Universidad Zaragoza

Proyecto Final de Carrera
Ingeniería Informática
Curso 2011-2012

Desarrollo de Funcionalidades, Plugins y Herramientas para Software de Neuroterapia

Sergio Serrano Sánchez

Diciembre de 2011

Director: Javier Mínguez Zafra

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

El dolor es inevitable,
el sufrimiento, opcional.

Agradecimientos

A mis compañeros de clase, por las tardes y noches de laboratorio.

A mis amigos, por ser ellos.

A Sergio y Ana, por darle luz a mis noches.

A Fernando, por enseñarme lo que es un gradiente, y por sus "terrorismos".

A los BBTs, por aceptarme con mis defectos y virtudes, por ayudarme en los momentos más dificiles.

A la sección de deportes San Agustín, por permitirme aprender de ellos tantos años.

A Carlos Sebastián, por enseñarme que, la lucha por llegar, nos hace fuertes.

A María, por compartir sus metas conmigo, y darme la oportunidad de trabajar cada día, por un mañana distinto.

A Javier, por dirigir este PFC y por su incansable esfuerzo en convertir todo lo que nos rodea en algo mejor.

A mis tíos, por sus llamadas furtivas, por tener siempre una palabra de aliento.

A mis padres, por celebrar mis triunfos, pero sobre todo por estar a mi lado en mis derrotas.

A mi hermano y a Eva, por mostrarme el camino de regreso.

A mis abuelos, por enseñarme que no hay nada imposible.

Y como no a Merche, por guiarme , por entenderme, por re-escribir el presente, por hacerme soñar con el futuro.

Resumen

Desarrollo de Funcionalidades, Plugins y Herramientas para un Software de Neuroterapia

El objetivo de este proyecto es el análisis, diseño e implementación de diversas herramientas que proporcionan funcionalidades para un software de neuroterapia por medio de tecnología BCI (Interfaz Cerebro Computador).

El contexto de este proyecto es utilizar BCI como herramienta adicional para el tratamiento de diversas patologías (TDA, accidente cerebro-vascular, depresión, fibromialgia) o incluso como herramienta orientada a la mejora de las capacidades cognitivas.

La forma mas básica de aplicación de la tecnología BCI para trabajar con cualquier tipo de característica o trastorno ha sido denominada neurofeedback, el cual es una forma de biofeedback ligado a aspectos específicos de la actividad eléctrica del cerebro, los cuales se sabe que están relacionados con aspectos cognitivos humanos que se desea potenciar. Se espera que una mejora en estos derive en una mejora de las capacidades cognitivas asociadas.

En concreto los objetivos del proyecto son:

- Analizar, diseñar e implementar plugins y funcionalidades necesarias dentro del software de neuroterapia.
- Analizar, diseñar e implementar una herramienta de comprobación de defectos de montaje, teniendo en cuenta aspectos de usabilidad y facilidad de manejo.
- Analizar, diseñar e implementar funcionalidades adscritas a la instalación del software BrainUp.
- Colaborar en las diversas etapas de ingeniería del software BrainUp desde su fase más temprana.

Índice

1. Introducción

El presente proyecto fin de carrera se ha realizado en estrecha colaboración tanto con el Grupo de Robótica, Percepción y Tiempo Real del Departamento de Informática e Ingeniería de Sistemas del Centro Politécnico Superior de la Universidad de Zaragoza, dentro del equipo de investigación de Interfaces Cerebro-Computador, así como BitBrain Technologies, spin-off de la Universidad de Zaragoza, empresa pionera en la investigación, diseño e implementación de Interfaces Cerebro-Computador.

El objetivo de este proyecto es realizar el análisis, diseño e implementación de diversas herramientas y funcionalidades tanto visuales como matemáticas, necesarias dentro de un sistema de gestión para Neurofeedback basado en tecnología de interfaces cerebro computador (BCI, del inglés Brain Computer Interface), y desarrollado por BitBrain Technologies.

Una interfaz cerebro computador o BCI[1] es un sistema basado en la adquisición de señal cerebral, la cual, pasa a ser inmediatamente procesada, con el fin de extraer actividades o patrones destacables conforme a ciertos ritmos cerebrales. Existen diversos medios encargados de la adquisición de esta señal cerebral, divididos en métodos invasivos y no invasivos dependiendo de su nivel de intrusión en el usuario/paciente. En el caso que nos ocupa, se utiliza un montaje de electroencefalograma (basado en la colocación de electrodos-sensores), ya que supone una solución con un tiempo de procesado aceptable, barata, no invasiva y cuyo coste de adaptación al usuario final es inferior al de otras soluciones.

Debido a que la señal de encefalograma (EEG[2]) es muy débil (baja amplitud, del orden de los microvoltios), necesita ser amplificada durante el periodo de adquisición, este proceso provoca del mismo modo un incremento en la amplitud de perturbaciones ajenas al montaje contaminando de este modo la señal. Éstas perturbaciones, denominadas también artefactos pueden ser de dos tipos, fisiológicas (parpadeos, movimientos musculares) o eléctricas (fluctuaciones de corriente), siendo en cualquier caso varios órdenes de magnitud superiores a la señal EEG. Es por ello por lo que han de implementarse métodos de filtrado y comprobación de artefactos y defectos del montaje.

Los patrones y actividades extraídos después del proceso de adquisición y procesado son utilizados con distintos objetivos, desde la tele-operación de vehículos no tripulados[3], control de interfaces en personas con movilidad reducida[4], rehabilitación de pacientes

con lesiones medulares o neurológicas[5], hasta aplicaciones basadas en neuromarketing o como en el caso que nos ocupa, neurofeedback.

El neurofeedback (NFB)[6], también conocido como neuroterapia, se basa en la capacidad por parte del usuario/paciente de adquirir auto control sobre ciertos patrones cerebrales mediante el condicionamiento operante. El entrenamiento sobre éstos refleja rendimiento cognitivo, como mejoras en memoria de trabajo, disminución del tiempo de respuesta ante un estímulo aleatorio, progreso asociado a habilidades motoras o trastornos depresivos, entre otros.

En la actualidad comienzan a adoptarse técnicas basadas en neurofeedback con el fin de tratar enfermedades como la epilepsia, deficit de atención[7][8][9], desordenes adictivos[10], depresión, o fibromialgia[11]. También es utilizado cada vez con mas frecuencia con fines no estrictamente médicos, como su presencia en entornos lúdicos (videojuegos[12], controladores musicales[13], robótica de consumo[14]), o en procesos de toma de decisión.

Con el objetivo de dar soporte a estos sistemas nace BrainUp, concebido como un software de creación, configuración y ejecución de terapias, orientado a usuarios/terapeutas sin conocimientos previos en materia de señal EEG, rápido y usable. En la actualidad, BrainUp y por consiguiente las tareas realizadas en este PFC son utilizadas en varios ensayos clínicos relacionados con trastornos depresivos y fibromialgia en colaboración con el Hospital Miguel Servet de Zaragoza, así como en sendos estudios de mercado (*neuromarketing*), se espera su lanzamiento al mercado en un plazo máximo de 3 semanas.

1.1. Alcance del proyecto

Las tareas desarrolladas en este PFC comprenden el análisis, diseño, implementación e integración de herramientas o interfaces de usuario completas, funcionalidades matemáticas y visuales dentro de BrainUp.

A continuación se detallan las tareas en el que se han estructurado este PFC. Para conocer su desarrollo temporal debe consultarse el diagrama de Gantt del producto, presente en la sección A.1.

Herramientas: Encargadas de dotar a la aplicación de una determinada funcionalidad.

1. **Herramienta de comprobación de defectos de montaje:** interfaz de usuario dedicada a la detección y notificación de los defectos detectados en un montaje de EEG (electrodos erróneos), tanto antes del entrenamiento, como durante el mismo, protegiendo así la integridad de la terapia.

2. **Localización del software:** algoritmo de adaptación y localización lingüística para BrainUp.

3. **Instalador:** interfaz de usuario dedicada a la instalación y configuración de la aplicación a las diferentes arquitecturas y sistemas operativos soportados.

Funcionalidades Matemáticas: Encargadas del procesado de señal EEG.

1. **Unidad de detección de defectos de montaje:** relacionada con la herramienta de comprobación de defectos de montaje antes descrita, realiza la operatoria matemática que reside bajo la misma, siendo la herramienta de comprobación una representación visual de sus resultados.

Funcionalidades Visuales: Encargadas de la representación gráfica dentro de las diferentes interfaces de usuario.

1. **Plugin de visualización de actividad cerebral:** representación espacial interpolada de la actividad cerebral en tiempo real, puede hacer las veces de feedback visual, así como ser utilizada como herramienta de calibración y comprobación.
2. **Informe:** representación visual resultados de neuroterapia, número de sesión en la que nos encontramos, datos relativos al paciente, gráficas de evolución entre fases, entre otros.

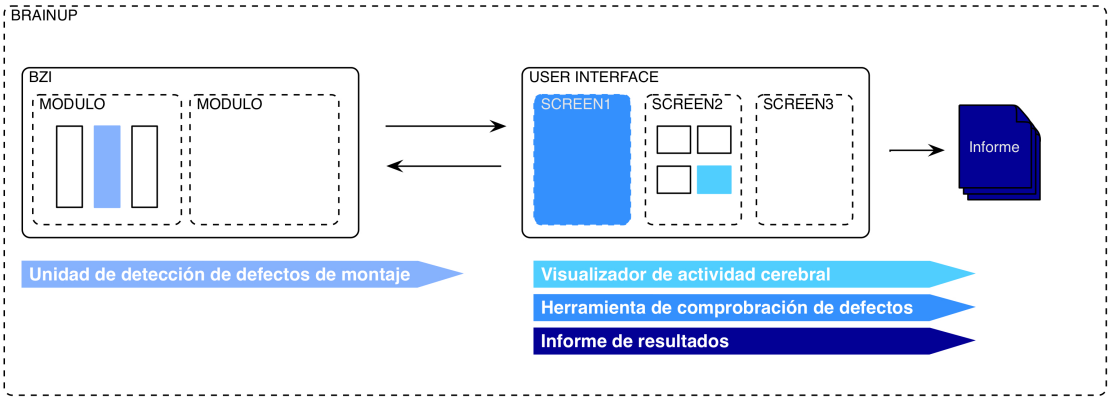


Figura 1.1: Alcance de las tareas.

2. Contexto

2.1. Conceptos sobre Neurofeedback

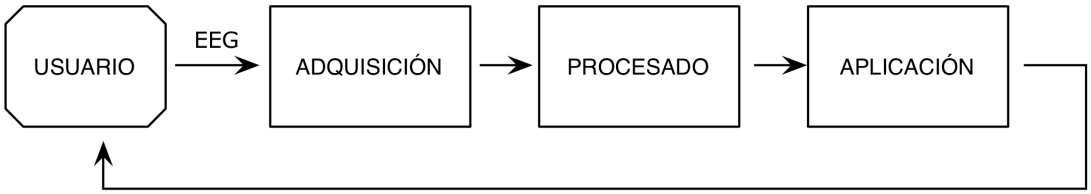


Figura 2.1: Esquema general del neurofeedback.

Una aplicación basada en neurofeedback, responde a la estructura básica de una interfaz cerebro-computador (Figura 2.1). En sucesivas secciones se concretará el diseño y funcionamiento particular de una terapia de neurofeedback, comenzaremos en ésta explicando qué elementos posee en común con cualquier interfaz cerebro-computador (BCI):

1. Adquisición: Se lleva a cabo mediante un gorro EEG (montaje), unido a un amplificador operacional encargado de amplificar la señal. En la actualidad comienzan a proliferar, sistemas hardware "secos" (sin necesidad de aplicar gel conductor), de fácil montaje y coste inferior a un equipo de EEG tradicional.

2. Procesado: Realizado con dos objetivos, por un lado la búsqueda de patrones relevantes de los que deseamos mejora y escogidos dentro del entrenamiento que nos ocupe (*Upper Alpha*, *Lower Alpha*) y por otro, la gestión de la señal necesaria para tales fines, ya sea filtrando, editando o almacenando la misma.

3. Aplicación: Parte esencial del sistema, pues proporciona al usuario feedback (visual/auditivo/táctil) dependiendo de su corrección/fallo en la realización de la tarea (control sobre ritmos determinados), consiguiendo así un mayor control en el usuario/paciente gracias al aprendizaje por refuerzo.

Como ya hemos comentado anteriormente el propósito general de una terapia basada en neurofeedback, es conseguir que el usuario adquiera gracias al condicionamiento operante cierto grado de auto-control sobre bandas o ritmos determinados que se considera reflejan rendimiento cognitivo, como por ejemplo la memoria de trabajo [15][16][17].

Definición de Banda: Existen diversas bandas o ritmos presentes en la actividad cerebral (Figura 2.2), pudiendo realizarse terapias/entrenamientos de neurofeedback en cualquiera de éstas y extrayendo mejoras cognitivas diferentes dependiendo de la banda o ritmo seleccionado. En el caso que nos ocupa, tanto por la experiencia acumulada por BitBrain Technologies como por la Universidad de Zaragoza, se escoge la banda *Alpha* como objeto de estudio.

Figura 2.2: Bandas de trabajo de una neuroterapia.

Las terapias basadas en neurofeedback desarrolladas hasta la fecha se realizaban sobre una banda *Alpha* fija, sin embargo, la existencia de problemas derivados de una metodología de banda fija, como por ejemplo, su ineficacia en grupos con importantes variaciones "inter-usuario", así como el desconocimiento de qué aspectos cognitivos eran mejorados realmente si se optaba por metodologías de banda completa[17][18], desembocaron en la aparición de nuevas metodologías de neurofeedback.

Como respuesta a este tipo de limitaciones, se introduce el concepto de *Individual Alpha Frequency (IAF)*, definido como punto de anclaje distintivo entre dos sub-bandas independientes, *upper-alpha (UA)* y *lower-alpha (LA)* y calculado de manera individual para cada sujeto permitiendo de este modo tener en cuenta variaciones "inter-usuario". Dichas sub-bandas han demostrado comportarse de manera distinta ante distintos tipos de tareas, siendo la banda *UA* la que parece más relacionada con mejoras cognitivas[18].

Su localización en cada sujeto requiere de la realización de tareas previas al entrenamiento. En nuestro caso, una tarea pasiva y una activa, con el fin de localizar la posición y potencia del *IAF* dentro de la totalidad de la banda *Alpha*, estas tareas, también conocidas como calibración de ritmos, se verán detalladas en secciones posteriores.

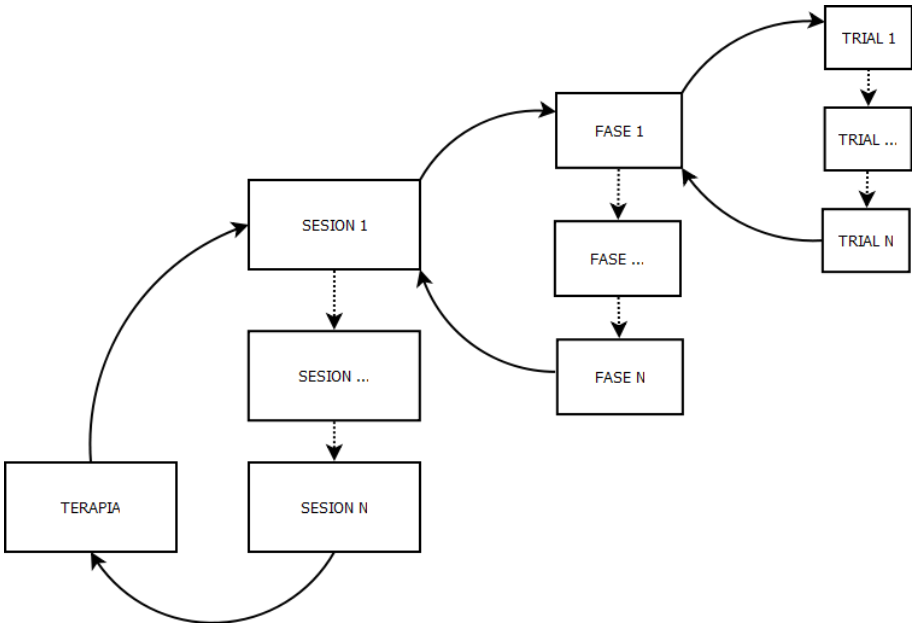


Figura 2.3: Esquema de una neuroterapia.

Concepto de terapia: Los interfaces cerebro-computador basados en neurofeedback se estructuran en terapias (Figura 2.3), una terapia se compone de una o varias sesiones, pudiendo realizarse en un espacio temporal reducido o bien dilatarse en el tiempo (días, semanas o meses).

Cada una de estas sesiones se encuentra estructurada en fases, siendo posible realizar trabajo común o específico (tabla 2.1), dotando así de versatilidad a la terapia. Tienen una duración aproximada de entre 10 y 30 minutos, contando con el tiempo programado de descanso entre fases.

Ejemplo	Protocolo para pacientes de depresión
	Fase 1: Mejora cognitiva. Fase 2: Refuerzo emocional.

Tabla 2.1: Ejemplo de diferencia entre fases.

Cada una de las fases esta compuesta por :

1. **Tarea Pasiva:** Realizada únicamente al inicio de la sesión, su duración aproximada es de 2 minutos, donde el usuario/paciente debe estar relajado y permanecer con los ojos cerrados, procurando aislarse en la manera de lo posible del exterior.

2. **Tarea Activa:** Realizada únicamente al inicio de la sesión, su duración aproximada es de unos 2 minutos, donde el usuario/paciente debe realizar una determinada

actividad, por ejemplo contar cambios de tonalidad dentro de una gama de colores ofrecida por pantalla, se trata de una tarea activa si bien todavía no forma parte del entrenamiento.

Como hemos comentado, la presencia de artefactos puede perturbar la señal dejándola inservible, es por ello por lo que durante la tarea activa también es realizada la calibración del filtro *ICA* cuyo objetivo es el filtrado automático de este tipo de componentes.

3. **Trial:** También llamados repeticiones, tras la ejecución de las tareas pasivas y activas, se procede a realizar tandas de entrenamiento, correspondientes al aspecto a reforzar en la fase (entrenamiento de memoria, de atención), el usuario recibe feedback positivo en caso de alcanzar el estado mental adecuado, y feeedback negativo en caso contrario, ya sea de tipo visual, auditivo o táctil. Todos los trials deben reforzar el mismo aspecto cognitivo, pues pertenecen a la misma fase.

El paciente no es informado en ningún instante de la estrategia a seguir para alcanzar el estado mental adecuado, se ha demostrado que no existe una única correcta[16], sino varias, y que a su vez, la búsqueda por parte del usuario/paciente de estrategias propias e individuales podría enriquecer y potenciar el entrenamiento frente a usuarios/pacientes informados con anterioridad.

La decisión de proporcionar feedback positivo o negativo es tomada en torno a un valor denominado *baseline*, valor de referencia al inicio de la sesión (calibración de ritmos). De esta forma nuestra terapia se convierte en un proceso dinámico, donde los resultados obtenidos en el día n estarán relacionados con los progresos particulares del sujeto en cuestión, adaptándose a éstos de forma transparente.

Validación: La validación de la terapia se realizó con 50 sujetos sanos, repartidos en semanas completas de experimentación y distribuidos bajo los siguientes roles:

1. **Grupo de pacientes:** Realizan una batería de test el primer día de entrenamiento y otra el último, con el fin de medir su evolución tras la terapia/entrenamiento.
2. **Grupo de control:** Realizan los mismos test pero sin recibir ningún tipo de entrenamiento, gracias a esto podemos estimar la capacidad de adaptación al test por parte de los sujetos, y con ello definir la mejora real del grupo de pacientes.

Los resultados arrojaron mejoras de entorno al **10 %** en el grupo de pacientes.

Por último cabría destacar que nos movemos en un terreno de gran complejidad, una terapia o protocolo de neurofeedback es el resultado de un intenso trabajo multidisciplinar y, si bien las fases de adquisición, procesado y aplicación son comunes a todos los interfaces BCI, la configuración y diseño de una terapia presenta unos niveles de heterogeneidad elevados, a esta y otras cuestiones pretende dar respuesta el software desarrollado por BitBrain Technologies de nombre BrainUp.

BrainUp: Quizás la mejor manera de explicar de qué se compone BrainUp, es remontarnos a la definición de interfaz cerebro computador dada en el capítulo anterior, donde quedaba dividida en tres fases bien diferenciadas, enriqueciéndolas ahora con nuevas necesidades como la creación y edición de una terapia, o la realización de tareas de localización del *IAF* y de filtrado automático de artefactos (*ICA*).

BrainUp se encuentra dividido en dos grandes bloques, por un lado la arquitectura BZI, encargada de las tareas de gestión de la señal cerebral, por otro la interfaz de usuario, encargada de la edición, configuración y ejecución de la terapia.

1. **Adquisición:** Realizada directamente por la arquitectura BZI proveniente del amplificador operacional.

2. **Procesado:** Realizado por la arquitectura BZI, encargada de las diferentes tareas de gestión de la señal, extracción de características y clasificación.

3. **Aplicación:** Realizado tanto por la arquitectura BZI, como la interfaz de usuario contenida en BrainUp

    a) **Arquitectura BZI:** Encargada de la visualización del feedback de usuario, conforme a la realización del entrenamiento por parte del sujeto

    b) **Interfaz de Usuario:** Monitorización y edición del entrenamiento, ya sea pausándolo, repitiéndolo o modificándolo.

2.2. Arquitectura BZI

Esta sección proporciona una descripción mas detallada de la plataforma para desarrollo de sistemas BCI proporcionada por BitBrain Technologies en colaboración con la Universidad de Zaragoza, y de nombre BZI.

Esta arquitectura responde al esquema general comentado anteriormente, donde puede observarse de manera diferenciada, módulos de adquisición en tiempo real, módulos correspondientes a procesado, así como elementos de visualización que asisten y proporcionan feedback al usuario o paciente.

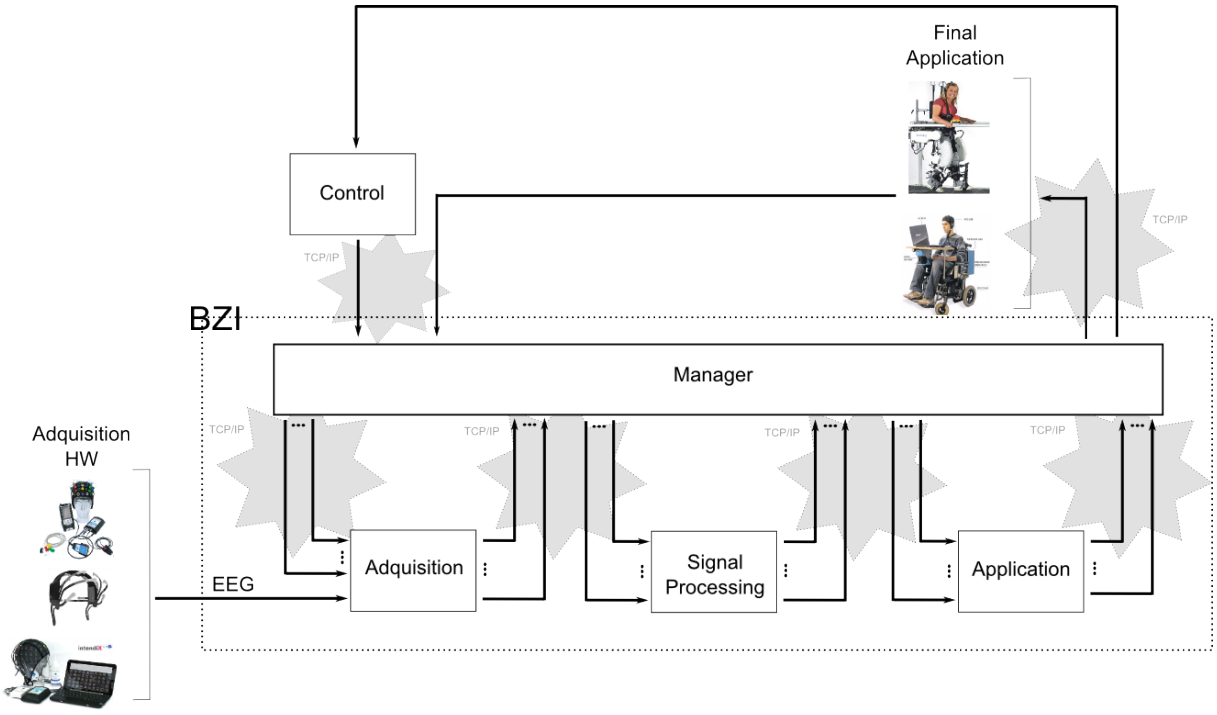


Figura 2.4: Esquema que describe la arquitectura de la plataforma BZI.

La arquitectura BZI contiene una serie de componentes básicos: módulos y un manager de propósito general (Figura 2.4).

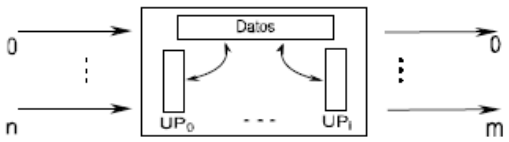


Figura 2.5: Esquema de un módulo dentro de la plataforma BZI.

Los módulos pueden ser de adquisición, procesamiento o aplicación final, y se encuentran comunicados mediante protocolos TCP/IP, es necesario aclarar que no existe comunicación entre ellos, sino que cada módulo adquiere y deposita la información pertinente en el manager, mediante mecanismos de suscripción (Figura 2.5).

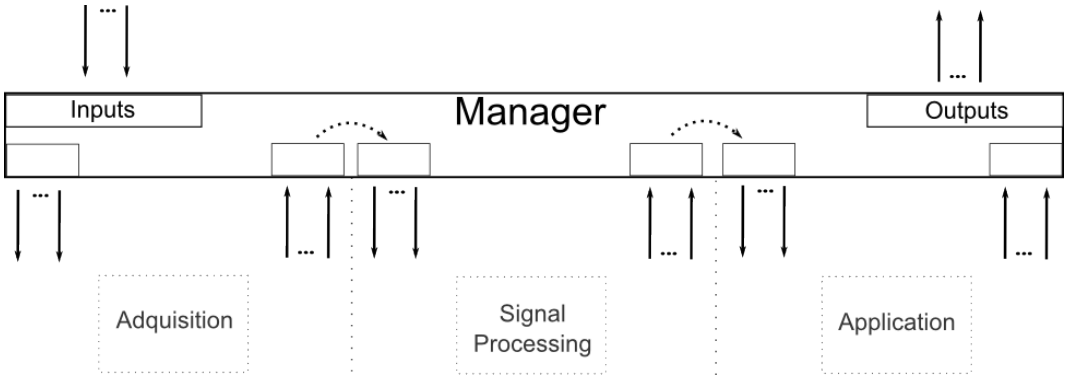


Figura 2.6: Esquema del manager dentro de la plataforma BZI.

Los módulos se encuentran formados por unidades de procesamiento, son unidades encargadas de realizar algún tipo de procesamiento de datos específico. Pueden ser encadenadas construyendo un tratamiento secuencial sobre la señal (Filtro paso banda + FFT + Detección de máximo). Al igual que en los módulos, no poseen comunicación directa entre ellas haciendo uso de un repositorio común, el control viene dado por la estructura que engloba varias unidades de procesamiento, dicese el módulo.

El manager ya mencionado, hace las veces tanto de concentrador de la información presente en el proceso, como labores de control, gestión y coordinación del proceso de neurofeedback, especial relevancia poseen las entradas y salidas del sistema, puesto que ademas de aquellas estándar del sistema definidas anteriormente, también podemos encontramos elementos de control y configuración capaces de adoptar este tipo de roles (Figura 2.6).

La dinámica de un sistema BCI desarrollado sobre BZI está dirigida por el flujo de datos, todos los módulos del sistema BCI y el manager se implementan como procesos que están dormidos hasta la llegada de los mismos.

El flujo de datos se inicia con la adquisición de señal de EEG, dicha señal es amplificada y muestreada a una frecuencia determinada (pre-procesado de señal en el hardware de adquisición). La señal digitalizada llega al módulo de adquisición, este módulo procesa el EEG para obtener un conjunto de muestras (bloque) que son almacenadas (disco y almacenadas en el manager).

Tras la adquisición el manager envía los datos al siguiente proceso en el flujo de ejecución: el módulo de procesado de señal, en este módulo se lleva a cabo la extracción de características y clasificación, dicha tarea es realizada en las unidades de proceso,cada una de las cuales implementa algún tipo de filtro de señal.

Finalmente el flujo de datos pasa al módulo de aplicación que aplica reglas y transforma los datos para emitir la respuesta del sistema.

2.3. BrainUp : Construya su propia Neuroterapia

BrainUp surge para dar respuesta a todas aquellas necesidades expuestas a lo largo de las secciones anteriores, se trata de un software de gestión de neuroterapia, donde el usuario/terapeuta, puede crear, gestionar y desarrollar las terapias o entrenamientos que considere adecuadas para su usuario/paciente.

A lo largo de esta sección, desglosaremos los elementos que componen el sistema de gestión de neuroterapia desarrollado por BitBrain Technologies, de nombre BrainUp.

2.3.1. Paso 1: Gestión de Usuarios

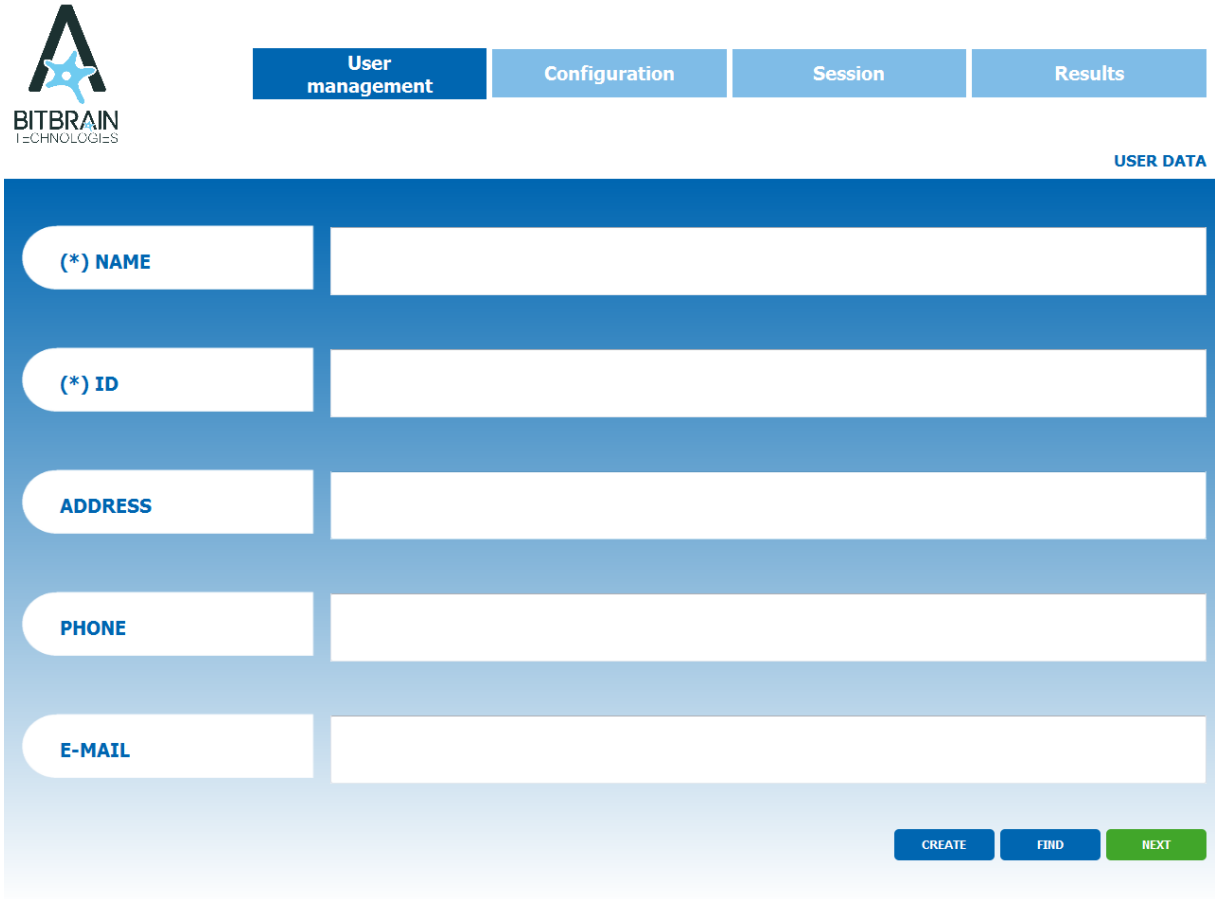


Figura 2.7: Aspecto de la interfaz de usuario correspondiente a la gestión y edición de usuarios.

Como se puede observar en la figura 2.7 se trata de la interfaz encargada del alta/edición de los usuarios dentro de BrainUp, muestra de manera clara los datos necesarios para el alta de un nuevo usuario, permitiendo también la búsqueda de aquellos

registrados anteriormente.

Todos los datos de usuario surgidos durante el proceso de terapia, así como los datos generados durante el proceso de alta, serán almacenados en una base de datos, cumpliendo además con los protocolos de protección de datos en vigor.

2.3.2. Paso 2: Gestión de Terapia

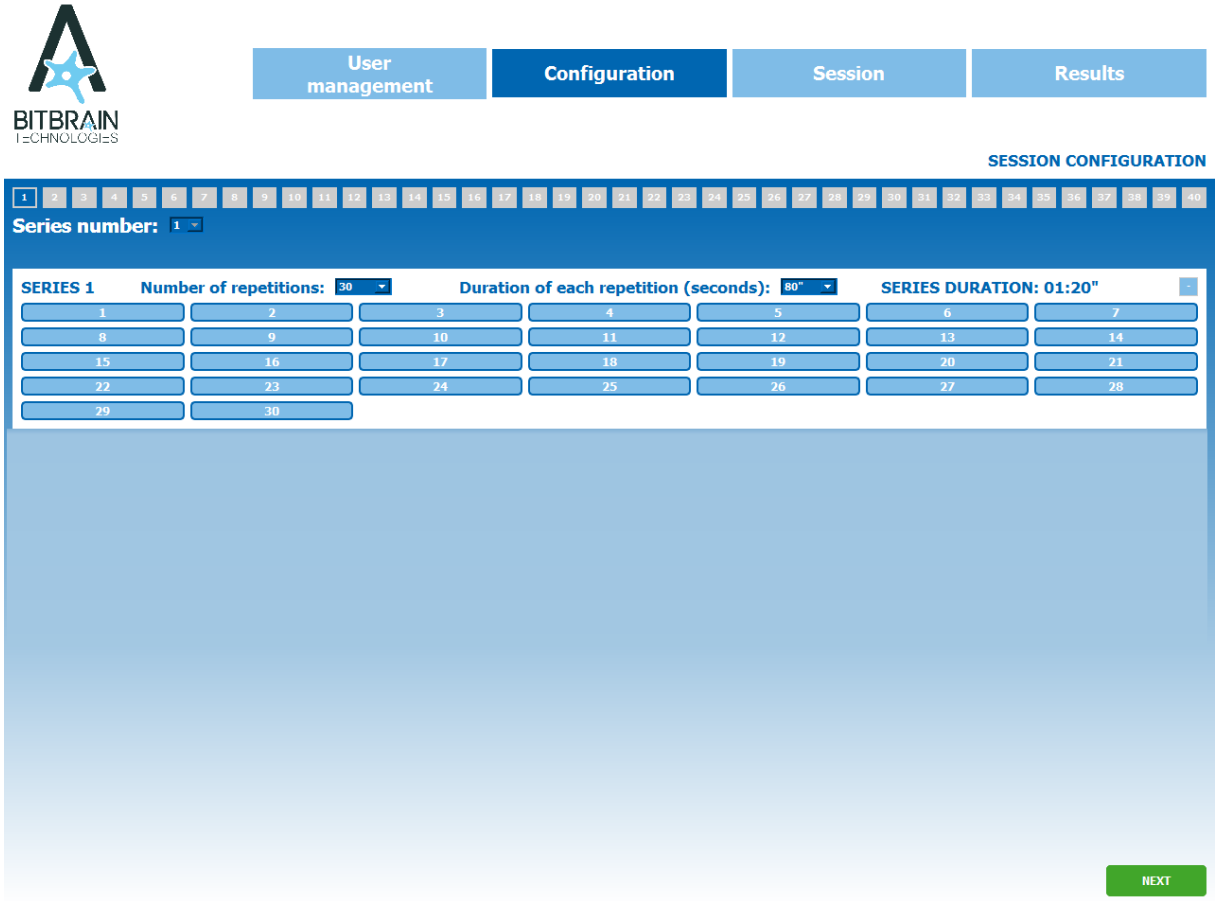


Figura 2.8: Aspecto de la interfaz de usuario correspondiente a la gestión y edición de la terapia.

Una vez seleccionado o creado el usuario destino del entrenamiento, realizaremos tareas de edición y configuración de la sesión en curso de la terapia (Figura 2.8), pudiendo modificar en ésta tanto el número de fases/repeticiones, como la duración de las mismas. También permite ejecutar la configuración de sesión por defecto del sistema.

La selección personalizada del número de fases o repeticiones correspondientes a la sesión en curso, nos permitirá personalizar la terapia a petición del usuario final, adecuándolo a sus necesidades. Por otro lado, la opción por defecto suministrada por BrainUp

responderá a los parámetros óptimos para la terapia.

2.3.3. Paso 3: Comprobación de defectos de montaje

Tras la edición, se deberán realizar comprobaciones acerca del estado del montaje, puesto que una mala colocación de los sensores, llevaría de manera unilateral a una errónea realización del entrenamiento.

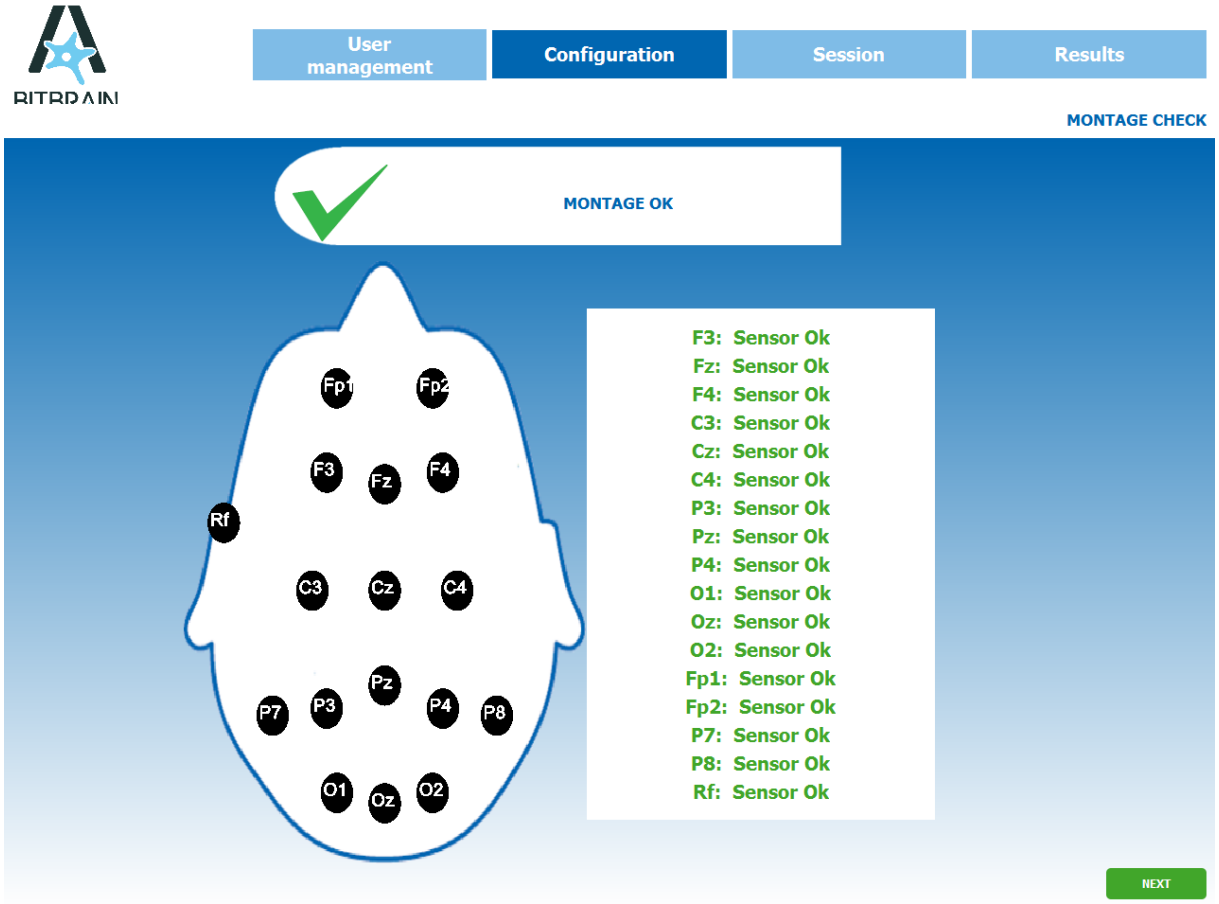


Figura 2.9: Aspecto de la interfaz de usuario correspondiente a la calibración de artefactos.

Conseguiremos la correcta colocación de los sensores de EEG de una manera fácil, usable e intuitiva, dejando en un segundo plano complejas técnicas basadas en la observación de señal EEG no filtrada en búsqueda de errores relevantes (Figura 2.9).

La detección de uno o varios sensores erróneos será notificada, tanto en la reproducción por pantalla del montaje, como en un área de texto anexa al montaje y habilitada para tales efectos.

2.3.4. Paso 4: Calibración de Ritmos

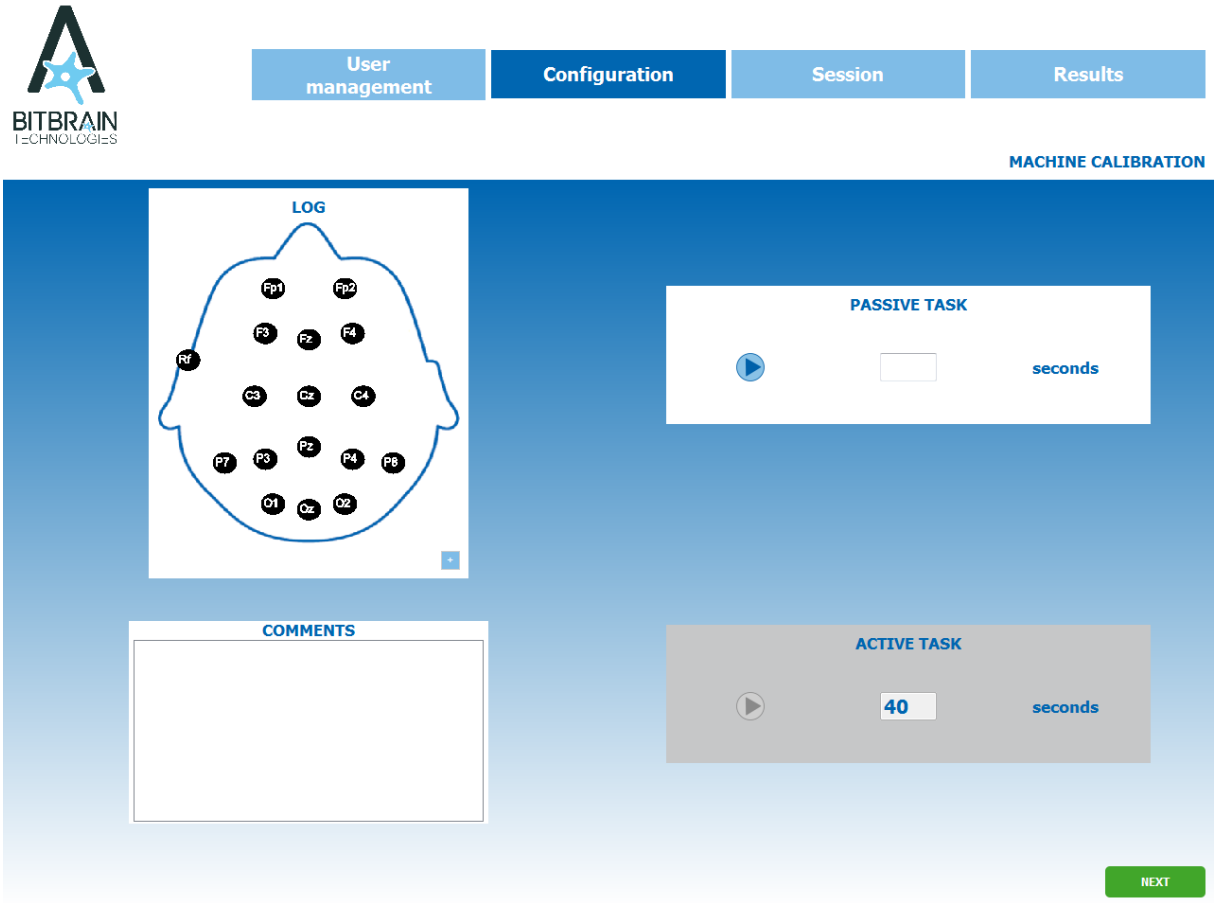


Figura 2.10: Aspecto de la interfaz de usuario correspondiente a la calibración de ritmos.

Si bien como hemos comentado anteriormente, la elección del *IAF*, permitía a BrainUp gozar de ciertos beneficios frente a otro tipo de aplicaciones, esta elección conlleva el obligado cumplimiento de una serie de tareas, recogidas en esta interfaz.

La localización del *IAF* necesita de la correcta realización de ambas tareas (Figura 2.10), así mismo, durante el proceso de tarea activa se realizará la calibración del filtro *ICA*.

En ocasiones, y debido a diversos factores como puede ser el cansancio o una errónea realización/ejecución de las tareas puede resultar imposible calcular *IAF*, en ese caso, sería aplicado el valor obtenido en sesiones anteriores, en caso de no haberlo, se aplicaría uno por defecto.

En el caso del filtro *ICA*, no existe un filtro por defecto, luego en caso de resultar imposible su cálculo, la señal quedaría sin filtrar.

Una vez aplicados ambos parámetros, puede realizarse la terapia con total normalidad.

2.3.5. Paso 5: Terapia

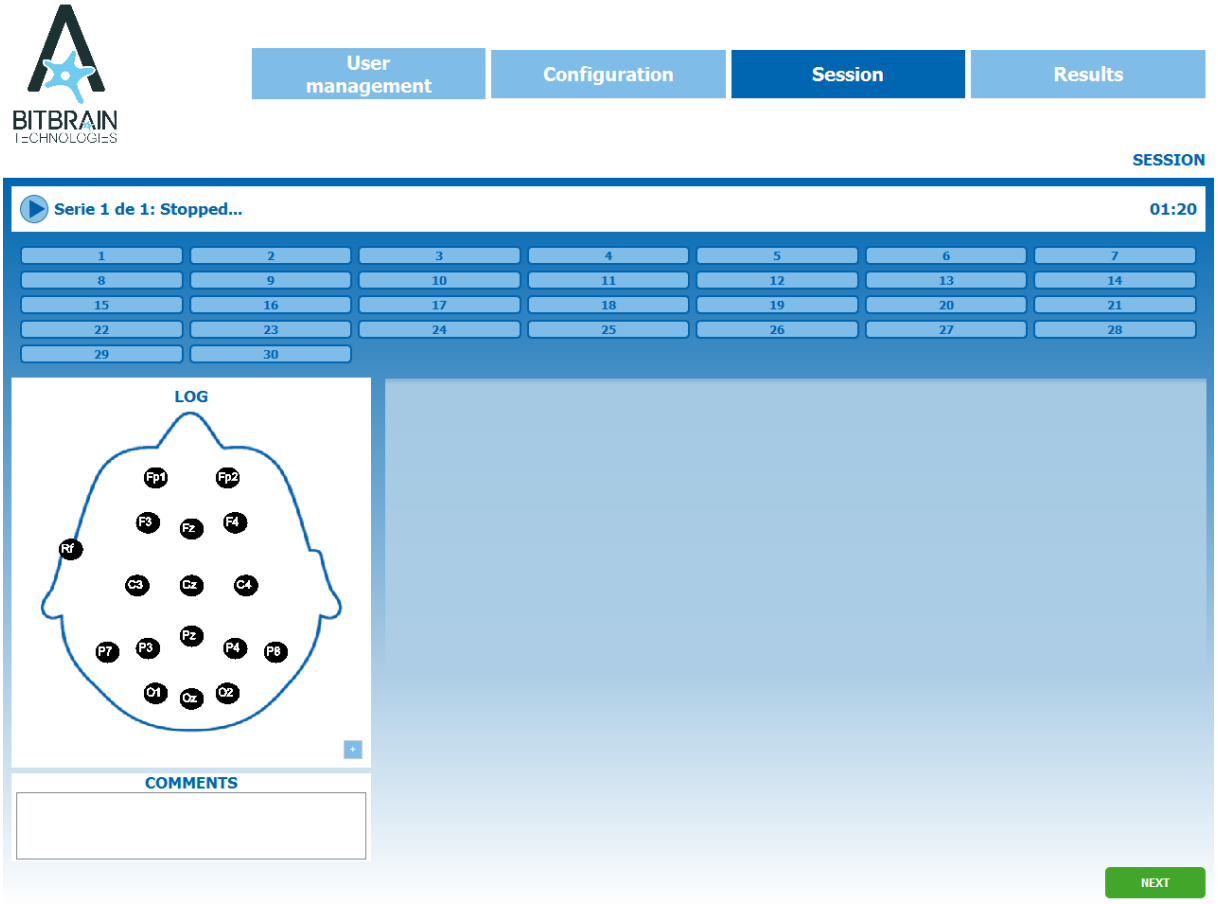


Figura 2.11: Aspecto de la interfaz de usuario correspondiente a la fase en curso.

Interfaz de usuario destinada al control del entrenamiento, posee a su vez herramientas de notificación de error, de manera que el terapeuta tenga conocimiento en todo momento de cualquier anomalía surgida durante la terapia.

Permite la pausa/cancelación de la sesión en ejecución (Figura 2.11), ya sea a petición del usuario (descanso), o bien a juicio del terapeuta (aparición de artefactos inesperados, insatisfacción con la sesión diseñada).

3. Desarrollo

La metodología utilizada a lo largo de esta fase ha sido el *proceso unificado de desarrollo software*, caracterizado por estar dirigido por los casos de uso, encontrarse centrado en la arquitectura, ser iterativo e incremental, constando de las siguientes fases:

1. **Análisis:** Fase cuyo objetivo es producir las tablas de requisitos del sistema previa identificación y generación de los casos de uso, describiendo la interacción del usuario con el sistema y pudiendo así evaluar sus necesidades.

   Los requisitos resultantes de esta fase se encuentran divididos en dos grandes grupos:

   a) **Requisitos funcionales:** Aquellos relativos a como serán satisfechos los casos de uso en el futuro sistema (RF-X).

   b) **Requisitos No funcionales:** Generalmente establecidos por el analista, representan características requeridas por el sistema, o el proceso de desarrollo, no producen efecto alguno sobre el actor del sistema (RNF-X).

2. **Diseño:** Consistente en presentar un modelo que permita satisfacer todos los requisitos funcionales y no funcionales recogidos durante la fase anterior, reflejados de manera estática por los diagramas de clase y de manera dinámica gracias a los diagramas de secuencia, actividad o estado. A la finalización de esta fase se realizará un prototipado de ventanas.

3. **Implementación:** Donde se procede a la codificación de las clases extraídas en el diagrama de clases, debiendo satisfacer a su vez la dinámica descrita en los diagramas de secuencia, actividad o estado.

4. **Pruebas:** Separadas en pruebas individuales, de integración y de usabilidad, realizadas a fin de certificar el correcto funcionamiento de la solución implementada.

   Como hemos comentado, se trata de un proceso iterativo, donde tras la fase de implementación se obtendrá una primera versión, la cual puede ser incluida de nuevo en el proceso con el fin de enriquecer sus funcionalidades.

3.1. Herramienta de comprobación de defectos de montaje

3.1.1. Introducción

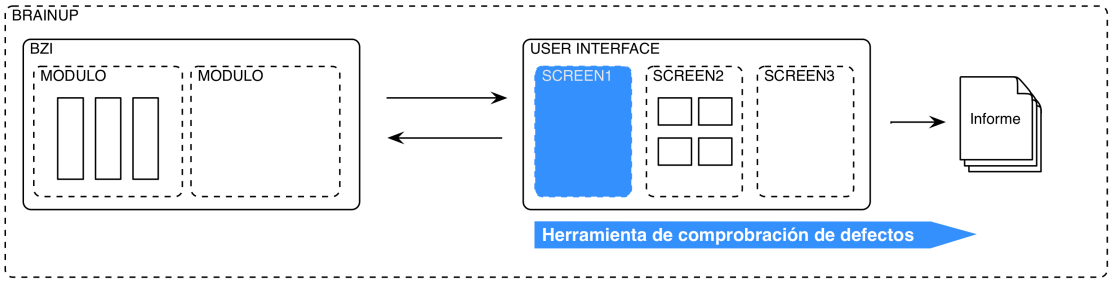


Figura 3.1: Herramienta de comprobación.

Se trata de una interfaz de usuario dedicada en exclusiva a la comprobación del estado del montaje previa al entrenamiento, como vimos, cualquier tipo de perturbación durante el mismo produciría resultados incongruentes con la actividad real del usuario/paciente, es por ello, por lo que se impide continuar la ejecución mientras no queden resueltos estos defectos.

El usuario/terapeuta puede realizar las comprobaciones pertinentes visualizando cualquiera de los siguientes indicadores:

1. Representación visual del montaje.
2. Notificación del error en cada uno de los sensores (modo texto).
3. Indicador general de estado.

La secciones B.1 y 2.3.3 detallan el funcionamiento de la herramienta de comprobación, desde el punto de vista del usuario/terapeuta.

3.1.2. Análisis

Definimos el caso de uso relativo a la comprobación de defectos en el montaje, procediendo a la posterior extracción de requisitos funcionales y no funcionales.

Nombre	Caso de uso 1
Actores que intervienen	Usuario/Terapeuta
Descripción	Comprobación del estado del montaje.
Precondición	El terapeuta se encuentra en la herramienta de comprobación.
Secuencia de acciones	**1.** Comprueba el estado del montaje en el indicador central. **2.** Revisa qué electrodos/sensores se encuentran erróneos y en qué posición se encuentran. **3.** Visualiza el texto correspondiente a la causa de los errores.
Resultados	El usuario/terapeuta ha comprobado el montaje.

Tabla 3.1: Caso de uso correspondiente a la comprobación del estado del montaje.

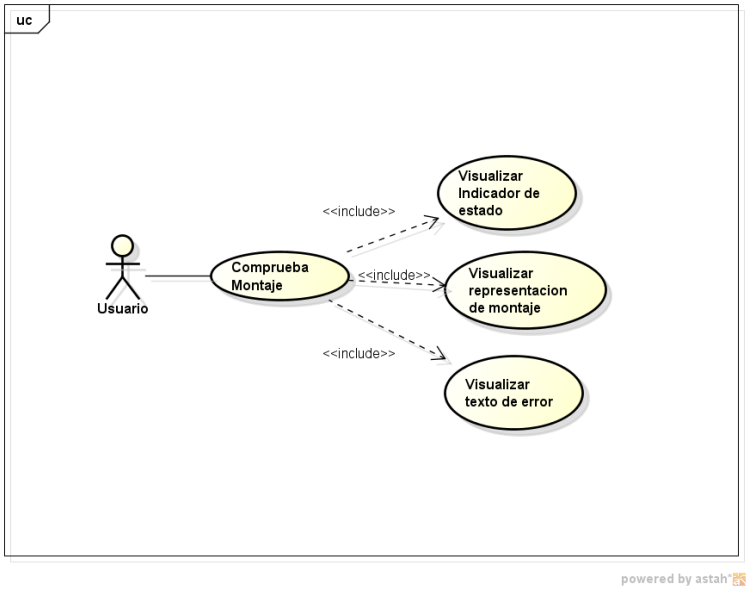


Figura 3.2: Gráfico de caso de uso correspondiente a la comprobación del montaje.

Obteniendo los siguientes requisitos a satisfacer:

Código	Descripción
RF-0	El sistema debe ofrecer una herramienta de comprobación del montaje
RF-1	La herramienta debe tener un indicador central de estado del montaje.
RF-2	El sistema debe tener una representación visual del montaje.
RF-3	El sistema debe ofrecer la causa de error de los diferentes sensores.
RNF-1	Debe ser intuitiva.
RNF-2	Debe ser muy eficiente y funcionar en tiempo real.
RNF-3	Debe estar disponible en diferentes idiomas.

Tabla 3.2: Requisitos de la herramienta de comprobación de defectos de montaje.

3.1.3. Diseño

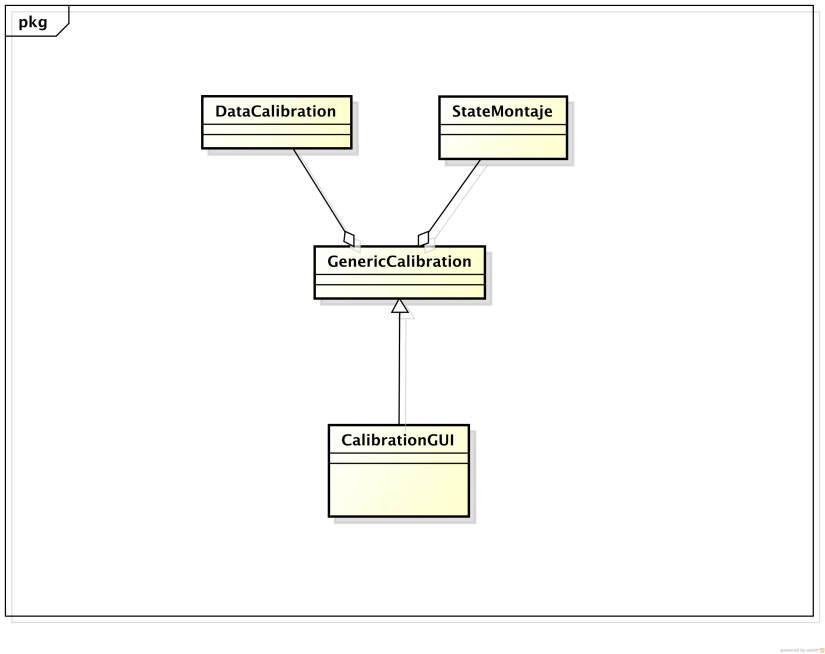


Figura 3.3: Diagrama de clase base *GenericCalibration*.

Como hemos comentado anteriormente, se trata de un sistema completo dentro de las interfaces de usuario presentes en BrainUp (figura 2.9), a tal efecto, deberá integrarse y adaptarse a la estructura diseñada para la aplicación.

Comenzaremos nuestro diseño definiendo la clase base de nuestra herramienta *GenericCalibration*, cuya misión es aglutinar elementos comunes a futuros diseños o especializaciones de la misma. En nuestro caso, cualquier calibración poseerá al menos dos elementos básicos:

1. Un objeto *Data*, contenedor de valores genéricos y encargado de dotar a cualquier especialización de esta clase de métodos de adquisición y distribución de datos.
2. Un indicador de estado general del montaje centralizado e independiente de las herramientas de visualización incluidas.

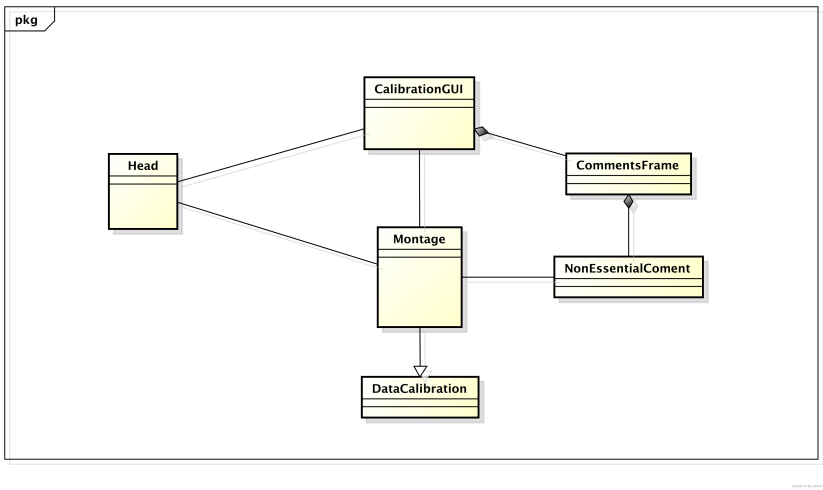


Figura 3.4: Diagrama de clase de la herramienta de comprobación de defectos de montaje

De la clase base definida anteriormente hereda *CalibrationGUI*, interfaz de usuario encargado de la comprobación del montaje acorde con los requisitos extraídos anteriormente. Deberá encontrarse debidamente comunicado con el resto de interfaces de BrainUp.

Entre sus componentes se encuentra:

- **Montage:** Se trata de la clase más importante dentro de nuestra herramienta. Actúa como almacén central de información entre la unidad de detección de electrodos erróneos (BZI) y el interfaz de usuario en el que nos encontramos, evitando así la necesidad de instanciar un contenedor de datos para cada uno de ellos.

  Al tratarse de un elemento común a ambas partes del sistema es creado por una instancia superior dentro de BrainUp llamada *InterfaceController* y ajena a este PFC, su misión es la creación y gestión de los diferentes interfaces de usuario, así como aquellos elementos compartidos con BZI.

- **Head:** Representación visual del montaje realizado, obtiene los datos necesarios para su representación del objeto *Montage*.

- **CommentsFrame:** Cuya misión es la creación y gestión de una o varias zonas de notificación en modo texto.

- **NonEssentialComment:** Representa el área de notificación de defectos de montaje (modo texto), así como sus operaciones de pintado. Adquiere sus datos del objeto *Montage*.

- **StateMontage:** Fruto de la herencia con la clase base, hace las veces de indicador central e inequívoco del estado del montaje. Este objeto tomará especial relevancia en futuras implementaciones, donde existirán diferentes niveles de error y por consiguiente, diferentes tipos de notificación.

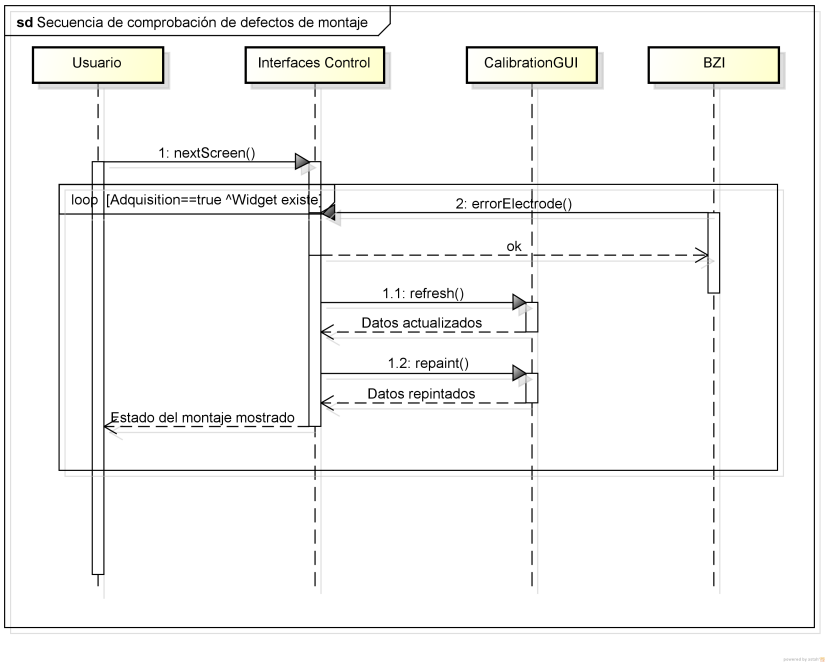


Figura 3.5: Secuencia de comprobación de defectos de montaje.

La adquisición/distribución de los valores dentro de *CalibrationGUI* se realiza a través del mecanismo heredado de *GenericCalibration*, gracias a este tipo de operaciones conseguimos operar los datos de forma genérica.

En la figura 3.5 y 3.6 podemos observar la secuencia de acciones relativa al caso de uso 3.2 donde un usuario que desea comprobar el estado del montaje accede al interfaz de usuario, previa creación del objeto montaje y de la interfaz de comprobación de defectos del montaje. Tras la comunicación por parte de BZI de qué electrodos se encuentran erróneos (realizada a *InterfacesControl* vía TCP/IP) procederemos a actualizar el objeto montaje y a repintar todos los elementos en pantalla. Estas dos últimas operaciones se realizarán de manera iterativa mientras se siga adquiriendo señal y procesando electrodos, o mientras alguno de los objetos de comprobación de montaje se encuentre instanciado en los interfaces de usuario.

Como puede observarse en la figura 3.6, es *InterfacesControl* el encargado de la creación del objeto *CalibrationGUI*, el objeto *Montage* y dos vectores de datos, un vector correspondiente a los electrodos erróneos y otro a los electrodos borrados. Tras esta operación, el interfaz queda a la espera de ser visualizado a petición del usuario.

Por último en la figura 3.7 definimos la secuencia de acciones a realizar para la representación visual de los electrodos erróneos, en primera instancia y tras la llegada desde BZI de los datos, procederemos a actualizar el montaje que como recordaremos almacena la información referenciada por todos las clases. Tras esto distribuiremos los datos entre los objetos de representación y procederemos a actualizar la representación cada uno de los indicadores visuales.

Las secuencias específicas de repintado de los elementos por pantalla se encuentran detalladas en al anexo A.

Tras esta fase obtenemos un primer prototipo de la herramienta (Figura 3.8)

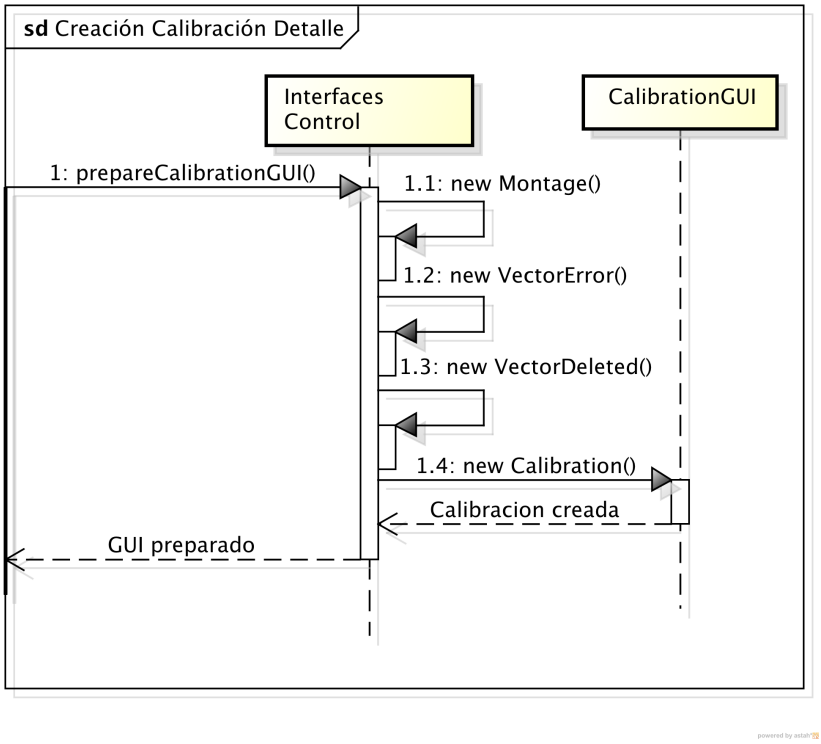


Figura 3.6: Prototipo de la herramienta de comprobación de defectos del montaje.

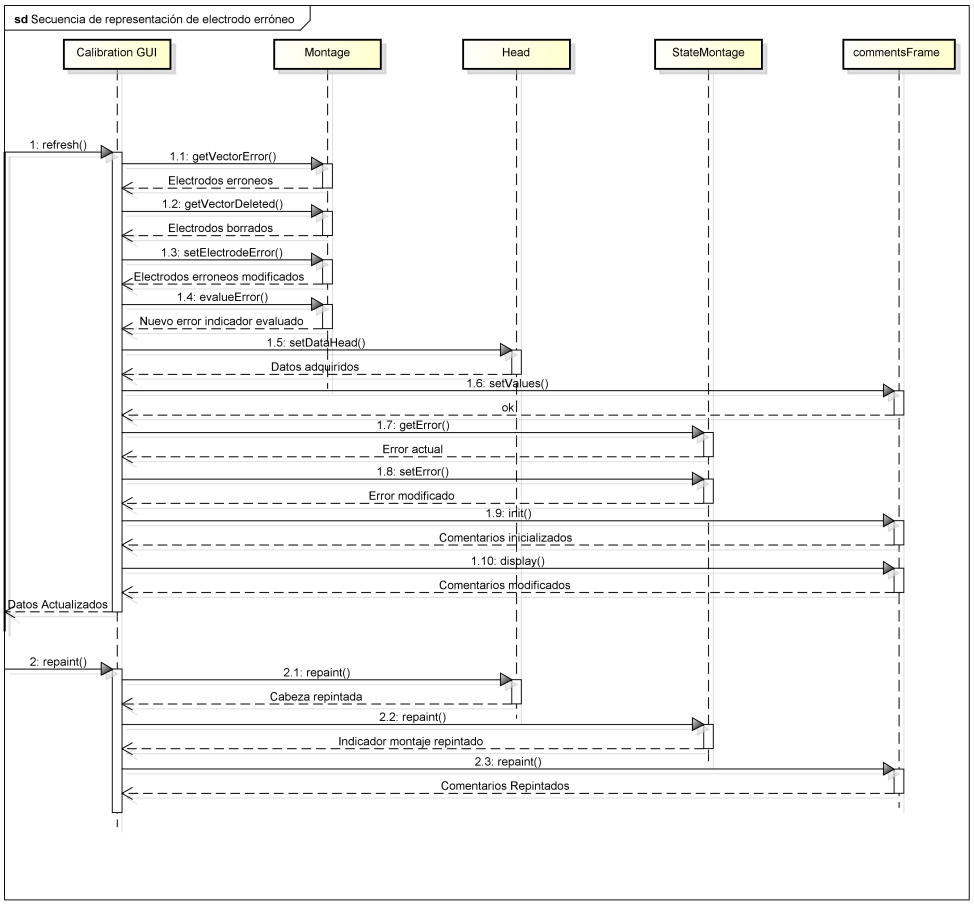


Figura 3.7: Secuencia de representación de electrodo erróneo

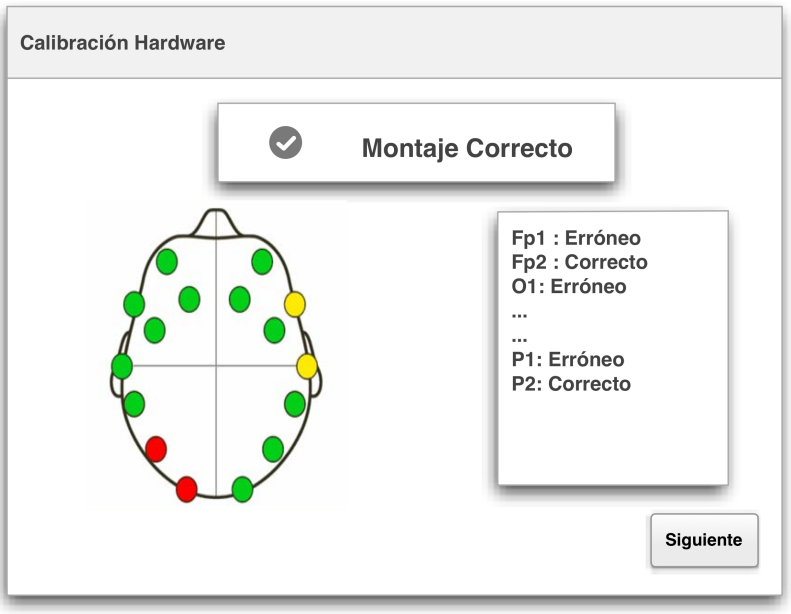


Figura 3.8: Secuencia de creación de la herramienta de comprobación.

3.1.4. Implementación

Realizada en C++ bajo el framework *Qt*[19], en el que se encuentran implementados todos las interfaces de usuario de BrainUp.

Debido a que la operación de representación/pintado es muy costosa en *Qt*, se utilizo para *Head* un tipo especial de elemento gráfico de nombre *QPainterPath*[20], donde en lugar de dibujar cada elemento individualmente, se permite agrupar gran cantidad de éstos en una única capa y dibujarla de manera atómica tantas veces como sea necesario.

En el caso que nos ocupa, donde la mayoría de los electrodos no van a modificar su estado en largos periodos de tiempo, y teniendo un tiempo de ciclo reducido, resulta más eficiente añadir elementos comunes (electrodos correctos o erróneos) en una variable *QPainterPath* que dibujarlos individualmente[21] en cada iteración de adquisición, tarea que resultaba crítica.

3.1.5. Pruebas

Las pruebas se realizaron en *Matlab*, comparando los electrodos erróneos provenientes de la unidad de detección de defectos de montaje (la cual explicaremos en la próxima sección) con los errores presentes en el objeto *Montage* en el instante previo a la invocación a *repaint()*. De esta manera somos capaces de asegurar que los datos generados

por la unidad de detección y los datos preparados para su pintado son iguales, y que por consiguiente la representación visual es correcta.

Las pruebas 3.3 fueron realizadas con cuatro ficheros de *EEG* grabados con anterioridad, y se encuentran disponibles en el anexo C.

Fichero:	*testS00R004.bzi*
Precondición:	El interfaz de comprobación y la unidad han procesado fichero.
Datos de entrada:	*unitChannelsS00R004* : contiene los canales detectados como erróneos calculadas por la unidad. *interfaceChannelsS00R004* : contiene los canales detectados como erróneos presentes en el interfaz.
	$A = loadFile(interfaceChannelsS00R004)$ $B = loadFile(matlabChannelsS00R004)$ $ans = A - B$
Resultado:	$ans = 0$ Los canales detectados como erróneos son iguales tanto en cuáles son detectados, como en qué instante temporal.

Tabla 3.3: Prueba 1 de la herramienta de comprobación de defectos de montaje.

A su vez, se realizaron pruebas con semántica diferencial con el fin de conocer el grado de aceptación de la herramienta, así como su calificación en materia de usabilidad, claridad y facilidad de uso. Estas pruebas, cuyo resultado puede observarse en la tabla 3.4 les fueron realizadas a 10 sujetos y se encuentran disponibles por cada uno de los usuarios en el anexo C.

La técnica se desarrolla proponiendo dos adjetivos al sujeto, que se han de relacionar con los conceptos propuestos siendo presentados de forma bipolar, mediando entre ambos extremos una serie de valores intermedios.
El sujeto procede puntuando así : *Bueno 3 2 1 0 -1 -2 -3 Malo.*

Pregunta	Respuesta
	3 2 1 0 -1 -2 -3
En general el interfaz de comprobación de montaje	Me gusta **2.3**
El interfaz me parece	Intuitivo **2.9**
El notificador de estado del montaje	Claro **2.6**
La representación visual del montaje	Claro **2.8**
La representación tipo texto de los errores	Claro **1.9**
En términos generales el funcionamiento me parece	Claro **2.7**

Tabla 3.4: Resultados globales de la evaluación.

3.2. Unidad de detección de defectos de montaje

3.2.1. Introducción

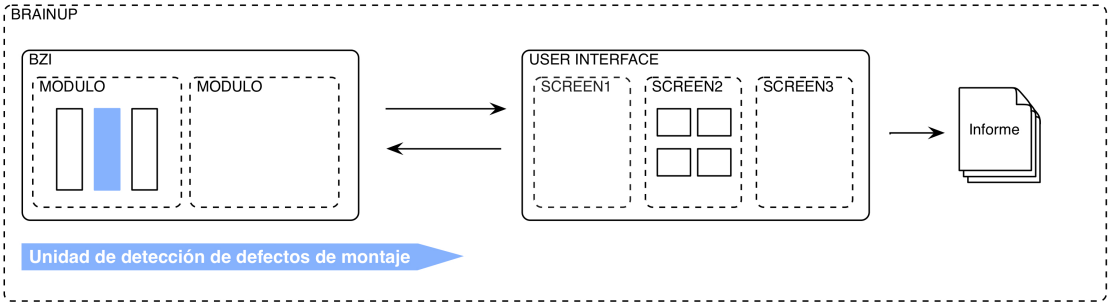


Figura 3.9: Unidad de comprobación.

Como en todo sistema de comprobación y chequeo, el apartado visual solo representa una pequeña parte de la herramienta, siendo necesario un algoritmo de procesamiento capaz de proveer datos a esta interfaz de usuario. En nuestro caso, versiones anteriores se limitaban a proporcionar EEG no filtrado como mecanismo de chequeo mediante inspección visual, resultando altamente ineficiente pues requería de conocimientos previos en material de señal.

Son estos conocimientos en materia de señal los que han permitido a BitBrain Technologies desarrollar un método de procesamiento donde como entrada tendremos la señal EEG tradicional, y como salida qué electrodos del montaje se encuentran erróneos así como su causa. Gracias a este tipo de algoritmos somos capaces de ofrecer una interfaz usable e intuitiva al usuario/terapeuta, sin necesidad de que éste posea conocimientos previos sobre encefalografía.

El método de forma genérica consta de lo siguientes pasos:

1. **Subsampleo:** Debido a que necesitamos una ventana temporal de n segundos, pero no deseamos procesar la totalidad de la información, por ello elegimos un factor de subsampleo m, según el cual serán adquiridos 1 de cada m samples.

2. **Cálculo de medias:** Tras subsamplear la señal procedente de la fase de adquisición se procederá al calculo de las medias por cada canal (sensor) para cualquier instante de tiempo y en valor absoluto, procediendo en última instancia a su ordenación de menor a mayor.

3. **Regresión:** Calcularemos la recta de regresión conforme a las num primeras medias anteriormente calculadas.

4. **Lógica:** Tras el cálculo de la recta de regresión se procede a aplicar lógica de decisión sobre la distancia de cada una de las medias a la recta, y en comparación con un *threshold* definido previamente. Si la distancia supera el *threshold*, se considera que en ese sensor y en ese instante temporal (sample) se ha producido *popping*.

Un electrodo puede producir error por las siguientes razones:

- **Ausencia de gel:** Para el correcto funcionamiento de los electrodos del montaje debe aplicarse en cada uno de ellos un gel conductor evitando así los problemas de medición producidos por el tejido epitelial o el cuero cabelludo.

- **Electrodo suelto (*Popping*):** Es el problema más común en un montaje de EEG, se produce cuando un electrodo pierde el contacto durante momentos puntuales del entrenamiento/terapia perturbando la señal.

- **Ruido:** Aumentando sustancialmente la frecuencia y amplitud de la señal, puede ser provocado por artefactos de tipo muscular (tensión involuntaria en la frente, pulso) o de tipo eléctrico (acoplamiento, electricidad estática).

- **Puente (*Bridge*):** Evidencia de interconexión en el gel aplicado en dos sensores próximos en el espacio, provocando un cortocirtuito y obteniendo en ambos la misma señal eléctrica.

El método expuesto anteriormente, permite la detección de *popping* de forma robusta, su eficacia ha sido probada en el entorno matemático *Matlab* con resultados satisfactorios. El procesamiento del resto de defectos de montaje se encuentra en fase de desarrollo.

3.2.2. Análisis

En este caso no existen nuevos casos de uso, pues ya han sido definidos para la herramienta de comprobación de defectos de montaje en la sección 3.1.2, sin embargo, como fruto del análisis, en esta sección si es generado un nuevo requisito no funcional la necesidad de implementar una unidad de detección de defectos de montaje dentro de la arquitectura BZI.

Código	Descripción
RF-0	El sistema debe ofrecer una herramienta de comprobación del montaje
RF-1	La herramienta debe ofrecer un indicador de estado del montaje.
RF-2	El sistema debe contener una representación visual del montaje.
RF-3	El sistema debe ofrecer la causa de error de los diferentes sensores.
RNF-1	**Debe procesarse la información con una unidad de BZI para obtener los sensores erróneos y poder representarlos.**
RNF-2	Debe ser intuitiva.
RNF-3	Debe ser muy eficiente y funcionar en tiempo real.
RNF-4	Debe estar disponible en diferentes idiomas.

Tabla 3.5: Requisitos completos de la herramienta de comprobación de defectos de montaje.

3.2.3. Diseño

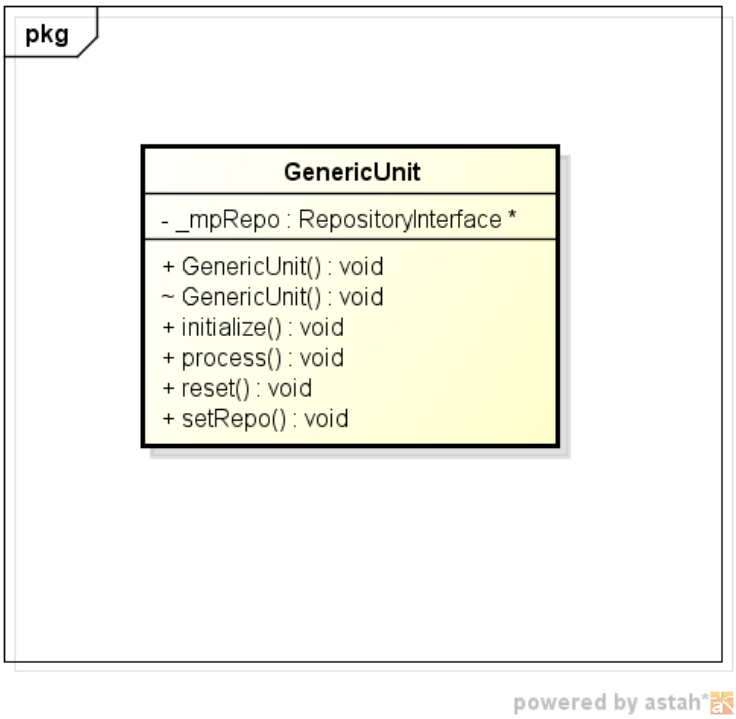


Figura 3.10: Estructura de la clase base GenericUnit

Comenzaremos comentando la estructura de clase base proporcionada por BZI para la creación de unidades de procesamiento (figura 3.10), como podemos observar se dota a cualquier especialización de ésta de mecanismos de inicialización, procesado y reseteo, así como acceso a la estructura de repositorio y compartición de datos presente en BZI.

Mientas los procedimientos de inicialización *(Initialize)* y reseteo (*Reset*) se encuentran definidos en la clase base, el mecanismo de procesado (*process*) es virtual puro, luego deberá ser reimplementado en cualquier de las especialización de la clase base, como en el caso que nos ocupa.

Como una especialización de *GenericUnit* y haciendo uso del mecanismo de herencia, definimos *MontageChecker* (Figura 3.11), unidad encargada de realizar el procesamiento descrito en la sección 3.2.1. Compuesta de un objeto *Subsampler* encargado del subsampleo de la señal y de *ElectrodeChecker* donde se realizará tanto el procesado, como la lógica de decisión.

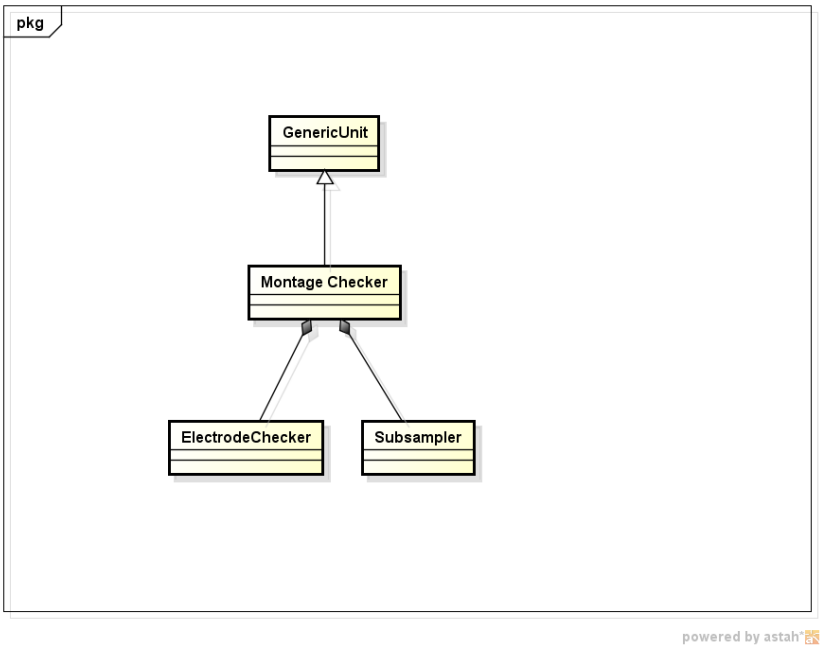


Figura 3.11: Diagrama de clase de MontageChecker

El diagrama 3.12 responde a la secuencia de acciones a realizar por la unidad de detección de electrodos erróneos. Tras la invocación a *Initialize*, se procede a subsamplear la señal mientras el criterio de parada de la ventana/buffer de recepción así lo indique, este criterio de parada ofrece dos vertientes:

- **Tamaño de ventana:** Es necesario completar una latencia inicial, el tiempo que tardará la ventana en llenarse de datos.
- **Frecuencia de refresco:** Tras completar esta latencia inicial, se lanzará un primer chequeo de electrodos, para posteriormente lanzar uno cada $frecuenciaDeRefresco$ samples.

Tras la invocación a la comprobación de electrodos y en caso de encontrarse electrodos erróneos, estos serán distribuidos a *NetModule*, módulo de red TCP/IP que posteriormente

los comunicará al *Manager* de BZI, quien por último informará al interfaz de comprobación de defectos de montaje, descrito en la sección 3.1.2.

La secuencia completa de comprobación de defectos de montaje, tanto por parte de BZI, como por parte de los interfaces de usuario, se encuentra presente en el anexo A (Figura A.10).

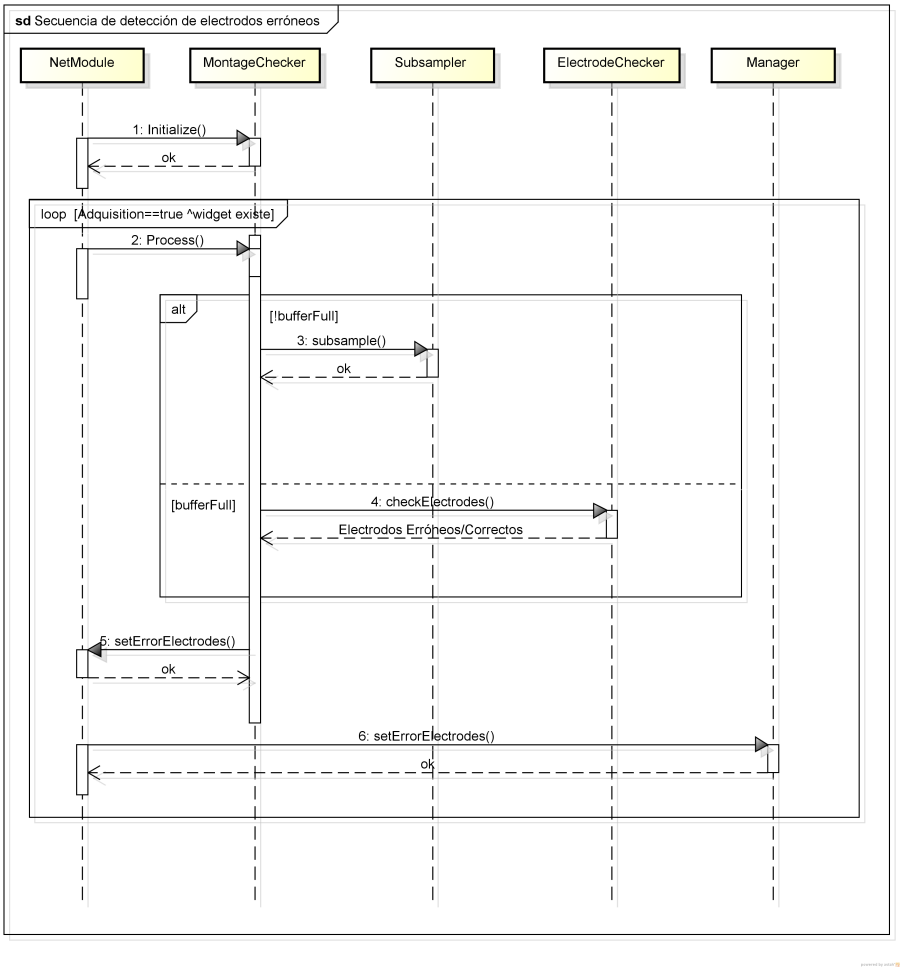


Figura 3.12: Diagrama de secuencia de detección de electrodos erroneos.

3.2.4. Implementación

Realizada respetando la estructura básica de toda unidad BZI , facilitando así su integración en la arquitectura y su implantación en el resto del sistema. Esta modularidad le permite así mismo estar abierta a futuras implementaciones, como la detección de ruido en la señal, sin que la dinámica del resto del sistema se vea comprometida.

Se encuentra implementada en *Qt*, siendo asistida por el framework matemático *Ar-*

madillo[22] en labores como la ordenación del vector de medias o la localización de la posición de las mismas tras el cálculo de la recta de regresión (*componente 1 corresponde a canal 3(FP1)*), evitando así tener que recurrir a una estructura de tipo *Map* para la indexación de las mismas.

Durante la fase de implementación se debe proceder a ajustar los parámetros de procesamiento convenientemente (ventana, refresco) puesto que una mala elección de los mismos podría desembocar en un mal funcionamiento de la unidad. Se debe gestionar a su vez la cadencia con la que la unidad de detección de defectos de montaje debe informar a los interfaces de usuario de la presencia de *popping* (Ej : no informar mientras no se produzcan cambios en el estado, o en qué electrodos producen *popping*).

3.2.5. Pruebas

En este caso, y al igual que con la herramienta de comprobación de defectos de montaje se realizaron pruebas de funcionamiento de la unidad, comparando los resultados obtenidos por la misma, con los resultados obtenidos tras la ejecución del mismo algoritmo en *Matlab*. Se diseñaron en dos fases, una primera donde se comprobó el valor de las medias, y otra donde se comprobó los electrodos erróneos resultantes.

En la tabla 3.6 y la figura 3.13 se observan las medias obtenidas por la unidad de detección, así como las resultantes del algoritmo de procesamiento codificado en *Matlab*.

Fichero:	*testS00R0015.bzi*
Precondición:	Matlab y la unidad han procesado el fichero.
Datos de entrada:	*unitMeansS00R0015* : contiene las medias calculadas por la unidad. *matlabMeansS00R0015* : contiene las medias calculadas por matlab.
	$A = loadFile(unitMeansS00R0015)$ $B = loadFile(matlabMeansS00R0015)$ $ans = A - B$
Resultado:	$ans = 0$ Las medias calculadas son iguales en ambos casos.

Tabla 3.6: Prueba de unidad de detección de electrodos medias 4.

En la tabla 3.7 y la figura 3.14 se observan los valores obtenidos por la unidad de detección, así como los resultantes del procesamiento con el mismo algoritmo en *Matlab*, aquellos bloques de señal donde se ha producido *popping* son marcados con una línea horizontal.

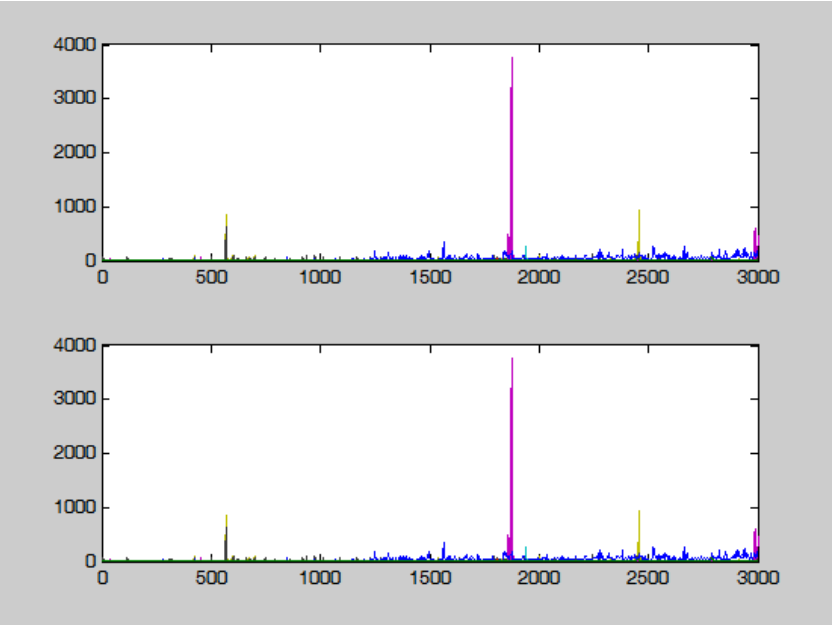
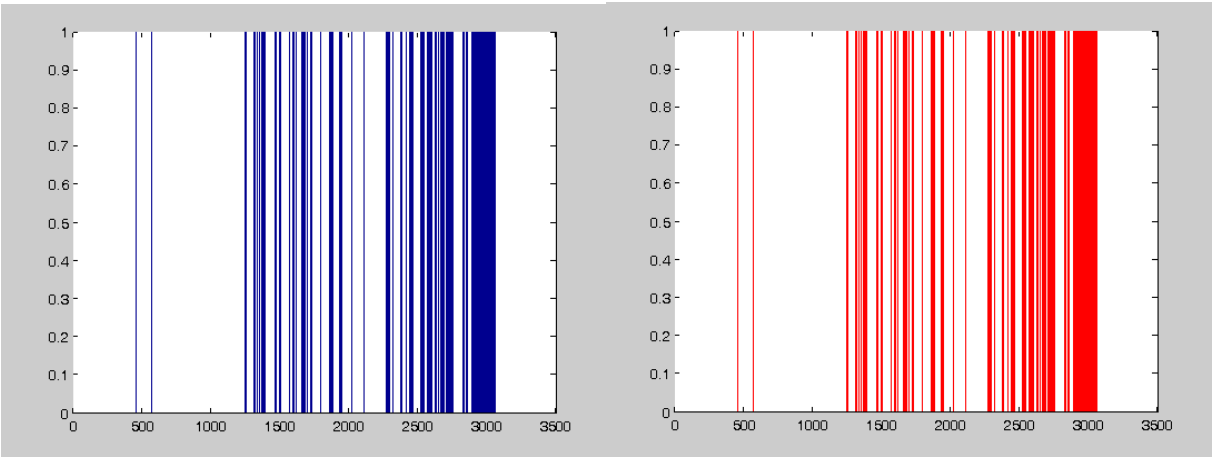


Figura 3.13: Comprobación de medias en fichero *testS00R015.bzi*.

Fichero:	*testS00R0015.bzi*
Precondición:	Matlab y la unidad han procesado el fichero.
Datos de entrada:	*unitChannelsS00R0015* : contiene los canales detectados como erróneos calculadas por la unidad. *matlabChannelsS00R0015* : contiene los canales detectados como erróneos calculadas por matlab.
	$A = loadFile(unitChannelsS00R0015)$ $B = loadFile(matlabChannelsS00R0015)$ $ans = A - B$
Resultado:	$ans = 0$ Los canales detectados como erróneos son iguales tanto en cuales son detectados, como en que momento temporal.

Tabla 3.7: Prueba de unidad de detección de electrodos 4.



(a) Unidad (b) Matlab

Figura 3.14: Comparación de popping en fichero *testS00R015.bzi*

3.3. Informe de resultados

3.3.1. Introducción

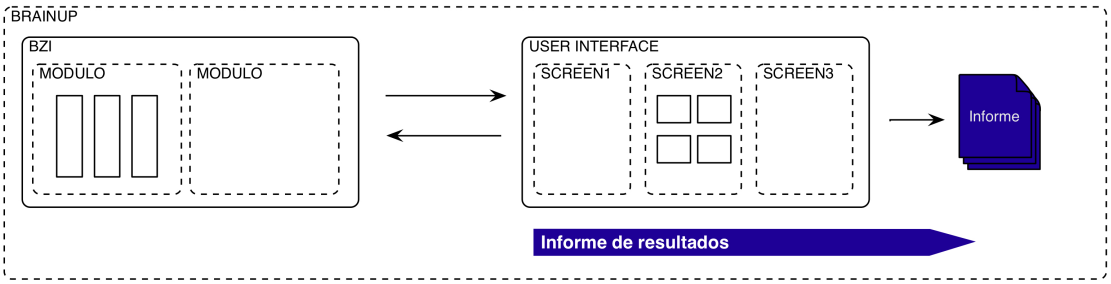


Figura 3.15: Informe de resultados.

Toda terapia necesita dejar constancia de sus resultados, ya sea con el fin de dimensionar el progreso, informar convenientemente al usuario/paciente o verificar su corrección. En este PFC se desarrolló el informe de resultados para BrainUp, tanto en su faceta visual, encuadrado dentro de las interfaces de usuario como impreso vía *PDF*, constando de los siguientes elementos:

- **Cabecera de datos de paciente:** Contiene los datos de usuario/paciente, el número de sesión en la que nos encontramos, su nombre y la fecha de realización.
- **Datos de terapia:** Contiene los datos de calibración de ritmos (descritas anteriormente *ICA+IAF*), la duración total del entrenamiento/terapia y la de cada repetición realizada.
- **Gráfica de rendimiento:** Representa la evolución del usuario/paciente en cada repetición con respecto al *baseline* tomado al inicio de la sesión.
- **Gráfica de permanencia temporal:** Muestra cuánto tiempo se ha ofrecido feedback positivo al usuario/paciente en cada repetición, tomando como valor de referencia $duracionEnSegundosDelTrial/2$
- **Gráfica de tres últimas sesiones:** Muestra la evolución en el usuario/paciente durante las tres últimas sesiones normalizado al valor del *baseline* de la primera sesión representada.

3.3.2. Análisis

Definimos en primera instancia el caso de uso relativo a la visualización del informe de resultados, para proceder a la posterior extracción de requisitos funcionales y no

Nombre	Caso de uso 1
Actores que intervienen	Usuario/Terapeuta
Descripción	Comprobación del estado del montaje.
Precondición	El terapeuta se encuentra en un interfaz de usuario.
Secuencia de acciones	**1.** Visualiza el número de sesión y los datos del paciente. **2.** Visualiza los datos de calibración de ritmos. **3.** Visualiza la gráfica de progreso. **4.** Visualiza la gráfica de tiempo. **5.** Visualiza la gráfica de las tres últimas sesiones. **6.** El usuario/terapeuta obtiene el informe en *PDF*.
Resultados	El usuario/terapeuta ha visualizado y obtenido el informe.

Tabla 3.8: Caso de uso correspondiente a visualización y impresión del informe.

funcionales del sistema.

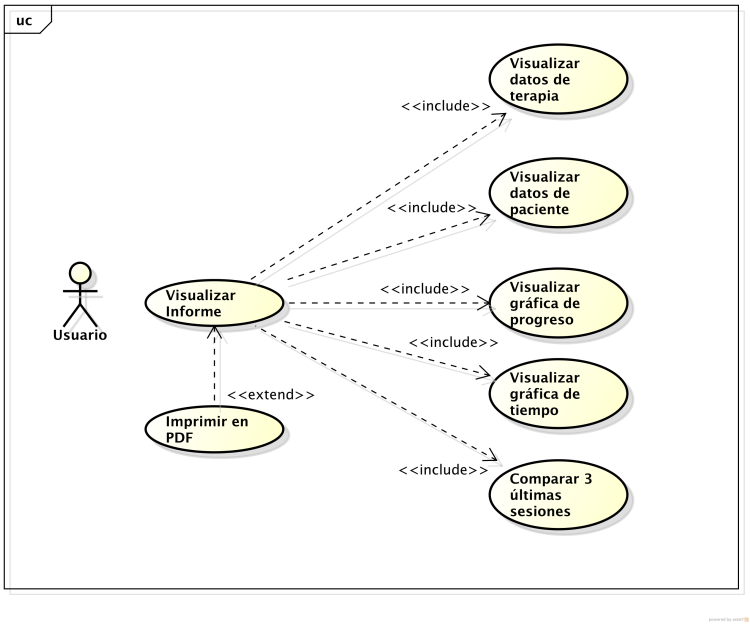


Figura 3.16: Gráfico de caso de uso correspondiente al informe.

Obteniendo los siguientes requisitos a satisfacer:

Código	Descripción
RF-0	Debe ofrecer el número de sesión y los datos de paciente
RF-1	Debe contener la duración de la terapia y de las repeticiones. así como la de la calibración de ritmos.
RF-2	Debe ofrecer una representación gráfica del progreso (rendimiento).
RF-3	Debe ofrecer una representación gráfica del progreso (tiempo).
RF-4	Debe contener una comparación de las tres últimas sesiones.
RF-4	El informe debe poder imprimirse.
RNF-1	Debe ser intuitivo.
RNF-2	Debe guardarse en PDF.
RNF-3	Debe estar disponible en diferentes idiomas.

Tabla 3.9: Requisitos del informe.

3.3.3. Diseño

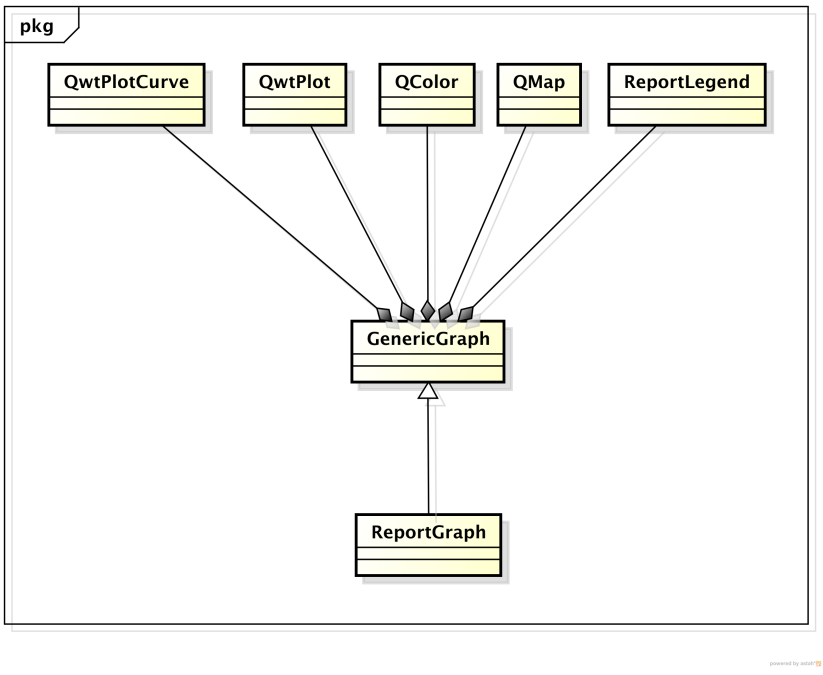


Figura 3.17: Diagrama de clase base GenericUnit

Comenzaremos diseñando la clase base de nombre *GenericGraph*, puesto que se debe permitir de cara a futuras implementaciones, un amplio espectro de representaciones gráficas (diagramas de barras, puntos, o funciones interpoladas), así como diferentes formatos, colores o estilos de línea.

Es por ello, por lo que además de disponer de los habituales métodos de adquisición/distribución de datos, contiene los siguientes elementos:

1. **QwtPlotCurve:** Perteneciente a *Qwt* (framework de representación gráfica x-y en C++ [23]), y cuya misión es proveer soporte matemático a la representación gráfica, cada objeto *QwtPlotCurve* representa una $f(x)$ diferente.

2. **QwtPlot:** Clase *Qwt* encargada de las labores de visualización, le pueden ser transmitidas de 1 a n *QwtPlotCurve* gracias al procedimiento *Attach()*. Una vez añadidas al objeto, pueden ser representadas invocando a la función *show()* de manera similar al resto de componentes de los diferentes interfaces de usuario, permitiendo además editar su estilo y su escala.

3. **QColor:** Debe contener todos aquellos colores que deseemos incluir en la representación gráfica.

4. **QMap:** Relaciona cada color incluido en *QColor* con una *QwtPlotCurve* diferente, de manera que podamos representar todos aquellos puntos pertenecientes a un determinado color con una única *QwtPlotCurve* Ej: *[1,3,5] Rojo; [2,4,6] Azul; Estilo='Barras'*.

5. **ReportLegend:** Proporciona la leyenda de la gráfica, su posición en la misma es configurable gracias a un atributo enumerado (*Above, Below, Left, Right*).

ReportGraph se define como una especialización de *GenericGraph*, donde deberá implementarse la lógica de decisión de qué puntos del eje x corresponden a qué colores, así como qué estilo debe ser aplicado (puntos,barras,líneas).

Contemplaremos dos casos:

Caso 1: Los valores en el eje x representarán repeticiones/trials, y los del eje y potencia media de todos los canales por cada repetición/trial. Aquellos puntos por encima del *baseline* inicial se representarán en rojo, mientras que aquellos que se encuentren por debajo, se representarán en azul, el *baseline* será representado con una línea amarilla horizontal (Estilo='Barras').

Caso 2: Los valores en el eje x representarán repeticiones/trials, y los del eje y el tiempo que ha permanecido el usuario/paciente por encima del *baseline* en cada repetición/trial. Aquellos puntos por encima de $duracionEnSegundosDelTrial/2$ se representarán en rojo, mientras que aquellos que se encuentren por debajo, se representarán en azul, $duracionEnSegundosDelTrial/2$ será representado con una línea amarilla horizontal (Estilo='Barras').

ThreeSessionGraph se diseña también como una especialización de *GenericGraph* bajo la siguiente premisa:

Caso 1: Los valores en el eje x representarán repeticiones/trials de las tres últimas sesiones, y los del eje y potencia media de todos los canales por cada repetición/trial en las tres últimas sesiones. Aquellos puntos por encima del *baseline* de cada sesión se representarán en rojo, mientras que aquellos que se encuentren por debajo, se representarán en azul, el *baseline* será representado como un punto en x adicional pero en color amarillo, todos los valores se encontrarán normalizados al *baseline*de la primera sesión representada (Estilo='Puntos').

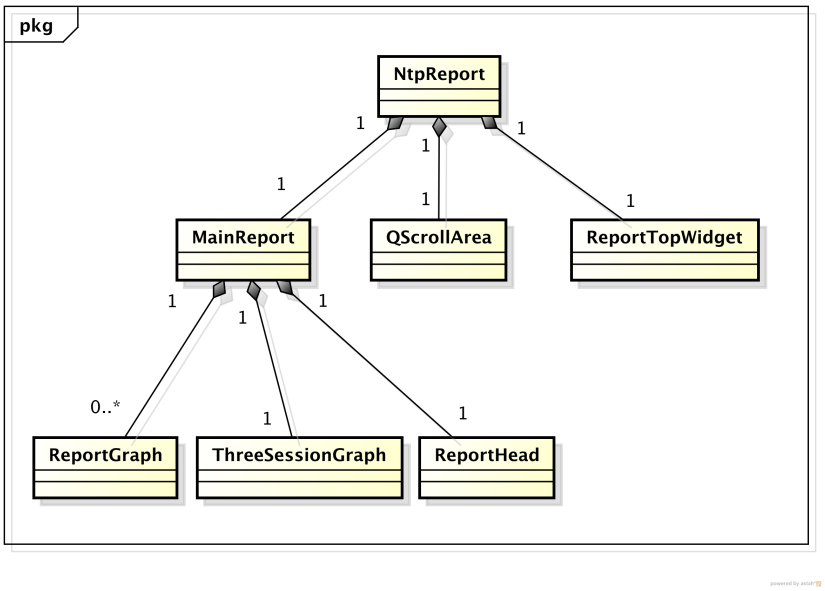


Figura 3.18: Diagrama de clases del informe de resultados.

La figura 3.19 ilustra la estructura del objeto principal *NtpReport*, compuesto a su vez de *QScrollArea* (necesario para desplazarnos dentro del documento formato *A4* generado), *ReportTopWidget* (encabezado donde aparecerán el número de sesión y los datos del usuario/paciente), y un objeto de tipo *MainReport* que contiene el informe en sí.

MainReport se compone de dos objetos de tipo *ReportGraph* (gráfica de rendimiento y de tiempo) y uno de tipo *ThreeSessionGraph* (gráfica de tres ultimas sesiones).

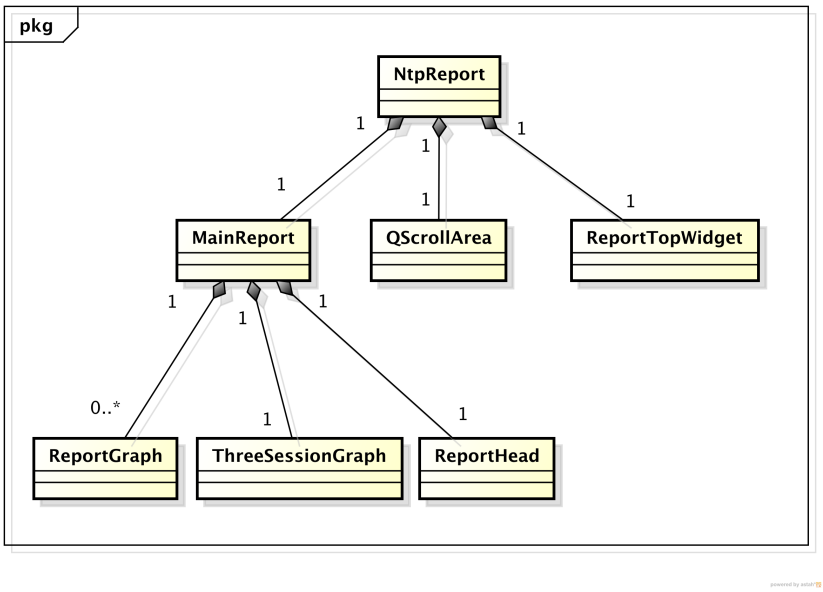


Figura 3.19: Diagrama de clases del informe de resultados.

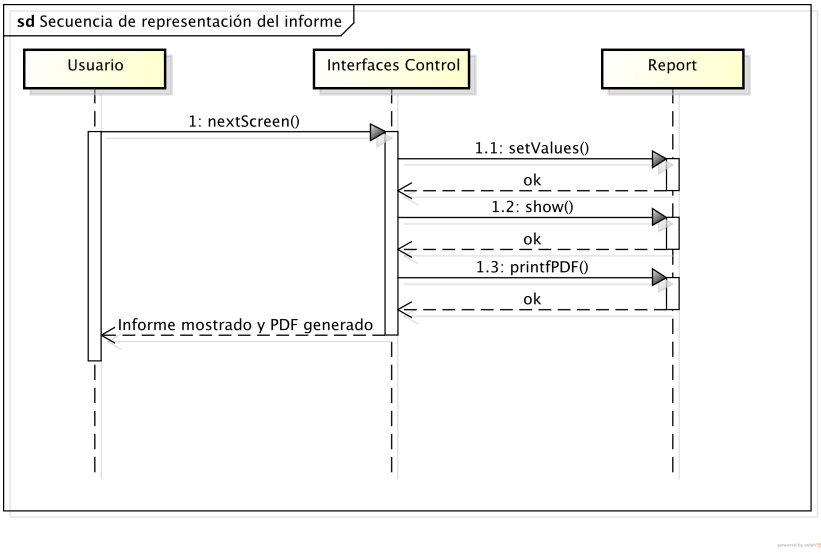


Figura 3.20: Diagrama de secuencia de representación del informe.

Como podemos observar en la figura 3.20 y 3.21 la estrategia utilizada para el informe de resultados es similar a la utilizada en apartado anteriores. Tras la petición por parte del usuario de "Siguiente pantalla", *InterfaceController* procede a servir a cada uno de los objetos mencionados los datos necesarios para su correcta representación, una vez visualizado el informe se procede a su impresión en formato *PDF*.

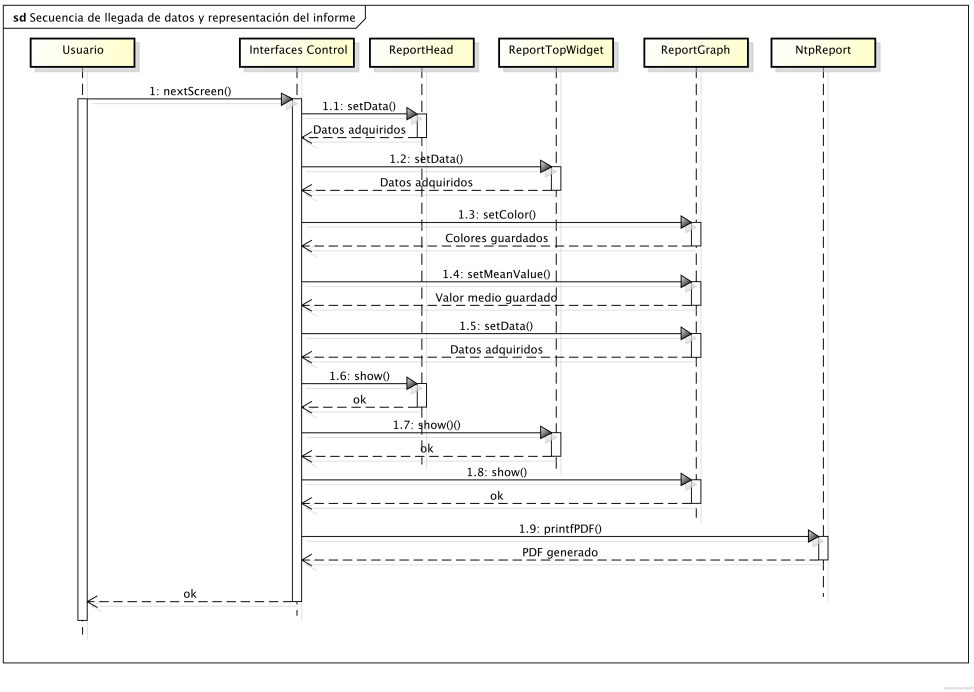


Figura 3.21: Diagrama de secuencia de llegada de datos y representación del informe.

El diagrama de secuencia correspondiente a la representación gráfica de *ReportGraph* se encuentra disponible en el anexo A, figura A.11.

Tras esta fase obtenemos un primer prototipo de la herramienta (Figura 3.22).

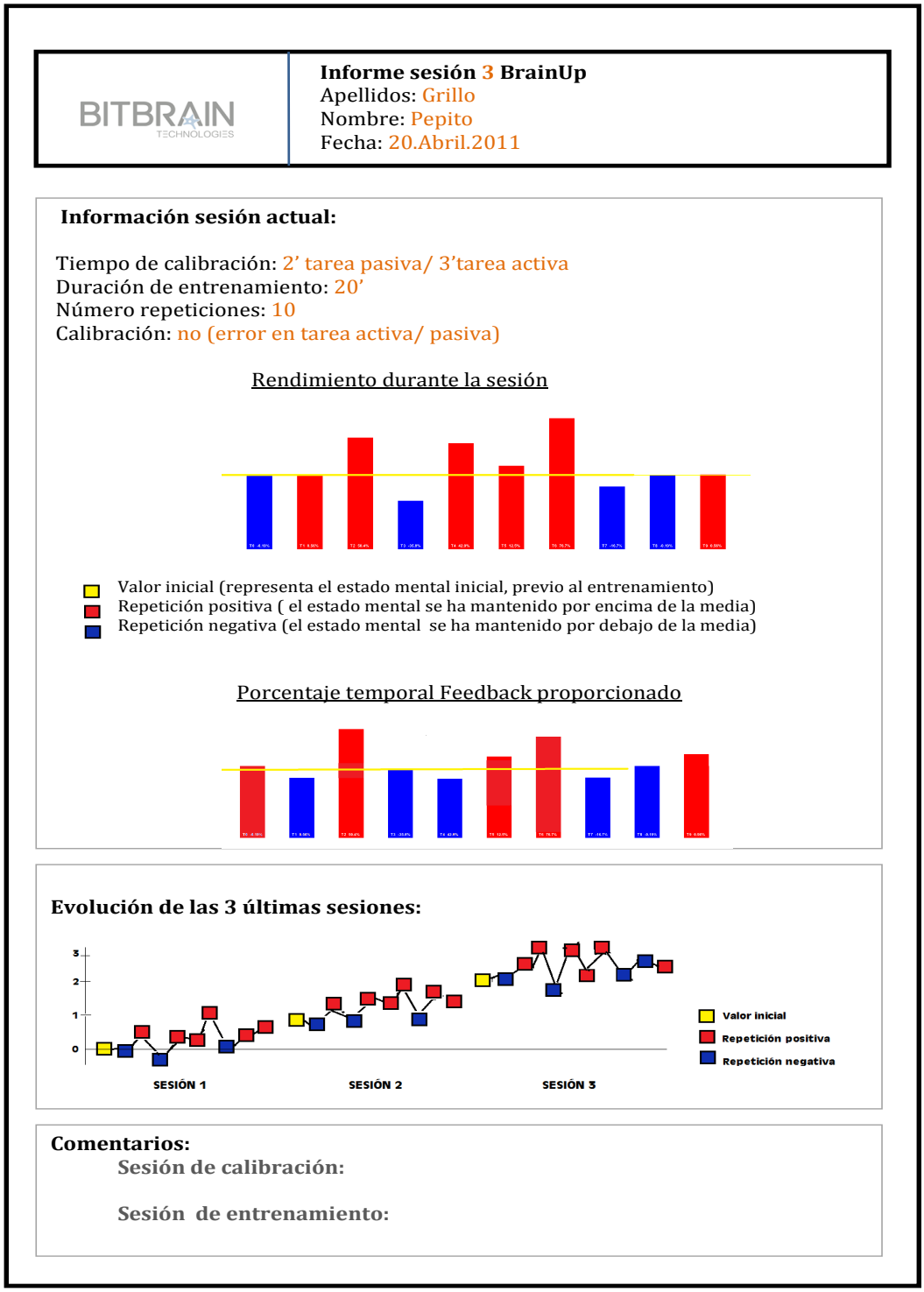


Figura 3.22: Prototipo del informe de resultados.

3.3.4. Implementación

Aquellas tareas relacionadas con la gestión y edición de los elementos a representar en el informe se desarrollaron con el framework *Qt*, mientras que las gráficas incluidas en el mismo se realizaron con el framework gráfico *Qwt*, compatible a todos los efectos con *Qt*.

Debido a la necesidad de imprimir el documento en formato *PDF* se utilizó una resolución de pantalla similar en dimensiones al *A4*, provocando que no fuera posible observar por pantalla el informe en su totalidad. A fin de que pudiera observarse el mismo antes de ser impreso se incluyó el objeto *QScrollArea*, permitiendo así desplazarse con libertad.

Así mismo hubo que definir los *ppp* (puntos por pulgada) a los que el documento sería impreso, siendo seleccionada una resolución de 300 *ppp*, por su excelente compromiso calidad-tiempo tanto en labores de creación y visualización, como de impresión.

3.3.5. Pruebas

Se realizaron pruebas con semántica diferencial a 10 sujetos, con el fin de conocer el grado de aceptación del informe, así como su calificación en materia de usabilidad, claridad y facilidad de uso, obteniendo los siguientes resultados:

Pregunta	Respuesta
	3 2 1 0 -1 -2 -3
En general el informe de resultados	Me gusta **2.4**
El informe me parece	Intuitivo **2.1**
La gráfica de rendimiento me parece	Claro **2.2**
La gráfica de tiempo me parece	Claro **2**
La gráfica de las tres últimas sesiones me parece	Claro **2.4**
En términos generales el funcionamiento me parece	Claro **2.3**

Tabla 3.10: Resultados globales de la evaluación del informe.

Se encuentran disponibles en el anexo de pruebas los resultados detallados para cada uno de los usuarios.

3.4. Plugin de visualización de actividad cerebral

3.4.1. Introducción

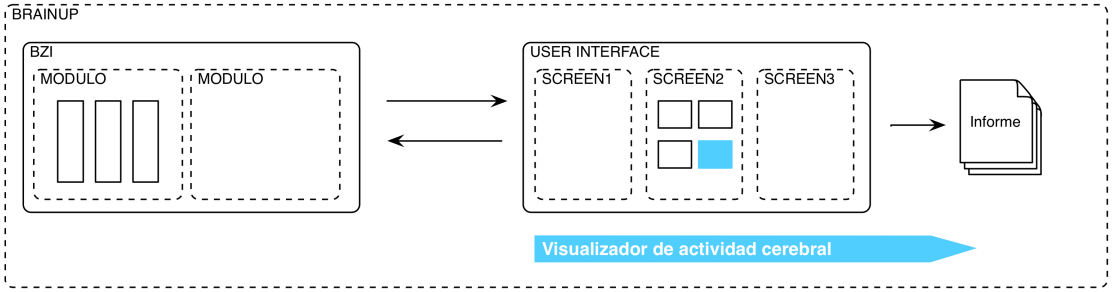


Figura 3.23: Visualizador de actividad.

En todo interfaz de usuario orientado a la usabilidad, deben existir tanto herramientas encargadas de la comprobación (representación visual del montaje, notificación modo texto) de manera intuitiva, como otras encargadas de la representación en tiempo real de cierto tipo de datos (EEG, FFT). En el caso que nos ocupa y como tarea dentro de este PFC, se ha desarrollado un plugin de visualización espacial, cuya misión es mostrar la actividad cerebral en tiempo real del usuario/paciente mediante un mapa de calor, donde el color rojo corresponderá a la cota superior y el azul a la inferior.

3.4.2. Análisis

La tabla 3.11 corresponde a la secuencia de acciones que satisfacen el caso de uso 3.24. El usuario ejecuta la interfaz donde se encuentra contenido el plugin de visualización, ésta procede a la creación del objeto visualizador y a la distribución de los datos pertinentes mientras se continúe adquiriendo señal o la interfaz se encuentre activa. Como resultado de esta secuencia el usuario observa la actividad correctamente.

La tabla 3.12 representa la extracción de requisitos funcionales y no funcionales realizada al caso de uso 3.11.

Nombre	Caso de uso 1
Actores que intervienen	Usuario/Terapeuta
Descripción	Visualización de la actividad cerebral.
Precondición	El terapeuta se encuentra en la interfaz de usuario. donde se encuentra el visualizador.
Secuencia de acciones	**1.** A través del plugin visualiza la actividad cerebral.
Resultados	El usuario/terapeuta ha visualizado la actividad cerebral.

Tabla 3.11: Caso de uso correspondiente a la visualización de la actividad cerebral.

Código	**Descripción**
RF-0	El plugin debe ofrecer un sistema de visualización espacial de la actividad cerebral.
RNF-0	Se debe poder incluir fácilmente en cualquier interfaz de usuario.
RNF-1	Se aproximarán el resto de elementos a visualizar conforme a los valores obtenidos del montaje.
RNF-2	Debe tener un tiempo de ejecución reducido y funcionar en tiempo real.

Tabla 3.12: Requisitos del plugin de visualización de actividad cerebral.

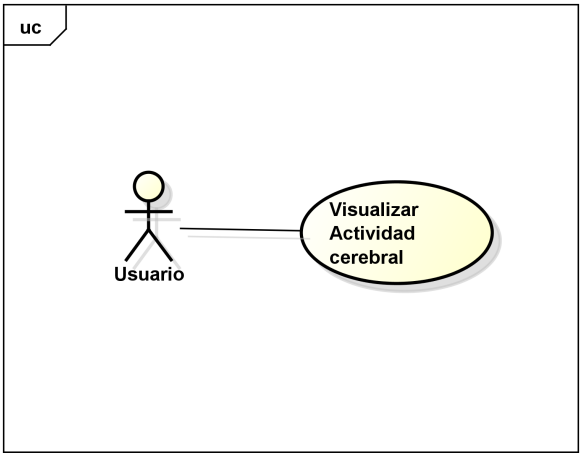


Figura 3.24: Gráfico de caso de uso correspondiente al Plugin de visualización de actividad cerebral.

3.4.3. Diseño

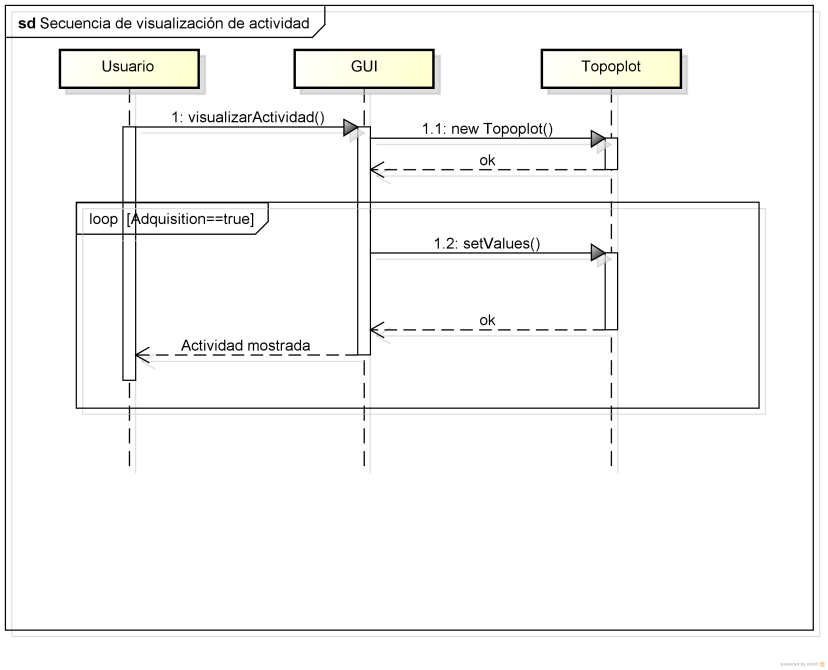


Figura 3.25: Diagrama de secuencia correspondiente la visualización de la actividad cerebral

La figura 3.25 corresponde a la secuencia de acciones que satisfacen el caso de uso 3.24. El usuario ejecuta la interfaz donde se encuentra contenido el plugin de visualización. Esta procede a la creación del objeto visualizador y a la distribución de los datos a representar, repitiéndose cada ciclo mientras se continúe adquiriendo señal o la interfaz se encuentre activa. Como resultado de esta secuencia el usuario visualiza la actividad correctamente.

La distribución de los datos de los electrodos se realiza mediante un método publico, que a su vez invoca al algoritmo de representación visual de la actividad. Este tipo de estructura confiere versatilidad al plugin de visualización, convirtiéndolo en una unidad independiente del contexto y permitiendo que sea instanciado en cualquiera de las interfaces de usuario disponibles.

La aproximación del resto de puntos a representar es obtenida mediante interpolación, en concreto mediante el método *inverse distance weighting* donde cada uno de los datos interpolados $u(x)$ responde al sumatorio del inverso de cada una de las distancias a los electrodos, entre el total de las mismas $w_i(x)$

$$u(x) = \sum_{i=0}^{N} \frac{w_i(x) u_i}{\sum_{j=0}^{N} w_j(x)}$$

donde

$$w_i(x) = \frac{1}{(x, x_i)^p}$$

.

La elección de esta técnica frente a otras se basa en dos aspectos importantes:

- **Disponibilidad:** La mayoría de los métodos de interpolación utilizados habitualmente (*Bilinear*,*Spline*,*Bezier*) tienen como precondición que la distribución espacial de los datos sea regular, sin embargo, los montajes no responden a una estructura de este tipo quedando reducido el abanico de métodos a unos pocos, muchos de ellos incompatibles con una ejecución en tiempo real (*Krigging*). Existen implementaciones *Bilinear*,*Spline* donde se convierte el montaje en un mallado regular, interpolando a posteriori, se desecharon estas modificaciones debido a la mas que destacable pérdida de precisión.

- **Eficiencia:** Se trata de un plugin que debe ejecutarse en tiempo real (aproximadamente 30 ms), luego necesita que el tiempo máximo de procesamiento durante la interpolación sea menor que éste. Esta restricción favorece la elección de *inverse distance weighting* frente a otros métodos con resultados similares pero un tiempo de procesamiento mucho mayor. Con el fin de agilizar su representación aún mas, se realizó una evaluación y almacenado de las distancias a los diferentes sensores ($w_i(x)$), evitando tener que computar de nuevo valores (distancia entre sensores) que permanecen constantes iteración tras iteración, y reduciendo su ciclo de ejecución de 210 ms a 2.18 ms.

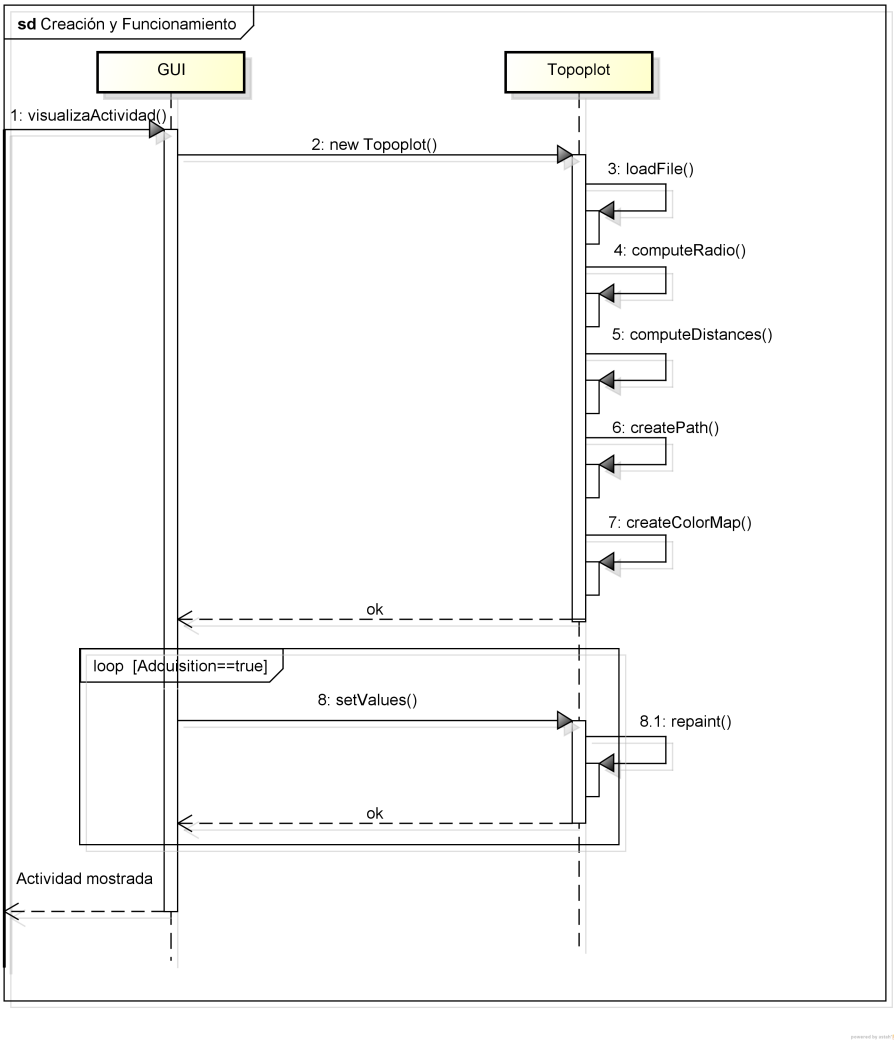


Figura 3.26: Diagrama de secuencia correspondiente a la creación del plugin de visualización de actividad cerebral.

Tras la ejecución del interfaz, se procede a la creación del visualizador (figura 3.26), en éste se cargará tanto la distribución de sensores presentes en el montaje (*loadFile*), como aquellos objetos donde se realizará la representación (*createPath*,*createColorMap*).

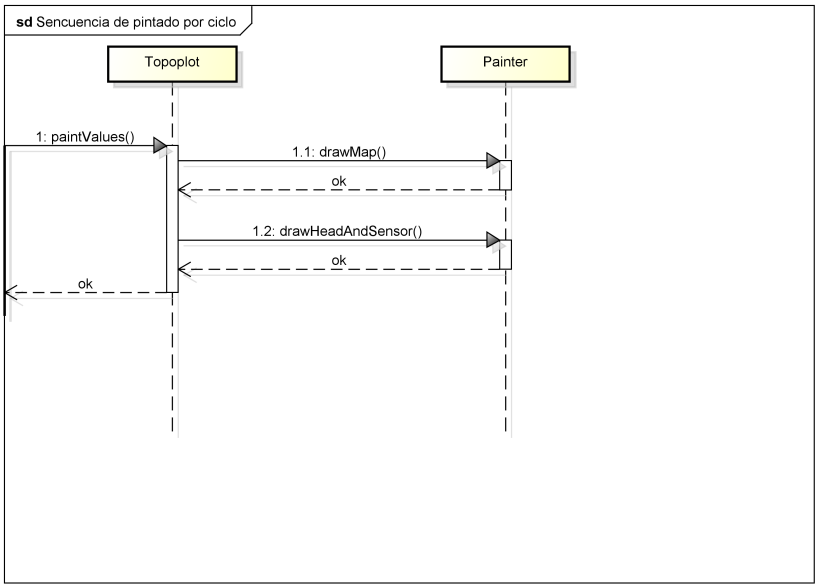


Figura 3.27: Diagrama de secuencia correspondiente al proceso de pintado del plugin de visualización de actividad cerebral.

Se representará la actividad cerebral tras cada adquisición pues funciona en tiempo real, en el diagrama de secuencia 3.27 se detalla que acciones se realizarán en cada ciclo de representación. Se procederá al pintado del montaje "base" (contorno + electrodos) para posteriormente proceder al pintado de los distintos valores interpolados.

Tras esta fase obtenemos un primer prototipo de la herramienta (Figura 3.8)

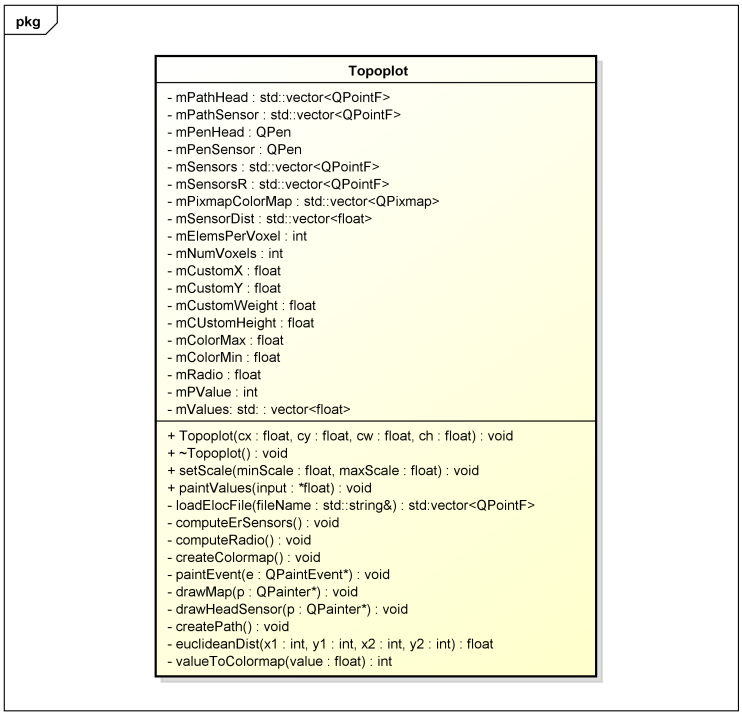


Figura 3.28: Diagrama de clases correspondiente al visualizador.

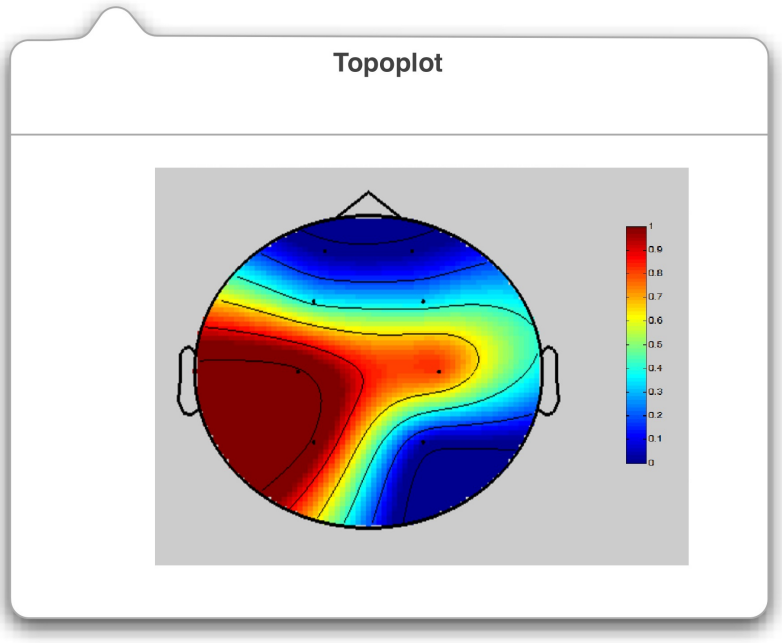


Figura 3.29: Prototipo del visualizador de actividad cerebral.

3.4.4. Implementación

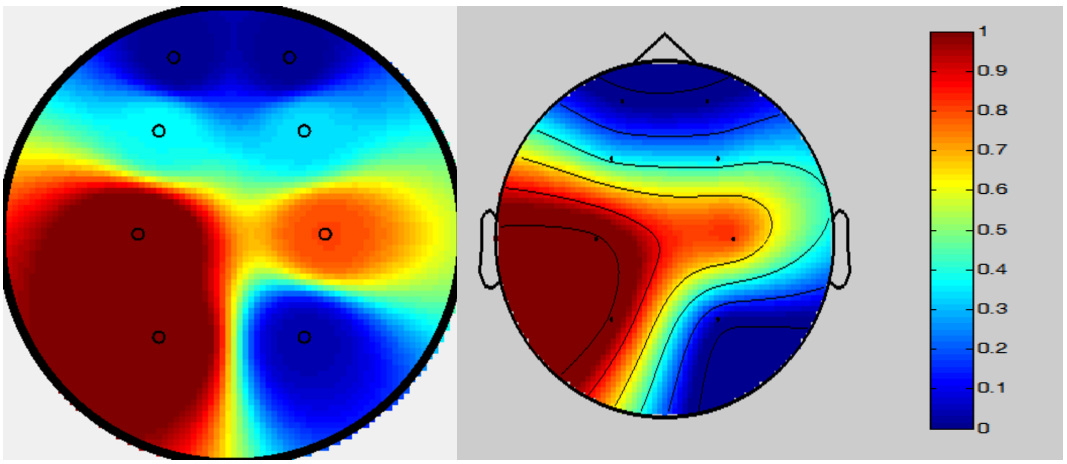
Procedemos a la implementación de la solución resultante de la fase de diseño, bajo el framework *Qt*. Se realizó un extenso proceso de documentación en busca de los elementos de representación mas eficientes disponibles dentro de Qt, tratando de minimizar las penalizaciones temporales producidas por una mala planificación del proceso de pintado.

La mayor parte de los elementos gráficos incluidos en BrainUp se encuentran desarrollados en *Qwt*, librería gráfica C++ compatible con *Qt*, sin embargo, la inexistencia de objetos similares a los requerimientos de usuario expuestos en la sección 3.4.2 provocó su rechazo en favor de una implementación en *Qt*.

3.4.5. Pruebas

Se realizaron diversas pruebas:

- **Pruebas de funcionamiento:** Realizadas entre nuestra solución y el visor distribuido por Matlab, se procesaron 5 ficheros de EEG con resultados similares, concluyendo que las ligeras diferencias encontradas entre ambas soluciones se debían a la utilización de diferentes algoritmos de interpolación, aunque como se observa en la figura 3.4.5 no afectaban a la correcta visualización de la actividad cerebral.

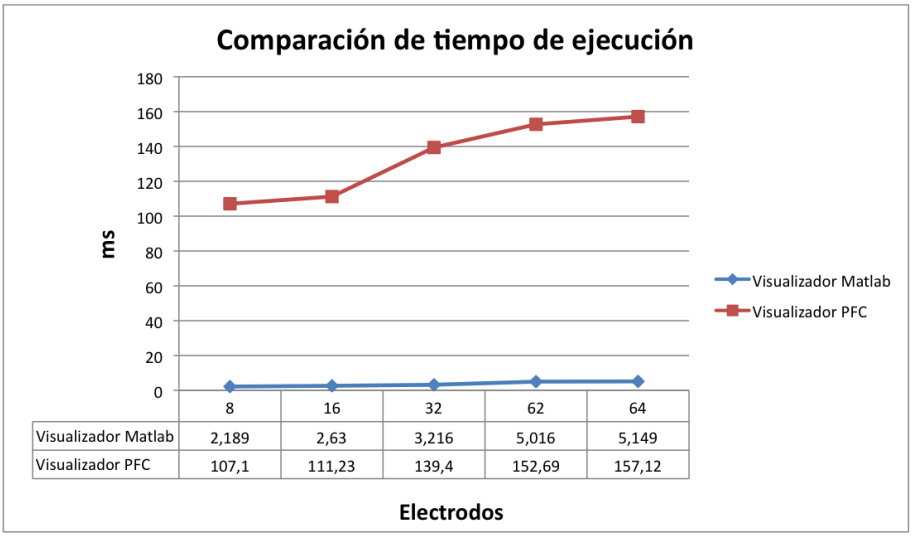


(a) Real (b) Matlab

Figura 3.30: Prueba con 8 electrodos

- **Pruebas de rendimiento:** Realizadas con diferentes configuraciones de electrodos (16 electrodos, 32 electrodos,62 electrodos y 64) comprobando que el ciclo de ejecución resultante es aceptable en todos los casos.

  Concluimos así que la implementación realizada en este PFC se mantiene dentro de unos parámetros aceptables, pues no sobrepasa en ninguno de los casos el tiempo



	8	16	32	62	64
Visualizador Matlab	2,189	2,63	3,216	5,016	5,149
Visualizador PFC	107,1	111,23	139,4	152,69	157,12

Figura 3.31: Comparación visualizador Matlab vs PFC.

de ciclo (30 *ms*) manteniéndose en un intervalo de 2-6 *ms*. La solución ofrecida por *Matlab* sin embargo posee un tiempo de ejecución mínimo de 107,1 *ms*.

El resto de pruebas de funcionamiento, y una prueba individual del visualizador, se encuentran presentes en el anexo C, sección C.4.

4. Localización del software

A la hora de desarrollar una solución informática de cualquier tipo, especialmente en fases tempranas de análisis o desarrollo, surgen inevitablemente cuestiones referidas al diseño, la eficiencia, o incluso a la usabilidad de la misma, sin embargo ¿Qué ocurre con la localización del software?, ¿Qué idioma acompaña por defecto a la aplicación? ¿Qué lenguas posibilitamos? y sobre todo ¿Cómo las incluimos?

Las limitaciones producidas por una incorrecta localización software pueden llegar a imposibilitar la utilización del mismo, disminuir su atractivo en ciertos entornos comerciales e incluso relegarla a un segundo plano frente a aplicaciones inferiores técnicamente, pero localizadas adecuadamente. En casos extremos una mala localización puede desembocar en una pésima o equívoca valoración del producto final(p.ej : Nissan Moco, Volkswagen Jetta).

Como tarea dentro de este PFC, se realizó la localización de BrainUp, atendiendo a su vez al requisito no funcional presente en todos los apartados anteriores (*"Debe estar disponible en diferentes idiomas"*). El idioma seleccionado por defecto para la aplicación es el inglés, se trata de una de las lenguas mas habladas y estudiadas del planeta, además de su ubicua presencia en los entornos de divulgación científica, y puesto que nuestro software se encuentra enmarcado dentro de un ámbito científico-técnico, convenimos que el inglés era el más adecuado a las necesidades del usuario final.

La localización y gestión de idiomas se realizó conforme al proceso descrito en la API de *Qt Qt Linguist*[24], gracias a este método conseguimos generalizar y desacoplar la localización software, haciéndola accesible incluso a personal no habituado a entornos de desarrollo software, como los lingüistas.

4. Localización del software

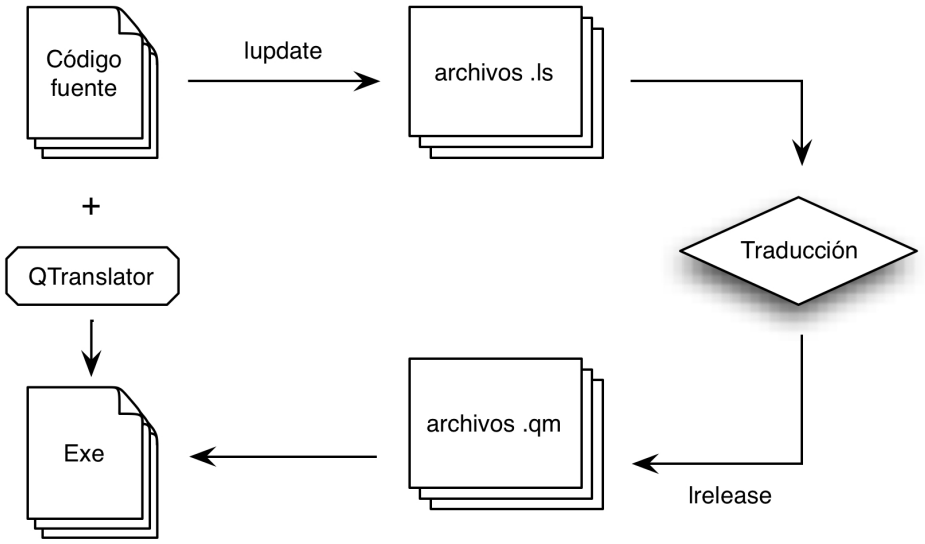


Figura 4.1: Esquema del proceso de localización.

El proceso como podrá observar en la figura 4.1 consta de las siguientes fases:

1. **Trabajo previo:** Dentro del documento de normas de programación, de obligado cumplimiento por parte de los ingenieros de BitBrain Technologies, incluimos una norma acerca de la necesaria utilización de la función "tr"previa cualquier cadena de texto que se desea mostrar por pantalla (p.ej. *tr("mitexto")*), permitiendo de este modo la correcta localización de cualquier aplicación generada ahora o en el futuro por la compañia.

2. **Modificación archivo .pro:** Este tipo de archivo es básico dentro de la creación de soluciones basadas en *Qt*, se trata de un *meta-makefile* en el que quedan definidas las librerías a incluir, las dependencias, así como el código fuente y cabeceras que se implementarán en el entorno de desarrollo elegido (en nuestro caso Visual Studio 2008), debemos modificarlo añadiendo los idiomas que deseamos generar y completar a posteriori, como por ejemplo:

   Translations:
   bUp-es_ES.ts
   bUp-fr_FR.ts
   bUp-de_DE.ts
   bUp-ja_JP.ts
   bUp-pt_PT.ts

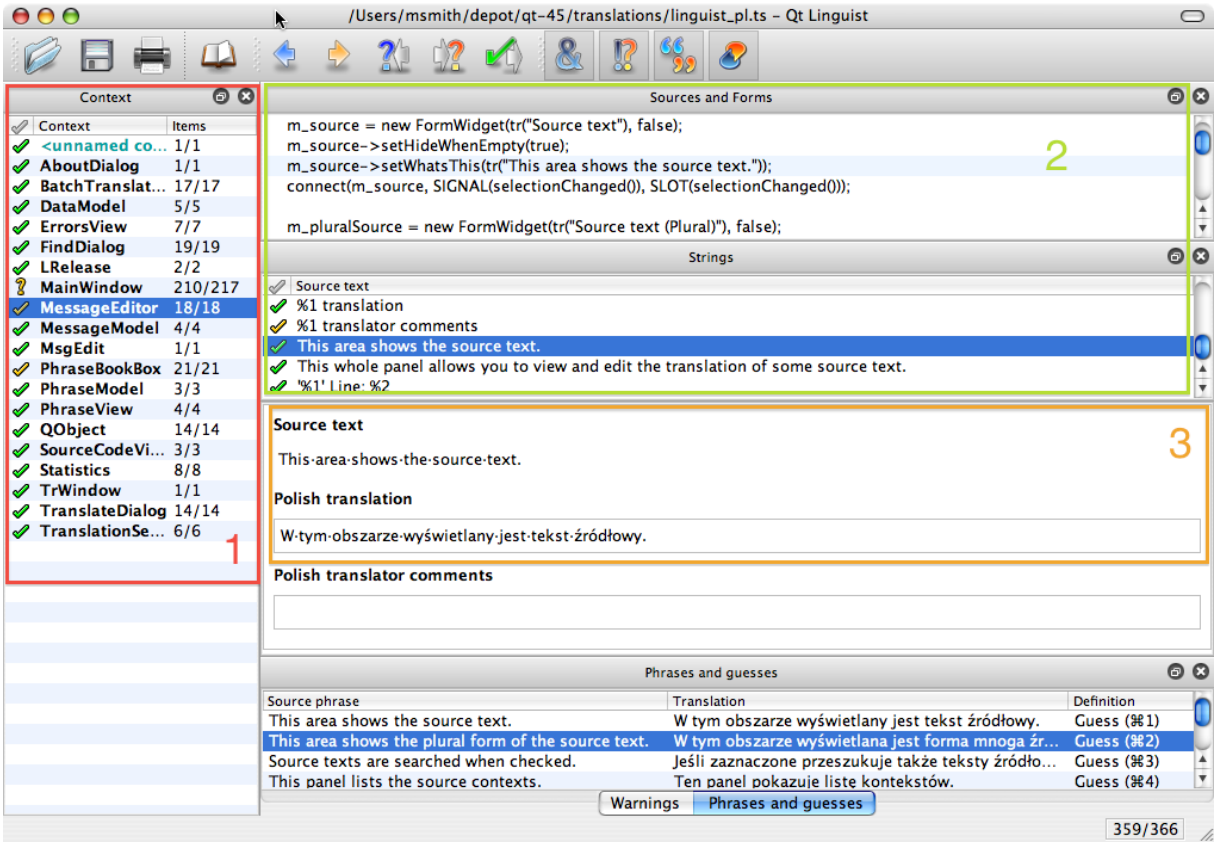


Figura 4.2: Aspecto del editor de lenguajes.

3. **Lupdate:** Invocación correspondiente a la API de Qt Linguist y realizada por línea de comando, su principal función es recorrer los diferentes archivos correspondientes al código fuente, en busca de cadenas de tipo *"tr(cadena)"* y sensibles de ser traducidas, generando los ficheros *.ts* adecuados a los idiomas definidos en el archivo *.pro*.

4. **Traducción:** Una vez generados los ficheros *.ts* para los distintos idiomas, procedemos a su traducción gracias a la herramienta de edición proporcionada por Qt, aunque también pueden ser modificados de manera directa, pues se trata simplemente de un fichero en formato *.xml*.

   Como se puede observar en la figura 4.2 el programa dispone de tres zonas principales:

   a) Zona 1: Selección de objeto cuyos elementos deseamos traducir.

   b) Zona 2: Selección de elemento a traducir dentro de un objeto.

   c) Zona 3:Realización de la traducción.

5. **Lrelease:** Invocación correspondiente a la API de Qt Linguist y realizada por línea de comando, su principal función es generar los diferentes archivos *.qm* correspon-

dientes a la traducción de las secuencias *tr("mitexto")*, estos serán cargados de manera dinámica por el ejecutable traduciendo el texto por pantalla.

Translations:

bUp-es_ES.qm

bUp-fr_FR.qm

bUp-de_DE.qm

bUp-ja_JP.qm

bUp-pt_PT.qm

6. **QTranslator:** Una vez confeccionado el fichero de traducción y generados los archivos *.qm*, debemos instanciar un objeto de tipo *QTranslator* en nuestro procedimiento principal, de manera que nuestra aplicación cargue automáticamente el idioma adecuado de acuerdo a nuestra configuración regional, en caso de que este no esté disponible, se hará uso del idioma por defecto.

   a) *QString locale = QLocale::system().name();* : Almacenamos gracias a esta invocación bajo qué código regional operamos (p.ej. es_ES fr_FR) el primero codifica la lengua en formato *ISO 639*, mientras el segundo codifica el país en *ISO 3166*, pudiendo así distinguir por ejemplo, entre inglés británico o americano (*en_GB en_US*).

   b) *QTranslator translator;* : Instanciamos un objeto de tipo *QTranslator*.

   c) *QString name = 'bUp'+ locale;* : Construimos una string en la que almacenaremos el idioma adecuado a nuestra región.

   d) *translator.load(name,"../")* : Cargamos el fichero correspondiente, en caso de no ser encontrado, se hará uso del idioma por defecto (en nuestro caso inglés).

   e) *app.installTranslator(translator);* : El traductor queda completamente operativo en nuestro ejecutable.

Conseguimos gracias a este método satisfacer los requisitos no funcionales antes expuestos y generalizar la localización de nuestra aplicación, de forma que la adición de un nuevo idioma a la misma no conlleve nuevos esfuerzos de implementación sobre el núcleo.

5. Instalador

5.1. Descripción

Tras varias iteraciones sobre el proceso unificado de desarrollo software descrito en capítulos anteriores desembocamos en la primera versión estable de la aplicación, sin embargo, quedan aún multitud de tareas a realizar hasta considerar a ésta como un producto terminado.

Tareas por ejemplo relativas al empaquetado y distribución de la aplicación, al método de obtención del hardware que la acompaña (amplificador operacional), a los manuales que deben servir de guía a usuarios noveles, o a su método de instalación y configuración.

En el caso que nos ocupa, y dentro de las tareas realizadas en este PFC se procedió al análisis diseño e implementación de un instalador para la aplicación BrainUp.

A pesar de que el framework sobre el que se implementa BrainUp es multiplataforma, se desarrolló el instalador únicamente para sistemas operativos Windows por tratarse del sistema en el que es comercializado BrainUp en primera instancia, teniendo como requisito mínimo Windows XP y realizando distinción entre las versiones de 32 y 64 bits, la motivación de esta será detallada en la sección 6.

Durante este proceso también se realizará la instalación del software correspondiente a terceros, necesario para la correcta ejecución de la aplicación, como por ejemplo:

1. **Drivers Gtec:** Drivers del fabricante necesarios al tratarse de un dispositivo de adquisición conectado a nuestro ordenador por USB.
2. **API Gtec:** Librerías utilizadas por BZI para la correcta adquisición de señal.
3. **Base de datos PostgreSQL:** Requerida para la gestión de datos relativos usuarios y terapia.

También se ejecutarán scripts encargados de la creación de una estructura de base de datos adecuada para la terapia, así como la inserción de las variables de entorno en el path del sistema, una por cada una de las librerías requeridas por la aplicación (*Qt,Qwt,Armadillo*).

Se implementó paralelamente al instalador, el desinstalador pertinente, aunque no se dotó a este último de estrategias de reparación o recuperación en caso de una instalación incompleta.

5.2. Análisis

Siguiendo la metodología expuesta en el capítulo 3 se definen dos casos de uso principales, iniciando así la fase de extracción de requisitos:

1. Instalación de la aplicación: El usuario procede a instalar y configurar BrainUp.

2. Desinstalación de la aplicación: El usuario procede a desinstalar BrainUp.

En las tablas 5.1 y 5.3 así como en las gráficas 5.1 y 5.3 pueden observarse los mismos.

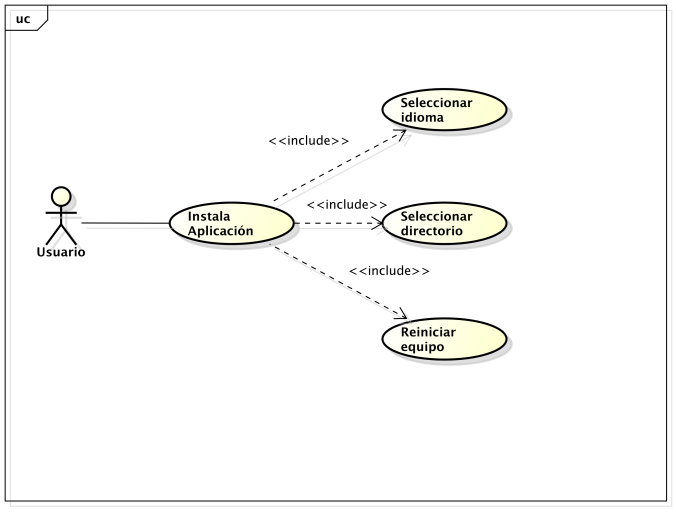


Figura 5.1: Diagrama de caso de uso de instalación.

Nombre	Caso de uso 1
Actores que intervienen	Usuario
Descripción	Instalación y configuración de la aplicación.
Precondición	El usuario posee el ejecutable de BrainUp.
Secuencia de acciones	**1.** Selecciona el idioma de instalación **2.** Acepta los términos de licencia. **3.** Selecciona el directorio destino de la aplicación. **4.** Finalmente el usuario acepta y se reinicia el equipo.
Resultados	Se ha instalado y configurado BrainUp correctamente.

Tabla 5.1: Caso de uso de instalación.

Nombre	Caso de uso 2
Actores que intervienen	Usuario
Descripción	Desinstalación de la aplicación.
Precondición	El usuario posee instalado BrainUp.
Secuencia de acciones	**1.** Selecciona el idioma de desinstalación **2.** Selecciona desintalar.
Resultados	Se ha desinstalado BrainUp correctamente.

Tabla 5.2: Caso de uso de desinstalación.

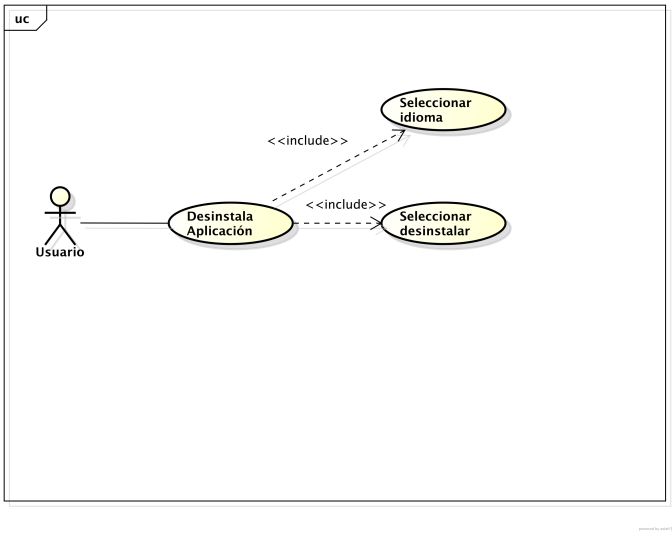


Figura 5.2: Diagrama de caso de uso de desinstalación.

Gracias a los casos de uso descritos anteriormente, sintetizamos los requisitos funcionales y no funcionales necesarios para el instalador.

Código	Descripción
RF-0	El instalador debe ofrecer selección de idioma.
RF-1	El instalador debe permitir la selección de la carpeta destino.
RF-2	El instalador debe obligar al reinicio del equipo tras la instalación.
RNF-0	Se debe detectar el sistema operativo destino de la aplicación copiando los archivos adecuados a cada versión.
RNF-1	Se debe realizar la instalación de componentes o tareas pertenecientes a terceros, de manera desatendida.
RNF-2	Debe ser usable e intuitivo.
RNF-3	Debe estar disponible en diferentes idiomas.

Tabla 5.3: Requisitos del instalador.

5.3. Diseño

Un diagrama de actividades es utilizado con el fin de modelar el comportamiento del sistema o describir como un sistema implementa su propia funcionalidad, cada diagrama representa una actividad, que a su vez puede estar formada por actividades más pequeñas, además están basados en redes de *petri*.

Mientras un diagrama de interacción muestra como los objetos gestionan los mensajes, uno de actividades muestra las operaciones ocurridas entre entidades de nuestro sistema, sirven para modelar la dinámica de un conjunto de objetos, el flujo de control de una operación, caso de uso, o bien un hilo de trabajo (*workflow*).

En la figura 5.3 podemos observar el diagrama de actividades correspondiente al instalador.

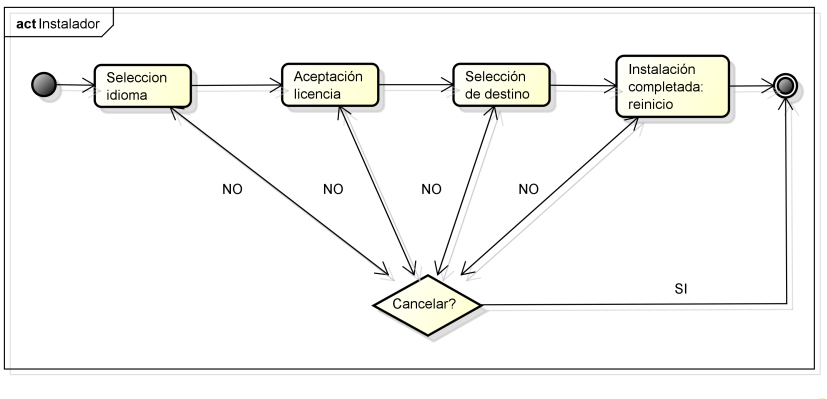


Figura 5.3: Diagrama de actividades del instalador.

Para finalizar el proceso de diseño se realizaron prototipos de las ventanas correspon-

dientes al proceso de instalación, incluidas en el anexo de desarrollo.

5.4. Implementación

La implementación fue realizada sobre *NSIS*[25], lenguaje de *script* de licencia open-source y desarrollado por NullSoft, creadores del afamado Winamp. La popularidad de éste ha crecido de forma exponencial en los últimos años, debido sobre todo a su versatilidad, el magnífico soporte ofrecido por una amplia comunidad de desarrolladores, y por tratarse de una alternativa libre y gratuita frente a otras como InstallShield de elevado coste.

Además de resultar una alternativa libre y gratuita, *NSIS* ofrece las siguientes funcionalidades:

1. Reducido tamaño: *NSIS* fue concebido para ser pequeño, rápido y eficiente, un instalador básico completamente funcional tendría un tamaño aproximado de 34 KB, muy inferior al resto de soluciones existentes.

2. Los ejecutables generados son compatibles con todas las versiones de Windows disponibles hasta la fecha, satisfaciendo así uno de los requerimientos de nuestro instalador.

3. Permite la ampliación de sus funcionalidades mediante invocaciones C, C++ o Delphi entre otros.

4. Al contrario que otras soluciones, genera ejecutables auto-contenidos, sin necesidad de extracción alguna previa a la instalación.

5. Posee soporte multilenguaje.

6. Como característica más destacada, las diferentes aportaciones realizadas por la comunidad de desarrolladores, ya sea en la creación de nuevos plugins o funcionalidades, como en la modificación del núcleo de lenguaje, dotándolo así de nuevas capacidades.

NSIS requiere definir previamente cada una de las páginas de las que constará el instalador, página de bienvenida, selección de ruta de destino, progreso de instalación, realizando una configuración de las mismas previa a su invocación (iconos a mostrar, texto por pantalla, colores).

Dentro de las soluciones suministradas por la comunidad de desarrollo, tres fueron de especial utilidad dentro de nuestra implementación:

- *LogicLib.nsh*[26] : Dota al lenguaje de sentencias "if-then-else", "case", o "unless", permitiendo así la utilización de estructuras de control y reduciendo ostensiblemente la complejidad del código fuente.
- *x64.nsh*[27] : Capaz de detectar sobre qué versión de sistema operativo nos encontramos (32-bit / 64-bit), facilitando así la elección de que librerías acompañarán al ejecutable.
- *EnvVarUpdate.nsh*[28] : Permite añadir o borrar variables de entorno al path del sistema operativo, ya sea a nivel de usuario o de administrador.

La invocación "ExecWait"[29] nos permite lanzar ejecutables/scripts ajenos a nuestro instalador, esperando a su correcta finalización para continuar, en el caso que nos ocupa, esta llamada fue usada para instalar el software relativo a la base de datos, ejecutar los scripts de configuración de la misma, e instalar la API y drivers de Gtec.

Ej. ExecWait msiexec /passive /i "INSTDIR/g.USBamp/driver/gUSBampDriver.msi"

La instalación de software ajeno a BitBrain Technologies se realiza de forma desatendida, dícese sin ningún tipo de interacción por parte usuario.

5.5. Pruebas

El instalador fue probado en diferentes sistemas operativos, tanto el proceso de instalación, como el de desintalación. Comprobando en ambos casos su correcta configuración y ejecución dependiendo de la versión instalada (Figura 5.4).

Sistema Operativo	Instalación
Windows XP 32-bit	**Correcta**.
Windows Vista 32-bit	**Correcta**.
Windows Vista 64-bit	**Correcta**.
Windows 7 32-bit	**Correcta**.
Windows 7 64-bit	**Correcta**.

Tabla 5.4: Resultados de instalación.

A su vez, se realizaron pruebas con semántica diferencial a 10 sujetos, evaluando el instalador en materia de usabilidad. (Figura 5.5).

Se encuentran disponibles en el anexo de pruebas, los resultados detallados de cada uno de los usuarios (Sección C.5).

Pregunta	Respuesta
	3 2 1 0 -1 -2 -3
En general el instalador	Me gusta **2.7**
El instalador me parece	Intuitivo **2.9**
La pantalla de selección de idioma me parece	Clara **2.4**
La aceptación de términos de licencia me parece	Clara **2.7**
La selección de carpeta destino me parece	Clara **2.8**
En términos generales el funcionamiento me parece	Claro **2.8**

Tabla 5.5: Resultados globales de la evaluación del instalador.

6. Aspectos relevantes

Como comentamos en el capítulo 5 existen aspectos relevantes dentro de la aplicación desarrollada que conviene sean definidos en detalle, pues su presencia paso desapercibida hasta fases tardías del desarrollo y ha motivado la adición o modificación de ciertos comportamientos dentro de BrainUp y de este PFC.

- **Armadillo:** Librería matemática fortran utilizada durante el procesado de la señal (FFT, cálculo filtro ICA), gran parte de sus beneficios se deben a las optimizaciones a bajo nivel realizadas sobre esta, como modificaciones en el acceso a memoria cache, o la gestión e indexación de elementos comunes en zonas contiguas de memoria, sin embargo, este tipo de optimizaciones requieren un nivel de compromiso "máquina" elevado, desembocando por ejemplo en la necesidad de distinguir en qué versión de sistema operativo nos encontramos.

  Debemos compatibilizar la librería de *Armadillo* distribuida con la versión de sistema operativo residente, ya que en caso contrario la aplicación no se ejecutará de manera correcta, la detección realizada durante el proceso de instalación responde a esta necesidad.

  Armadillo distribuye únicamente versiones pre-compiladas para 32-bit, se necesitó recompilar la librería para conseguir una versión de 64-bit compatible. Se desconoce en la actualidad la influencia de las diferentes arquitecturas hardware en su funcionamiento.

- **Resolución:** El framework de *Qt*, usado para la implementación de los interfaces de usuario, posee una potente metodología, encargada de la correcta colocación de los elementos en pantalla de manera dinamica, sin embargo, en muchas ocasiones adolece de una cierta incapacidad para distribuir correctamente los elementos en bajas resoluciones, o en ratios diferentes a las establecidos previamente (16:9, 4:3).

  Es por ello, por lo que se decidió limitar las interfaces de usuario a una serie de resoluciones establecidas, cubriendo la mayor parte de los tamaños existentes en el mercado y garantizando así su correcto visionado.

7. Conclusiones y trabajo futuro

- **Desarrollo de tareas:** Concluimos que todos las tareas de este PFC se han desarrollado siguiendo una metodología de proceso unificado, resultando en sistemas que satisfacen los requisitos fijados al inicio del mismo.

- **Eficiencia:** Se trataba de uno de los puntos importantes al inicio de este PFC, todas las herramientas, plugins o funcionalidades desarrolladas debían ser extremadamente eficientes debido a lo reducido del ciclo de adquisición de señal (30 ms). Gracias a la excelente documentación que acompaña al framework *Qt*, las soluciones resultantes gozan de un tiempo de ejecución aceptable.

- **Usabilidad y diseño:** Se ha puesto especial cuidado en aquellos aspectos relativos a la usabilidad y el diseño, pues se pretende que BrainUp sea usado en entornos sin conocimientos informáticos previos. Gracias al extenso proceso de prototipado realizado por BitBrain Technologies, los elementos incluidos en este PFC y la aplicación en general resultan muy usables así como atractivas al usuario.

- **Desarrollo software:** Se ha podido estar presente en todas las fases del desarrollo de un software comercial, desde los primeros análisis, hasta el empaquetado final, actúando sobre muchas de ellas con las tareas de este PFC. A su vez, se ha podido comprobar el correcto funcionamiento de las distintas funcionalidades implementadas en este proyecto en entornos profesionales (ensayos clínicos, demos, experimentos con psicólogos) verificando que satisfacen en todo momento los requerimientos del usuario final.

7.1. Trabajo futuro

1. **Herramienta de comprobación de defectos de montaje:** En la actualidad adopta una postura meramente informativa, sin embargo futuras versiones requerirán, por ejemplo, eliminar electrodos, o definir sensores esenciales (permitiendo continuar la ejecución con errores en electrodos no-esenciales). A pesar de que estos cambios han sido tenidos en cuenta en la implementación realizada, se debería proceder a su adaptación y al desarrollo de las nuevas funcionalidades.

2. **Unidad de detección de defectos de montaje:** No todos los defectos o errores presentes en un montaje poseen las mismas características, ya sea en términos de amplitud, frecuencia o rango temporal, luego su detección no conlleva el mismo tipo de procesamiento. Como futura tarea queda la adición de nuevos métodos de detección de defectos de montaje.

3. **Plugin de visualización de actividad cerebral:** Se trata de un plugin altamente optimizado, la visualización en tiempo real de la actividad cerebral conlleva el interpolado y representación de gran cantidad de puntos en cada uno de los ciclos de adquisición (30 ms), a tal efecto y como ya se ha comentado en secciones anteriores, se fijó el tamaño del mismo con el fin de pre-calcular los coeficientes de interpolación, disminuyendo así en gran medida el tiempo de cálculo y permitiendo su ejecución en un tiempo estimado de entre 2 y 6 ms.

   Es esta extrema especialización la que lo convierte en un plugin rígido para ciertos interfaces de usuario, quedando pues como trabajo futuro la implementación de una solución intermedia, donde se sacrifique cierto tiempo de ejecución en pro de una mayor flexibilidad.

4. **Informe:** Como comentamos en el capítulo 2 las terapias basadas en neurofeedback son el resultado de un intenso trabajo multidisciplinar, y como resultado, los avances realizados en cada una de ellas influyen enormemente sobre el resto. Quedaría pues como trabajo en un futuro, la adaptación del report a estas nuevas exigencias, inclusive su reimplementación.

5. **Instalador:** En su caso, futuras versiones deben implementarse sobre la estructura proporcionada por "Modern User Interface"[30] correspondiente también a NSIS, pero más cercano visualmente a los interfaces modernos. Se trata entonces, de modificaciones en el aspecto y no en la estructura o funcionamiento del instalador.

Bibliografía

[1] J.R.Wolpaw, N.Birbaumer, D.J.McFarland, G.Pfurtscheller, and T.M.Vaughan. Brain-computer interfaces for communication and control. *Clinical Neurophysiology.*, 113(6), 2002.

[2] T.H.Budzynski, H.K.Budzynski, J.R.Evans, and A.Abarbanel. *Introduction to Quantitative EEG and Neurofeedback, Second Edition: Advanced Theory and Applications.* Academic Press, 1999.

[3] C.Escolano, J.Antelis, and J.Mínguez. Human Brain-Teleoperated Robot between Remote Places. *IEEE International Conference on Robotics and Automation.*, September 2009.

[4] I.Iturrate, J.Antelis, A.Kübler, and J.Mínguez. Non-Invasive Brain-Actuated Wheelchair based on a P300 Neurophysiological Protocol and Automated Navigation. *IEEE Transaction on Robotics*, June 2009.

[5] T.Elbert, N.Birbaumer, P.Wolf, A.Duchting-Roth, M.Reker, I.Daum, W.Lutzenberger, and J.Dichgans. Cortical sefl-regulation in patients with epilepsies. *Epilepsy res*, 14:63–72, 1993.

[6] J.N. Demos. *Getting started with neurofeedback.* WW Norton, 2005.

[7] H.Gevensleben, B.Holl, and B.Albrecht. Is neurofeedback an efficacious treatment for ADHD? A randomised controlled clinical trial. *Journal of Child Psychology and Psychiatry and Allied Disciplines.*, 50(7), 2009.

[8] M.Abikoff. Cognitive training in ADHD children: less to it than meets the eye. *Journal of Learning Disabilities*, 24:205–209, 1991.

[9] U.Strehl. Self-regulation of Slow Cortical Potentials: A New Treatment for Children With Attention-Deficit/Hyperactivity Disorder. *Pediatrics*, 2006.

[10] E.G.Peniston and P.J.Kulkosky. Alpha-theta brainwave training and beta-endorphin levels in alcoholics. *Alcolism: Clinical and Experimental Research*, 13:271–279, 2007.

[11] Actualidad Económica. Sistema para hacer gimnasia mental en pacientes de fibromialgia y depresión. http://bitbrain.es/wp-content/uploads/2011/09/ActualidadEconomicaBBT.pdf, 2011.

[12] Neurosky. Brainwave sensors for everybody. http://www.neurosky.com/, 2010.

[13] B. Hamadicharef, Xu Mufeng, and S.Aditya. Brain-Computer Interface (BCI) Based Musical Composition. *Cyberworlds (CW), 2010 International Conference on* , pages 282–286, 2010.

[14] Neurowear. Nekomimi. http://neurowear.com/, 2011.

[15] C.Escolano, M.Aguilar, and J.Mínguez. Effects of Upper Alpha Neurofeedback Training on Working Memory Performance and on Electrophysiology. *33 rd Annual International IEEE EMBS Conference*, April 2011.

[16] B. Zoefel, R.J.Huster, and Christoph S.Herrmann. Neurofeedback training of the upper alpha frequency band in EEG improves cognitive performance. *NeuroImage*, August 2010.

[17] S. Hanslmayr, P. Sauseng, M. Doppelmayr, M. Schabus, and W. Klimesch. Increasing individual upper alpha power by neurofeedback improves cognitive performance in human subjects. *Applied Psychophysiology and Biofeedback*, 2005.

[18] W. Klimesch. EEG alpha and theta oscillations reflect cognitive and memory performance: a review and analysis. *Brain Research Reviews*, 1999.

[19] Nokia. Qt libraries. http:/qt.nokia.com, 1992.

[20] Nokia. Qpainterpath library. http://doc.qt.nokia.com/latest/qpainterpath.html, 1992.

[21] Nokia. Qpainter library. http://doc.qt.nokia.com/stable/qpainter.html, 1992.

[22] Conrad Sanderson. Armadillo libraries. http://arma.sourceforge.net/, 2007.

[23] Uwe Rathmann. Qwt libraries. http://qwt.sourceforge.net/, 1997.

[24] Nokia. Qt linguist. http://doc.qt.nokia.com/latest/linguist-manual.html, 1992.

[25] NullSoft. Nsis reference. http://nsis.sourceforge.net/Main_Page.html, 2001.

[26] Dselkirk and Eccles. Logiclib plugin. http://nsis.sourceforge.net/LogicLib.html, 2004.

[27] NullSoft. x64 plugin. http://nsis.sourceforge.net/Include/x64.nsh, 2004.

[28] NullSoft. Environment path manipulation plugin. http://nsis.sourceforge.net/Path_Manipulation.html, 2004.

[29] NullSoft. Script reference. http://nsis.sourceforge.net/Docs/Chapter4.html, 2004.

[30] NullSoft. Nsis mui reference. http://nsis.sourceforge.net/Docs/Modern%20UI/Readme.html, 2007.