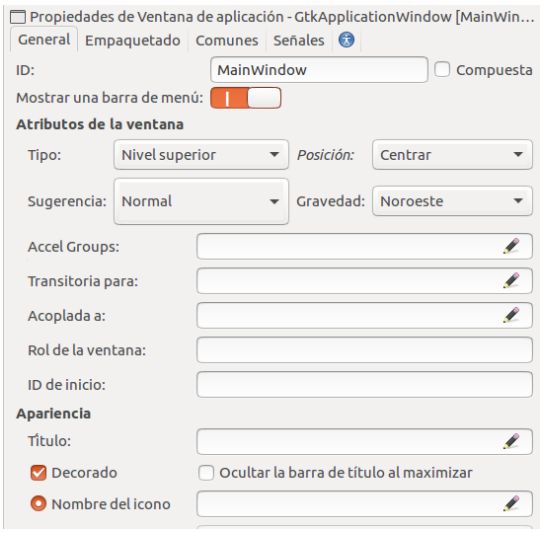
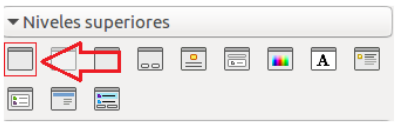


Figura 2.5: Propiedades propias

NIVELES SUPERIORES

GtkWindow

Una GtkWindow [25] es una ventana de nivel superior que puede contener otros widgets. Normalmente, las ventanas presentan decoraciones que están bajo el control del sistema de ventanas y permiten al usuario manipular la ventana (redimensionarla, moverla, cerrarla, ...). En Glade se añade pulsando en el botón de la Figura 2.6.




```
<object class="GtkWindow" id="window_devices">
    <property name="can_focus">False</property>
    <property name="window_position">center</property>
    <property name="default_width">460</property>
    <property name="default_height">400</property>
    <property name="gravity">center</property>
    <child>
      ...
    </child>
</object>
```

Figura 2.6: Botón GtkWindow en Glade y descripción XML.

GtkApplicationWindow

GtkApplicationWindow [26] es una subclase GtkWindow que ofrece alguna funcionalidad adicional para una mejor integración con las características de GtkApplication. Puede manejar tanto el menú de la aplicación como la barra de menú. Para lo que a este proyecto respecta, esta clase será la correspondiente a cada ventana de la aplicación. Para usar esta ventana en Glade se pulsa en el botón de la Figura 2.7.

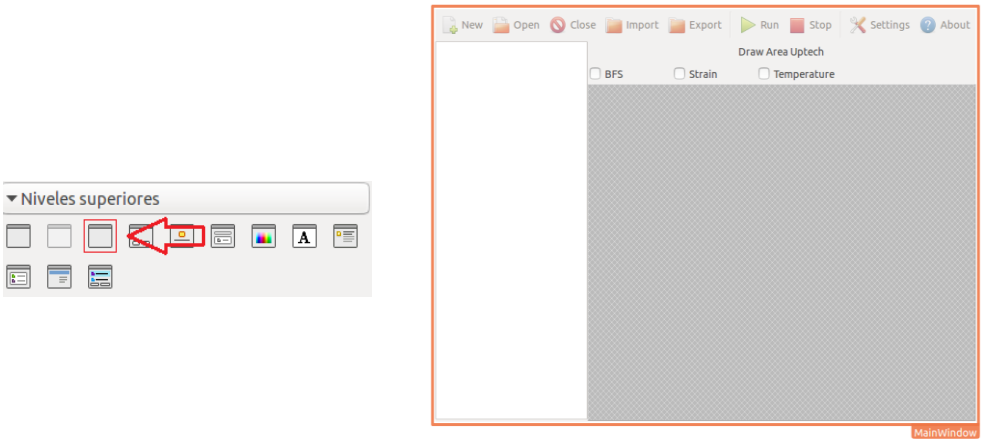


Figura 2.7: Botón GtkApplicationWindow en Glade y ventana MainWindow.

GtkDialog

Los cuadros de diálogo son una forma conveniente de solicitar al usuario una pequeña cantidad de entrada, por ejemplo, para mostrar un mensaje o hacer una pregunta [27]. En nuestro caso ha sido usado como entrada de valores de configuración del sensor, de medidas, selección de dispositivo para importación y exportación de archivos, etc. Su correspondiente botón en Glade y un ejemplo podemos verlo en la Figura 2.8.

Figura 2.8: Botón GtkDialog en Glade y diálogo ejemplo.

```
// Con este while(true) lo que conseguimos es que el diálogo aparezca
siempre hasta que se rellenen los valores, se presione cancelar o la X
while (true)
{
    //Mostramos diálogo y esperamos respuesta del usuario
    int result = NewConfigurationDialog->run();

    //Manejador de la respuesta
    switch(result)
    {
        case(0): // Accept clicked
        {
            …
            return;
        }
        case(1): // Cancel clicked
        {
            …
            return;
        }
        default: // X clicked
        {
            …
            return;
        }
    }
}
```

Cómo vemos en el código fuente descrito, se lanza el diálogo esperando una respuesta mediante la estructura de control switch. En función del botón pulsado por el usuario se podrá realizar diferentes acciones. Los id de las respuestas se han aplicado mediante una propiedad de los botones en Glade.

GtkFileChooserDialog

GtkFileChooserDialog [28] es un cuadro de diálogo adecuado para usar con los comandos "Archivo/Abrir" o "Archivo/Guardar como". Expone la interfaz GtkFileChooser dentro de un GtkDialog donde GtkFileChooser permite accesos directos a varios lugares en el sistema de archivos. El botón en Glade es el de la Figura 2.9.

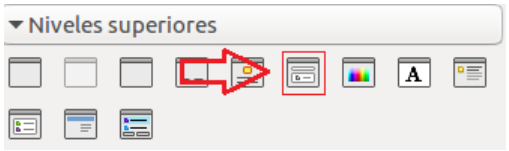


Figura 2.9: Botón GtkFileChooserDialog en Glade.

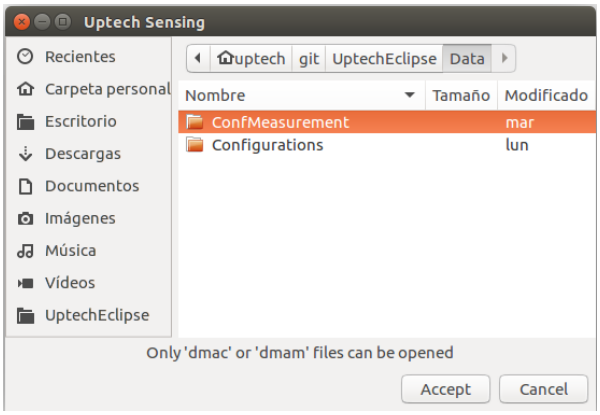


Figura 2.10: Diálogo de selección de ficheros.

Este widget al ser de tipo diálogo se aplica el mismo método de estructura de control que en GtkDialog. Para la apertura de ficheros se ha aplicado un filtro GtkFileFilter para archivos dmac y dmam que será explicado más adelante.

GtkMessageDialog

GtkMessageDialog [29] presenta un diálogo con algún texto de mensaje. Es un widget de conveniencia ya que podría construir el equivalente de GtkMessageDialog desde GtkDialog sin demasiado esfuerzo, pero GtkMessageDialog guarda el tipado. En la Figura 2.11 vemos el correspondiente botón en Glade y un ejemplo de GtkMessageDialog.

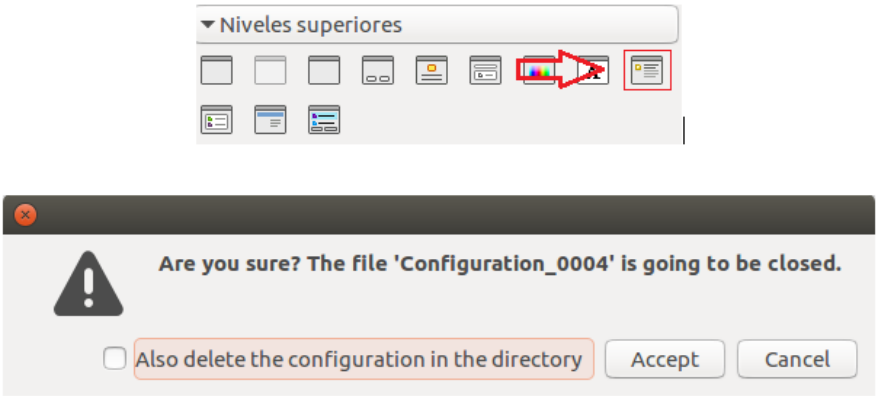


Figura 2.11: Botón GtkMessageDialog en Glade y diálogo de mensaje ejemplo.

El archivo es seleccionado mediante un iterador de TreeView y se ha vuelto a utilizar el mismo sistema de bucles que en los anteriores diálogos descritos en este documento.

CONTENEDORES

GtkBox

GtkBox [30] es un contenedor que organiza otros widgets en un área rectangular. Dentro de este contenedor podrás dividirlo en pequeños contenedores horizontales o verticales. GtkBox utiliza una noción de embalaje, el cual, se refiere a agregar widgets con referencia a una posición particular en un contenedor(GtkContainer). En el diálogo de la Figura 2.12 se ve un GtkBox con dos elementos (en este caso, botones) dispuestos en vertical.

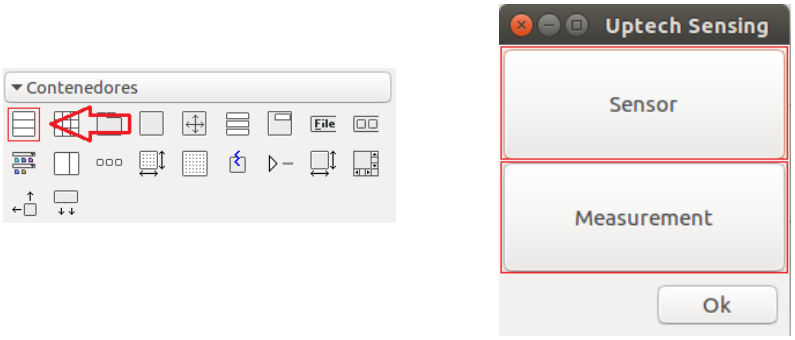


Figura 2.12: Botón GtkBox en Glade y diálogo ejemplo de dos botones en vertical.

A este tipo de widget se le añaden tantos contenedores como se necesiten mediante las propiedades generales del GtkBox en Glade. Cómo se ve en la Figura 2.13 la orientación está en vertical correspondiendo con la figura superior.

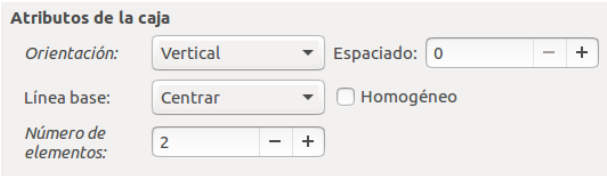


Figura 2.13: Atributos del GtkBox.

GtkNoteBook

El widget GtkNoteBook [31] es un GtkContainer cuyos elementos secundarios son páginas que se pueden cambiar entre etiquetas de pestaña a lo largo de un borde. Como se ve en la Figura 2.14 tenemos dos páginas, Settings y DiskQuote.

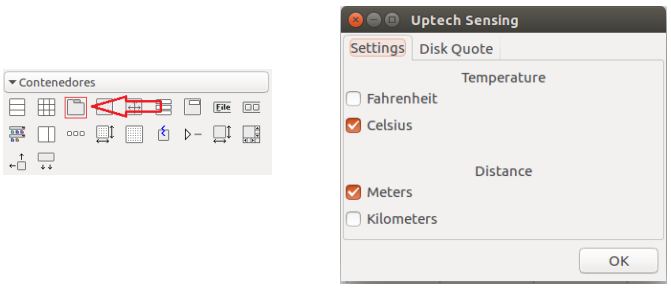


Figura 2.14: Botón GtkNoteBook en Glade y diálogo ejemplo de notebook.

GtkToolbar

GtkToolbar genera barras de botones y otros widgets [32]. Los elementos de la barra de herramientas se pueden agrupar visualmente al agregar instancias de GtkSeparatorToolItem. De forma predeterminada, una barra de herramientas puede reducirse, sobre lo cual se agregará un botón de flecha para mostrar un menú de desbordamiento que ofrece acceso a cualquier elemento secundario.

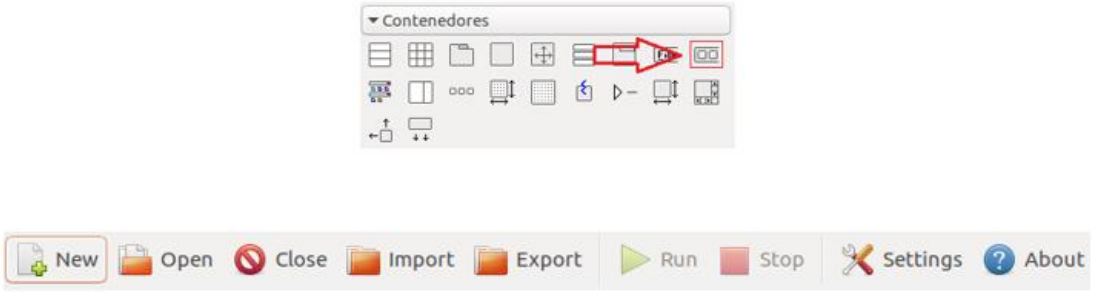


Figura 2.15: Botón GtkToolbar en Glade y ejemplo de paleta de herramientas.

Se pueden añadir tantos elementos como se quiera y el propio contenedor tiene una librería de iconos y separadores. En cuanto al código necesario cabe destacar que para usar cada botón se añade una acción a una función descrita. El nombre de la acción deberá ser del estilo win.nombre para que la ventana pueda utilizar dicha acción.

```
//Añadimos la acción (win.new) del botón de la barra a una función
add_action("new", sigc::mem_fun(*this, &MainWindow::on_new_clicked));
```

GtkScrolledWindow

GtkScrolledWindow [33] es un contenedor que acepta un único widget secundario, lo hace desplazable usando barras de desplazamiento agregadas internamente o ajustes asociados externamente y, opcionalmente, dibuja un marco alrededor del elemento secundario. En este caso se ha usado para añadir los TreeView, de los cuales se hablará más adelante, y así poder desplazarse por la lista.

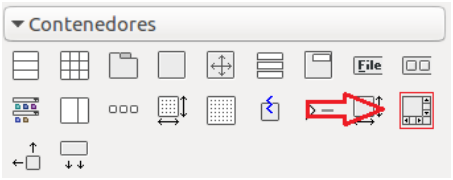


Figura 2.16: Botón GtkScrolledWindow en Glade

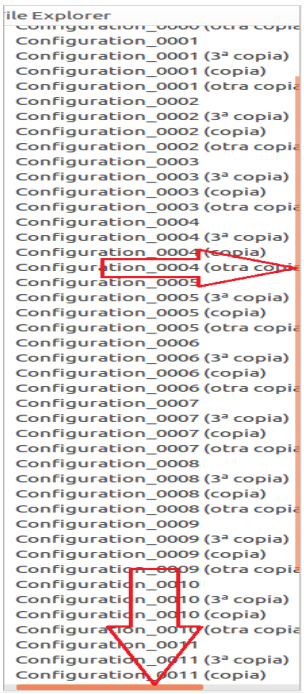
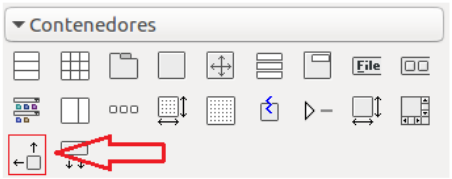


Figura 2.17: Scrolled Window.

GtkAlignment

El widget GtkAlignment controla la alineación y el tamaño de su widget hijo [34]. Tiene cuatro configuraciones: xscale, yscale, xalign y yalign. La configuración de la escala se usa para especificar cuánto debe expandirse el widget hijo para llenar el espacio asignado a GtkAlignment. La configuración de alineación se utiliza para colocar el widget hijo dentro del área disponible. Usado para la zona de dibujado.




```
<object class="GtkAlignment" id="align_multipleFit">
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <child>
        <placeholder/>
    </child>
</object>
```

Figura 2.18: Botón GtkAlignment en Glade y descripción XML.

CONTROL Y EXHIBICIÓN

GtkButton

El widget GtkButton [35] generalmente se utiliza para activar una función que se invoca cuando se presiona el botón ya que emite una señal cuando se hace clic en él. GtkButton puede contener casi cualquier otro GtkWidget hijo estándar. El hijo más comúnmente usado es GtkLabel. En la Figura 2.19 se ven dos botones de un diálogo.

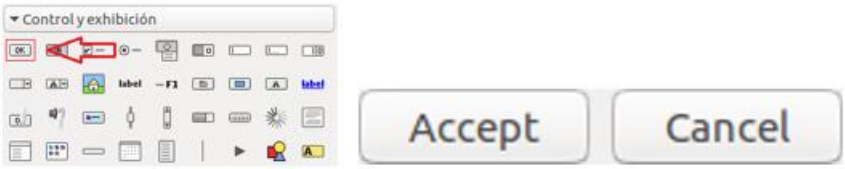


Figura 2.19: Botón GtkButton en Glade y dos botones ejemplo.

Los botones pueden recoger respuesta mediante un id aplicado con las propiedades de los botones en Glade o mediante una señal, normalmente un clic, añadida a una función como se ve a continuación.

```
btn_applyConfMeasure->signal_clicked().connect(sigc::mem_fun(*this,
&MainWindow::on_confmeasurement_apply));
```

GtkCheckButton

Un GtkCheckButton [36] coloca un GtkToggleButton (botón que conserva su estado) junto a un GtkLabel. Esto ha sido utilizado en diversos diálogos con el fin de verificar una acción como se puede ver en la figura

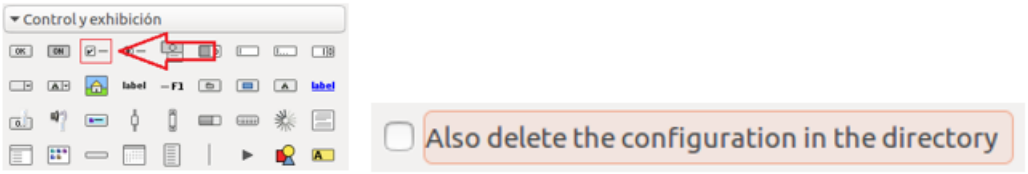


Figura 2.20: Botón GtkCheckButton en Glade y ejemplo de uso.

Este tipo de botón tiene dos posiciones: activo o desactivo; y se puede manejar mediante una función propia del widget la cual devolverá una variable de tipo booleano, verdadero en caso de que esté activo y falso en caso de no estarlo.

```
checkbuttonDelConfig->get_active();
```

GtkSwitch

GtkSwitch [37] es un widget que tiene dos estados: encendido o apagado. El usuario puede controlar qué estado debe estar activo haciendo clic en el área vacía o arrastrando el controlador. Un ejemplo lo tenemos en la figura.

Figura 2.21: Botón GtkSwicth en Glade y ejemplo de conmutador.

Al igual que pasa con los botones anteriores, el GtkSwitch tiene dos posiciones que se pueden alcanzar mediante la misma función. Además, ambos widgets tienen una función muy utilizada durante la aplicación y es la de darle un estado inicial, es decir, al iniciar cierto diálogo que el switch esté desactivado por defecto. Para esto usamos set_active(). True para active y false para desactivo.

```
swrepeatdaily->set_active(false);
```

GtkEntry

El widget GtkEntry [38] es un widget de entrada de texto de una sola línea. Si el texto ingresado es más largo que la asignación del widget, el widget se desplazará para que la posición del cursor sea visible. En la Figura 2.22 ejemplo vemos un GtkEntry con la palabra “Entry” escrita en su interior.

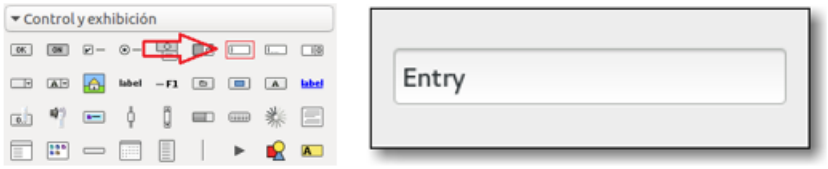


Figura 2.22: Botón GtkEntry en Glade y ejemplo de entrada.

En este tipo de widgets es necesario la obtención del texto introducido para su posterior uso. Este tipo de adquisición del texto se realiza mediante la función get_text() propia de GtkEntry.

```
info->get_text();
```

GtkSpinButton

Un GtkSpinButton [39] es una forma ideal de permitir al usuario establecer el valor de algún atributo y permitirle hacer clic en una de las dos flechas para incrementar o disminuir el valor mostrado. En la Figura 2.23 vemos como marcar la hora a la que realizar una medida.

Figura 2.23: Botón GtkSpinButton en Glade y ejemplo usado en formato hora.

En el caso que respecta, se utilizan dos GtkAdjustment para no dejar que el usuario añadiese valores no deseados. Al tratarse de horas del día, de 0 a 23 horas y de 0 a 59 minutos respectivamente.

GtkComboBox

Un GtkComboBox [40] es un widget que permite al usuario elegir de una lista de opciones válidas. GtkComboBox muestra la opción seleccionada. Cuando se activa, muestra una ventana emergente que permite al usuario hacer una nueva elección. Se puede ver un ejemplo en la Figura 2.24.

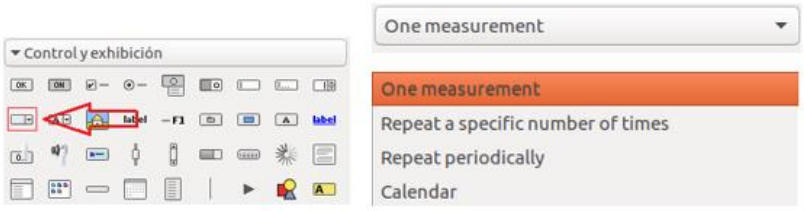


Figura 2.24: Botón GtkComboBox en Glade y ejemplo hecho en diálogo de medida.

Este tipo de widget es ligeramente diferente a los anteriores en cuanto a la obtención de datos. Cada elemento tendrá un valor de id el cual es capturado mediante get_active_id().

```
ent_choosetypemeasurement->get_active_id()
```

GtkImage

El widget GtkImage [41] muestra una imagen. Varios tipos de objetos pueden mostrarse como una imagen como por ejemplo GdkPixbuf ("buffer de píxeles") aunque en este proyecto se van a cargar imágenes .png. Un ejemplo se presenta en el recuadro rojo de la Figura 2.25 añadido mediante las propiedades de GtkImage en Glade.

Figura 2.25: Botón GtkImage en Glade y ejemplo de imagen en diálogo.

GtkLabel

El widget GtkLabel [42] muestra una pequeña cantidad de texto. La mayoría de las etiquetas se utilizan para etiquetar otro widget como GtkButton, GtkMenuItem o GtkComboBox.

Figura 2.26: Botón GtkLabel en Glade y ejemplo de etiqueta en diálogo.

Este es uno de los elemento más utilizados y más sencillos de usar. Básicamente con una función propia de GtkLabel se puede asignar cualquier texto además de formatearlo.

```
Label1.set_text("Esto es un ejemplo");
Label2.set_markup("<b>"+nombre+"</b> \n");
```

GtkSeparator

GtkSeparator [43] es un widget de separador horizontal o vertical, según el valor de la propiedad "orientación" , que se utiliza para agrupar los widgets dentro de una ventana. Muestra una línea con una sombra para que parezca hundida en la interfaz.

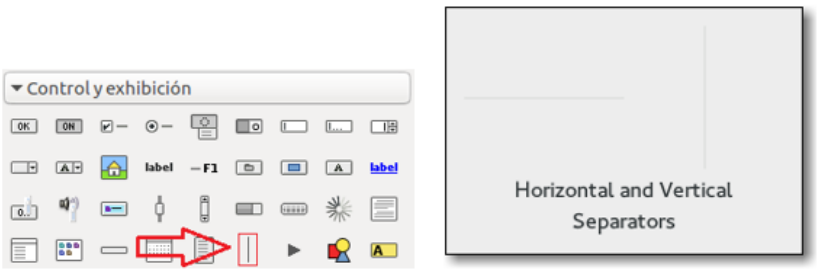


Figura 2.27: Botón GtkSeparator en Glade y ejemplo de separadores.

GtkCalendar

GtkCalendar [44] es un widget que muestra un calendario gregoriano, un mes a la vez. El mes y el año que se muestran actualmente se pueden modificar y el día exacto se puede seleccionar del mes mostrado además de colocar marcadores en el día seleccionado. La forma en que se muestra el calendario se puede modificar. Vemos un ejemplo en la Figura 2.28 del calendario utilizado para la configuración de medida.

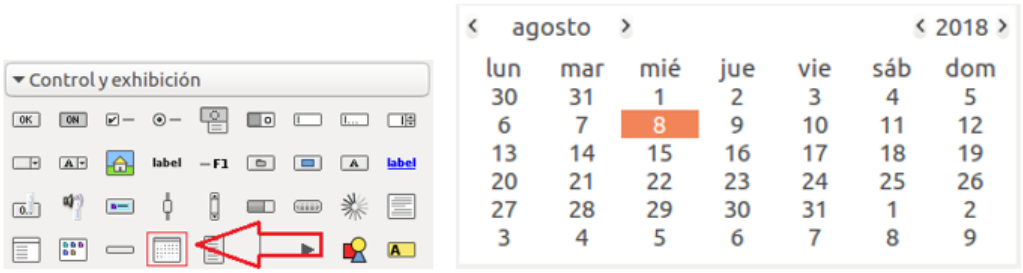


Figura 2.28: Botón GtkCalendar y ejemplo de calendario usado en la aplicación.

Para manejar este widget se ha utilizado unas funciones propias que han sido añadidas al Apéndice C pero cabe destacar la señal del doble clic sobre un día del calendario aplicándole una función.

```
Calendario->signal_day_selected_double_click().con-
nect(sigc::mem_fun(*this, &MainWindow::on_calendarpress_event));
```

GtkInfoBar

GtkInfoBar [45] es un widget que te permite mostrar un mensaje al usuario sin tener que mostrar un diálogo. Normalmente se muestra en la parte superior o inferior de un documento. Como vemos en la Figura 2.29, mostramos un mensaje para avisar al usuario de algo sin cambiar de diálogo.

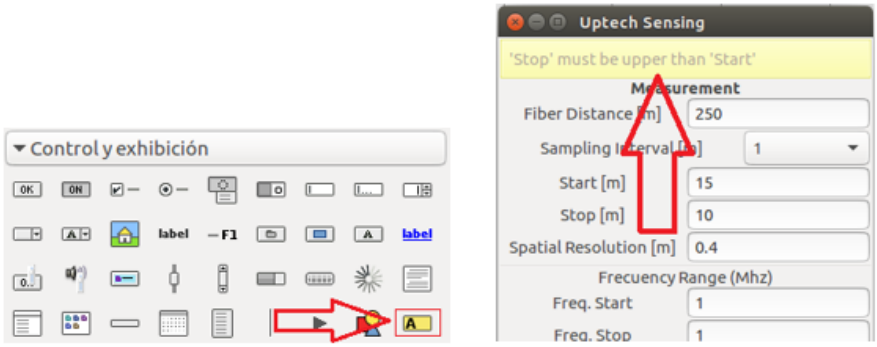


Figura 2.29: Botón GtkInfoBar en Glade y ejemplo en diálogo.

Cuando mostramos el diálogo es importante tener el infobar en oculto mediante la función hide(). Cuándo ocurre algo que necesite de un aviso, se añade texto a la etiqueta del infobar y se muestra.

```
lbl_infodialogconfig->set_text("Fill all boxes (Positive numbers)");
infobarconfig->show();
```

GtkTreeView

Un GtkTreeView y sus widgets asociados son extremadamente potentes a la hora de mostrar datos.

Cada GtkTreeView [46] tiene un GtkTreeModel asociado, que contiene los datos mostrados por el GtkTreeView. Cada GtkTreeModel puede ser utilizado por más de un GtkTreeView. Normalmente se utiliza las clases de modelo GtkListStore o GtkTreeStore, las cuales implementan la interfaz GtkTreeModel. GtkListStore contiene filas simples de datos, y cada fila no tiene hijos, mientras que GtkTreeStore contiene filas de datos, y cada fila puede tener filas secundarias. Para agregar datos al modelo usamos GtkListStore.append() o GtkTreeStore.append(), dependiendo de qué tipo de modelo se creó.

Para añadir el modelo al GtkTreeView usamos GtkTreeView.set_model(GtkTreeModel) y tras ello deberá saber cómo mostrar el modelo. Lo hace con columnas y renderizadores de celdas. En este caso solo se utilizan columnas. Un GtkTreeViewColumn es el objeto

que GtkTreeView utiliza para organizar las columnas verticales en la vista de árbol. En este proyecto se ha hecho una clase propia con una columna y esta clase al crear el modelo se la he añadido. Mediante GtkTreeModel::Row podemos ir añadiendo filas a esta columna. Un ejemplo con el modelo GtkTreeStore se puede ver en la Figura 2.30.

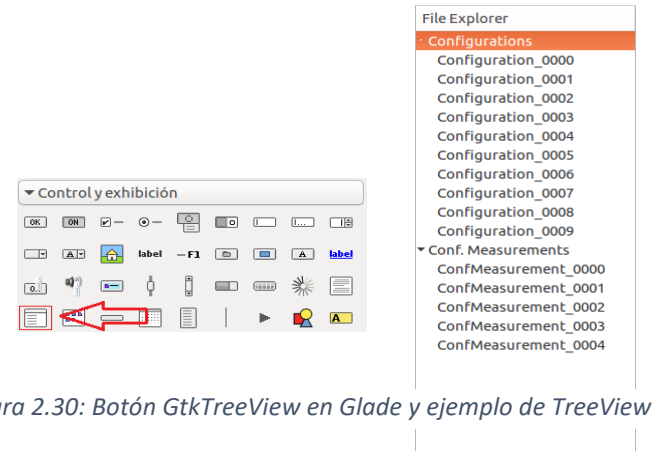


Figura 2.30: Botón GtkTreeView en Glade y ejemplo de TreeView en árbol.

Todo lo anteriormente explicado se puede ver más claro en el ejemplo de código, el cual, añade los ficheros de configuración al TreeView. Para añadir los ficheros de configuración de medida se procede de la misma manera.

```
//Creamos el Tree model y lo añadimos al TreeView
TreeModelTree = Gtk::TreeStore::create(m_Columns);
TreeModelTree->set_sort_column(m_Columns.m_col_name, Gtk::SORT_ASCEN-
DING);
treeviewMain->set_model(TreeModelTree);

//Rellenamos el TreeView's model
rowConfigurations = *(TreeModelTree->append());
rowConfigurations[m_Columns.m_col_name] = "Configurations";

if(DIR* dir = opendir(pathFilesConfigurations.c_str())){
    while(dirent* ent = readdir(dir) ){
        string nombre = ent->d_name;
        if(IsFile(nombre)){
            if(nombre.substr(nombre.find_last_of(".") + 1) == "dmac"){
                childrowConfigurations = *(TreeModelTree->ap-
pend(rowConfigurations.children()));
                childrowConfigurations[m_Columns.m_col_name] = nom-
bre.substr(0,nombre.find_last_of("."));
            }
        }
    }
    closedir(dir);
}
```

Al hacer clic derecho en uno de los elementos del TreeView se mostrará un GtkMenu [47] que básicamente es un GtkMenuShell que implementa un menú desplegable que consiste en una lista de objetos GtkMenuItem que pueden ser navegados y activados por el usuario para realizar funciones de la aplicación. Para añadir elementos a este menú se procede de la siguiente forma, descrita a continuación.

```
//Fill popup menu:
auto item = Gtk::manage(new Gtk::MenuI-
tem("_Edit", true));
item->signal_activate().con-
nect(sigc::mem_fun(*this, &MainWindow::edit_fi-
les) );
m_Menu_Popup.append(*item);

item = Gtk::manage(new Gtk::MenuItem("_View",
true));
item->signal_activate().con-
nect(sigc::mem_fun(*this, &MainWindow::view_fi-
les) );
m_Menu_Popup.append(*item);

...

m_Menu_Popup.accelerate(*this);
m_Menu_Popup.show_all();
```

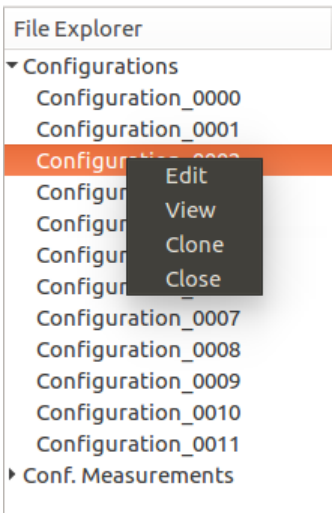


Figura 2.31: Popup menú.

VARIOS

GtkAdjustment

El objeto GtkAdjustment [48] representa un valor que tiene un límite inferior y superior asociado, junto con incrementos de paso y página, y un tamaño de página. Se usa en varios widgets GTK +, incluidos GtkSpinButton, GtkViewport y GtkRange.

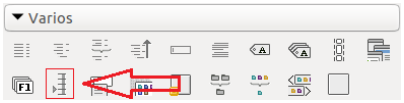


Figura 2.32: Botón GtkAdjustment en Glade.

Cómo podemos ver en la Figura 2.33 para realizar un ajuste en las horas correspondientes a las medidas, en el GtkSpinButton hemos aplicado un GtkAdjustment de 24 horas que tiene un día.

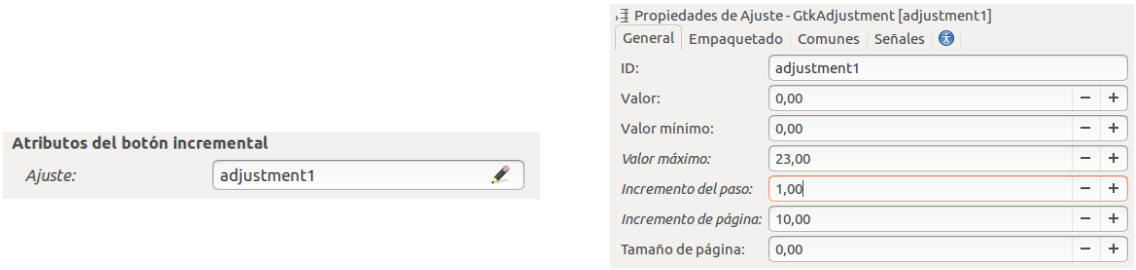


Figura 2.33: Atributo de ajuste del SpinButton y propiedades del ajuste.

GtkFileFilter

Un GtkFileFilter [49] se puede usar para restringir los archivos que se muestran en un GtkFileChooser. Los archivos se pueden filtrar en función de su nombre, en su tipo de mime o mediante una función de filtro personalizada. Un ejemplo lo tenemos en la Figura 2.34.

Figura 2.34: Botón GtkFileFilter en Glade y ajustes del filtro archivos dmac y dmam.

2.3. Organización del proyecto

A la hora de realizar un software, la organización del proyecto [Figura 2.35] es un punto importante para tener en cuenta. Tener una estructura bien organizada de todo un proyecto ayudará en un futuro a desarrolladores que entren a formar parte del equipo de trabajo a entender de manera rápida y sencilla el funcionamiento del software.

La estructura que se ha llevado a cabo es un sistema de carpetas y ficheros por el que sea intuitivo moverse. En un primer lugar, la carpeta configure tiene el fichero configure.cpp el cual nos dirá que tarjetas de adquisición hay instaladas. La aplicación tiene dos pantallas principales donde se ha optado por hacer una carpeta para cada una, en cuyo interior podemos encontrar el código C++ pertinente para el control de esta, más el archivo Glade que describe la interfaz gráfica de la ventana. Además de tener las carpetas correspondientes a cada una de las ventanas, también tenemos otras carpetas como Objects o Functions donde se almacenan ficheros más genéricos o que van a ser usados por más de una ventana en un futuro. Finalmente, para organizar los datos de configuración y medidas se optó por una carpeta Data donde guardar los ficheros.

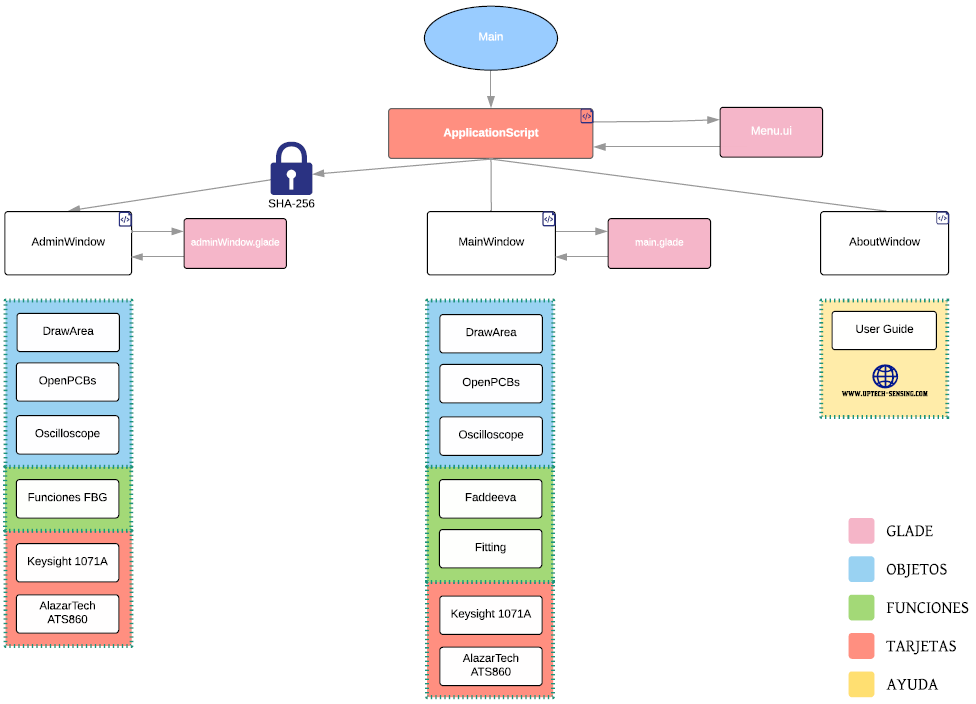


Figura 2.35: Organización del software desarrollado.

Configure

En esta carpeta se almacena el fichero correspondiente a la exploración en el sistema de las tarjetas instaladas. Su funcionamiento se basa en el empleo del fichero configure.cpp que al ser ejecutado nos retorna un ejecutable (debido a que C++ es un lenguaje compilado) que ejecutamos con el fin de recibir el archivo configure.h, el cual describe que tarjetas tenemos instaladas en nuestro sistema.

```
#include <vector>
#include <string>
#include <sys/stat.h>
#include <iostream>
#include <fstream>      //ofstream

using namespace std;

int main()
{
    vector<string> headerFiles;
    headerFiles.push_back("/usr/include/AgMD1.h");
    headerFiles.push_back("AGMD1_EXISTS_");
    headerFiles.push_back("/usr/include/AlazarApi.h");
    headerFiles.push_back("ALAZARTECH_EXISTS_");

    ofstream configFile ("./Configure/configure.h", ofstream::out);

    struct stat buffer;

    for (int i = 0; i < headerFiles.size(); i = i + 2)
    {
        if (stat ((headerFiles[i]).c_str(), &buffer) == 0)
        {
            configFile << "#ifndef " << headerFiles[i+1] << endl <<
"#define " << headerFiles[i+1] << endl << "#endif" << endl;
            cout << headerFiles[i+1] << endl;
        }

        configFile << endl;
    }
    configFile.close();

    return 0;
}
```

El código básicamente es una manera sencilla de comprobar que las librerias correspondientes están instaladas. Mediante la función stat comprobamos si las tarjetas necesarias, anteriormente añadidas las rutas de estas a un vector de strings llamado headerFiles, están instaladas mirando si en dicha ruta, especificada para las librerías, existe el fichero de las tarjetas.

Main

La función *main* es la encargada de realizar el lanzamiento de la aplicación a través de la generación de un objeto de la clase ApplicationScript que es la clase coordinadora de toda la aplicación. Esta función ejecuta el run de dicha clase, la cual hereda de la clase genérica de GTK+, GtkApplication.

```
#include "ApplicationScript.h"

int main(int argc, char* argv[])
{
    auto application = ApplicationScript::create();

    //Inicia la aplicación, mostrando la ventana inicial.
    //run () retornará del bucle principal cuando la última ventana
esté cerrada.
    const int status = application->run(argc, argv);
    return status;
}
```

ApplicationScript

Esta clase es la encargada de toda la coordinación de la aplicación; abrir y cerrar ventanas, conexión con los módulos del equipo, mirar ventanas abiertas, etc.

Existen dos partes que son importantes dentro de este código y que merecen ser comentadas con especial interés. La primera es la función *on_startup*. En esta función se carga el menubar a través del objeto builder de GTK+, que será el que posteriormente también se use para cargar todas las interfaces gráficas generadas por Glade. Además de cargarse el menubar y configurarlo como menú de la aplicación, se asocian funciones para cada una de las opciones descritas en el menubar a partir de la función *add_action* propia de GTK+.

```
//Call the base class's implementation:
Gtk::Application::on_startup();

//Añadimos acciones al menú (Solo descrita una)
add_action("newConfiguration", sigc::mem_fun(*this, &ApplicationS-
cript::on_fileNewConfiguration_select));

//Creamos el builder que vamos a usar en todas las ventanas para gene-
rarlas a partir de los ficheros glade
m_refBuilder = Gtk::Builder::create();


try
{
    //Cogemos la información de la barra de menú del siguiente fichero
    m_refBuilder->add_from_file("Glade/menu.ui");
}
catch (const Glib::Error& ex)
{
    std::cerr << "Building menus failed: " << ex.what();
}
```

```
//Obtenemos el objeto mediante el id del archivo XML, obtenemos el
menú a partir del objeto creado de menú, y lo añadimos a la aplica-
ción.
auto object = m_refBuilder->get_object("menuBar_global");
auto gmenu = Glib::RefPtr<Gio::Menu>::cast_dynamic(object);

if (!(gmenu)) {
    g_warning("GMenu or AppMenu not found");
}
else
{
    set_menubar(gmenu);
}
```

La otra función interesante y que merece ser comentada se llama *on_activate*. En esta función, a partir del builder se carga la información de las interfaces gráficas generadas por Glade y se pasa a las clases propias de cada ventana (usando la función de GTK+ *get_widget_derived* descrita más atrás) para que puedan trabajar con ella. Tras haber generado las funciones y haberles pasado a los objetos de cada ventana la información de sus respectivas interfaces de usuario se añaden estas ventanas al sistema de ventanas de la aplicación y se lanza la principal.

```
// Aquí cargamos todos los ficheros Glade de todas las ventanas para
ya tenerlos en el m_refBuilder y no tener que recargar
m_refBuilder->add_from_file("AdminWindow/Glade/adminWindow.glade");
m_refBuilder->add_from_file("MainWindow/Glade/main.glade");

// Definimos los punteros de cada ventana para que carguen los datos
correspondientes
// Admin
m_adminWin = nullptr;
m_refBuilder->get_widget_derived("AdminWindow", m_adminWin);
// MainWin
m_mainWin = nullptr;
m_refBuilder->get_widget_derived("MainWindow", m_mainWin);

// Admin Dialog
m_refBuilder->get_widget("dialog_AdminPassword", m_adminDialog);
m_refBuilder->get_widget("ent_adminPassword", ent_adminPassword);
m_refBuilder->get_widget("lbl_wrongPassword", lbl_wrongPassword);

//Añadimos los punteros a las ventanas como nuevas ventanas de la
aplicación.
add_window(*m_adminWin);
add_window(*m_mainWin);

// Escondemos las ventanas que no son main que es la ventana de por
defecto
m_adminWin->hide();

// Mostramos la pantalla por defecto de la aplicación
m_mainWin->maximize();
m_mainWin->show();
```

Es interesante también conocer que en esta clase hay una variable global (m_contVentanas) que se encarga de contar cuantas ventanas hay abiertas (nunca puede haber más de 1 abierta) para que en el momento que dicho contador llegue a cero se pueda cerrar la aplicación, abortando todos los procesos que están en ejecución.

```
// Aumentamos en una unidad el número de ventanas abiertas
++m_contVentanas;
```

Además de esta variable global hay otras variables globales (una por cada ventana) para saber si estas están activas en ese momento. Dichas variables realizan la llamada propia de GTK+ property_is_active() para obtener en estas variables una variable de tipo booleano de si están o no abiertas.

```
// Variables para monitorizar que ventana está en uso en cada momento
m_showingMain = m_mainWin->property_is_active();
```

Por último, se debe comentar que para hacer la visualización del pdf de ayuda de usuario se usa la aplicación evince de visualización de pdfs.

```
// Abrimos el pdf de ayuda usando evince...
int openPDF = system("evince AboutWindow/user_guide.pdf &");
```

AdminWindow

Esta es la carpeta que contiene los archivos que se encargan de generar la pantalla de Admin y todas sus funcionalidades. Debido a que es una pantalla orientada a administradores, se ha realizado un sistema de autenticación mediante una contraseña. Además, esta pantalla está dividida en 4 partes claramente diferenciables; una pantalla de dibujado donde se presentarán los datos capturados por la tarjeta, un área donde podemos seleccionar los parámetros de captura de la tarjeta bajo la pantalla. Y en la mitad derecha de la pantalla se encuentra el área de encendido del equipo y bajo él, un área de texto que actúa como terminal y por el que se van sacando avisos varios para guiar al administrador de la aplicación. Todo esto se puede ver en la Figura 2.36.

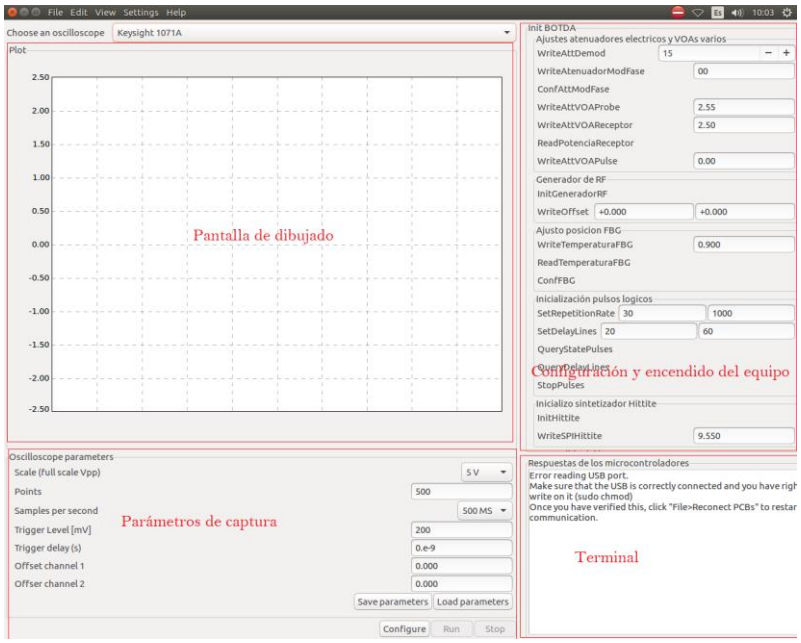


Figura 2.36: Ventana de administrador.

- Sha256

La contraseña que se solicita para acceder a la aplicación está codificada usando el algoritmo de encriptación sha256, mediante un fichero llamado igual que el propio algoritmo, ya que de esta manera si alguien accede al código fuente de la aplicación encontrará una cadena codificada que necesita decodificar para acceder a la parte de admin. La contraseña será solicitada una única vez desde que se inicia la aplicación, una vez se ha introducido la contraseña se guardará en una variable la sesión de admin y se le permitirá reabrir la ventana indefinidas veces.

Este fichero, es el encargado de "codificar" la cadena de texto que se introduzca por el campo de la contraseña mediante sha256 para que pueda ser contrastado con el hash previamente computado y proporcionar los permisos de admin o no.

El fichero LICENSE-SHA256.txt es el fichero de licencia del codificador sha256.

MainWindow

Esta ventana es la principal de la aplicación, la pantalla que se ejecuta al iniciar la aplicación y la que el usuario general va a visualizar. Esta pantalla está divida en 3 partes, un área de explorador de archivos, una barra de herramientas y un área de dibujado.

MainWindow tiene ciertas funciones que merecen ser comentadas para una mayor comprensión del uso de la ventana. La creación de configuración de sensor y de medidas son dos funciones similares que ambas desembocan en la función *save_files* guardando los ficheros en las rutas Data/Configurations y Data/ConfMeasurement respectivamente. Las variables de tipo booleano de la función *save_files* describen si los datos pasados son nuevos o editables/clonados/importados y si son de sensor o de medida. Además de guardar los ficheros, añade mediante las funciones *add_configuration* y *add_measurement* el nombre del fichero al TreeView. En la parte de nueva configuración del sensor hay una función llamada *check_if_filled* la cual comprueba que todos los campos de la configuración estén correctamente rellenados.

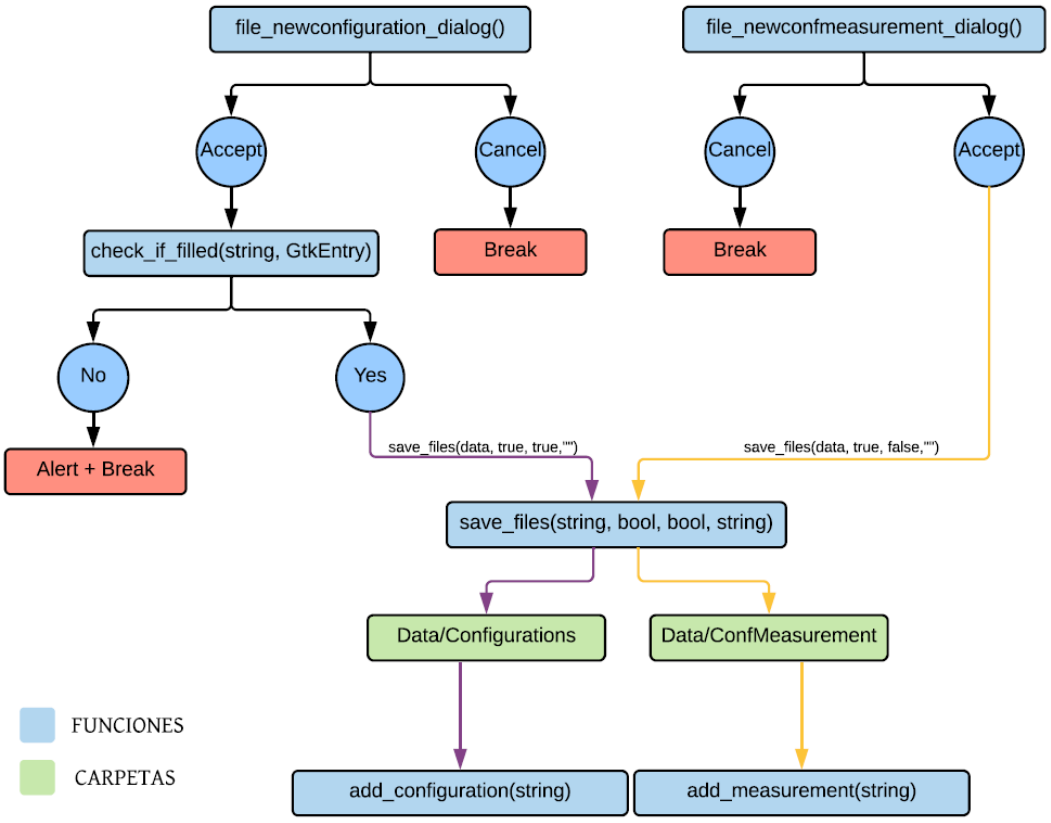


Figura 2.37: Organización de las funciones de nuevo sensor y medida.

Consecuentemente, la aplicación consta de una función para editar estos datos ya creados. Esta función *edit_files()* es parecida a las anteriores a diferencia de que la diferenciación de sensor y medida se hace dentro de la misma y no en funciones separadas. En la función *save_files()* se puede observar el primer booleano como false, indicando así que es un archivo editable, clonado o importado. Además, la última cadena ya no es pasado en blanco, sino que se le pasa el nombre del archivo que vamos a editar.

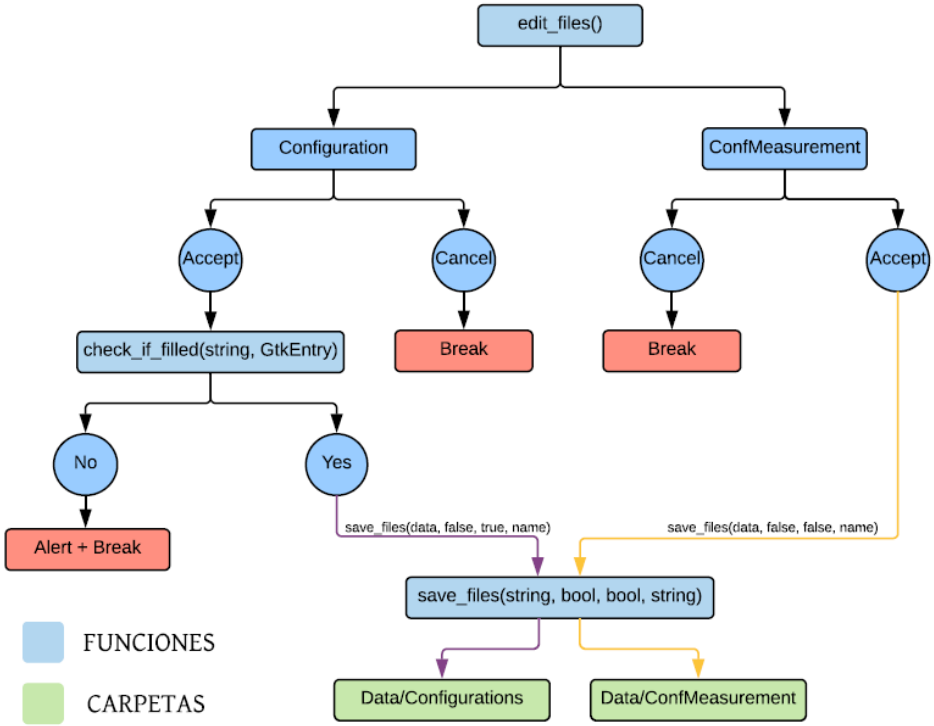


Figura 2.38: Organización de la función de editar ficheros.

La visualización se da de manera sencilla ya que, al pulsar sobre el botón correspondiente, observamos el fichero que vamos a mostrar y lanzamos un diálogo con la información correspondiente a dicho fichero.

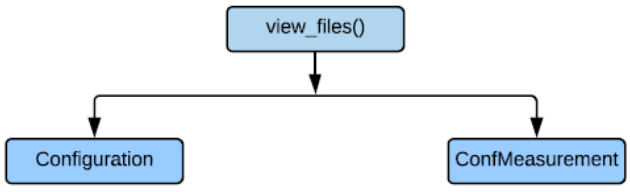


Figura 2.39: Organización de la función de visualizar ficheros.

La función *open_files()* lanzará un diálogo de elección de ficheros el cual solo mostrará los ficheros dmac y dmam. Una vez seleccionado el fichero correspondiente y pulsado en aceptar, diferenciamos entre configuración de sensor o de medida y lo añadimos a su zona correspondiente del TreeView.

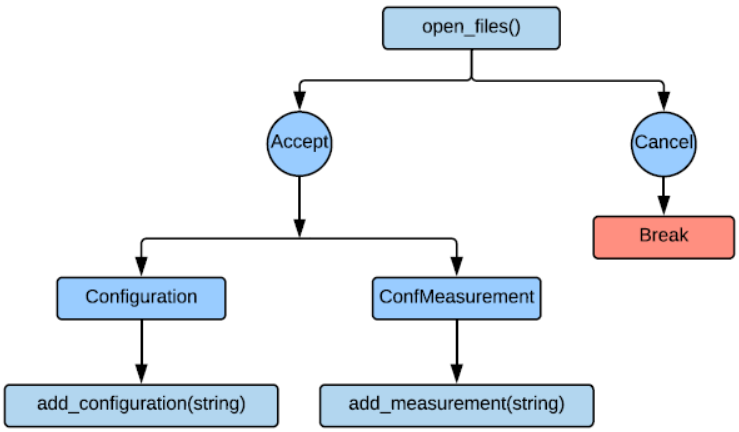


Figura 2.40: Organización de la función de abrir ficheros.

La exportación de ficheros se da de manera sencilla ya que, al pulsar sobre el botón correspondiente de elección de dispositivo externo donde exportar, observamos mediante un TreeView los archivos posibles de exportar. Una vez seleccionado uno de ellos y pulsado sobre Ok, la función *on_devicesfiles_export* guardará el fichero en el dispositivo.

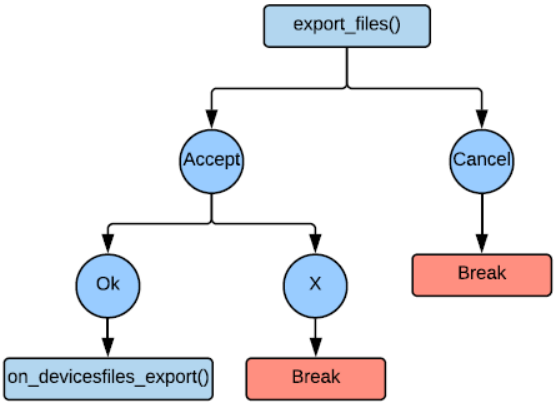


Figura 2.41: Organización de la función de exportar ficheros.

La importación de archivos mediante USB es un tanto más compleja que la apertura y la exportación. En *Import_files* al pulsar sobre el botón correspondiente de elección de dispositivo externo del cual importar, la función *fileUsb* recorrerá el dispositivo en busca de archivos dmac y dmam. Una vez seleccionado el fichero deseado por el usuario y pulsado en Ok, la función *on_devicesfiles_import* se encarga de llamar a las siguientes funciones responsables de buscar el fichero (importFilesUsb), guardarlo (*save_files)* y añadirlo al TreeView (*add_configuration)*.

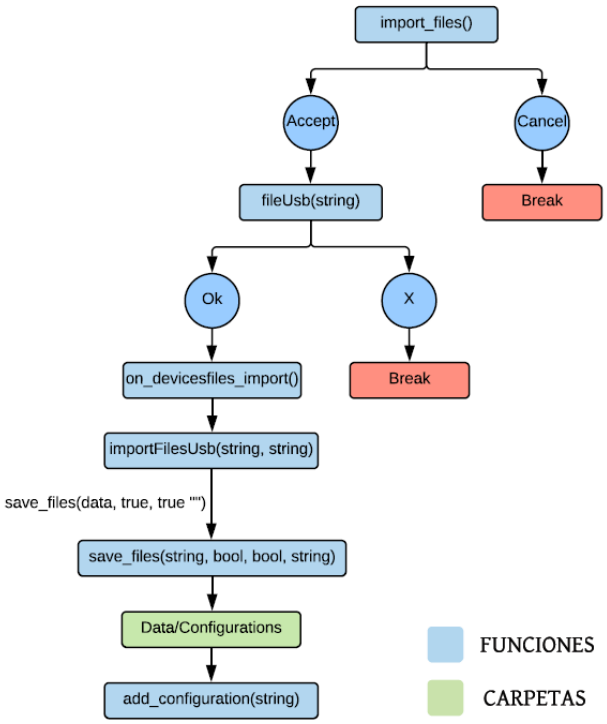


Figura 2.42: Organización de la función de importar ficheros.

AboutWindow

AboutWindow es la carpeta donde se encuentran los ficheros necesarios para generar la ventana de About. Esta ventana presenta el logo de la empresa, la versión en la que está el software, el año en el que se desarrolló dicha versión y un link a la página web de UptechSensing como vemos en la Figura 3.17.

Además, esta carpeta contiene un archivo pdf que es la guía de uso de la aplicación y el equipo. La visualización de esta guía de ayuda al usuario, como ya se ha comentado en este documento, se realiza mediante evince.

Capítulo 3

Uso de la aplicación

El software desarrollado hasta el momento consta de dos ventanas (Admin y Main) además de un diálogo de ayuda (About). Estas dos ventanas y el diálogo de ayuda están controladas desde una clase central (ApplicationScript) que se encarga de la correcta apertura de las ventanas, así como de la transmisión de datos entre ellas y generación de objetos compartidos.

La primera ventana que se visualiza es MainWindow, lanzada desde la clase predeterminada para el manejo de la aplicación (ApplicationScript). En esta ventana se puede ver 4 partes claramente diferenciables, la barra de herramientas, el panel de visualización, el explorador de archivos y el menú de la aplicación.

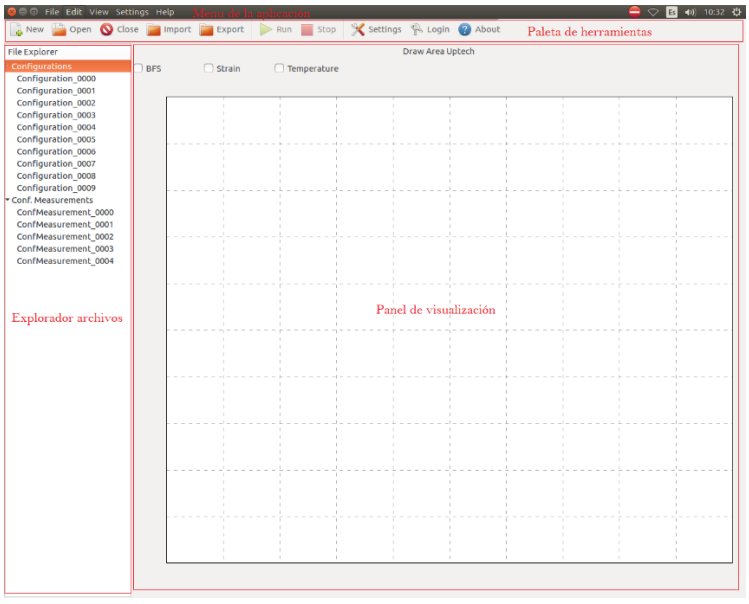


Figura 3.1: Ventana principal de la aplicación.

3.1 Barra de herramientas

La paleta de herramientas es un conjunto de atajos que nos proporciona diversas funcionalidades con respecto a los ficheros. Se muestra en pantalla a modo de fila y contiene iconos o botones, que al ser presionados, activan ciertas funciones que van a ser explicadas a continuación. El primer elemento es New, encargado de lanzar un diálogo para la elección de la creación de sensor o medida.

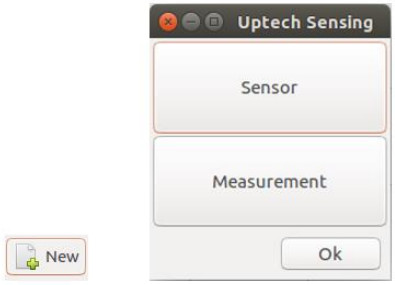


Figura 3.2: Botón de nueva configuración y su correspondiente diálogo.

En caso de pulsar en sensor, se abrirá el diálogo de nueva configuración de sensor el cual tiene diversos campos de entrada que serán guardados en un fichero de extensión dmac tras pulsar en aceptar y que el software verifique que todos los campos correctamente rellenados.

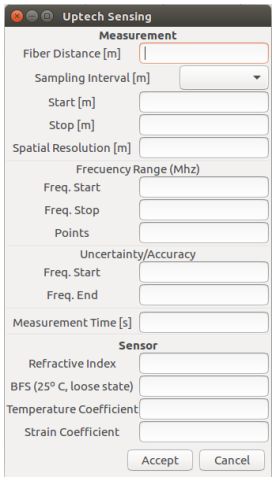


Figura 3.3: Diálogo de nueva configuración de sensor.

En caso de pulsar en Measurement, se abrirá el diálogo de nueva configuración de medida el cual tiene diversos tipos de medida que se pueden seleccionar mediante un desplegable en la parte superior derecha del diálogo. La primera opción por defecto es la de una medida. Al pulsar sobre una configuración de sensor en el panel de la izquierda y en aceptar guadaremos un fichero con extensión dmam correspondiente a ese tipo de configuración de sensor.

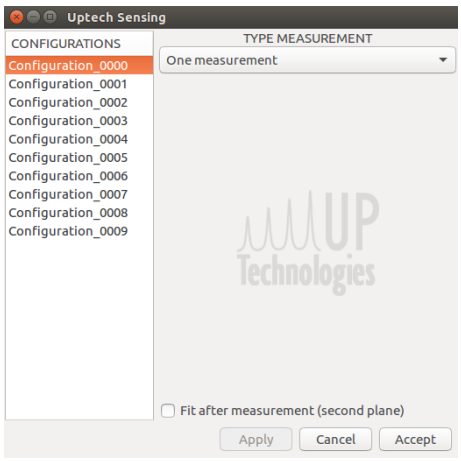


Figura 3.4: Diálogo de configuración de una medida.

La segunda opción repite la medida con la configuración empleada un numero especifico de veces. Se selecciona la configuración de sensor y en la entrada de texto ponemos el número de medidas que queramos.

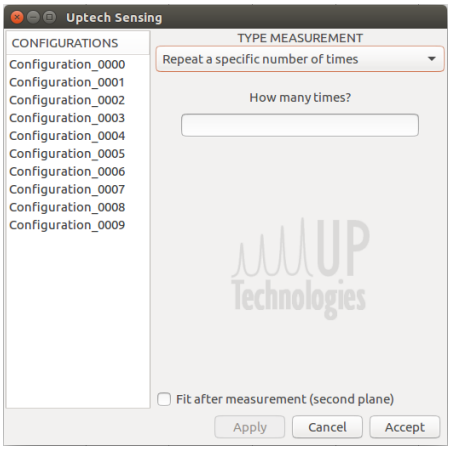


Figura 3.5: Diálogo de configuración de medida un número especifico de veces.

La tercera opción nos da opción a realizar la medida periodicamente un número especifico de veces. En las entradas de texto ponemos respectivamente el número de medidas que se quieran realizar y la periodicidad deseada.

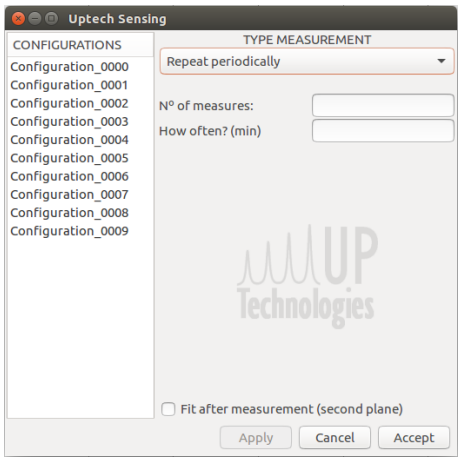


Figura 3.6: Diálogo de conf. medida periódicamente un número especifico de veces.

La cuarta y última opción nos presenta un calendario el cual muestra la fecha actual y es posible seleccionar un día marcandolo y accionando otros campos vistos en la figura x.x. En estos campos podemos seleccionar la hora de la medida además de repetir diariamente o semanalmente. Además, hay opción de hacer un ajuste después de realizar la medida.

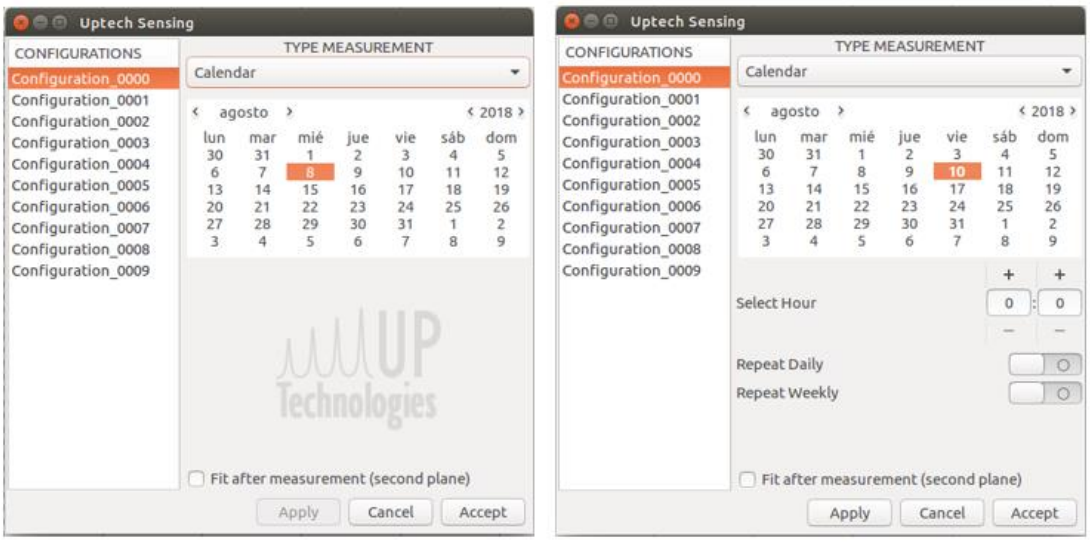


Figura 3.7: Diálogo de configuración de medida conforme al calendario.

El segundo elemento es Open, el cual lanza el diálogo de apertura de ficheros. Este diálogo tiene un filtro de extensión de ficheros para solo mostrar los correspondientes a la aplicación, en este caso dmac y dmam. Además, dentro del código fuente hay implementada una función la cual guarda el directorio del último fichero abierto permitiendo la próxima vez que se abra el diálogo estar en esa misma carpeta.

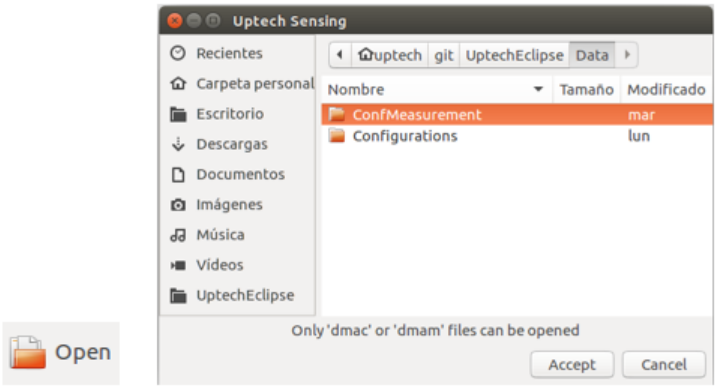


Figura 3.8: Botón de apertura de ficheros y su correspondiente diálogo.

El tercer elemento es Close y se corresponde con la eliminación de ficheros. El usuario tiene posibilidad de solo cerrar el fichero y no mostrarlo en el explorador de archivos o borrarlo completamente del directorio y del explorador. Esta opción se le da al usuario mediante un diálogo el cual avisa de que el archivo va a ser cerrado y en caso de querer borrarlo del directorio marcar la casilla correspondiente.

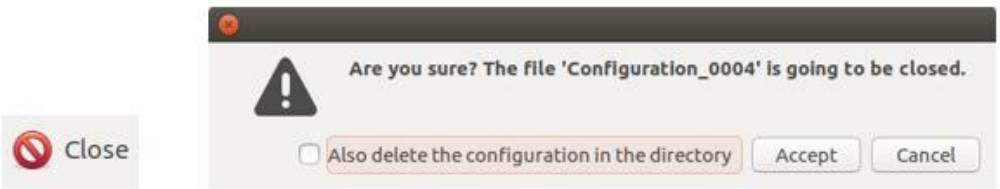


Figura 3.9: Botón de cerrar/borrar fichero y su correspondiente diálogo.

El cuarto elemento es Import el cual permite la importación de ficheros correspondientes a la configuración de sensor. Al pulsar sobre el botón se lanza un diálogo de elección de dispositivo externo para la importación de ficheros dmac.

Figura 3.10: Botón de importar ficheros y su correspondiente diálogo.

Una vez seleccionando el dispositivo del cual queremos importar y pulsado en aceptar, se lanza otro diálogo el cuál muestra todos los ficheros con la extensión dmac sin distinción de si están en una carpeta u en otra, son mostrados en forma de lista.

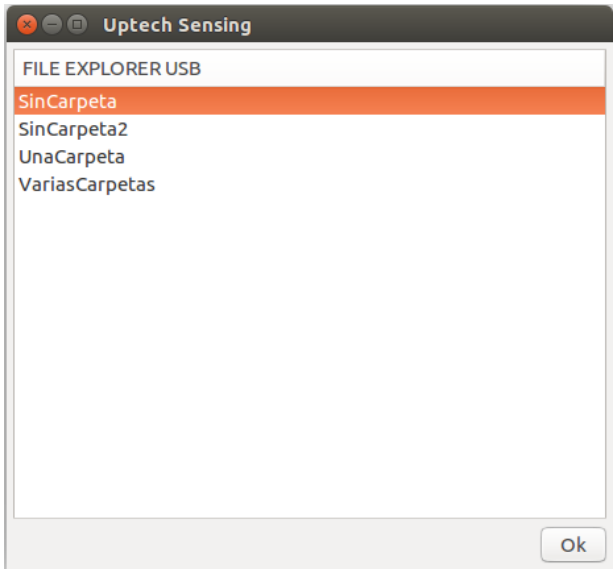


Figura 3.11: Ventana de importación de ficheros.

El quinto elemento es Export el cual permite la exportación de ficheros correspondientes a la configuración de sensor y la configuración de medida. Al pulsar sobre el botón es lanzado un diálogo de elección de dispositivo externo para la exportación de ficheros.

Figura 3.12: Botón de exportar ficheros y su correspondiente diálogo.

Una vez seleccionando el dispositivo en el cual queremos exportar y pulsado en aceptar, se lanza otro diálogo el cuál muestra todos los ficheros con la extensión dmac y dmam de la aplicación, mostrados en forma de lista. Una vez seleccionado un fichero de la lista y pulsado en el botón Ok, se guardará en el dispositivo externo.

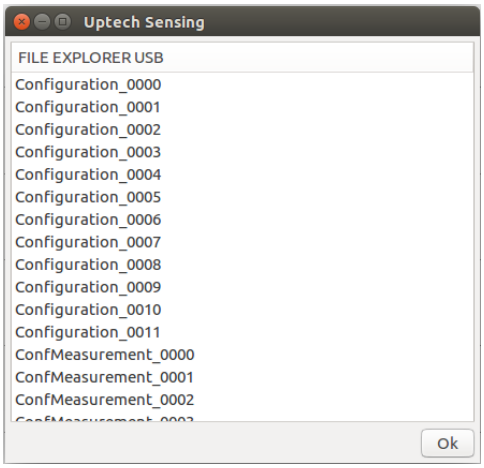


Figura 3.13: Ventana de exportación de ficheros.

El sexto y séptimo elementos son dos botones no activos por el momento y aunque las rutinas ya están escritas, no se han incluido al programa. Los botones de Run y Stop son los correspondientes al lanzamiento de una medida y a la parada de esta respectivamente. En un futuro próximo serán implementados.

Figura 3.14: Botones correspondientes a lanzar medida y detenerla.

El octavo elemento es Settings y se corresponde con las opciones disponibles en la aplicación. Al pulsar sobre él se observa un diálogo con varias pestañas en su interior, la primera corresponde con las unidades de medida de la aplicación y la segunda con la cuota de disco.

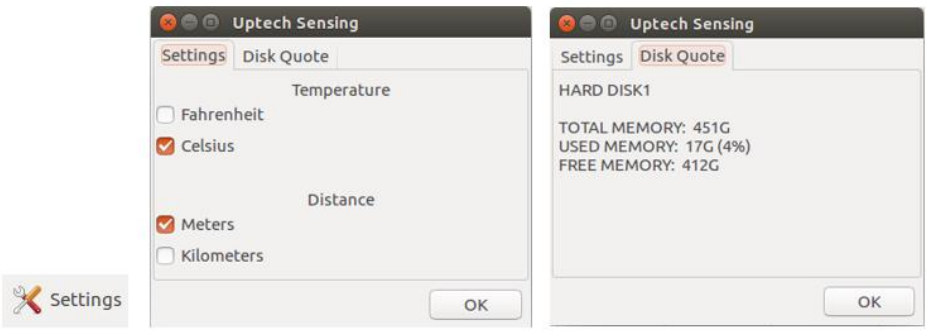


Figura 3.15: Botón de ajuste y sus correspondientes diálogos.

El noveno elemento es Login, el cual nos da la posibilidad de entrar en la ventana de administrador. La funcionalidad de Admin está implementada con anterioridad a este proyecto y el acceso a ella se hace mediante una contraseña codificada usando el algoritmo de encriptación sha256. Una vez pulsado el botón correspondiente en la paleta de herramientas se lanza un diálogo pidiendo la contraseña de administrador. En caso de insertar correctamente la contraseña y dado en aceptar, la ventana de administrador será lanzada.

Figura 3.16: Botón de login y su correspondiente diálogo de contraseña.

El décimo y último elemento es About y se corresponde con el menú de ayuda. La ventana de About nos enseña un diálogo el cual muestra una imagen de Uptech Sensing, la versión del software, la página web de la empresa y el año de desarrollo.

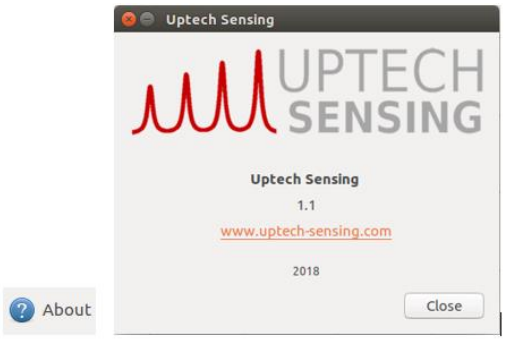
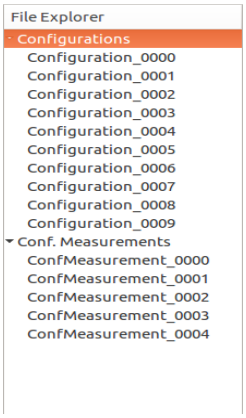
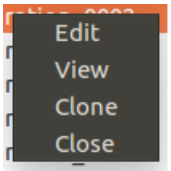


Figura 3.17: Botón acerca de y su correspondiente diálogo.

3.2 Explorador de archivos



El explorador de archivos muestra mediante el TreeView los ficheros de configuración de sensor y de configuración de medidas. Es por ello por lo que tenemos dos padres en el TreeViewModel, Configurations y Conf. Measurement. Al pulsar sobre ellos se desplegará los ficheros correspondientes que hay en los directorios Data/Configurations y Data/ConfMeasurement respectivamente. La principal funcionalidad de este elemento es la de otorgar al usuario de manejabilidad y facilidad de uso con respecto a los ficheros creados.



En este elemento se ha implementado un menú de clic derecho el cual lanza 4 atajos (Edit, View, Clone y Close). Al pulsar con el clic derecho sobre un nombre dentro del TreeView se busca ese fichero dentro del directorio y en caso de pulsar cualquiera de estos atajos, se maneja el archivo consecuentemente.

El primer botón del menú es Edit, el cual abre un diálogo de edición de datos correspondientes a la configuración de sensor (ver Figura 3.18) o medida (ver Figura 3.18).

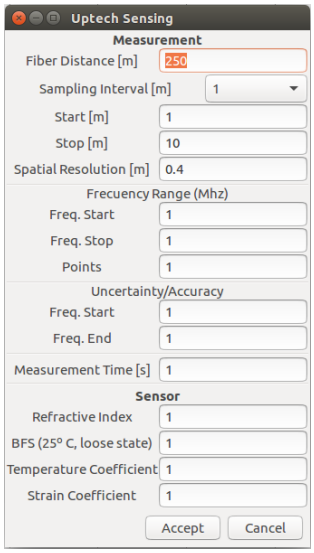


Figura 3.18: Edición de configuración de sensor.

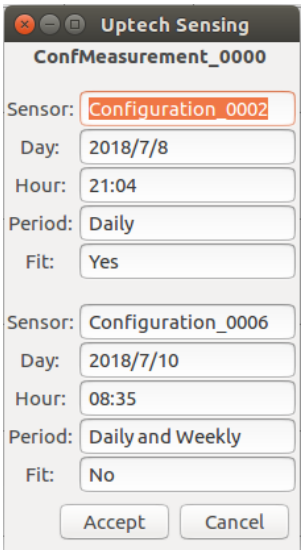


Figura 3.19: Edición de configuración de medida.

El segundo botón del menú llamado View abre un diálogo de visualización de los datos de configuración de sensor (ver Figura 3.20) o de medida (ver Figura 3.20).

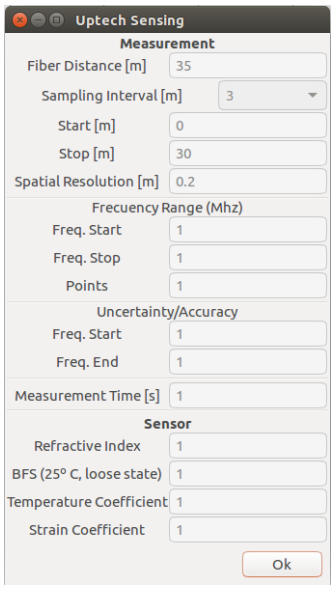


Figura 3.20: Visualización de configuración de sensor.

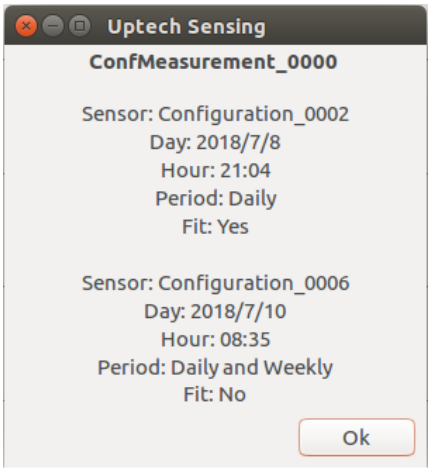


Figura 3.21: Edición de configuración de medida.

El tercer botón llamado Clone básicamente hace una copia del archivo seleccionado añadiéndole al nombre "_Clone" y lo guarda en el mismo directorio que el archivo original.

El cuarto y último botón llamado Close se corresponde con la eliminación de ficheros. El usuario tiene posibilidad de solo cerrar el fichero y no mostrarlo en el explorador de archivos o borrarlo completamente del directorio y del explorador. Un ejemplo podemos verlo en la Figura 3.9.

3.3 Panel visualización

El panel de visualización se basa en un objeto llamado DrawArea anteriormente programado por otro desarrollador. En dicho panel es posible pinchar en cualquier zona obteniendo sus coordenadas pudiendo ser un punto de inflexión para la realización de muchas funcionalidades futuras como el zoom de la traza mostrada en el panel.

A pesar de no estar implementado, las rutinas ya están escritas en un archivo llamado DrawArea. La conexión con esta clase se conseguirá en Líneas abiertas.

Figura 3.22: Panel de visualización de señales.

3.4 MenuBar

El menú de la aplicación es un elemento que destacar ya que está en todas las ventanas de la aplicación. Las funcionalidades del menú son las mismas que las comentadas más atrás en la barra de herramientas y se pueden ver a continuación las figuras correspondientes.

La generación de archivos de datos correspondientes a las configuraciones de sensores y medidas se realiza mediante archivos de texto plano bajo las extensiones “dmac” y “dmam” respectivamente y pueden ser abiertos mediante el botón open del menú de la aplicación o de la paleta de herramientas, el cual, tiene un filtro para estas dos extensiones de ficheros. Además, es posible importar y exportar este tipo de ficheros, filtrando los ficheros mostrados en el diálogo correspondiente, mostrando solo los de extensión “dmac” y “dmam”, haciendo así más sencillo para el usuario la elección del archivo a importar/exportar.

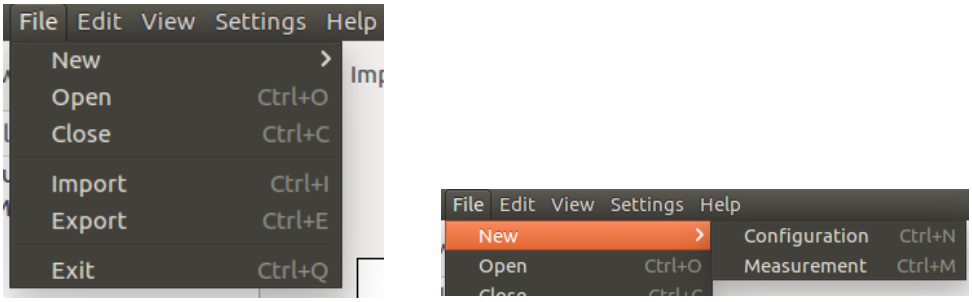


Figura 3.23: Sección File del menú de la aplicación y nueva configuración

Las funciones de editar y visualizar ficheros descritas más atrás son accesibles como se ve en la Figura 3.24 desde el menú de la aplicación.

Figura 3.24: Secciones de edición y visualización de ficheros respectivamente.

El menú de la aplicación también tiene un elemento de 'Settings' el cual contiene dos elementos: Login y Options explicados más atrás.

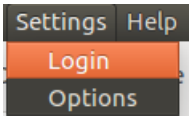


Figura 3.25: Elemento del menú correspondiente con los ajustes.

Help es un elemento el cual posee dos opciones, una guía de usuario que es abierta mediante evince como se describió en este documento más atrás. About también ha sido descrito con anterioridad.

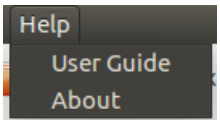


Figura 3.26: Elementos de ayuda del menú.

En un futuro próximo, cómo se ha explica en las Líneas abiertas, es posible una implementación de un menú propio para cada ventana permitiendo así la implementación de la pantalla completa de la aplicación, actualmente todas las acciones de este menú compartido son implementadas en la clase ApplicationScript

Capítulo 4

Conclusiones y líneas abiertas

4.1. Conclusiones

Los interfaces gráficos de usuario (GUI) son un medio muy interesante para la representación de las medidas distribuidas y dinámicas de deformación unitaria y temperatura realizadas por un equipo interrogador. La realización de una aplicación de estas características para este tipo de equipamientos nos aporta una gran productividad ya que permite el manejo y el orden de todos los recursos e información aportados por el equipo interrogador, facilitando así al usuario poder identificar los puntos clave en cada activo y así poder realizar un correcto mantenimiento del mismo. Además, el control de los datos de un equipo de tales características desde una aplicación hace posible implementar una correcta gestión de alertas que permitan activar determinados protocolos de seguridad, protegiendo de esta manera la integridad de la estructura y de su entorno. Este tipo de monitorización debe ser sencilla para un usuario cualquiera y es en la consecución de estos aspectos en lo que se centra el desarrollo de esta aplicación realizada en este trabajo fin de grado. Durante la realización del mismo, se ha trabajado en varias líneas de desarrollo que mejoran la comunicación usuario-equipo.

En primer lugar, se ha realizado un interfaz amigable e intuitivo en el cual el usuario pueda interactuar con el equipo interrogador para la consecución de la monitorización de la fibra óptica en la estructura deseada mediante, por ejemplo, la configuración del equipo, la realización de medidas, el análisis y visualización de los datos obtenidos por el equipo interrogador o la configuración de alertas ante determinados eventos.

Esta aplicación permite la creación, edición, clonación y borrado de nuevas configuraciones de sensor y de medidas, además de aportarnos opciones de importación y exportación a USB externos. La pantalla de administrador permite ciertas funcionalidades adicionales para el control de módulos del equipo interrogador tales como ajustar atenuadores ópticos, inicializar generador de RF, ajustar posición de filtros sintonizables, inicializar pulsos, el encendido del láser y del EDFA, entre otros.

La aplicación desarrollada en este Proyecto Fin de Grado realizada en lenguaje C++, se ejecuta en un ordenador externo con sistema operativo Ubuntu. La comunicación entre el ordenador y el equipo interrogador se realiza por conexión serie.

4.2. Líneas abiertas

La continuación natural de este trabajo fin de grado se basa en el avance y perfeccionamiento de las características de la correspondiente aplicación. A continuación, se presentan líneas de desarrollo para la consecución de una mejora futura del software.

- La aplicación realizada hasta el momento podría definirse como una versión Alpha de la misma, necesitando así la correcta conexión con el área de dibujado y la comunicación entre el ordenador y el equipo interrogador realizada por conexión serie.
- La mejora de un menú capaz de aceptar una pantalla completa mediante la ayuda de GtkToolbar es un hecho necesario debido al carácter que la aplicación debe presentar, y es que tiene que estar abierta y en uso continuamente, sin posibilidad de cerrarla.
- La posibilidad de edición en los nombres de los ficheros. Es posible que esta solución pueda darse mirando el nombre del padre en el TreeView, facilitando así la comprobación de si un fichero es configuración de sensor o de medida. Además, supone un cambio agradable para el usuario ya que le aportas cierto control sobre la aplicación.

- Una vez conseguida la conexión con el área de dibujado y comprobar que funciona, realizar algún tipo de función capaz de hacer zoom-in y zoom-out en el dibujado, permitiendo así al usuario poder observar las trazas representadas mejor.
- La implementación de una papelera en cuyo directorio guardar los archivos borrados por el usuario, la cual, sea accesible por el administrador para poder recuperar ciertos ficheros que por cualquier situación se necesiten.
- La posibilidad de cambiar el directorio de guardado de ficheros es una opción futura a tener en cuenta ya que es posible que un usuario del software desee guardar los ficheros en algún lugar concreto del sistema.
- En el diálogo de settings, la posibilidad de cambiar las unidades es un hecho, pero sin función escrita en código fuente actualmente. En un futuro próximo implementar funciones para manejar las unidades. Por ejemplo,

```
//Función para pasar grados fahrenheit a celsius
void MainWindow::FtoC(float fahrenheit)
{
    cel = (fahrenheit - 32) * 5/9;
}
```

- El panel visualización tiene tres botones en la zona superior que deben ser excluyentes entre sí, pudiendo visualizar diferentes tipos de gráficos según esté la correspondiente casilla seleccionada.
- La realización de un explorador de archivos en los dispositivos externos conectados para conseguir que sea mejor visualmente que el diálogo con TreeView simple actual.
- Un instalador de la aplicación para en un futuro próximo tener el software en un disco de instalación el cual poder distribuirlo junto con el equipo interrogador.

Apéndices

Apéndice A

Importación del proyecto en IDE

A la hora de la importación del proyecto programado mediante C/C++ en el IDE Eclipse se ha llevado a cabo unos rigurosos pasos que se describen a continuación.

En primera instancia se abre el entorno de desarrollo integrado Eclipse y en la parte superior se clica en el siguiente elemento: File > New > Makefile Project with Existing Code. Los datos necesarios que se deben rellenar los detallamos a continuación:

Proyect name: Se aplica el nombre del proyecto.

Existing Code Location: Se escoce la localización del código de nuestro proyecto.

Languages: C & C++

Toolchain for Indexer Settings: Linux GCC

Una vez creado, se pulsa sobre el proyecto y en el menú superior: Project > Properties. Primero, en C/C++ Build en la pestaña Builder Settings se marca "Generate Makefiles automatically". Segundo, en C/C++ Build > Settings en la pestaña Build Artifact se pone Artifact Type Executable. Tercero, en C/C++ Build en la pestaña Configuration se clica en Manage Configurations > New y se pone un nombre a la carpeta donde se van a crear los makefiles. Tras crearla se activa mediante Set Active.

Cuarto y último, en C/C++ Build > Settings, pestaña Tool Settings se aplica lo siguiente:

- En GCC C++ Compiler se pone lo siguiente al final de Command line pattern: `pkg-config gtkmm-3.0 --cflags --libs`
- En GCC C++ Compiler > Dialect se pone lo siguiente: -std=c++11
- En GCC C++ Compiler > Preprocessor se añade en Defined symbols (-D) lo siguiente: _LINUX
- En GCC C++ Compiler > Optimization y se pone Optimization Level -03
- En GCC C++ Compiler > Miscellaneous se borra fmessage-lenght=0 dejando el -c del principio
- En GCC C++ Compiler los warnings(-Wall) y el flag(-c) se deja de momento por defecto.
- En GCC C++ Linker se pone lo siguiente al final en Command line pattern: `pkg-config gtkmm-3.0 --cflags --libs`
- En GCC C++ Linker > Libraries se añade en Libraries(-l) las librerias que se utilicen al compilar.
- En GCC C++ Linker > Libraries se añade en Libraries search path lo siguiente: /usr/local/lib

Tras todo esto: Apply and Close.

En la parte izquierda donde se tiene el proyecto en Configure > Configure.cpp se pulsa click derecho sobre él y en Resource Configurations > Exclude from build > Select All > Ok. Esto es necesario para excluir de la compilación ese archivo el cual debemos compilarlo a parte.

Para finalizar se hace el build de la aplicación: Project > Build All

Para hacerla correr en Run > Run Configurations > New Configuration se selecciona la aplicación que va a correr y listo.

Apéndice B

Configuración de GIT

Este tipo de sistema de control llamado EGit es utilizado de manera local en el proyecto siendo propio del IDE utilizado. Cómo ya se explicó más atrás en este documento, se puede trabajar con distintas perspectivas con lo que puedes tener distintas configuraciones de tu pantalla de trabajo según tu tipo de proyecto o según la acción que vayas desarrollar en ese momento. Una de estas perspectivas es la del SVN local el cual es de gran ayuda para proyectos de gran envergadura y en los que vayan a trabajar varios desarrolladores.

EGit ya está incluido en Eclipse Oxygen [50], por lo que no es necesario su instalación. Si usa una versión anterior a Eclipse Juno tendrá que instalar el software abriendo el Asistente de Eclipse y pulsando en '*Help -> Install new software'.*

Si se quiere configurar con el repositorio GitHub en línea, el nombre de usuario y el correo electrónico deben ser los mismos que usa para su cuenta de Git, es decir. su cuenta de GitHub. Cada confirmación en EGit incluirá el nombre del usuario y su dirección de correo electrónico. Estos atributos se pueden establecer en la ventana de preferencias*: Window -> Preferences*. Navegue a *Team -> Git -> Configuration* y presione el botón *Add Entry.* En primera instancia añada el *usuario.nombre* como *Key* y su nombre como *Value*. Repita este procedimiento con *user.email* y su dirección de correo electrónico y haga *clic* en *Ok*. En la ventana de preferencias pulsamos en Apply and Close.

Una ventaja importante de Git es que puede crear fácilmente repositorios locales, incluso antes de compartirlos con otras personas. De esta forma, puede versionar su trabajo localmente para tener un constante control del avance de la aplicación. Primero, debe crear un proyecto en Eclipse que desee compartir a través de su repositorio local.

Después de haber creado el proyecto, seleccione el menú contextual haciendo clic derecho y navegue a *Team -> Share proyect*. Seleccione Git como tipo de repositorio y presione Siguiente. En la siguiente ventana, seleccione su proyecto, presione el botón Crear repositorio y haga clic en Finalizar.

El repositorio recién creado estará vacío, aunque el proyecto esté asignado a él. Antes de que se pueda comprometer los archivos al repositorio, hay que agregarlos. Simplemente haga clic derecho en el proyecto compartido y navegue a *Team -> Add*. Después de esta operación, el signo de interrogación debería cambiar a un símbolo más. Lo último que debe hacer es comprometer el proyecto haciendo clic con el botón derecho en el nodo del proyecto y seleccionando *Team -> Commit* desde el menú contextual. En el asistente de confirmación, todos los archivos deben seleccionarse automáticamente. Ingrese un mensaje de confirmación (la primera línea debe ser similar a un titular, como aparecerá en la vista del historial) y *presione el* botón *Confirm*. Si la confirmación salió correctamente, los símbolos más se abran convertido en símbolos de repositorio.

Para hacer el commit de lo programado hasta el momento vamos a la perspectiva de Git y veremos en la zona de abajo diversas pestañas. Seleccionamos Git Starting si no lo está por defecto y se verán los cambios que se han hecho y no han sido cometidos en ‘Unstaged Changes’. Para añadirlos clicamos en el icono verde de añadir de la derecha y se observará como se añaden a ‘Staged Changes’. Una vez en este punto se pone un ‘Commit Message’ y se hace el commit pulsando en el botón del mismo nombre en la esquina derecha inferior.

Apéndice C

Programación

En cuanto a la programación cabe destacar el uso de ciertas funciones realizadas a lo largo del proyecto para el correcto funcionamiento de la aplicación desarrollada. Estas funciones creadas no tienen por qué estar implementadas de la mejor manera posible y es muy probable que haya formas de mejorarlas.

Cuota de disco

Para el mantenimiento de la cuota de disco se ha realizado una clase propia llamada HardDisk la cual almacena los datos correspondientes al disco del equipo.

```
class HardDisk {
    public:
        long memtotal;
        long memusada;
        long memlibre;
        long memporc;
};

//Escribimos en el sistema el siguiente comando el cual saca la infor-
mación del disco y lo añade al fichero descrito.
int resultado = system("df -h /dev/sd*1 > ~/git/UptechEclipse/MainWin-
dow/harddiskinfo.txt");

//En caso de que el comando se haya ejecutado correctamente entramos
if(resultado==0){
    string token, word;
    //Fichero de entrada
    ifstream fileI("MainWindow/harddiskinfo.txt");
    int num_disp=0, aux=0;
    getline(fileI,token); //Cogemos la primera línea
```

```
    //Leemos línea a línea mientras no sea el final del fichero
    while(!fileI.eof()){
        //Miramos que no sea USB ni fichero
        if(!(token.find("/media/")!=string::npos) && !(token.find("fi-
cheros")!=string::npos)){
            istringstream ss(token);
            //Añadimos a la clase los datos del disco leido
            while(ss >> word){
                if(aux==1) harddisk[num_disp].memtotal = stol(word);
                if(aux==2) harddisk[num_disp].memusada = stol(word);
                if(aux==3) harddisk[num_disp].memlibre = stol(word);
                if(aux==4) harddisk[num_disp].memporc = stol(word);
                aux++;
            }
            aux=0;
            num_disp++;
        }
        getline(fileI,token);
    }
    fileI.close();

    //En caso de tener menos de 10 gigas de espacio, sacamos diálogo.
    if(harddisk[0].memlibre<10){
        int result = dialogDiskQuote->run();

        //Handle the response:
        switch(result)
        {
            case(0): // Ok clicked
            {
                dialogDiskQuote->close();
                return;
            }
            default: // X clicked
            {
                dialogDiskQuote->close();
                return;
            }
        }
    }
}
```

Fecha actual

La función de get_date(string) retorna el date actual que le preguntes(day, month, year, hour, min, seg).

```
double MainWindow::get_date(string dateAux)
{
    //Cogemos la fecha actual
    time_t time_now_t = chrono::system_clock::to_time_t(chrono::sys-
tem_clock::now());
    tm now_tm = *localtime(&time_now_t);
    stringstream dia, mes, ano, horas, minutos, segundos;
    double day, month, year, hour, min, seg;
```

```
    //Asignamos a cada variable su valor de fecha
    dia << put_time(&now_tm, "%d");
    mes << put_time(&now_tm, "%m");
    ano << put_time(&now_tm, "%Y");
    horas << put_time(&now_tm, "%H");
    minutos << put_time(&now_tm, "%M");
    segundos << put_time(&now_tm, "%S");

    //Pasamos las variables a double
    istringstream(dia.str()) >> day;
    istringstream(mes.str()) >> month;
    istringstream(ano.str()) >> year;
    istringstream(horas.str()) >> hour;
    istringstream(minutos.str()) >> min;
    istringstream(segundos.str()) >> seg;

    //Retornamos lo que el usuario haya pedido
    if(!dateAux.compare("day")) return day;
    if(!dateAux.compare("month")) return month;
    if(!dateAux.compare("year")) return year;
    if(!dateAux.compare("hour")) return hour;
    if(!dateAux.compare("min")) return min;
    if(!dateAux.compare("seg")) return seg;
    return;
}
```

Fecha válida

La función isValidDate le pasas una fecha en formate año/mes/dia y te devuelve true en caso de ser fecha válida, false en caso contrario.

```
bool MainWindow::isValidDate(int y, int m, int d){
    //Seteamos máximo y mínimo valor posible de año
    int MAX_VALID_YR = get_date("year") + 20;
    int MIN_VALID_YR = get_date("year");

    //Miramos que el año, el mes y el día sean correctos
    if(y > MAX_VALID_YR || y < MIN_VALID_YR) return false;
    if(m < 1 || m > 12) return false;
    if(d < 1 || d > 31) return false;

    //Si el mes es febrero, comprobamos si es bisiesto
    if(m == 2){
        if(((y%4 == 0) && (y%100 != 0)) || (y%400 == 0)) return (d <=
29);
        else return (d <= 28);
    }

    //Meses con 30 días
    if (m == 4 || m == 6 || m == 9 || m == 11) return (d <= 30);

    return true;
}
```

Cadena es un número

Check_if_numeric es una función que retorna true si el string pasado es un numero.

```
bool MainWindow::check_if_numeric(string text)
{
    bool isNumeric=false;

    for(int i=0; i<text.length(); i++)
    {
        if(+((text.at(i)))==46 || (+((text.at(i)))>=48 &&
+((text.at(i)))<=57) || +((text.at(i)))==69 || +((text.at(i)))==101 ||
+((text.at(i)))==45 || +((text.at(i)))==43)
        {
            isNumeric = true;

            if((+((text.at(i)))==69 || +((text.at(i)))==101) &&
(text.length()<=(i + 1) || ((+((text.at(i + 1)))==69 &&
+((text.at(i+1)))==101))))
            {
                isNumeric=false;
                break;
            }
        }
        else
        {
            isNumeric=false;
            break;
        }
    }
    return isNumeric;
}
```

Comprobar si está correctamente rellenado

Función que retorna true si todos los campos (Configuración Sensor) están correctamente rellenados.

```
bool MainWindow::check_if_filled(string configuracion[], Entry
*info[])
{
    double auxiliar;
    // SI NO ESTAN TODOS LOS CAMPOS COMPLETOS Y SOLO NUMEROS, SALE IN-
FOBAR //
    for (int i=0; i<15; i++)
    {
        istringstream(configuracion[i]) >> auxiliar;
        if(configuracion[i].empty() || !check_if_numeric(configura-
cion[i]) || auxiliar < 0)
        {
            lbl_infodialogconfig->set_text("Fill all boxes (Positive
numbers)");
            infobarconfig->show();
            return false;
```

```
        }
    }

    // Fiber Distance -- Redondeamos el entero por encima
    double fiberdistance;
    istringstream(configuracion[0]) >> fiberdistance;
    configuracion[0] = to_string((int)ceil(fiberdistance));

    // Sampling Interval (Identificar tarjeta y mostrar configuracio-
nes de tarjetas)
    double samplinginterval;
    istringstream(configuracion[1]) >> samplinginterval;

    // Start -- Debe ir de 0 a Fiber Distance
    double start;
    istringstream(configuracion[2]) >> start;
    if(start>=(int)ceil(fiberdistance))
    {
        lbl_infodialogconfig->set_text("'Start' must be lower than
'Fiber Distance'");
        infobarconfig->show();
        return false;
    }

    // Stop -- Debe ir de 0 a Fiber Distance
    double stop;
    istringstream(configuracion[3]) >> stop;
    if(stop>(int)ceil(fiberdistance))
    {
        lbl_infodialogconfig->set_text("'Stop' must be lower than
'Fiber Distance'");
        infobarconfig->show();
        return false;
    }
    else if(stop<=start)
    {
        lbl_infodialogconfig->set_text("'Stop' must be upper than
'Start'");
        infobarconfig->show();
        return false;
    }

    // Spatial Resolution -- Posibilidad de decimales, redondeando a 1
decimal
    double spatialresolution;
    stringstream stream;
    istringstream(configuracion[4]) >> spatialresolution;
    spatialresolution = (int)ceil(spatialresolution*10);
    spatialresolution/=10;
    stream << fixed << setprecision(1) << spatialresolution;
    stream >> spatialresolution;
    configuracion[4] = stream.str();
    if(spatialresolution>(int)ceil(fiberdistance))
    {
        lbl_infodialogconfig->set_text("'SpatialResolution' must be
lower than 'Fiber Distance'");
        infobarconfig->show();
        return false;
    } else if(spatialresolution>(stop-start)){
        lbl_infodialogconfig->set_text("'SpatialResolution' must be
lower than 'Start-Stop'");
```

```
        infobarconfig->show();
        return false;
    }

    double freqstart;
    istringstream(configuracion[5]) >> freqstart;

    double freqstop;
    istringstream(configuracion[6]) >> freqstop;

    infobarconfig->hide();
    NewConfigurationDialog->hide();
    for (int i=0; i<15; i++)
    {
        if(i==0){info[i]->set_text("");}
        else if(i==1){ent_choosesamplinginterval->set_ac-
tive_text("");}
        else{info[i-1]->set_text("");}
    }
    return true;
}
```

Menú pop-up TreeView

Función para mostrar el menu-popup del click derecho en el TreeView principal.

```
void MainWindow::on_treeview_clicked(GdkEventButton* button_event){
    if((button_event->type == GDK_BUTTON_PRESS) && (button_event->but-
ton == 3))
    {
        m_Menu_Popup.popup(button_event->button, button_event->time);
    }
}
```

Calendario

El calendario ha sido implementado mediante dos funciones. En la primera de ellas marcamos el día seleccionado y mostramos los campos requeridos por la medida.

```
void MainWindow::on_calendarpress_event()
{
    int day = Calendario->property_day();

    //Mostrar o quitar el box, el boton de apply, el dia en el calen-
dario y el boton de fit
    if(boxOptionsCalendar->get_visible() && btn_applyConfMeasure-
>get_sensitive() && Calendario->get_day_is_marked(day)){
        boxOptionsCalendar->hide();
        btn_applyConfMeasure->set_sensitive(false);
        Calendario->unmark_day(day);
        imageLogoMeasurement->show();
```

```
        spinhour->set_value(0);
        spinmin->set_value(0);
        swrepeatdaily->set_active(false);
        swrepeatweekly->set_active(false);
        checkbuttonFitSP->set_active(false);
    } else {
        boxOptionsCalendar->show();
        btn_applyConfMeasure->set_sensitive(true);
        Calendario->mark_day(day);
        imageLogoMeasurement->hide();
    }
}
```

En la segunda función va aplicando cada medida seleccionada en el calendario cuando se pulsa el botón apply.

```
void MainWindow::on_confmeasurement_apply()
{
    Glib::ustring nameFile;
    // Fit //
    string fit;
    if(checkbuttonFitSP->get_active()) fit = "Yes";
    else fit ="No";

    // Sensor //
    auto refSelection = treeviewDialog->get_selection();
    if(refSelection)
    {
        Gtk::TreeModel::iterator iter = refSelection->get_selected();
        if(iter)
        {
            nameFile  = (*iter)[m_Columns.m_col_name];
        }
    }

    // Hora //
    string hour = to_string(spinhour->get_value_as_int());
    if(hour.length()==1) hour = "0" + hour;
    string min = to_string(spinmin->get_value_as_int());
    if(min.length()==1) min = "0" + min;
    string hora = hour + ":" + min;

    // Año-Mes-Dia // Según ISO 8601
    int day = Calendario->property_day();
    int month = Calendario->property_month();
    int year = Calendario->property_year();
    string dia =
to_string(year)+"/"+to_string(month)+"/"+to_string(day);

    string repeat;
    // Repetición diaria/semanal //
    if(swrepeatdaily->get_active() & swrepeatweekly->get_active()) re-
peat = "Daily and Weekly";
    else if (swrepeatdaily->get_active()) repeat = "Daily";
    else if (swrepeatweekly->get_active()) repeat = "Weekly";
    else repeat = "None";
```

```
    if(Calendario->get_day_is_marked(day)){
        Calendario->unmark_day(day);
    } else {
        Calendario->mark_day(day);
    }

    // Reiniciamos los valores
    swrepeatdaily->set_active(false);
    swrepeatweekly->set_active(false);
    btn_applyConfMeasure->set_sensitive(false);
    checkbuttonFitSP->set_active(false);
    spinhour->set_value(0);
    spinmin->set_value(0);
    boxOptionsCalendar->hide();

    // Guardamos en fichero los datos recogidos
    ficheroConfMeasurement[auxiliarConfMeasure] = nameFile.raw()  +
"\n" + dia  + "\n" + hora + "\n" + repeat + "\n" + fit + "\n";

    // Sumamos una medida para tener constancia del tamaño del array
    auxiliarConfMeasure++;
}
```

Función encriptación contraseña admin

```
bool ApplicationScript::admin_password_dialog()
{
    // Seteamos para que los caracteres del password no sean legibles
    Glib::PropertyProxy<bool> caracteresLegibles = ent_adminPassword-
>property_visibility();
    caracteresLegibles = false;

    // Eliminamos el label de wrong password de la vista
    lbl_wrongPassword->hide();

    // Con este while true lo que conseguimos es que el diálogo apa-
rezca siempre hasta que se ponga la contraseña correcta
    // se presione cancelar o la X
    while (true)
    {
        //Show the dialog and wait for a user response:
        int result = m_adminDialog->run();

        ent_adminPassword->grab_focus();

        //Handle the response:
        switch(result)
        {
            case(0): // Accept clicked
            {
                std::cout << "Accept clicked." << std::endl;
                std::cout << "Result: " << result << std::endl;
                // Comprobamos si el texto introducido y haseado coin-
cide con nuestro hash
                if ( (sha256(ent_adminPassword->get_text())).com-
pare("cc3cd70c58b18b93f5ff9792afb46e11a8574ff10d29aa7761dd15a38f176a1a
") == 0 ) // Contraseña: Upna2017
```

```
                {
                    m_adminDialog->hide();
                    return true;
                }
                else
                {
                    lbl_wrongPassword->show();
                    break;
                }
            }
            case(1): // Cancel clicked
            {
                std::cout << "Cancel clicked." << std::endl;
                std::cout << "Result: " << result << std::endl;
                m_adminDialog->hide();
                return false;
            }
            default: // X clicked
            {
                std::cout << "Unexpected button clicked." <<
std::endl;
                m_adminDialog->hide();
                return false;
            }
        }
    }
}
```

Lista de Figuras

Figura 1.1: Gráfica SO más utilizados actualmente. Eje X: Año-Mes; Eje Y: Porcentaje. . 6

Figura 1.2: Tabla de porcentaje SO más utilizados en la actualidad. 7

Figura 1.3: Porcentaje de uso en distribuciones Linux. 8

Figura 1.4: Lenguajes de programación. De izquierda a derecha: C++, Java y Python. ... 9

Figura 1.5: Entornos de desarrollo integrados 17

Figura 1.6: Diversos widgets de GTK+ 18

Figura 1.7: Interfaz de Glade. 20

Figura 2.1: Principio de funcionamiento de los sistemas BOTDA. 21

Figura 2.2: BOTDA en ganancia. 22

Figura 2.3: Simplificación sistema BOTDA. 23

Figura 2.4: Propiedades comunes.....28

Figura 2.5: Propiedades propias 29

Figura 2.6: Botón GtkWindow en Glade y descripción XML. 29

Figura 2.7: Botón GtkApplicationWindow en Glade y ventana MainWindow..... 30

Figura 2.8: Botón GtkDialog en Glade y diálogo ejemplo. 30

Figura 2.9: Botón GtkFileChooserDialog en Glade. 31

Figura 2.10: Diálogo de selección de ficheros. 32

Figura 2.11: Botón GtkMessageDialog en Glade y diálogo de mensaje ejemplo. 32

Figura 2.12: Botón GtkBox en Glade y diálogo ejemplo de dos botones en vertical. 33

Figura 2.13: Atributos del GtkBox. 33

Figura 2.14: Botón GtkNoteBook en Glade y diálogo ejemplo de notebook. 34

Figura 2.15: Botón GtkToolbar en Glade y ejemplo de paleta de herramientas. 34

Figura 2.16: Botón GtkScrolledWindow en Glade 35

Figura 2.17: Scrolled Window. 35

Figura 2.18: Botón GtkAlignment en Glade y descripción XML. 35

Figura 2.19: Botón GtkButton en Glade y dos botones ejemplo. 36

Figura 2.20: Botón GtkCheckButton en Glade y ejemplo de uso. 36

Figura 2.21: Botón GtkSwicth en Glade y ejemplo de conmutador. 37

Figura 2.22: Botón GtkEntry en Glade y ejemplo de entrada. 38

Figura 2.23: Botón GtkSpinButton en Glade y ejemplo usado en formato hora. 38

Figura 2.24: Botón GtkComboBox en Glade y ejemplo hecho en diálogo de medida. .. 39

Figura 2.25: Botón GtkImage en Glade y ejemplo de imagen en diálogo. 39

Figura 2.26: Botón GtkLabel en Glade y ejemplo de etiqueta en diálogo. 40

Figura 2.27: Botón GtkSeparator en Glade y ejemplo de separadores. 40

Figura 2.28: Botón GtkCalendar y ejemplo de calendario usado en la aplicación. 41

Figura 2.29: Botón GtkInfoBar en Glade y ejemplo en diálogo. 42

Figura 2.30: Botón GtkTreeView en Glade y ejemplo de TreeView en árbol. 43

Figura 2.31: Popup menú. 44

Figura 2.32: Botón GtkAdjustment en Glade. 44

Figura 2.33: Atributo de ajuste del SpinButton y propiedades del ajuste. 45

Figura 2.34: Botón GtkFileFilter en Glade y ajustes del filtro archivos dmac y dmam. . 45

Figura 2.35: Organización del software desarrollado. 46

Figura 2.36: Ventana de administrador. 51

Figura 2.37: Organización de las funciones de nuevo sensor y medida. 52

Figura 2.38: Organización de la función de editar ficheros. 53

Figura 2.39: Organización de la función de visualizar ficheros. 53

Figura 2.40: Organización de la función de abrir ficheros. 54

Figura 2.41: Organización de la función de exportar ficheros. 54

Figura 2.42: Organización de la función de importar ficheros. 55

Figura 3.1: Ventana principal de la aplicación. 57

Figura 3.2: Botón de nueva configuración y su correspondiente diálogo. 58

Figura 3.3: Diálogo de nueva configuración de sensor. 58

Figura 3.4: Diálogo de configuración de una medida. 59

Figura 3.5: Diálogo de configuración de medida un número especifico de veces. 59

Figura 3.6: Diálogo de conf. medida periódicamente un número especifico de veces. 60

Figura 3.7: Diálogo de configuración de medida conforme al calendario. 60

Figura 3.8: Botón de apertura de ficheros y su correspondiente diálogo. 61

Figura 3.9: Botón de cerrar/borrar fichero y su correspondiente diálogo. 61

Figura 3.10: Botón de importar ficheros y su correspondiente diálogo. 62

Figura 3.11: Ventana de importación de ficheros. 62

Figura 3.12: Botón de exportar ficheros y su correspondiente diálogo. 63

Figura 3.13: Ventana de exportación de ficheros. 63

Figura 3.14: Botones correspondientes a lanzar medida y detenerla. 63

Figura 3.15: Botón de ajuste y sus correspondientes diálogos. 64

Figura 3.16: Botón de login y su correspondiente diálogo de contraseña. 64

Figura 3.17: Botón acerca de y su correspondiente diálogo. 65

Figura 3.18: Edición de configuración de sensor. 66

Figura 3.19: Edición de configuración de medida. 66

Figura 3.20: Visualización de configuración de sensor. 66

Figura 3.21: Edición de configuración de medida. 66

Figura 3.22: Panel de visualización de señales. 67

Figura 3.23: Sección File del menú de la aplicación y nueva configuración 68

Figura 3.24: Secciones de edición y visualización de ficheros respectivamente. 68

Figura 3.25: Elemento del menú correspondiente con los ajustes. 69

Figura 3.26: Elementos de ayuda del menú. 69

Bibliografía

[1] Gnu [En línea]. https://www.gnu.org/home.es.html. [08/18].

[2] Debian [En línea]. https://www.debian.org/releases/stable/s390x/ch01s02.html.es. [08/18].

[3] ADSLZone [En línea]. https://www.adslzone.net/2017/01/19/windows-vs-linux-vs-macos-consume-mas-memoria-ram/. [08/18].

[4] NetMarketShare [En línea]. https://www.netmarketshare.com/operating-system-market-share.aspx. [08/18].

[5] Genbeta [En línea]. https://www.genbeta.com/linux/31-distribuciones-de-linux-para-elegir-bien-la-que-mas-necesitas. [08/18].

[6] Cloudbalkan [En línea]. https://www.cloudbalkan.com/most-popular-linux-distributions-from-2017/. [08/18].

[7] B. Stroustrup, El lenguaje de programación C++, Madrid: Addison Wesley, 1998.

[8] Opera [En línea]. https://www.opera.com/es/about. [08/18].

[9] Android [En línea]. https://www.android.com/. [08/18].

[10] Ableton [En línea]. https://www.ableton.com/en/live/what-is-live/). [08/18].

[11] Python [En línea]. https://www.python.org/doc/essays/blurb/. [08/18].

[12] Calibre [En línea]. https://calibre-ebook.com/. [08/18].

[13] Flumotion [En línea]. http://www.flumotion.com/. [08/18].

[14] Twisted [En línea]. https://twistedmatrix.com/users/glyph/ipc10/paper.html. [08/18].

[15] Java [En línea]. https://www.java.com/es/download/faq/whatis_java.xml. [08/18].

[16] Minecraft [En línea]. https://minecraft.net/es-es/what-is-minecraft/. [08/18].

[17] Eclipse [En línea]. http://help.eclipse.org/photon/index.jsp. [08/18].

[18] Veracode [En línea]. https://www.veracode.com/security/integrated-development-environments. [08/18].

[19] Netbeans [En línea]. https://netbeans.org/projects/www/. [08/18].

[20] GTK+ [En línea]. https://www.gtk.org/. [08/18].

[21] Glade [En línea]. https://glade.gnome.org/. [08/18].

[22] GtkBuilder [En línea]. https://developer.gnome.org/gtk3/stable/GtkBuilder.html. [08/18].

[23] GtkApplication [En línea]. https://developer.gnome.org/gtk3/stable/gtk-getting-started.html. [07/18].

[24] GtkBuilder [En línea]. https://developer.gnome.org/gtk3/stable/GtkBuilder.html. [06/18].

[25] GtkWindow [En línea]. https://developer.gnome.org/gtk3/stable/GtkWindow.html. [06/18].

[26] GtkApplicationWindow [En línea]. https://developer.gnome.org/gtk3/stable/-GtkApplicationWindow.html. [06/18].

[27] GtkDialog [En línea]. https://developer.gnome.org/gtk3/stable/GtkDialog.html. [07/18].

[28] GtkFileChooserDialog [En línea]. https://developer.gnome.org/gtk3/stable/-GtkFileChooserDialog.html. [07/18].

[29] GtkMessageDialog [En línea]. https://developer.gnome.org/gtk3/stable/-GtkMessageDialog.html. [07/18].

[30] GtkBox [En línea]. https://developer.gnome.org/gtk3/stable/GtkBox.html. [06/18].

[31] GtkNoteBook [En línea]. https://developer.gnome.org/gtk3/stable/GtkNotebook.html. [07/18].

[32] GtkToolbar [En línea]. Available: https://developer.gnome.org/gtk3/stable/GtkToolbar.html. [07/18].

[33] GtkScrolledWindow [En línea]. https://developer.gnome.org/gtk3/stable/GtkScrolledWindow.html. [07/18].

[34] GtkAlignment [En línea]. https://developer.gnome.org/gtk3/stable/GtkAlignment.html. [07/18].

[35] GtkButton [En línea]. https://developer.gnome.org/gtk3/stable/GtkButton.html. [06/18].

[36] GtkCheckButton [En línea]. https://developer.gnome.org/gtk3/stable/GtkCheckButton.html. [07/18].

[37] GtkSwitch [En línea]. https://developer.gnome.org/gtk3/stable/GtkSwitch.html. [07/18].

[38] GtkEntry [En línea]. https://developer.gnome.org/gtk3/stable/GtkEntry.html. [07/18].

[39] GtkSpinButton [En línea]. https://developer.gnome.org/gtk3/stable/GtkSpinButton.html. [07/18].

[40] GtkComboBox [En línea]. https://developer.gnome.org/gtk3/stable/GtkComboBox.html. [07/18].

[41] GtkImage [En línea]. https://developer.gnome.org/gtk3/stable/GtkImage.html. [07/18].

[42] GtkLabel [En línea]. https://developer.gnome.org/gtk3/stable/GtkLabel.html. [07/18].

[43] GtkSeparator [En línea]. https://developer.gnome.org/gtk3/stable/GtkSeparator.html. [08/18].

[44] GtkCalendar [En línea]. https://developer.gnome.org/gtk3/stable/GtkCalendar.html. [07/18].

[45] GtkInfoBar [En línea]. https://developer.gnome.org/gtk3/stable/GtkInfoBar.html. [07/18].

[46] GtkTreeView [En línea]. https://python-gtk-3-tutorial.readthedocs.io/en/latest/treeview.html. [07/18].

[47] GtkMenu [En línea]. https://developer.gnome.org/gtk3/stable/GtkMenu.html. [07/18].

[48] GtkAdjustment [En línea]. https://developer.gnome.org/gtk3/stable/GtkAdjustment.html. [07/18].

[49] GtkFileFilter [En línea]. https://developer.gnome.org/gtk3/stable/GtkFileFilter.html. [07/18].

[50] Egit [En línea]. https://eclipsesource.com/blogs/tutorials/egit-tutorial/. [08/18].