
# Trabajo Fin de Grado

Integración de tecnologías de medida de actividad fisiológica y de EEG en los productos NeuroLab, usenns y BrainUp

Autor
Adrián Sánchez Recuero

Director
Javier Minguez Zafra

Ponente
Luis Montesano del Campo

Escuela de Ingeniería y Arquitectura/ Universidad de Zaragoza
2014

# Integración de tecnologías de medida de actividad fisiológica y de EEG en los productos NeuroLab, usenns y BrainUp

## RESUMEN

La empresa *BitBrain Technologies* es una empresa de nuerociencia y neurotecnología centrada actualmente en el desarrollo de una tecnología propia de neuromarketing. El trabajo presentado en este documento se encuentra dentro de este marco, en el cual se ha realizado la integración de los diversos dispositivos hardware inalámbricos (utilizando WIFI y Bluetooth) desarrollados en *BitBrain* con toda la arquitectura software de tiempo real existente en la empresa .

Para ello se ha construido una arquitectura software que abstrae a los desarrolladores del bajo nivel de las comunicaciones y que permite una escalabilidad y mantenimiento sencillos, además de desarrollar el firmware que irá embebido en los dispositivos.

A nivel metodológico global, se ha trabajado dentro de la metodología *scrum* implementada en la empresa, siguiendo un proceso de integración continua, realizando pruebas unitarias de cada parte desarrollada y pruebas de integración en las fases finales del proyecto.

# Índice general

# Capítulo 1

# Introducción

## 1.1. Contexto

*BitBrain Technologies* es una empresa de I+D+I especializada en neurociencia. Con su equipo multidisciplinar (cuenta con psicólogos, ingenieros informáticos, en telecomunicaciones y electrónicos) cuenta con dos lineas principales de negocio: la salud y el *neuromarketing*, la más importante en la actualidad.
Para poder avanzar en estas líneas de trabajo, se encuentran desarrolladas, o en fase de desarrollo distintas aplicaciones y dispositivos hardware. Todo el software desarrollado se encuentra construido sobre *BziFramework*, un *framework* de tiempo real concebido en la empresa. Las aplicaciones en las que se encuentran trabajando son *NeuroLab*, *BrainUp* y *ussens*.
*NeuroLab* permite la creación de experimentos en los cuales se requieran obtener distinta información del sujeto mediante hardware de adquisición de señales fisiológicas, *eyetrackers*, webcams, etc. *BrainUp* se centra en la mejora cognitiva del paciente. Permite la creación de una terapia personalizada pudiendo configurar las sesiones, fases y *trials* en función de las necesidades del paciente. El entrenamiento del paciente se realiza ofreciendo un estímulo visual en función del análisis de la actividad eléctrica del cerebro. Por último, *ussens* es una adaptación de *NeuroLab* para experimentos de *neuromarketing*. El programa cuenta con plantillas predefinidas en las cuales se añadirán los elementos audiovisuales a analizar.

Complementando al material software, se ha desarrollado un anillo WIFI de adquisición de medidas fisiológicas como son *Blood Volume Pulse* (BVP) y *Galvanic Skin Response*(GSR) (pensado para aplicaciones de *neuromarketing*) y un amplificador Bluetooth de

señales biomédicas, como son *electroencefalograma* (EEG) y *electromiograma* (EMG) (que complementaría al software *BrainUp*).

## 1.2. Necesidades identificadas y objetivos

Una vez analizado el ecosistema de tecnologías existentes, se identifico la principal necesidad. Dotar al framework *BziFramework* de una arquitectura escalable que permita gestionar las comunicaciones con los distintos dispositivos (los que se encuentran en desarrollo o los que se creen en un futuro), para que todas las aplicaciones que se construyan sobre este framework cuenten con la posibilidad de utilizar los distintos dispositivos de manera transparente. Este es el gran objetivo que se abordará en este proyecto.

Para la consecución de este gran objetivo, se marcaron distintos subobjetivos que permitieran valorar el avance del proyecto.

- Diseño de la arquitectura planteada.
- Implementación de librerías de conexión Bluetooth y WIFI, testeadas con servidores *dummy*
- Integración de las dos librerías en *BziFramework*. Pruebas de verificación de la integración.
- Desarrollo y depuración del firmware embebido en el anillo WIFI y en el amplificador Bluetooth.
- Diseño e implementación de adquisición simultanea de varios anillos.
- Integración de la adquisición WIFI en *NeuroLab*

Una vez alcanzados estos objetivos, la empresa cuenta con un producto comercial, que se encuentra en estos momentos con un programa de partners en el cual, empresas e instituciones puede hacer uso de la tecnología. Este programa tiene como finalidad adaptar y depurar las funcionalidades del sistema a las necesidades reales de los clientes.

## 1.3. Alcance del trabajo realizado

En la figura 1.1 tenemos una descripción de la arquitectura del software de *BitBrain* y los roles que se pueden adoptar, dependiendo de en que parte se desarrolle o que aplicaciones se utilicen. Rodeados con un círculo se encuentran los roles que se han adoptado en

la realización de este PFG. En el rol de *BZI core developer* se han realizado dos librerías

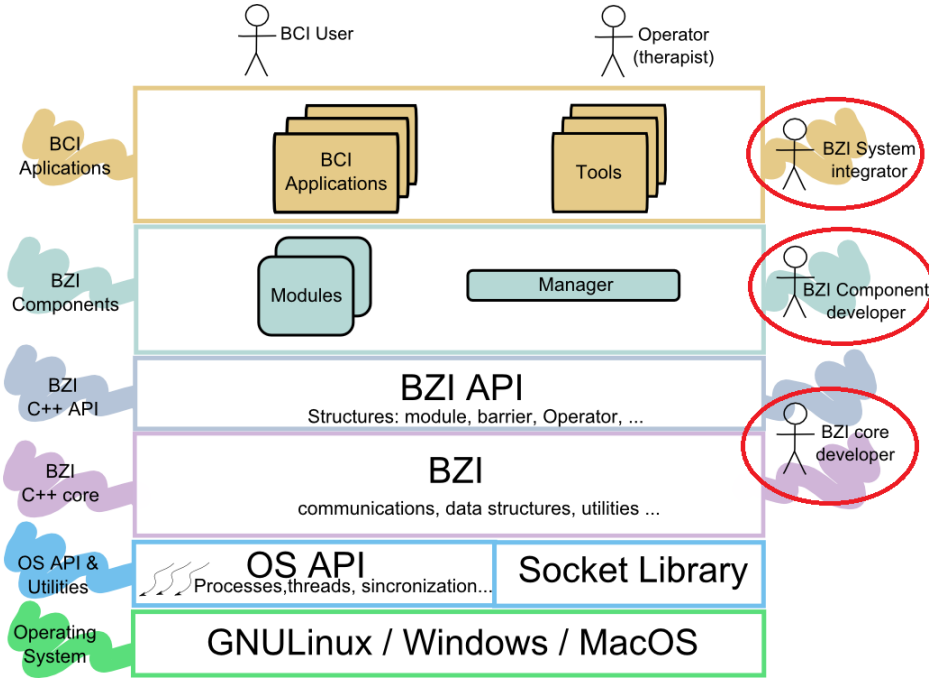


Figura 1.1: Capas de la arquitectura software

de comunicación, una WIFI y otra Bluetooth para permitir la comunicación con los dispositivos.

En el rol de *Bzi component developer* se ha realizado la integración de dichas librerías en los *Modules* representados en la figura.

Como *Bzi System integrator*, se ha integrado en la aplicación *NeuroLab* la adquisición de los anillos WIFI. También se han realizado distintos visores (interfaces gráficas), tanto para los dispositivos WIFI como para el dispositivo Bluetooth.

# Capítulo 2

# Sistemas existentes

## 2.1. BziFramework y NeuroLab

### 2.1.1. BziFramework

En esta sección vamos a explicar las características del *Framework Bzi* en el cual hemos integrado el trabajo realizado en este proyecto.

La figura 2.1 muestra los distintos componentes que conforman la arquitectura de este framework. Estos componentes son *Manager,* un conjunto de *módulos (Modules)* y un *Controller*. Los módulos realizan tareas como la adquisición de señal, tratamiento de señal, etc. El Manager controla el flujo de ejecución de los módulos y el intercambio de información entre ellos. El Controller trabaja como una API controlando la información entre el Manager y aplicaciones de terceros. A través del Controller y el Manager el flujo de ejecución puede ser modificado, activando, desactivando y reconfigurando los módulos.

El Manager y los módulos son procesos que se comunican a través de la red mediante el protocolo TCP/IP. Los módulos son los componentes encargados de realizar tareas de alto nivel, como por ejemplo, adquirir y preprocesar señal. Cada módulo tiene un conjunto de datos de entrada, que procesa y genera un conjunto de datos de salida. Tantos los datos de entrada como de salida son enviados a través de la red, supervisado por el Manager.

Las unidades de procesamiento (*units*) son los componentes básicos de la arquitectura. Una unidad se encarga de realizar una única tarea, como por ejemplo, adquirir señal, aplicar un filtro o eliminar artefactos. Las unidades están conectadas secuencialmente, obteniendo los datos da la unidad predecesora en el orden de ejecución y poniendo sus

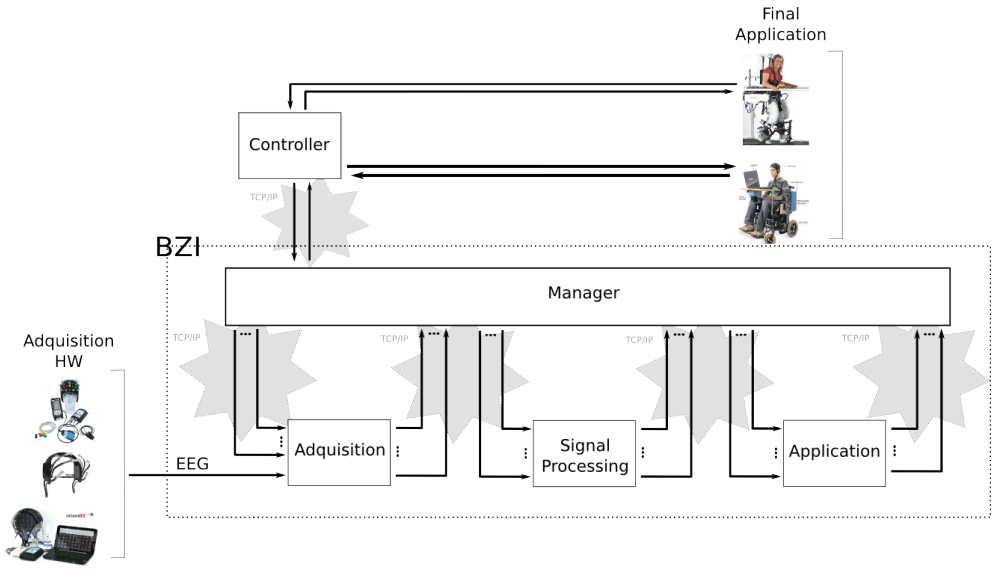


Figura 2.1: Arquitectura Bzi Framework

datos de salida disponibles a la siguiente unidad, para lo cual se usa un repositorio de datos común. Las unidades se encuentran agrupadas dentro de un módulo, que es quien tiene control de la información intercambiada. Las unidades que terminan su función de ejecución esperan dormidas a que comience un nuevo ciclo de ejecución.

Las figuras 2.2 2.3 muestran este esquema de componentes.

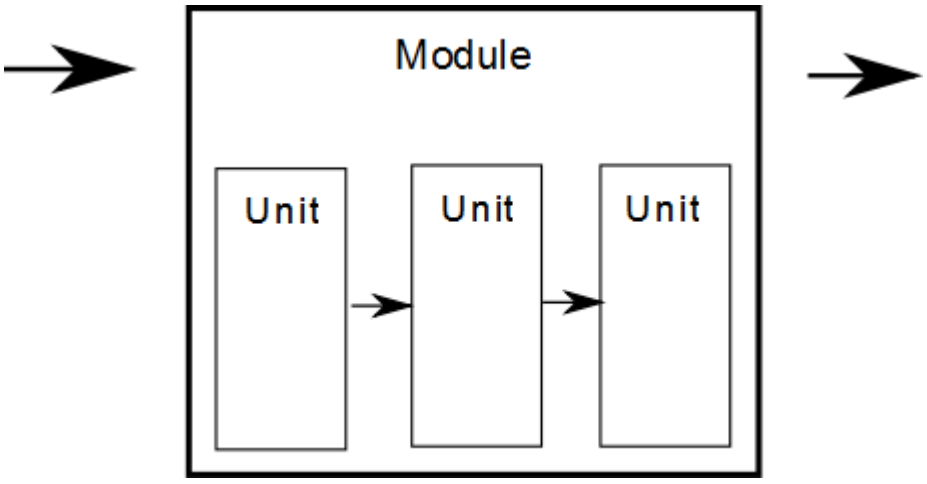


Figura 2.2: Módulo y unidades

Los módulos están organizados en etapas. Cada etapa puede contener mas de un módulo, que se ejecutaran de manera paralela. Todos los módulos de una etapa comienzan

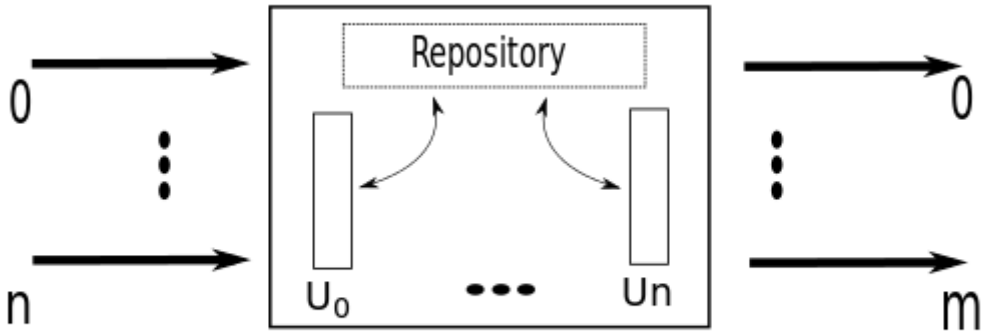


Figura 2.3: Estructura módulo

en el mismo momento. La etapa *i+1* solo empieza a ejecutar cuando todos los módulos de la etapa *i* han finalizado su ejecución. Esta sincronización es controlada por el manager.

La figura 2.4 muestra lo explicado anteriormente.

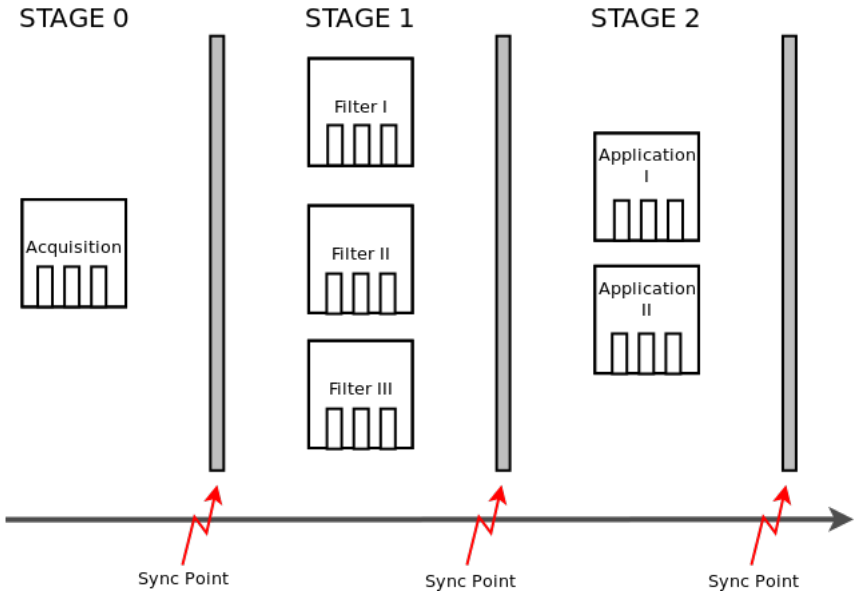


Figura 2.4: Estructura etapas

### 2.1.2. NeuroLab

NeuroLab es una aplicación que permite crear experimentos y estudios centrados en las señales fisiológicas del sujeto a estudiar. Un experimento consistirá de una o más tareas a realizar, y en cada tarea se podrán presentar estímulos audiovisuales. En estos experimentos se podrán agregar diversos dispositivos de captura de señales fisiológicas del sujeto que está realizando el experimento (dipsositivos WIFI *usenns*, amplificadores de Tmsi, Gtec) u otros dispositivos como son eyetrackers o cámaras Ip. Una vez terminado el experimento, los datos se exportan a fichero para poder realizar un análisis de los datos.

## 2.2. Dispositivo WIFI

### 2.2.1. Características generales

A continuación vamos a explicar el funcionamiento del dispositivo *wearable* ussens, especializado en la adquisición de medidas fisiológicas, como son *Blood Volume Pulse* (BVP) y *Galvanic Skin Response*(GSR), además de contar con un acelerómetro. El envío de los datos se realiza mediante un salto WIFI

El diseño del dispositivo esta basado en el hadware libre Flyport, que cuenta con una antena WIFI G y un procesador de la empresa Microchip. Este procesador tiene una frecuencia efectiva de 16 MHz.

A nivel de software, al basarse en Flyport se puede utilizar el entorno de desarrollo que proporcionan. Ente entorno proporciona un interfaz gráfico para la configuración de la conexión de red y una serie de funciones que simplifican la conexíon vía red. Estas librerías están construidas sobre el sistema operativo FreeRTOS (`http://www.freertos.org/`).

### 2.2.2. Software embebido

La adquisición de datos se realiza en una interrupción que ocurre a una frecuencia de 16 Hz. En esta interrupción se leen los distintos sensores del dispositivo y se encolan los valores en una estructura de datos que sirve de sincronización entre la interrupción y el código principal. A partir de este momento denominaremos *sample* al conjunto de señales adquiridas por el dispositivo en un instante *t*. Esta estructura de datos fue introducida a raíz de detectar un problema de concurrencia en la realización de pruebas. La estructura de sincronización viene dada por el sistema operativo.

En la figura 2.5 podemos observar como es cada sample enviado por el dispositivo, con la separación de las distintas señales enviadas. Cada paquete que el dispositivo envía consta de dos samples.

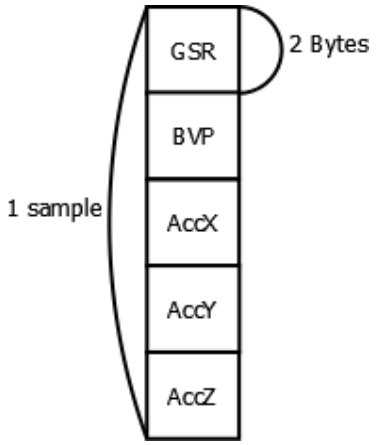


Figura 2.5: Sample enviado por el dispositivo WIFI

## 2.3. Amplificador EEG Bluetooth

### 2.3.1. Características generales

El segundo dispositivo tratado es un amplificador de señales de EEG y EMG, además de contar con entrada para señales digitales, como puede ser un fotodiodo o un pulsador. El envío de datos se realiza mediante un salto Bluetooth.

El procesador que controla la placa es un PIC de Microchip. La antena Bluetooth (versión 2.1 del protocolo) es un modelo de la empresa Roving Networks. A diferencia del dispositivo WIFI, en esta ocasión no se cuenta con una arquitectura software como la que ofrece Flyport.
Debido a los amplificadores de señal utilizados, nos encontramos con que cada canal de adquisición producirá un valor de tres bytes en *Big Endian* que deberá ser procesado en la recepción.

### 2.3.2. Software embebido

La aportación en el desarrollo firmware del dispositivo fue la creación de los paquetes de información que se enviarán al cliente, buscando que el tratamiento de la información

en el cliente fuera lo más ligero posible.

En la figura 2.6 podemos observar como es cada sample enviado por el dispositivo.

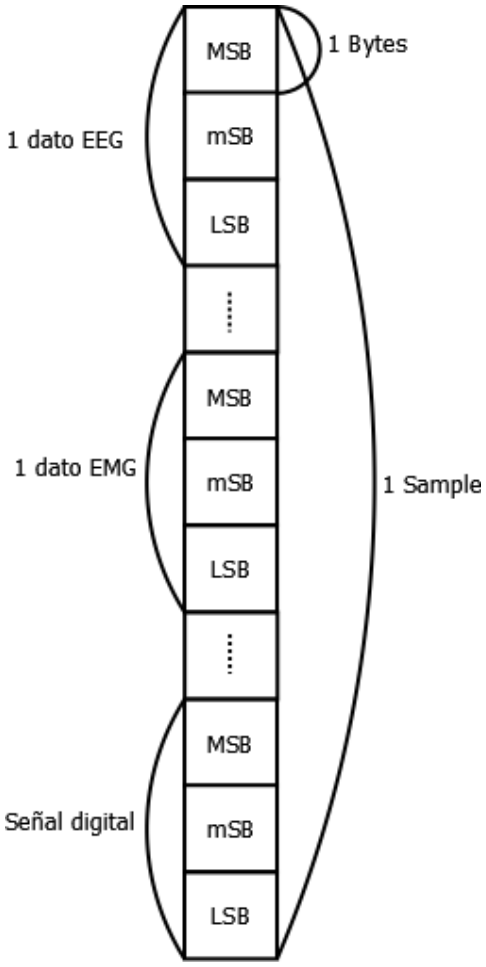


Figura 2.6: Sample enviado por el dispositivo Bluetooth

# Capítulo 3

# Requisitos

En este capítulo se van a explicar los requisitos obtenidos tras las primeras reuniones mantenidas, que permitirán establecer y desarrollar la solución que mejor se adapte al problema a resolver.

## 3.1. Requisitos definidos

### Lenguaje y framework de desarrollo

Todo el código desarrollado debe de estar hecho en el lenguaje C++ y el framework Qt5. Además deberá de estar integrado en el framework *BziFramework* y en la aplicación *NeuroLab*.

### Tiempo real

La arquitectura en la cual se integrará la solución aportada, es una arquitectura de tiempo real. Este tiempo real viene marcado por la frecuencia de envío de los dispositivos con los cuales mantengamos comunicación. En las aplicaciones de *BitBrain* este tiempo será de 31.25 ms. Esto repercute en que la eficiencia, tanto en tiempo (coste computacional de adquisición y tratamiento de la información recibida) como en gestión de la memoria (creación y destrucción de objetos, estructuras de datos utilizadas) sea muy importante.

## Modularidad, escalabilidad, usabilidad, robustez

La solución deberá de dar soporte a los dos dispositivos hardware (anillo WIFI y amplificador Bluetooth) que cuenta actualmente la empresa, pero también deberá de facilitar todo lo posible la integración de nuevo hardware que se pudiera desarrollar en un futuro. También se debe de dar soporte a la utilización de varios dispositivos simultáneamente (en el caso de anillos WIFI). Se deberá crear una capa de abstracción del bajo nivel de las comunicaciones (configuración, comienzo y terminación de la conexión, inicialización de variables y objetos que intervengan en la conexión) proporcionando una interfaz que abstraiga de todo este proceso. Por último todos los módulos deberán de implementar un plan de pruebas que garanticen su correcto funcionamiento y gestionar los errores y eventos que puedan ocurrir, de manera que se informe del tipo de error para poder elegir la respuesta adecuada.

## Anillo WIFI de medición de señales

El módulo WIFI deberá obtener los datos de los diferentes sensores (sensor de GSR, BVP y acelerómetro), empaquetarlos y enviarlos al cliente que solicita los datos mediante una conexión TCP, para asegurar la recepción y el orden de los paquetes enviados.

## Amplificador Bluetooth

Reordenar la lectura de los sensores conectados para facilitar la interpretación de los datos en el cliente.

# Capítulo 4

# Metodología y herramientas utilizadas

## 4.1. Metodología utilizada

Todo el trabajo realizado ha estado dentro de los ciclos de *scrum* [2] (metodología ágil implementada en la empresa). Esta metodología paso por dos fases muy diferenciadas. En la primera, las reuniones diarias se celebraban entre el director de proyecto, el jefe de equipo y trabajador para definir el trabajo a seguir a lo largo del día. Una vez que se adaptó el trabajo de los distintos equipos a esta manera de trabajo, se pasó a realizar las típicas reuniones de Scrum enfrente de un tablón de trabajo en el cual cada jefe de equipo explicaba los avances, problemas y trabajo futuro del equipo.
A la hora de planificar el trabajo, a nivel individual, se ha seguido una metodología de desarrollo incremental, realizando extensas pruebas de cada parte desarrollada (pruebas unitarias) y de la integración de los módulos realizados (pruebas de integración), buscando el detectar los fallos en fases tempranas, y de esta manera poder acotar y determinar la causa del fallo más fácilmente.

## 4.2. Herramientas utilizadas

Para la realización de este proyecto, se han utilizado diversas tecnologías, utilidades y lenguajes de programación. Para el desarrollo de los programas clientes, tanto para los dispositivos WIFI como para el dispositivo Bluetooth, se ha utilizado el lenguaje C++ sobre el framework QT en su versión 5 (`https://qt-project.org/`), ya que el sistema en el cual se realizará la integración de lo desarrollado está realizado con estas tecnologías. Este

*framework opens source* proporciona una mejora de los tipos de datos y estructuras que ofrece el lenguaje (tipo string, listas o vectores, por ejemplo) además del potente sistema de *signal-slots* explicado en el apéndice A.
El amplificador Bluetooth está desarrollado en el lenguaje C, en el entorno MPLAB para la programación de microprocesadores. El anillo WIFI está desarrollado en el lenguaje C en el IDE de Flyport(`http://www.openpicus.com/site/products`), que nos abstrae de la configuración y comunicación con la antena WIFI del micro.
Para las pruebas del cliente Bluetooth se utilizó Arduino (`http://arduino.cc/`) para simular el servidor y el programa TeraTerm (`http://ttssh2.sourceforge.jp/`), que permite abrir una terminal en un puerto COM del computador) para validar el programa de Arduino.

Para las pruebas del cliente WIFI se utilizó el programa TCP/IP Builder (`http://www.drk.com.ar/builder.php`), programa que puede actuar como cliente o servidor en comunicaciones TCP) para validar los programas que se desarrollaron tanto en Flyport como los clientes en C++.

# Capítulo 5

# Diseño de la solución

Es este capítulo vamos a detallar la solución aportada para cumplir los requisitos mencionados en el capítulo anterior, explicando las distintas opciones disponibles y el porqué de la solución adoptada.

## 5.1. Arquitectura para la adquisición de un dispositivo.

En la figura 5.1 observamos la arquitectura creada para resolver el problema. Contamos con dos librerías separadas. Una de las librerías se encargará de gestionar la comunicación con dispositivos WIFI y la otra será la encargada de comunicar con dispositivos Bluetooth. Esta división en dos librerías separadas se debe, por un lado a la necesidad de exportar la librería Bluetooth para un proyecto externo. Por otro lado, la utilización de librerías de conexión distintas (para la librería WIFI se utilizan las utilidades que ofrece Qt para la conexión TCP mientras que para la librería Bluetooth, se utiliza una utilidad multiplataforma implementada en *BziFramework* para el acceso a puertos series del computador). En la figura 5.2 podemos ver los diagramas de clases seguidos para la implementación de estas librerías.

Sin embargo, a pesar de contar con dos librerías separadas, la arquitectura es similar en ambas.
El diseño está basado en el patrón *facade* [1], que nos permite abstraer al usuario de todo el bajo nivel de la comunicación a través de una única interfaz, que ofrece la posibilidad de comenzar y terminar la conexión con el dispositivo y la obtención de la información recibida.

A nivel de la gestión de la conexión con el dispositivo, se cuenta con un *thread* dedicado exclusivamente a la adquisición de datos del dispositivo. Estos datos son los que la

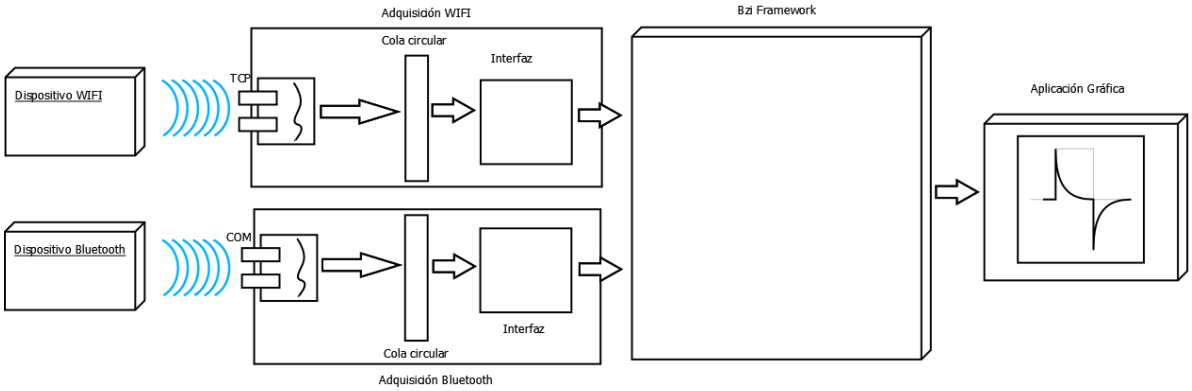


Figura 5.1: Arquitectura solución

librería ofrecerá al exterior. Con esta arquitectura planteada, se presenta el problema de concurrencia de *productores-consumidores*. Para solucionar este problema, se encuentra implementada una estructura de datos que cumplirá con una doble función. La primera, servir de almacén donde el hilo de adquisición podrá volcar los datos que reciba por la red para que el hilo principal los consuma cuando los necesite. En segundo lugar, toda la gestión de bloques de señales (cuando un bloque esta completo, gestión de la acumulación de bloques, informar de la pérdida de información) la realiza la estructura, de manera que el hilo principal leerá siempre bloques completos de información. Esta estructura de datos se explicará en detalle a continuación. El acceso a la estructura de datos por parte del *thread* de adquisición y del hilo principal se realiza en exclusión mutua, gracias a la utilización de un *mutex* compartido.
Estas librerías no realizan ningún tratamiento en la señal, por lo que se podrán utilizar para conectar con cualquier dispositivo, siempre y cuando el formato de los datos de envío se corresponda con el modelo de envío de $n$ datos (señales) de igual tamaño (en bytes).

A nivel de gestión de eventos y errores, la librería WIFI cuenta con el tratamiento de pérdidas de conexión inesperadas. Si se recibe eventos que indiquen que el servidor al cual estábamos conectados ha cerrado la conexión o no responde, se intentará recobrar la comunicación. Estos eventos de conexiones, desconexiones y reconexiones se marcan con una serie de eventos creados que pueden ser adquiridos a través de la interfaz de la librería. Los eventos son los presentados en el cuadro 5.1

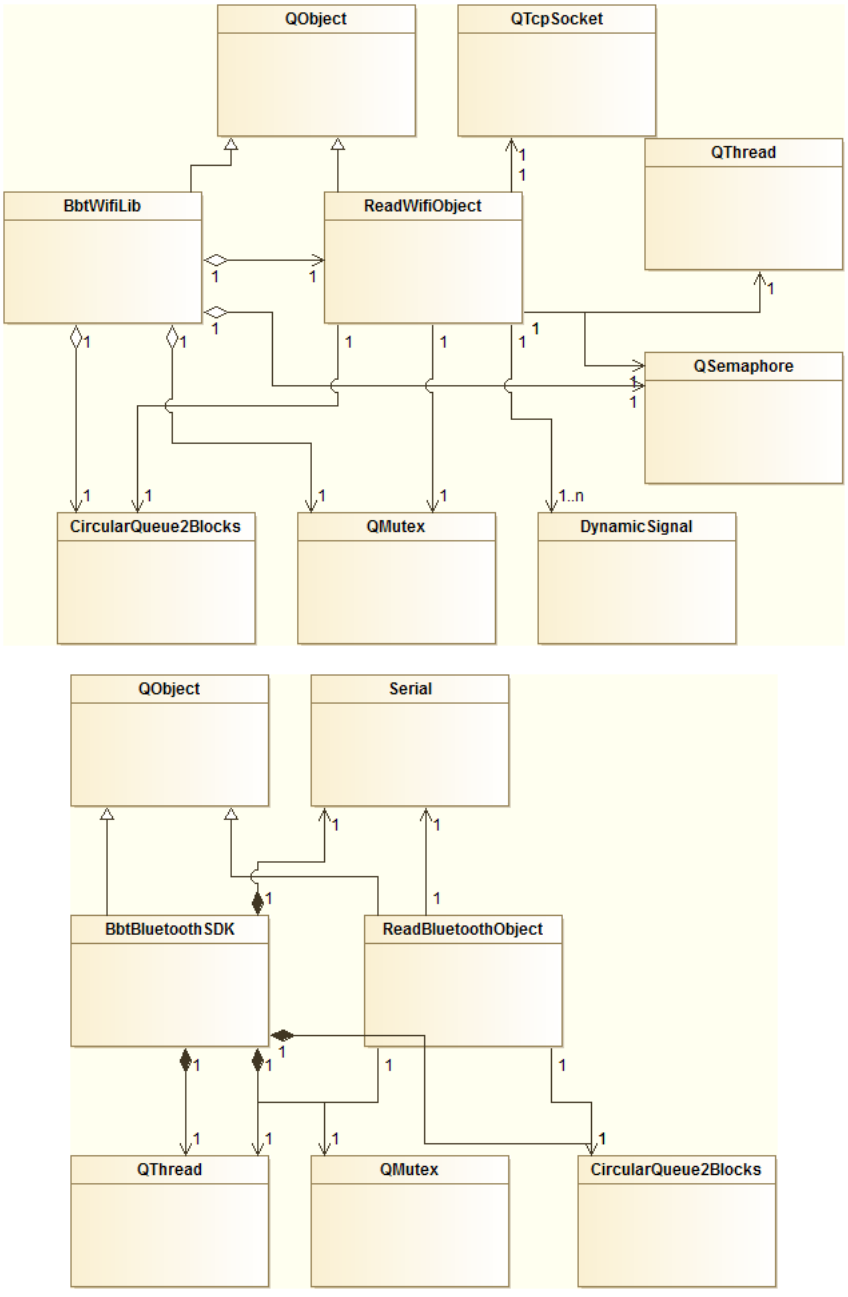


Figura 5.2: Diagrama de clases de las librerías implementadas

| Nombre evento | Descripción evento | Valor |
|---|---|---|
| NO_EVENT | Ningún evento relevante | 0 |
| CONNECTION_DONE | Primera conexión realizada con éxito | 1 |
| CONNECTION_FAIL | Fallo en la primera conexión | 2 |
| STARTING_RECONNECT | Comienza intento de reconexión (por demasiado tiempo sin recibir datos) | 3 |
| RECONNECTION_DONE | Reconexión realizada con éxito | 4 |
| RECONNECTION_FAIL | Fallo en la reconexión | 5 |
| THREAD_RECONNECTING_ALONE | Hilo de lectura empieza reconexión (detectado perdida del servidor) | 6 |
| THREAD_RECONNECTED_ALONE_DONE | Reconexión realizada con éxito | 7 |
| THREAD_RECONNECTED_ALONE_FAIL | Fallo en la reconexión | 8 |
| SOCKET_ERROR_DETECTED | Fallo en la red | 9 |

Cuadro 5.1: Eventos de red detectados

### 5.1.1. Cola circular

Como se ha explicado anteriormente, la cola circula se implementó para solventar el paso de información recibida por el hilo dedicado a la red y el resto del sistema. A continuación se pasará a explicar las características de esta estructura de datos.

#### Funcionamiento general de la cola circular

La estructura funciona como un vector FIFO donde tendremos almacenados como máximo dos bloques de samples. Mientras que no dispongamos de un bloque completo en el vector, la función de lectura devolverá un código de error, indicando que no se ha producido la lectura. Si la lectura es satisfactoria, el bloque leído se eliminará del vector. Es posible que se añadan nuevos samples cuando no se ha producido ninguna lectura y tengamos el vector completo con dos bloques. La solución adoptada, opta por ir eliminando los samples más antiguos e ir anotando el número de eliminados. De esta manera intentamos minimizar el número de información perdida. ¿Por qué eliminar samples y no hacer que la estructura almacene más bloques de información? Las aplicaciones que se construyan por encima trabajan en ciclos periódicos de ejecución, y en cada ciclo consumen un bloque de información. El hecho de que se acumulen más de dos bloques de información quiere decir que ha ocurrido algún problema o retraso en la aplicación y esta no ha sido capaz de consumir datos a la frecuencia de tiempo real necesaria (en el caso de las aplicaciones de *BitBrain*, 31.25 ms). Si esto ocurre es mejor leer información reciente y no la información del pasado.

#### Constructor e inicialización

El constructor de la estructura recibe tres parámetros: el número de samples por bloque, el número de señales que conforman un sample y los bytes que ocupa cada señal. El coste

computacional de la creación de la estructura es O(n) en el tamaño del vector circular, ya que debemos reservar su espacio en memoria.

### Inserción de datos

La inserción de datos en la cola circular, solo se produce cuando hemos recibido un sample completo. Mientras tanto, la información se almacena en una estructura auxiliar. La inserción de un sample en la cola circular puede producir las siguientes situaciones.

- Sin bloque completo.
- Un único bloque completo.
- Un bloque completo y parte del segundo.
- Dos bloques completos.
- Desbordamiento de los dos bloques.

A continuación analizaremos los casos más interesantes.

#### Un único bloque completo

El bloque entero estará disponible para su lectura. Una vez leído, será eliminado de la cola. En la figura 5.3 tenemos una representación de la cola en este estado.

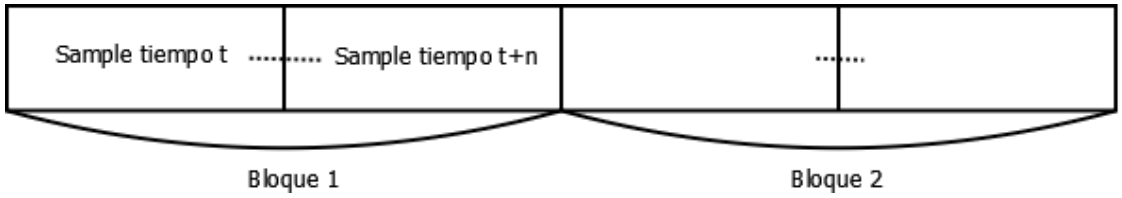


Figura 5.3: Cola circular con una bloque completo

#### Dos bloques completos

En este caso podremos hacer dos lecturas consecutivas sin tener fallo. El primer bloque leído será el bloque más antiguo (el bloque 1 de la figura 5.4).

#### Desbordamiento de los dos bloques

Descartamos el sample más antiguo, como podemos ver en la figura 5.5.

El tiempo computacional de insertar un sample en la cola es lineal en el tamaño del sample.

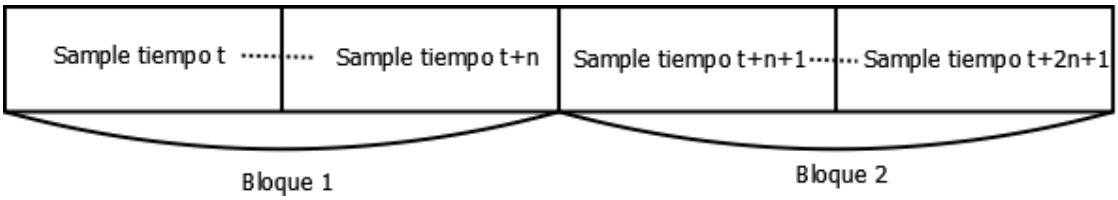


Figura 5.4: Cola circular con dos bloques completos

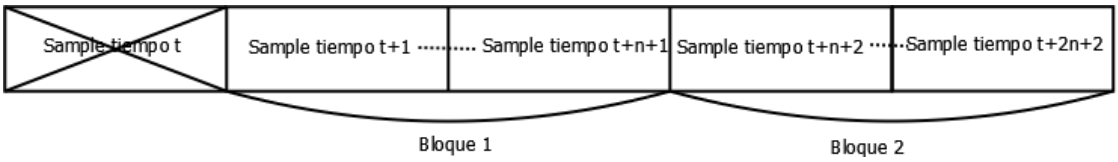


Figura 5.5: Descarte del sample más antiguo

**Lectura de datos**

La función de lectura de datos devolverá el bloque más antiguo (si existiera) y el número de samples descartados desde al última lectura. Esta lectura se realiza en *O(n)*, siendo $n$ número de samples dentro de un bloque.

## 5.2. Integración BziFramework

Para la integración en el framework desarrollado en *BitBrain* se ha construido una unidad de adquisición por cada tipo de conexión (una unidad para la conexión con el anillo WIFI y otra para la conexión con el amplificador Bluetooth). Esta unidad será la encargada de configurar la librería de conexión con los parámetros necesarios, realizar un primer tratamiento de los datos (conversión de los datos para realizar un cambio de escala, o la normalización de datos en caso del amplificador Bluetooth) y enviar la información al resto de unidades y módulos que los necesiten.

Al estar en un sistema de tiempo real, tenemos una ventana de tiempo en la cual se espera que se reciban los datos. El tratamiento de este tiempo es diferente dependiendo del dispositivo que estemos tratando.
En el caso del amplificador Bluetooth, la adquisición es bloqueante. La unidad se quedará esperando recibir datos del dispositivo. Se tomó esta decisión puesto que la aplicación para la que está pensado el dispositivo Bluetooth necesita tener siempre datos con los que trabajar y no se pueden permitir ciclos de ejecución en los cuales no se disponga de datos.
Para la adquisición WIFI se utiliza la frecuencia de muestreo y el número de *samples* por bloque para calcular el tiempo de ciclo. Este tiempo no puede ser utilizado para gestionar

las esperas en la adquisición, ya que no tiene en cuenta el tiempo de transmisión por la red. Para ajustar el tiempo, se aumenta el tiempo de ciclo en un 10 % del tiempo exacto. Una vez definido el tiempo de ciclo, se plantearon diferentes estrategias referentes a cuándo leer los datos, y en caso necesario, qué hacer con el tiempo sobrante. Las estrategias valoradas fueron las siguientes:

- Lectura al final del ciclo. Tengamos o no tengamos datos se continua con la ejecución.
- Lectura al principio del ciclo. Si existen datos se leen y se continúa con la ejecución. Si no existen, se lee al final de ciclo.
- Lectura al principio del ciclo. Si existen datos se leen y se continúa con la ejecución. Si no existen, se realizan lecturas sucesivas, hasta agotar tiempo de ciclo. Si se obtienen datos antes de la finalización del tiempo de ciclo, se agotará el tiempo restante.
- Lectura sucesiva de datos mientras se disponga de tiempo. En el momento en el que se reciben datos, se continúa con la ejecución.

La solución elegida ha sido la última enumerada, puesto que proporciona una mejor respuesta en caso de que se produzcan retardos en el ciclo del sistema o del dispositivo. Mientras que se espera la recepción de datos, se realiza una espera semi-activa. Los sucesivos intentos de lectura de datos se realizan con una diferencia de tiempo entre uno y otro igual a la cuarta parte del tiempo de ciclo restante. En el caso de que no se disponga de datos al finalizar el tiempo de ciclo, se modificó la unidad de visualización de señales para que pinte la señal en color rojo hasta que se vuelva a disponer de datos. Esta modificación en la unidad de visualización no afecta al funcionamiento de los programas ya existentes. También se encuentra implementado un sistema de reconexión con el dispositivo. Si se suceden varios ciclos de ejecución (los correspondientes a 30 segundos) en los cuales no se han recibido datos, se procederá a intentar reconectar con el dispositivo. En la figura 5.6 podemos ver la interfaz de un programa de visualización de la señal transmitida por el anillo WIFI de medición de señales, fruto de esta integración.

## 5.3. Arquitectura para la adquisición de múltiples dispositivos WIFI

En la figura 5.7 se muestra la arquitectura creada para la comunicación con múltiples dispositivos WIFI de manera simultánea. En esta solución, la unidad de adquisición delega la conexión y gestión de la información con el dispositivo en una clase que se ejecuta en un nuevo hilo. Esta clase recibirá la información necesaria para configurar la conexión,

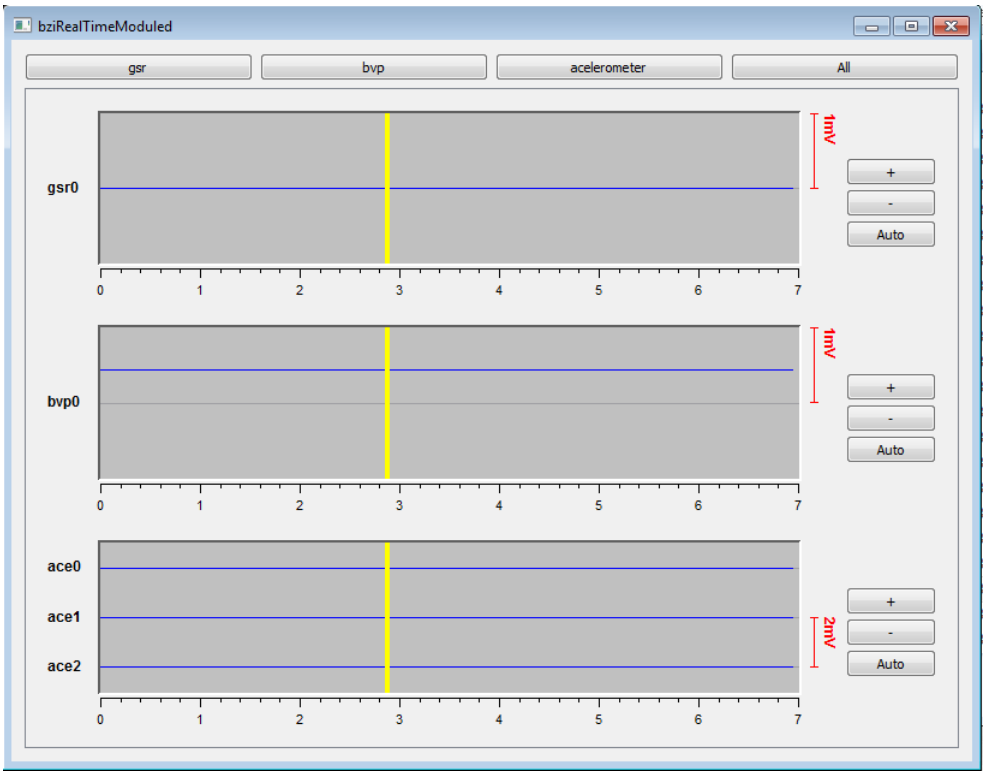


Figura 5.6: Visor de un anillo

así como las variables del repositorio de *BziFramework* en la cual deberá de volcar los datos recibidos para que el resto de elementos del sistema pueda acceder a ellos. En este punto nos encontramos con un problema. ¿Cuándo consideramos que la unidad ha terminado su ejecución y puede dejar actuar a las unidades y módulos siguientes? Para resolver este hecho, los distintos objetos que controlan los dispositivos y la unidad comparte un semáforo. En el momento en el que se produzca la lectura de datos, o bien expire el tiempo de ciclo asignado a cada dispositivo, se marcará este hecho en el semáforo. La unidad esperará a que existan tantos recursos en el semáforo como dispositivos tenemos conectados para terminar su ciclo de ejecución. También se utilizan semáforos para la destrucción de los objetos, para garantizar que ningún objeto intenta acceder a un atributo compartido que haya sido eliminado por otra entidad. Para sincronizar la comunicación de la unidad con todos los objetos encargados de controlar los dispositivos, se cuenta con una clase (a partir de ahora *manager*) encargada de conectar sus señales a los *slots* de los objetos de control de dispositivos que se le indiquen (para más información sobre el mecanismo *signal-slot* de QT ver el apéndice A). De esta manera, cuando se quiera comenzar la comunicación, leer datos o detener la conexión, el *manager* emitirá una señal que estará asociada al *slot* correspondiente de cada objeto controlador. Así se consigue sincronizar la invocación de estos métodos.

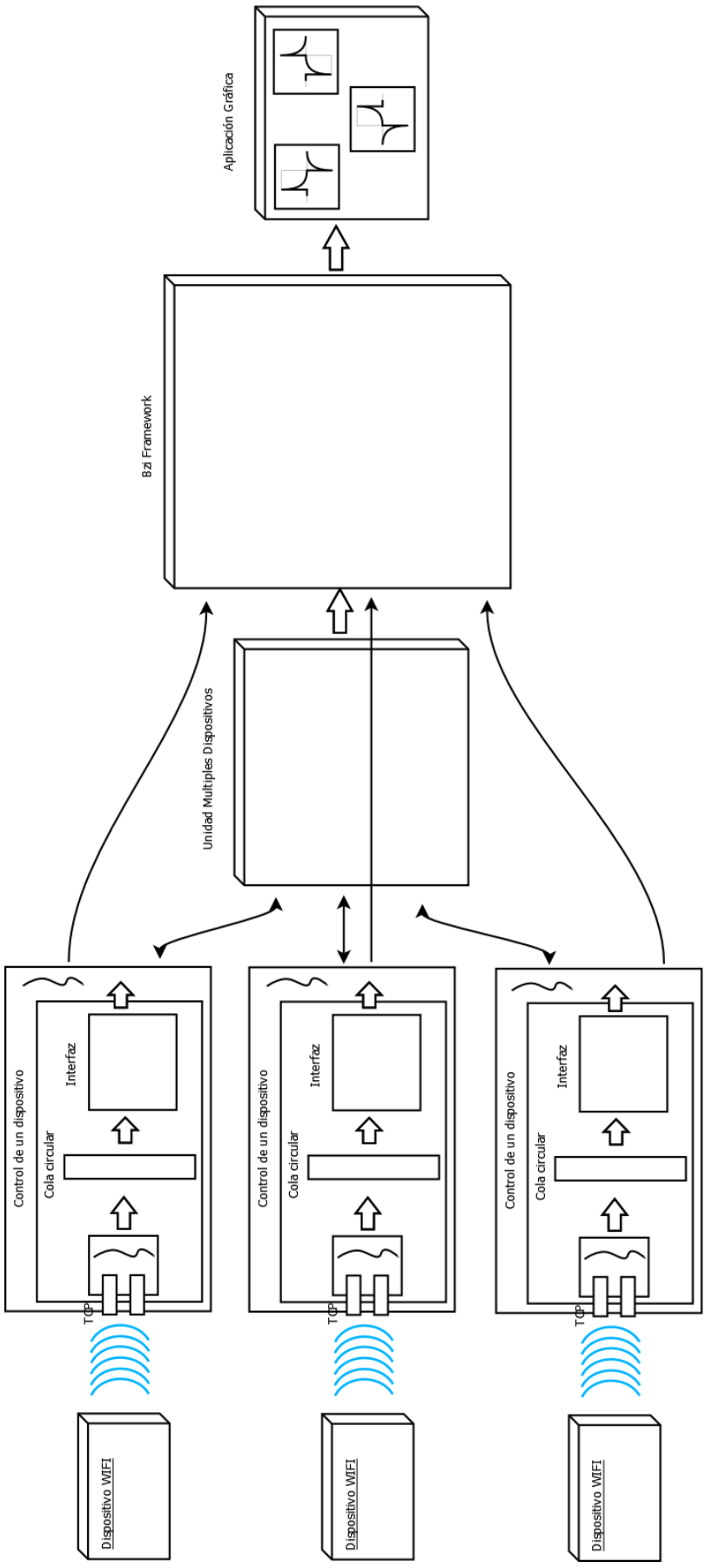


Figura 5.7: Arquitectura para varios dispositivos simultáneos

## 5.4. Integración en NeuroLab

Dado el funcionamiento de la aplicación *NeuroLab*, la integración de la adquisición de los dispositivos consiste en generar una configuración de la unidad de adquisición que se corresponda con los elemento que el usuario requiera en la interfaz gráfica. En la imagen 5.8 vemos el resultado de esta integración, con la visualización de dos anillo WIFI.

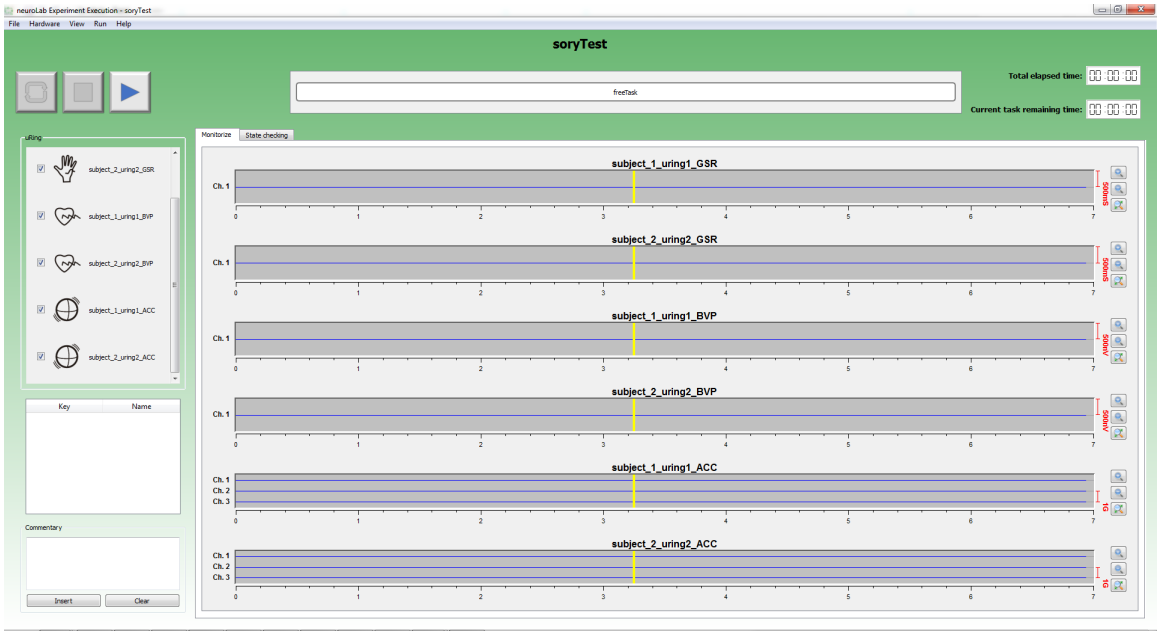


Figura 5.8: NeuroLab y adquisición multi-dispositivo

## 5.5. Anillo y red WIFI

El firmware desarrollado para el anillo WIFI consiste en la gestión de la comunicación con el cliente. En esta comunicación se enviarán los datos obtenidos por los distintos sensores, empaquetando varios *samples* en un solo envío. En el capítulo de pruebas se explica más detalladamente la problemática encontrada en la depuración del firmware embebido.
También se planteó la eliminación del router externo para conectar los anillos y el software. Para dar solución a este requerimiento, se exploró la posibilidad de crear una red WIFI privada utilizando al tarjeta de red inalámbrica de un equipo, encontrando en el sistema operativo Windows la herramienta *Netsh*, que permite crear redes WIFI con cifrado WPA2-Personal y capacidad para 100 usuarios.

# Capítulo 6

# Pruebas

En este capítulo se van a explicar las distintas pruebas y tests (pruebas unitarias, pruebas de integración, pruebas de carga) que se han realizado, tanto en los componentes software desarrollados como en los dispositivos hardware.

## 6.1. Pruebas unitarias

Durante la fase de pruebas de la estructura de datos implementada, se realizaron las pruebas unitarias de la tabla 6.1 con distintos tamaños de bloque ($n$ en la tabla) y distintos valores de bytes por sample (concretamente con 2, 3 y 4 bytes).

Con esta batería de pruebas se consigue verificar el correcto funcionamiento de la estructura de datos en los momentos críticos que pueden aparecer a lo largo de la vida de la estructura.

## 6.2. Pruebas librería WIFI y su integración en BziFramework y en NeuroLab

Para validar la implementación realizada se utilizaron tres tipos de servidores: uno lógico, un servidor hardware dummy y el dispositivo final.

El servidor lógico (ejecutado en la misma máquina que el servidor cliente) fue construido para que nos permitiera configurar elementos tales como la frecuencia de envío, canales, tamaño de los datos a enviar y samples empaquetados en cada envío. Este servidor fue validado con la aplicación *TCP/IP Builder*, para verificar que los paquete enviados por la

| Prueba realizada | Resultado esperado |
|---|---|
| Lectura | No disponemos de datos, por lo que devuelve error. |
| Inserción de un sample | Actualización de índices. |
| Lectura | No disponemos de datos suficientes, por lo que devuelve error. |
| Inserción de n-2 samples | Actualización de índices. |
| Lectura | No disponemos de datos suficientes, por lo que devuelve error. |
| Inserción de un sample | Actualización de índices. Tenemos bloque completo. |
| Lectura | Se lee el bloque almacenado. Se devuelven 0 samples perdidos. Se actualizan índices. |
| Inserción de n+1 samples | Actualización de índices. Tenemos bloque completo. |
| Dos lecturas consecutivas | La primera lectura lee el bloque más antiguo. La segunda lectura devuelve error. Se actualizan índices. |
| Inserción de 2n-2 samples | Actualización de índices. Tenemos bloque completo. |
| Dos lecturas consecutivas | La primera lectura lee el bloque más antiguo. La segunda lectura devuelve error. Se actualizan índices. |
| Inserción de n+1 samples | Actualización de índices. Tenemos dos bloques completos. |
| Dos lecturas consecutivas | La primera lectura lee el bloque más antiguo. La segunda lectura lee el bloque más reciente. Se actualizan índices. |
| Inserción 2n+1 samples | Se produce descarte del sample más antiguo. Tenemos dos bloque completos. Se actualizan índices. |
| Dos lecturas consecutivas | La primera lectura lee el bloque más antiguo. Esta lectura avisa del descarte de un sample. La segunda lectura lee el sample más reciente. |

Cuadro 6.1: Pruebas unitarias realizadas en la cola circular

red tiene en tamaño y contenido correspondiente a la configuración. Con este servidor se comprobó la creación y lanzamiento del hilo encargado de atender la red, la recepción de paquetes y el paso de información entre los dos hilos de ejecución a través de la cola circular.

Para comprobar el trasiego de datos a través de la red se diseñó un programa de envío simple en la tecnología Flyport (tecnología en la cual está baso el dispositivo final). Igual que en la prueba anterior, se validó el servidor creado con la aplicación *TCP/IP Builder*. Lo que se buscaba en este caso era comprobar que no se producían perdidas de datos en el envío. En la realización de estas pruebas se detectaron fallos en Flyport, que serán explicados en la sección 6.4 de este capítulo. Gracias a esta prueba se detectó un problema grave, y era el envío en el servidor de paquete más grande de lo normal, debido a los algoritmos de gestión de la congestión del protocolo TCP. Cuando se producía esta acumulación y envío de paquetes en el servidor, la recepción marcaba pérdida de datos, ya que la cola circular almacena dos bloques de información. Para solucionar este problema, se incluyeron en la solución las *DynamicSignal*, un tipo de dato de *BziFramework* que permite la persistencia de toda la información que se reciba por la red. Estos datos se

rellenan en el hilo de adquisición con toda la información procedente de la red. El envío de datos por la red se comprobó con tres clases de redes distintas. La primera, una red de trabajo en la cual conviven distintos elementos (computadores o móviles, por ejemplo). La segunda, una red dedicada, en la que solo existen en la red los dispositivos inalámbricos y el computador. En tercer lugar, una red creada por el propio computador cliente.

A continuación se pasó a comprobar casos de envíos con el dispositivo y software real. Se realizaron envíos con señales adquiridas y con señales simuladas. Al igual que en la prueba anterior se detectaron fallos en el hardware (sección 6.4 de este capítulo). Durante las pruebas con señales simuladas, se detectó un problema de concurrencia en el dispositivo, ya que en el envío de una señal incrementa se producían saltos inesperados.
En esta etapa de las pruebas se comprobó la sincronización de los datos. Se hizo que el dispositivo enviara un tiempo determinado (5, 10, 15, 30 minutos), y se comprobó que en recepción los datos grabados a fichero correspondían a ese tiempo. Se realizaron también medidas simulando pérdidas de conexión para probar distintas casuísticas. También se probaron la detección de eventos de red y la reconexión con el dispositivo cuando se detectaba un cierre inesperado de la conexión por parte del servidor.

Finalmente, para comprobar la adquisición de varios dispositivos simultáneamente se realizaron pruebas de funcionamiento largas (varias horas) para comprobar que la sincronización entre señales y dispositivos era correcta. También se prepararon despliegues de aplicaciones básicas que utilicen estas adquisiciones para uso interno de la empresa, de manera que se pudieran encontrar fallos o proponer posibles mejoras a la implementación realizada.

Respecto a la integración de NeuroLab, una vez comprobada la correcta integración y configuración de la unidad de adquisición, se utilizó para los fines comerciales de la empresa.

## 6.3. Pruebas Librería Bluetooth y su integración en BziFramework

En este caso se construyeron dos servidores distintos: un servidor hardware dummy y el dispositivo final.

El servidor dummy está implementado en una placa Arduino. Se escogió esta tecnología por la sencillez de programación y conexión. La conexión entre cliente y servidor se realizó a

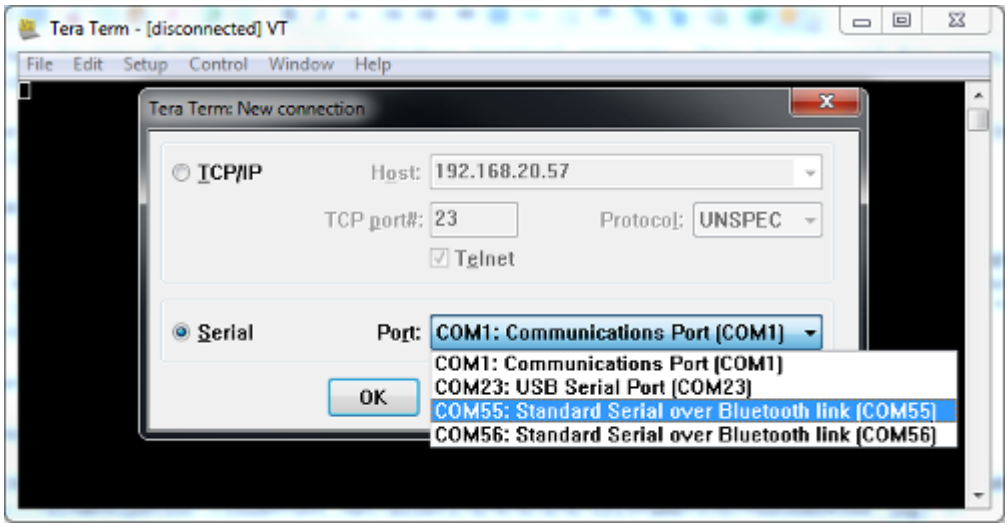


Figura 6.1: Interfaz del programa TeraTerm

través de cable, ya que el objetivo de las pruebas con este servidor eran la comprobación de la correcta apertura de la conexión a diferentes velocidades (1200, 2400, 4800, 9600, 19200, 57600 y 115200 baudios). También se comprobó, al igual que en el servidor lógico WIFI, el lanzamiento del hilo dedicado y la comunicación entre hilos de ejecución.
Para validar el programa de la placa Arduino se utilizó la aplicación *TeraTerm* (figura 6.1), que nos permite monitorizar los datos que circulan por un puerto COM.

Una vez comprobado el funcionamiento de la librería, se pasó a comprobar la integración en BziFramework utilizando el dispositivo real. En esta prueba se prestó especial atención a la conversión de los datos recibidos a datos nativos del lenguaje, debido a las características explicadas en la sección 2.3 del capítulo 1. Para comprobar esta conversión se enviaron datos simulados, concretamente un seno, para comprobar la conversión de datos positivos y negativos. Finalmente se realizaron pruebas con la adquisición de señales reales.

## 6.4. Pruebas anillo WIFI

Durante la realización de tests se descubrieron distintos fallos en el firmware del anillo. Con el envío de señal simulada (señal incremental) al cliente, se detectaron discontinuidades en la señal. Después de analizar las causas se llegó a la conclusión de que nos encontrábamos ante un problema de concurrencia, se estaban modificando datos que estaban preparados para ser enviados. La solución a este problema pasó por utilizar una estructura de datos ofrecida por el sistema operativo que permite a la interrupción almacenar los

datos, y al proceso principal obtenerlos, solventando el problema (nos encontramos otra vez ante un problema de *productor-consumidor*).

El problema más grave que se detectó fue la detención en la recepción de datos desde el anillo. Gracias a la herramienta *wireshak* se determinó que el fallo se encontraba en el servidor que no enviaba ningún dato por la red. Llegado un momento, la instrucción de escritura en el *socket* indicaba que este se encontraba lleno, entrando en un estado en el cual no se enviaban datos por la red y no se podían volcar datos en el *socket*. Para determinar el origen de este problema se realizaron exhaustivas pruebas, como fueron configuraciones con distintas capacidades de los buffers asociados al envío TCP o el incremento progresivo de la complejidad del software embebido (primero un bucle principal sencillo, complicar el bucle hasta el bucle definitivo, añadido de la interrupción, pero sin comunicación entre tarea principal e interrupción, añadido de la adquisición de datos y el software completo). Durante las pruebas ocurría el mismo error, pero era más frecuente cuanto más complejo era el código ejecutado. Durante la ejecución de estas pruebas nos encontramos con otro problema que complicó la tarea de depuración del anillo. Al intentar mostrar por pantalla el estado por el cual el sistema estaba pasando en cada momento, la función de escritura podía, aleatoriamente fallar y producir una excepción que reiniciaba el anillo. Todo esto nos indicó que nos encontrábamos antes un problema de carga del sistema.
Con la ayuda del osciloscopio se midió el tiempo en el cual el procesador no estaba ocupado con la interrupción. Este tiempo debería ser suficiente para que la rutina principal del código se ejecutara, por lo que deducimos que el problema estaba relacionado con el controlador de la antena WIFI. Visto esto nos pusimos en contacto con la empresa que produce las placas para solucionar el problema. Paralelamente, se optó por actualizar el sistema operativo a una versión más moderna e intentar evitar el llenado del buffer del *socket*, produciendo una desconexión con el cliente y limpiando los buffers cuando se empezaban a detectar muchas situaciones en las cuales el *socket* se encontraba lleno. Esto, junto a una actualización del entorno de desarrollo que incluía una actualización para el controlador de la antena WIFI solucionó el problema.

## 6.5. Pruebas amplificador Bluetooth

Las pruebas realizadas en el amplificador se centraron en verificar que el cambio introducido en el envío de datos era correcto, sin alterar la señal original, puesto que el código de la placa ya había sido testeado anteriormente.

# Capítulo 7

# Conclusiones y trabajo futuro

Los objetivos marcados para este proyecto fueron la integración de los dispositivos hardware inalámbricos (con la utilización de tecnologías WIFI y Bluetooth) que se encontraban en desarrollo en la empresa con la arquitectura software de *BitBrain*.

Estos objetivos han sido cumplidos, ofreciendo una solución escalable y eficiente, que abstrae al programador de todo el proceso de gestión de las comunicaciones, ofreciendo además soporte a futuros dispositivos que pudieran ser desarrollados (se ha integrado ya un nuevo amplificador EEG que utiliza un salto WIFI para transmitir la información). Además, gracias a la integración realizada en el sistema *NeuroLab*, la integración en el software de *ussens* fue inmediata.

Además durante la realización de este proyecto, se encontraron y solventaron problemas relacionados con el firmware del hardware, lo que derivó en la obtención de unas comunicaciones robustas.

El trabajo realizado ha aportado valor a la empresa, puesto que se enmarca en la estrategia de negocio principal de la misma y que actualmente se encuentra inmersa en un programa de parteners, en el que empresas e instituciones pueden utilizar la tecnología.

A la finalización de este proyecto se han quedado abiertos los puntos expuestos a continuación:

- Integración de unidad de adquisición bluetooth en Neurolab
- Integración de unidad de adquisición bluetooth en BrainUp

La integración en Neurolab contaría con parte del trabajo hecho, ya que la configuración a generar es similar a la generada para la unidad WIFI. El trabajo para integrarla en BrainUp sería más laborioso, puesto que se debe realizar una migración del código existente de Qt4 a Qt5. Una vez realizada esta migración, la integración seguiría los mismos pasos que en Neurolab.

# Apéndice A

# Signal y Slots.

En este anexo vamos a explicar una de las principales características del framework Qt, su mecanismo de *Signal-Slot*.

Qt utiliza las señales y los slots para comunicar objetos entre si (por ejemplo los *widgets* gráficos de una interfaz). Un slot es una función que atiende a una señal en concreto. Para conectar una señal con un slot se utiliza la función *connect*, definida en *QObject* (clase básica de Qt). En la imagen A.1 podemos ver como se realizan estas conexiones.

Este mecanismo de comunicación tiene comprobación de tipos. El tipo de datos de una señal debe de coincidir con el dato esperado por un slot. El objeto que emite una señal, no tiene por que conocer qué slot recibirá la señal, de eso se encargan los mecanismos de Qt.

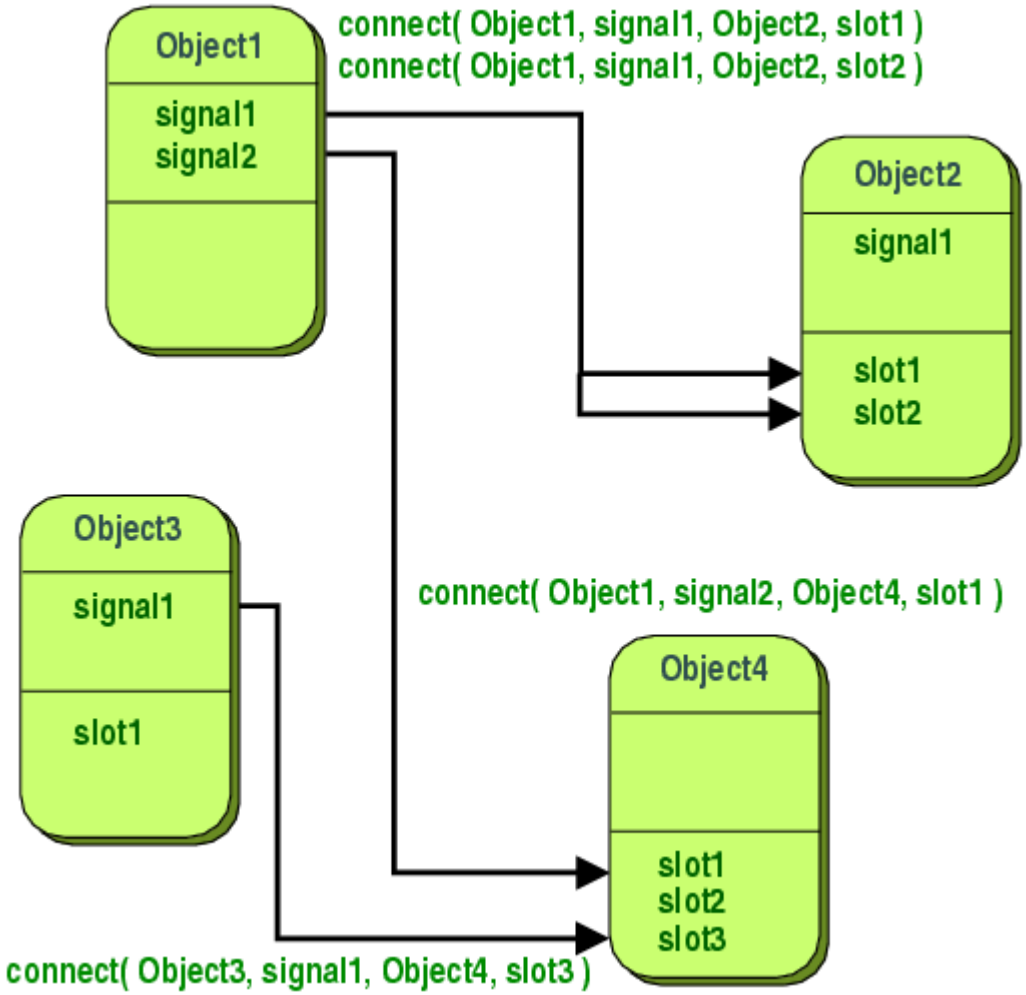


Figura A.1: Estructura Signal-Slot

# Índice de figuras

# Índice de cuadros

# Bibliografía


[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Patrones de Diseño. Elementos de software orientado a objetos reutilizable. Editorial Addison Wesley. Madrid, 2003 15

[2] Jeff Sutherland: Scrum Handbook 13

[3] Qt5 documentation: `https://qt-project.org/doc/qt-5/classes.html`

[4] Flyport API v2.2: `https://www.iiitd.edu.in/~amarjeet/EmSys2013/Flyport%20API%202.2%20Quick%20reference.pdf`

[5] freeRTOS API reference: `http://www.freertos.org/modules.html#API_reference`