UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

CRET

Can-Bus Reverse Engineering Toolkit
Documentación Técnica

Presentado por Adrián Marcos Batlle
en Universidad de Burgos — 8 de febrero de 2019
Tutor: Álvar Arnaiz González y César Represa Pérez

Índice general

Índice general I

Índice de figuras III

Índice de tablas V

Apéndice A Plan de Proyecto Software 1
A.1. Introducción 1
A.2. Planificación temporal 2
A.3. Estudio de viabilidad 5

Apéndice B Especificación de Requisitos 9
B.1. Introducción 9
B.2. Objetivos generales 9
B.3. Catalogo de requisitos 9
B.4. Especificación de requisitos 12

Apéndice C Especificación de diseño 23
C.1. Introducción 23
C.2. Diseño de datos 23
C.3. Diseño procedimental 24
C.4. Diseño arquitectónico 25
C.5. Diseño de interfaces 30

Apéndice D Documentación técnica de programación 33
D.1. Introducción 33
D.2. Estructura de directorios 33

D.3. Manual del programador 34
D.4. Compilación, instalación y ejecución del proyecto 37
D.5. Producción del *hardware* 37

Apéndice E Documentación de usuario **45**
E.1. Introducción 45
E.2. Requisitos de usuarios 45
E.3. Instalación 45
E.4. Manual del usuario 45

Índice de figuras

A.1. *Sprint* 0 2
A.2. *Sprint* 1 3
A.3. Resumen de horas dedicadas al proyecto. 5

B.1. Diagrama UML general de los casos de uso. 13

C.1. Diagrama relacional de la base de datos 24
C.2. Diagrama de secuencia de la aplicación. 25
C.3. Modelo MVC 26
C.4. Arquitectura general del proyecto. 27
C.5. Paquetes de la aplicación 27
C.6. Diagrama de clases y paquetes de la aplicación. 28
C.7. Diagrama de clases y paquetes de la interfaz gráfica. 29
C.8. Diagrama UML de las clases general de la aplicación. 30
C.9. Diagrama UML de las clases dedicadas a la comunicación con el bus CAN. 30
C.10. Diagrama de navegabilidad 31

D.1. Comando Git sobre WSL. 35
D.2. SceneBuilder 10.0 35
D.3. Ejemplo del diseño de uno de los elementos en el esquema. . . . 38
D.4. Icono para asignar las huellas al esquema. 38
D.5. Relación entre los componentes y su *footprint*. 39
D.6. Icono para generar el fichero *netlist*. 39
D.7. Diseño inicial de la *PCB* 40
D.8. Diseño final de la *PCB*. 40
D.9. *Pinout* del *PICKit3*. 42
D.10. Cargando *firmware* en el PIC. 42

D.11.Configuración avanzada de *MPLAB*. 43
D.12.Salida del comando *mphidflash*. 44

Índice de tablas

A.1. Horas dedicadas al proyecto. 4
A.2. Costes de personal. 5
A.3. Costes de personal. 6
A.4. Costes totales. 6
A.5. Dependencias del proyecto y sus licencias 7

B.1. Caso de uso 1: Gestión de proyectos 13
B.2. Caso de uso 2: Listar proyectos 14
B.3. Caso de uso 3: Crear proyecto 14
B.4. Caso de uso 4: Eliminar proyecto 15
B.5. Caso de uso 5: Abrir proyecto 15
B.6. Caso de uso 6: Cerrar proyecto 16
B.7. Caso de uso 7: Importar proyecto 17
B.8. Caso de uso 8: Exportar proyecto 18
B.9. Caso de uso 9: Obtención de puertos serie disponibles 18
B.10. Caso de uso 10: Configuración de la interfaz CAN 19
B.11. Caso de uso 11: Etiquetado de datos 20
B.12. Caso de uso 12: Monitorización de los datos 21

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Una de las fases más importantes de cualquier proyecto es la planificación. En esta fase se estima el tiempo y los requisitos necesarios para la realización del proyecto. Para ello es necesaria una idea clara de lo que se quiere realizar de manera que todas las partes que componen el proyecto puedan ser analizadas individualmente.

Podemos dividir este anexo en dos apartados más pequeños:

- Planificación temporal.
- Estudio de viabilidad.

En la primera fase se realiza una planificación de los tiempos esperados. Se especifican los tiempos necesarios para cada una de las partes que componen el proyecto, a la vez que una fecha de inicio y de fin de las mismas.

La segunda fase se centra en un estudio de la viabilidad del proyecto. En esta fase se pueden diferenciar dos partes:

- **Viabilidad económica:** Estimación de los costes y beneficios del proyecto.
- **Viabilidad legal:** Regulaciones legales que pueden afectar al proyecto. En este caso serían la Ley de Protección de Datos y las licencias del software.

A.2. Planificación temporal

Para realizar una planificación del proyecto adecuada, se decidió aplicar *Scrum* (una metodología ágil de desarrollo, explicada en la memoria).

Para ello, se dividió el trabajo en *sprints* los cuales iban siendo planificados según se terminaba el anterior. La duración media de estos era de una semana de trabajo.

En cada uno de los *sprint* se generan una serie de tareas las cuales tienen que ser realizadas en ese intervalo de tiempo. Para ello se utiliza la plataforma *Zenhub*, en la cual las *issues* hacen de tareas, y los *Milestones* de *Sprints*.

Sprint 0

Este primer *sprint* fue el más complicado de planificar. Estaba incluido el desarrollo del *hardware* el cual no sabíamos seguro si se conseguirían los conocimientos necesarios para diseñarlo, ni si después funcionaría correctamente. Además, el desarrollo de la aplicación dependía del resultado de esta parte ya que iba a basarse en ello.

Al final, sobrepasando un poco el tiempo planificado, se consiguió poner el *hardware* en marcha, con lo que se podía comenzar con el desarrollo del *software*.

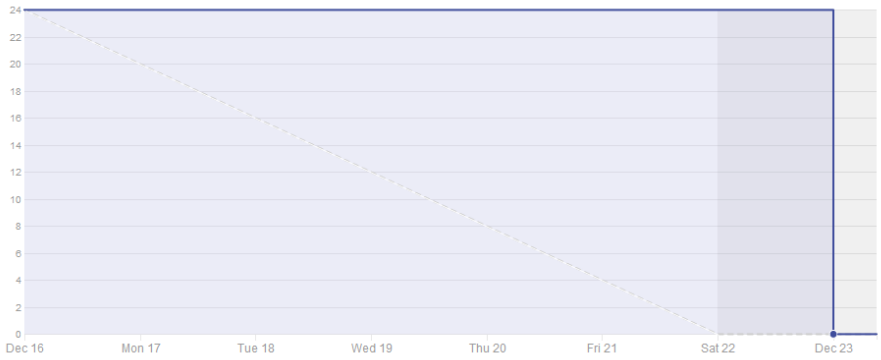

Figura A.1: *Sprint* 0

Sprint 1

Los objetivos de este *sprint* fueron asentar la idea general de la aplicación a desarrollar, su estructura principal y funcionalidades.

Además se comenzó a diseñar la estructura de la base de datos y el acceso al *hardware* a través de la comunicación con los puertos serie.

Las horas de trabajo fueron sobrepasadas sobre lo planificado en el *Sprint*.

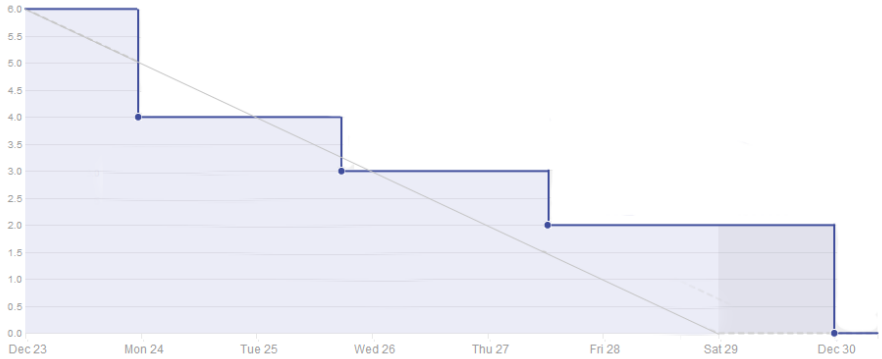

Figura A.2: *Sprint* 1

Sprint 2

Los objetivos de este *sprint* fueron comenzar con el desarrollo de la interfaz gráfica. Para ello era necesario la creación de gráficas de forma dinámica. Estas serían utilizadas para monitorizar los datos que fluían por el bus CAN en tiempo real.

Esta fase llevó un poco más tiempo de lo esperado.

Sprint 3

En este *sprint* se planteó la posibilidad de poder importar y exportar un proyecto de la aplicación. Para ello, se decidió usar ficheros JSON, los cuales contendrían todos los datos del proyecto.

En este punto se planteó la posibilidad de realizar una parte de la aplicación sobre una tecnología web. La idea era generar los ficheros JSON desde la aplicación Java para posteriormente poder importarlos a la web para su monitorización desde la misma.

Sprint 4

El objetivo de este *sprint* era la generación de un gestor de proyectos interno en la aplicación. Para ello era necesario diseñar las funcionalidades para la creación de un nuevo proyecto, la asignación de valores identificados al mismo, así como sus opciones de guardar y eliminar de la base de datos.

Sprint 5

En este *sprint* se desarrolló la interacción con la base de datos, tanto para obtener datos de ella como para recuperarlos. Además se realizó un tratado de las posibles excepciones que puedan surgir por problemas internos de la aplicación.

Sprint 6

En este *sprint* se centraron los esfuerzos en tener un borrador de la memoria y los anexos. Además, el desarrollo de la aplicación continuaba añadiendo opciones como la de ver los datos de una ID y un *byte* concretos, en tiempo real. Esta opción estaría disponible para todos los datos mostrados por pantalla.

Sprint 7

En este último *sprint* se trató de concluir con la documentación y terminar de pulir las pequeñas características que quedaban pendientes del proyecto. También se centraron esfuerzos en concluir la documentación y los vídeos de prueba de la aplicación.

Resumen

En la siguiente tabla se muestran los tiempos dedicados a cada uno de los distintos tipos de tareas:

Categoría	Tiempo (horas)
Documentation	50
Features	150
Research	20
Bug Fix	20
TOTAL	240

Tabla A.1: Horas dedicadas al proyecto.

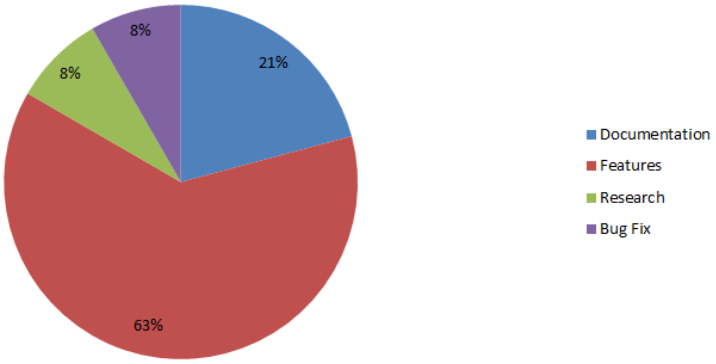

Figura A.3: Resumen de horas dedicadas al proyecto.

A.3. Estudio de viabilidad

En este punto se van a analizar por separado la viabilidad económica y la viabilidad legal del proyecto.

Viabilidad económica

Costes

Los costes van a ser desglosados en dos partes:

Costes de personal:

El proyecto ha sido desarrollado por una sola persona, empleada a tiempo completo durante tres meses. Considerando los siguientes valores:

Concepto	**Coste**
Salario mensual neto	1008,15 €
IRPF (9,64 %)	115,68 €
SS (30 %)	360 €
Salario mensual bruto	1200 €
Total tres meses	3600 €

Tabla A.2: Costes de personal.

Costes de *hardware*:

En este apartado se revisan los costes del *hardware* desarrollado en el proyecto.

Concepto	**Coste**
PCB (10 prototipos)	1,75 €
Transporte	7,11 €
Componentes (para 2 uds)	51,90 €
Aduanas	15 €
Soldador	15 €
Estaño	5 €
Flux	3 €
Total	98,76 €

Tabla A.3: Costes de personal.

Costes totales:

Concepto	**Coste**
Personal	3600 €
Hardware	98,76 €
Total	3698,76 €

Tabla A.4: Costes totales.

Teniendo en cuenta los datos anteriores, para hacer rentable el proyecto con un periodo de tiempo de un año, sería necesario vender tanto el *hardware* como licencias de la aplicación.

Viabilidad legal

A continuación se exponen todos los temas relacionados con las licencias del software y *hardware* de proyecto, así como de la documentación e imágenes.

Tabla de licencias de las dependencias utilizadas:

Dependencia	Versión	Descripción	Licencia
Java	8.0.191	JDK	GPL
USBtinLib	1.2.0	Librería para la conexión con el *hardware*.	GPL

Dependencia	Versión	Descripción	Licencia
JSSC	2.8.0	Librería para la búsqueda de puertos serie.	LGPL
sqlite-jdbc	3.23.1	Librería para realizar la conexión con la base de datos.	Apache 2.0
json	1.6	Librería para la importación y exportación de ficheros JSON.	JSON

Tabla A.5: Dependencias del proyecto y sus licencias

Apéndice B

Especificación de Requisitos

B.1. Introducción

A continuación se realiza la especificación de requisitos que define el comportamiento del sistema desarrollado en el proyecto.

B.2. Objetivos generales

Los objetivos generales del proyecto son los siguientes:

- Desarrollar una aplicación para el análisis y monitorización de los datos que fluyen por el bus CAN.
- Desarrollo de un *hardware* libre el cual permita conectarse y monitorizar dos buses de datos de forma simultánea.

B.3. Catalogo de requisitos

En este apartado se describen los requisitos específicos del proyecto, divididos en funcionales y no funcionales.

Requisitos funcionales

- **RF-1: Gestor de proyectos:** La aplicación debe de ser capaz de gestionar proyectos.

- **RF-1.1: Crear proyecto:** El usuario debe de poder crear un nuevo proyecto con un nombre específico.
  - **RF-1.1.1: Añadir datos al proyecto:** El usuario debe de poder añadir los datos identificados al proyecto.
- **RF-1.2: Listar proyectos:** El usuario debe de poder listar los proyectos existentes en la aplicación.
- **RF-1.3: Eliminar proyectos:** El usuario debe de poder eliminar los proyectos existentes en la aplicación.
- **RF-1.3.1: Confirmar la eliminación:** El usuario debe de poder confirmar si desea eliminar el proyecto.
- **RF-1.4: Abrir proyecto:** El usuario debe de poder abrir un proyecto existente en la aplicación.
- **RF-1.5: Cerrar proyecto:** El usuario debe de poder cerrar un proyecto que esté abierto.
- **RF-1.6: Importar proyecto:** El usuario debe ser capaz de importar un proyecto a la aplicación.
- **RF-1.7: Exportar proyecto:** El usuario debe ser capaz de exportar un proyecto de la aplicación.

- **RF-2: Configuración de las interfaces:** El usuario debe de poder configurar las interfaces CAN correspondientes.
  - **RF-2.1: Obtención de los puertos serie:** El usuario debe de poder obtener un listado de los puertos serie disponibles en el equipo que ejecuta la aplicación.
  - **RF-2.2: Configuración de velocidad:** El usuario debe de ser capaz de configurar la velocidad con la que una interfaz va a ser conectada.
  - **RF-2.3: Configuración del modo:** El usuario debe de ser capaz de configurar el modo con el que quiere conectarse al bus CAN.

- **RF-3: Configuración del análisis:** El usuario debe de ser capaz de configurar el tipo de análisis que quiere realizar.
  - **RF-3.1: Ignorar *bytes* con valor cero:** El usuario debe ser capaz de seleccionar si desea ignorar los *bytes* que no envíen ningún valor por el bus.

        - **RF-3.2: Separar los *bytes* en dos partes:** El usuario debe de ser capaz de separar los *bytes* en dos partes para mejorar el análisis si fuera necesario.

- **RF-4: Etiquetado de datos:** El usuario debe de ser capaz de etiquetar los datos una vez identificados.
    - **RF-4.1: Monitorización de los datos:** El usuario debe de ser capaz de monitorizar los datos una vez han sido etiquetados.

Requisitos no funcionales

- **RNF-1: Monitorización:** La aplicación debe monitorizar correctamente los datos que están siendo analizados siempre que se esté conectado al bus.
- **RNF-2: Rendimiento:** La aplicación debe funcionar fluidamente, sin que la interfaz gráfica se quede bloqueada.
- **RNF-3: Escalabilidad:** La aplicación debe permitir la incorporación de módulos de forma sencilla.
- **RNF-4: Usabilidad:** La aplicación debe ser fácil de utilizar, intuitiva y con una estructura clara.

B.4. Especificación de requisitos

En esta sección se exponen los casos de uso de la aplicación.

Actores

El único actor del sistema será la persona que maneja la aplicación y que identificará las señales importantes.

Casos de uso

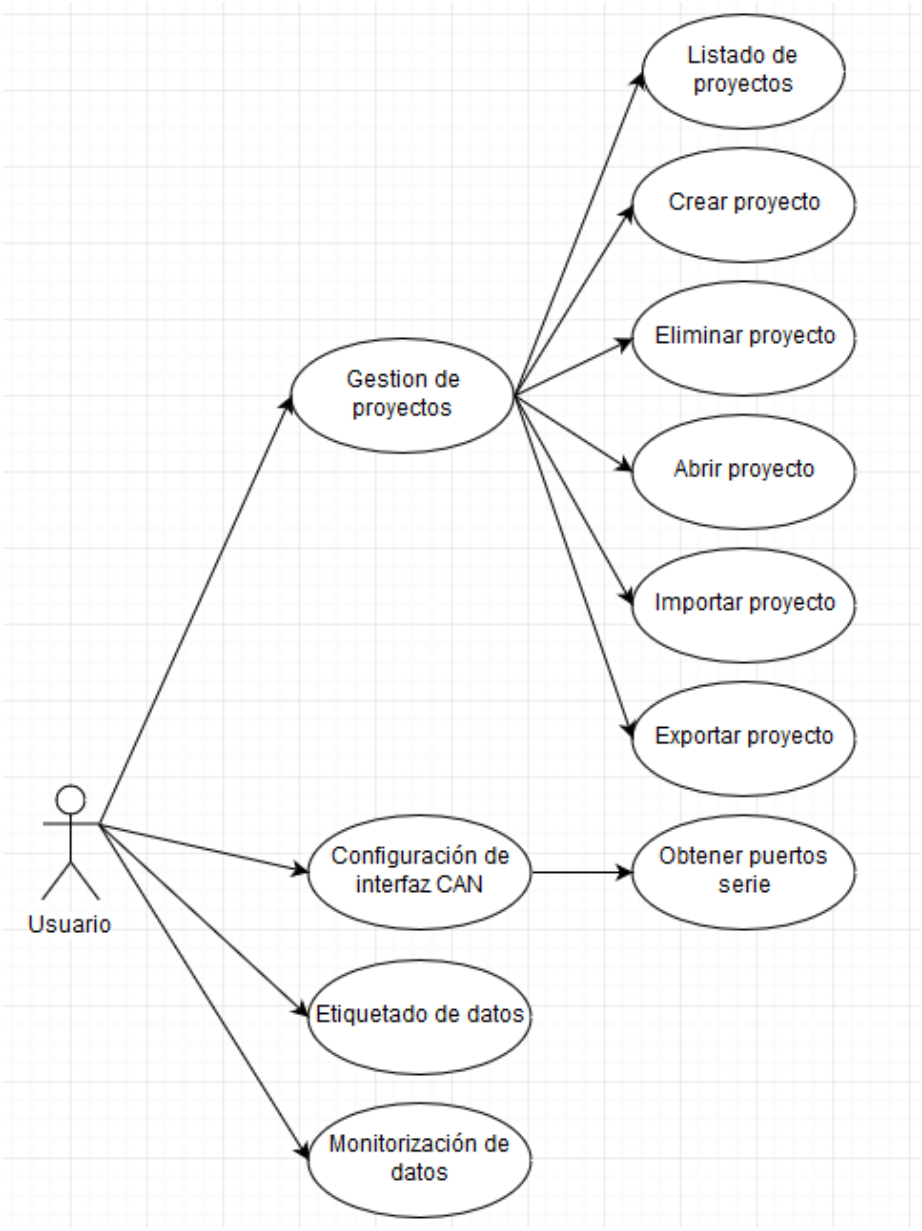

Figura B.1: Diagrama UML general de los casos de uso.

Caso de uso 1: Gestión de proyectos	
Descripción	Permite gestionar los proyectos de la aplicación.
Requisitos	RF-1
	RF-1
	RF-1.1
	RF-1.2
	RF-1.3
	RF-1.4
	RF-1.5
	RF-1.6
	RF-1.7
Precondiciones	Existe una conexión a la base de datos
Secuencia normal	Paso Acción
	1 El usuario ejecuta la aplicación

Caso de uso 2: Listar proyectos		
Descripción	Permite listar los proyectos de la aplicación.	
Requisitos	RF-1.2	
Precondiciones	Existe una conexión a la base de datos	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	El usuario lista los proyectos disponibles.
Postcondiciones	Se muestran todos los proyectos.	
Excepciones	Error en la conexión con la base de datos.	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.2: Caso de uso 2: Listar proyectos

Caso de uso 3: Crear proyecto		
Descripción	Permite crear un nuevo proyecto y añadirle datos.	
Requisitos	RF-1.1	
	RF-1.2	
Precondiciones	Existe una conexión a la base de datos	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	Se crea un nuevo proyecto.
	3	Se guardan los datos del proyecto.
Postcondiciones	Se muestran todos los proyectos.	
Excepciones	Error en la conexión con la base de datos.	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.3: Caso de uso 3: Crear proyecto

Caso de uso 4: Eliminar proyecto		
Descripción	Permite eliminar un proyecto existente.	
Requisitos	RF-1.3	
	RF-1.3.1	
Precondiciones	Existe una conexión a la base de datos	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	Se crea un nuevo proyecto.
	3	Se guardan los datos del proyecto.
Postcondiciones	Se elimina el proyecto seleccionado.	
Excepciones	El proyecto no existe.	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.4: Caso de uso 4: Eliminar proyecto

Caso de uso 5: Abrir proyecto		
Descripción	Permite abrir un proyecto existente.	
Requisitos	RF-1.4	
Precondiciones	Existe una conexión a la base de datos	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	El usuario lista los proyectos disponibles.
	3	El usuario selecciona el proyecto que desea abrir.
	4	El usuario abre el proyecto.
Postcondiciones	Se abre el proyecto seleccionado.	
Excepciones	El proyecto no existe.	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.5: Caso de uso 5: Abrir proyecto

Caso de uso 6: Cerrar proyecto		
Descripción	Permite cerrar un proyecto abierto.	
Requisitos	RF-1.5	
Precondiciones	Existe un proyecto abierto.	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	El usuario tiene un proyecto abierto en la aplicación.
	3	El usuario cierra el proyecto.
	4	El usuario confirma que desea guardar los cambios.
	5	Se cierra el proyecto.
Postcondiciones	Se cierra el proyecto abierto.	
Excepciones	No hay conexión con la base de datos.	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.6: Caso de uso 6: Cerrar proyecto

Caso de uso 7: Importar proyecto		
Descripción	Permite importar un proyecto.	
Requisitos	RF-1.6	
Precondiciones	Existe conexión a la base de datos.	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	El usuario lista los proyectos disponibles.
	3	El usuario selecciona el fichero que desea importar
	4	El usuario lista los proyectos disponibles de nuevo.
	5	El proyecto importado se encuentra en la aplicación.
Postcondiciones	Existe un nuevo proyecto en la aplicación.	
Excepciones	No hay conexión con la base de datos.	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.7: Caso de uso 7: Importar proyecto

Caso de uso 8: Exportar proyecto		
Descripción	Permite exportar un proyecto.	
Requisitos	RF-1.7	
Precondiciones	Existe conexión a la base de datos.	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	El usuario lista los proyectos disponibles.
	3	El usuario selecciona el proyecto que desea exportar.
	4	El usuario selecciona dónde desea guardar el proyecto.
	5	El proyecto es exportado a un fichero.
Postcondiciones	Se exporta un proyecto de la aplicación.	
Excepciones	No existe ningún proyecto	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.8: Caso de uso 8: Exportar proyecto

Caso de uso 9: Obtención de puertos serie disponibles		
Descripción	Permite obtener los puertos series disponibles en el equipo.	
Requisitos	RF-2	
	RF-2.1	
Precondiciones	Existe conexión a la base de datos.	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	El usuario se dispone a configurar una de las interfaces.
	3	El usuario lista los puertos serie disponibles.
Postcondiciones	Se listan los puertos serie.	
Excepciones	No existe ningún puerto serie disponible	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.9: Caso de uso 9: Obtención de puertos serie disponibles

Caso de uso 10: Configuración de la interfaz CAN		
Descripción	Permite configurar y conectarse a una interfaz CAN.	
Requisitos	RF-2.1	
	RF-2.2	
	RF-2.3	
	RF-3	
	RF-3.1	
	RF-3.2	
Precondiciones	Existe un puerto serie en la máquina.	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	El usuario selecciona un puerto serie de la máquina.
	4	El usuario configura la velocidad de conexión.
	5	El usuario configura el modo de conexión.
	6	El usuario configura si desea ignorar los *bytes* con valor cero.
	7	El usuario configura si desea dividir los *bytes* en dos partes.
Postcondiciones	Se conecta a una interfaz CAN.	
Excepciones	El puerto serie está en uso	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.10: Caso de uso 10: Configuración de la interfaz CAN

Caso de uso 11: Etiquetado de datos		
Descripción	Permite asignar etiquetas a los datos identificados por el usuario.	
Requisitos	RF-4	
Precondiciones	Existe un dato en el programa.	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	El usuario configura una interfaz.
	4	El usuario crea un nuevo proyecto.
	5	El usuario comienza el análisis.
	6	El usuario identifica uno de los datos.
	7	El usuario establece una etiqueta al dato.
	8	El usuario envía el dato al dashboard.
Postcondiciones	Etiqueta un dato identificado.	
Excepciones	No se ha creado un proyecto.	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.11: Caso de uso 11: Etiquetado de datos

Caso de uso 12: Monitorización de los datos		
Descripción	Permite monitorizar los datos identificados	
Requisitos	RF-4.1	
Precondiciones	Existen datos identificados.	
Secuencia normal	Paso	Acción
	1	El usuario ejecuta la aplicación.
	2	El usuario configura una interfaz.
	4	El usuario lista los proyectos disponibles.
	5	El usuario abre uno de los proyectos.
	6	El usuario comienza la monitorización de los datos.
Postcondiciones	Se muestran los datos que circulan por el bus.	
Excepciones	No existe un proyecto..	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Tabla B.12: Caso de uso 12: Monitorización de los datos

Apéndice C

Especificación de diseño

C.1. Introducción

En esta sección se expone cómo se han resuelto las especificaciones indicadas anteriormente.

Se definen los datos que va a utilizar la aplicación, el diseño de sus interfaces y su arquitectura.

C.2. Diseño de datos

Base de datos

La aplicación utiliza una base de datos SQLite para almacenar los proyectos. La estructura de la base de datos se compone de dos tablas:

- **projects**: En esta tabla se almacenan los nombres de los proyectos. Dispone de un único campo llamado *name*, el cual es clave primaria de la tabla.
- **data**: Esta tabla almacena los datos identificados en cada proyecto. Dispone de 4 campos:
  - **name**: Almacena la etiqueta con la que se ha identificado al dato.
  - **ID**: Almacena la ID del *frame* que contiene el dato identificado.
  - **byte**: Almacena el *byte* del frame que contiene el dato identificado.

- **projectName**: Clave foránea a la tabla *projects*. Nos indica el proyecto al que pertenece el dato identificado.

A continuación se expone un diagrama relacional de la base de datos:

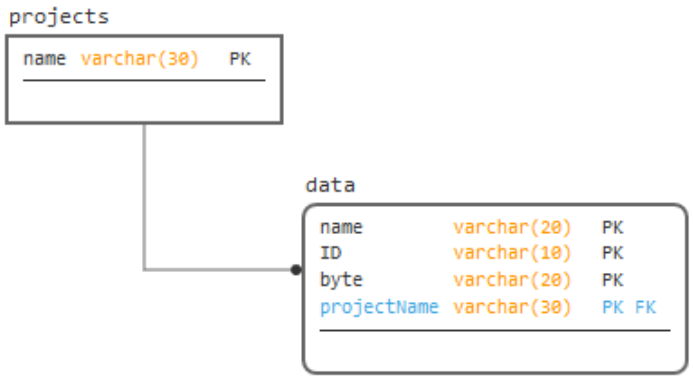

Figura C.1: Diagrama relacional de la base de datos

C.3. Diseño procedimental

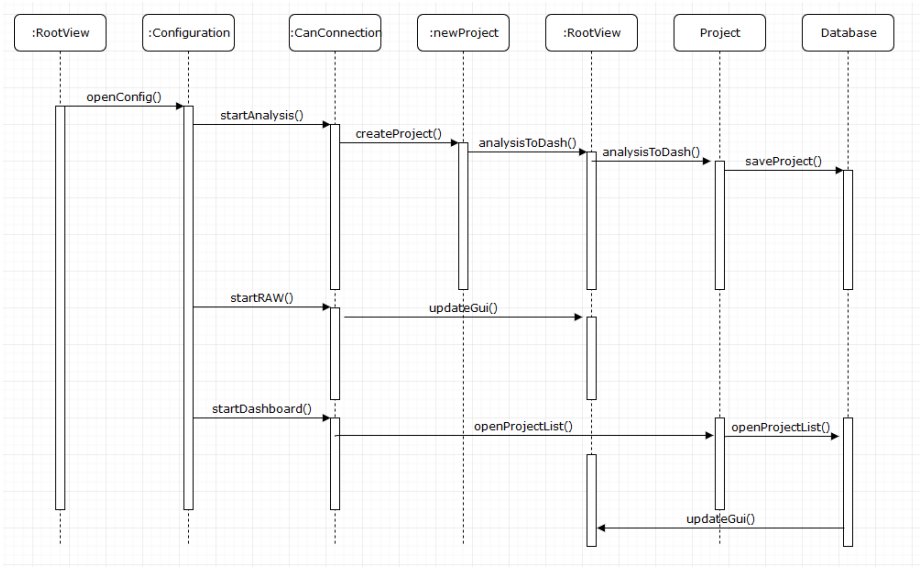

Figura C.2: Diagrama de secuencia de la aplicación.

Como podemos observar en el diagrama de secuencia, desde el puto de inicio del programa, podemos seleccionar tres ramas:

- Comenzar un análisis
- Comenzar una monitorización.
- Comenzar una lectura de los datos en formato *bruto*.

Para llegar a todas las opciones anteriores, es necesario haber configurado la interfaz con la que nos vamos a conectar al bus CAN.

Tanto la primera como la segunda línea del procedimiento hacen uso de la base de datos y de la gestión de proyectos, sin embargo la última rama no hace uso de la base de datos, ya que simplemente se encarga de monitorizar todos los datos en tiempo real en una tabla, sin almacenarlos.

C.4. Diseño arquitectónico

Model-View-Controller (MVC)

Aplicando este patrón de diseño, se utilizan 3 componentes, las vistas, los modelos y los controladores, de tal forma que se separa la lógica de la vista

de la aplicación. Con el uso de este método, si realizamos una modificación en una parte de nuestro código, la otra parte no se ve afectada.

Por ejemplo, si modificamos la base de datos, solo modificaríamos el modelo encargado de esa acción, sin tener que modificar por ejemplo, la parte de la vista.

Está compuesto por tres componentes:

Model: Normalmente esta parte se encarga de los datos (no siempre). Esto puede ser por ejemplo, consultando a una base de datos para obtener información.

View: Componen la representación visual de los datos, es decir, todo lo que tenga que ver con la interfaz gráfica.

Controller: Es un mediador entre los modelos y las vistas. Se encarga de recibir las órdenes del usuario a través de la vista, realizar la petición de datos al modelo y devolverlo de nuevo a la vista para que sea mostrado al usuario.

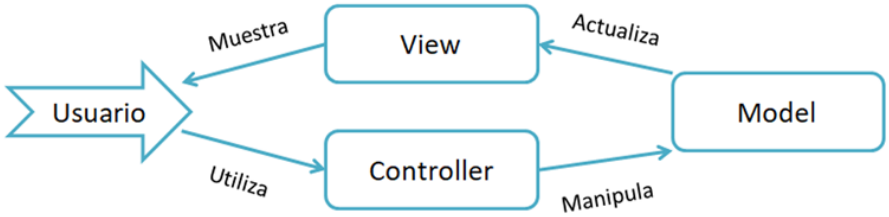

Figura C.3: Modelo MVC

.

Arquitectura general

A continuación se describe de manera general la arquitectura de la aplicación. Como se ha indicado anteriormente, se utiliza el modelo MVC para el funcionamiento interno de la aplicación.

Además de eso, cabe destacar la interacción con la base de datos, tanto de lectura y escritura, como la interacción con la interfaz CAN encargada de leer los datos que fluyen por el bus.

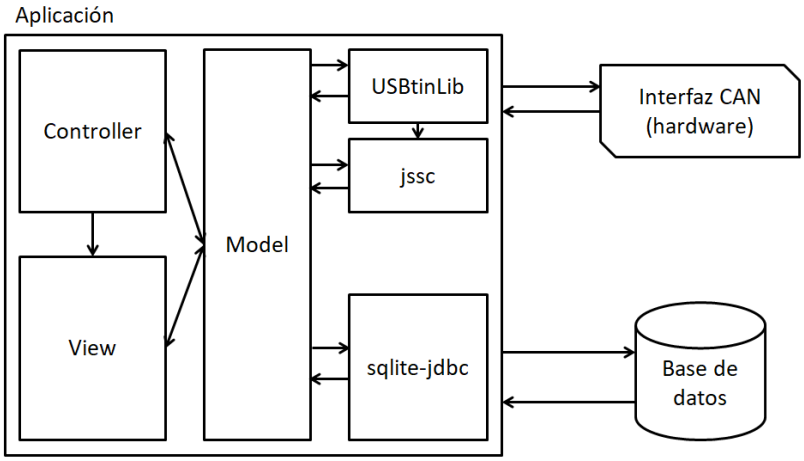

Figura C.4: Arquitectura general del proyecto.

Diseño de paquetes

Para conseguir una organización interna del proyecto, se agruparon las distintas funcionalidades de la aplicación en paquetes. De esta manera se consigue una aplicación modular y con funcionalidades independientes.

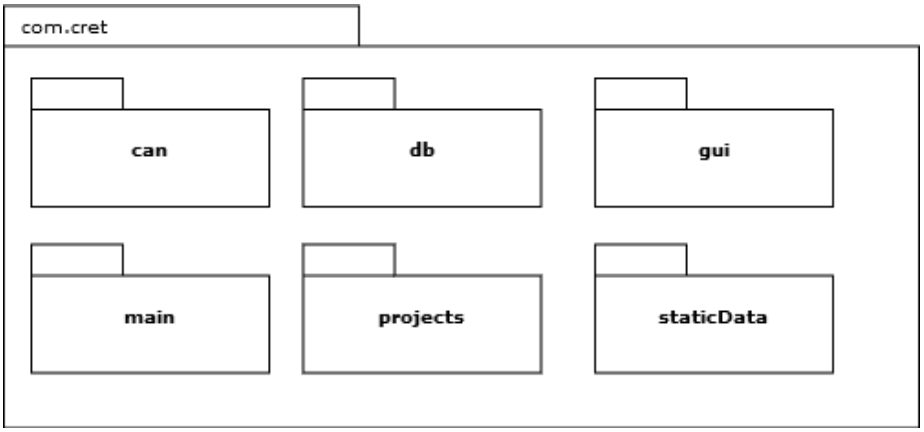

Figura C.5: Paquetes de la aplicación

A continuación se describen cada uno de los paquetes:

- **can**: En este paquete se almacena todo lo relacionado con la interacción con las interfaces CAN.

- **db**: En este paquete se almacena todo lo relacionado con la conexión a la base de datos, la obtención y la inserción de datos en la misma.
- **gui**: En este paquete se almacena todo lo relacionado con la interfaz gráfica.
- **main**: En este paquete se encuentra el *main* de la aplicación, así como el *preloader*.
- **projects**: En ete paquete se almacena todo lo relacionado con la gestión de proyectos de la aplicación.
- **staticData**: En este paquete se almacenan una serie de estructuras las cuales son utilizadas por distintos componentes del proyecto. En el se almacenan todos los datos que son representados por las gráficas de la aplicación.

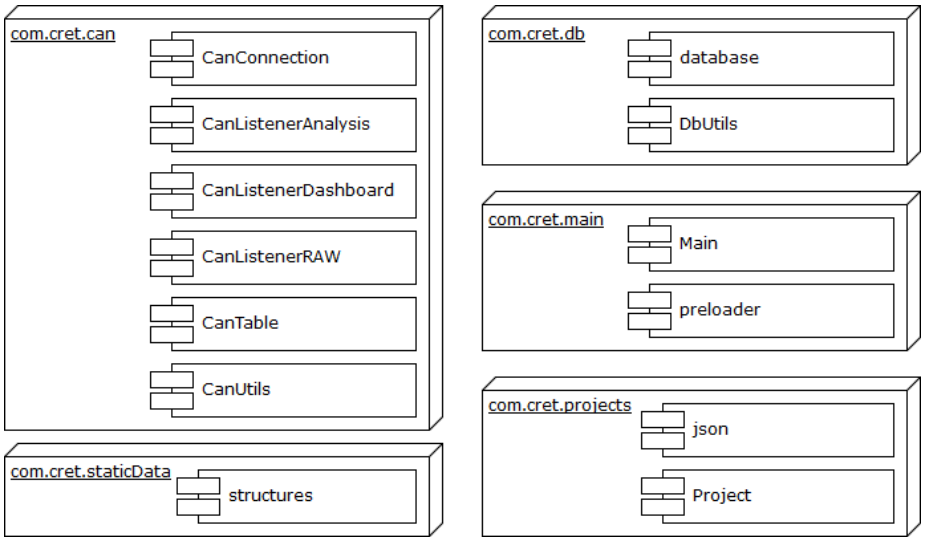

Figura C.6: Diagrama de clases y paquetes de la aplicación.

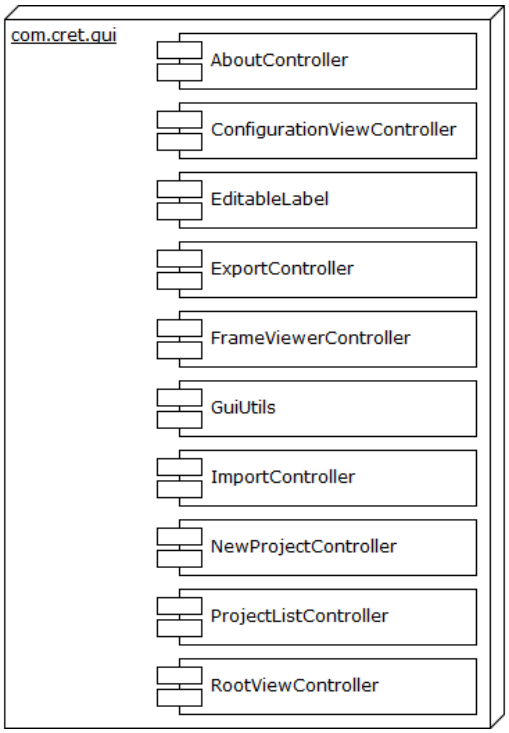

Figura C.7: Diagrama de clases y paquetes de la interfaz gráfica.

Diseño de clases

En este apartado se describe cada una de las clases que forman la aplicación y su funcionamiento.

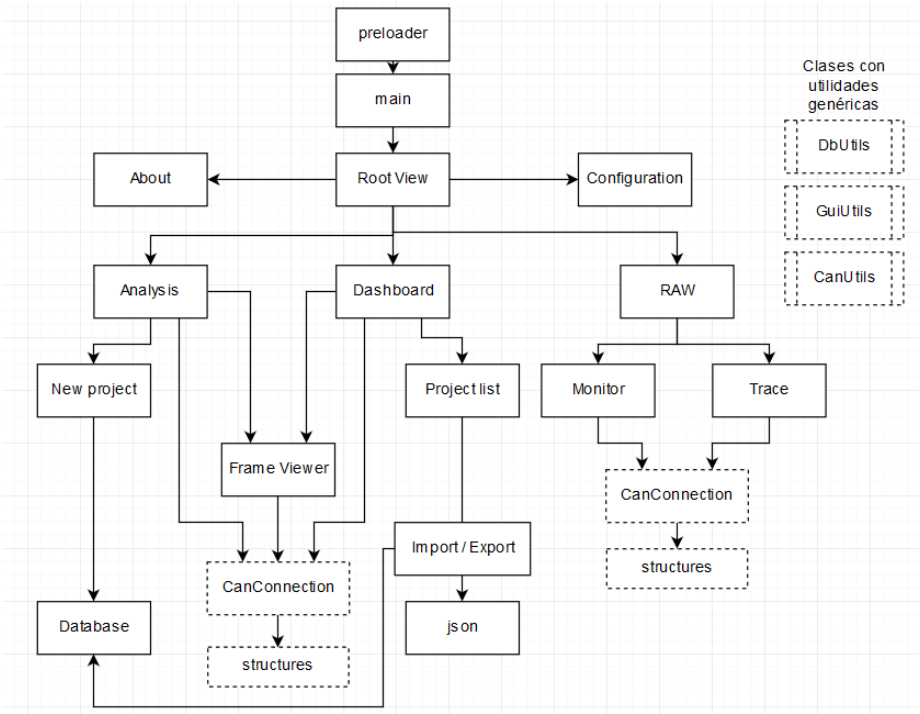

Figura C.8: Diagrama UML de las clases general de la aplicación.

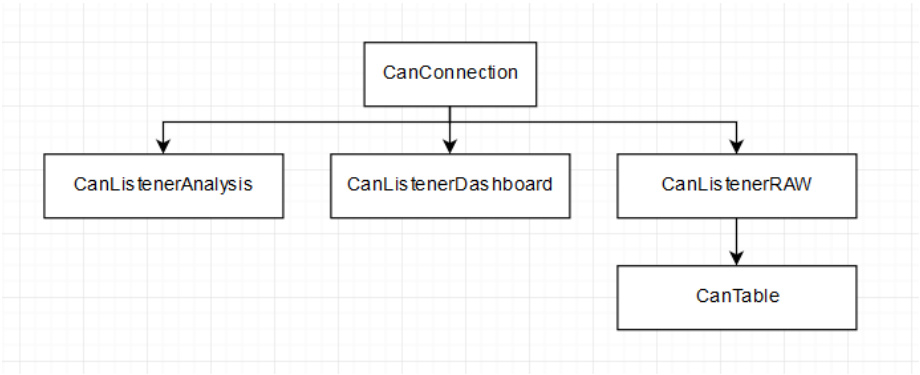

Figura C.9: Diagrama UML de las clases dedicadas a la comunicación con el bus CAN.

C.5. Diseño de interfaces

El diseño de las interfaces se ha llevado a cabo a través de la utilidad SceneBuilder. Se ha intentado seguir una estructura limpia y clara, para

facilitar su interacción con el usuario.

A continuación se muestran algunas de las vistas de la aplicación:

CAPTURAS DE LA APP

El siguiente diagrama nos muestra el flujo de navegabilidad por la aplicación.

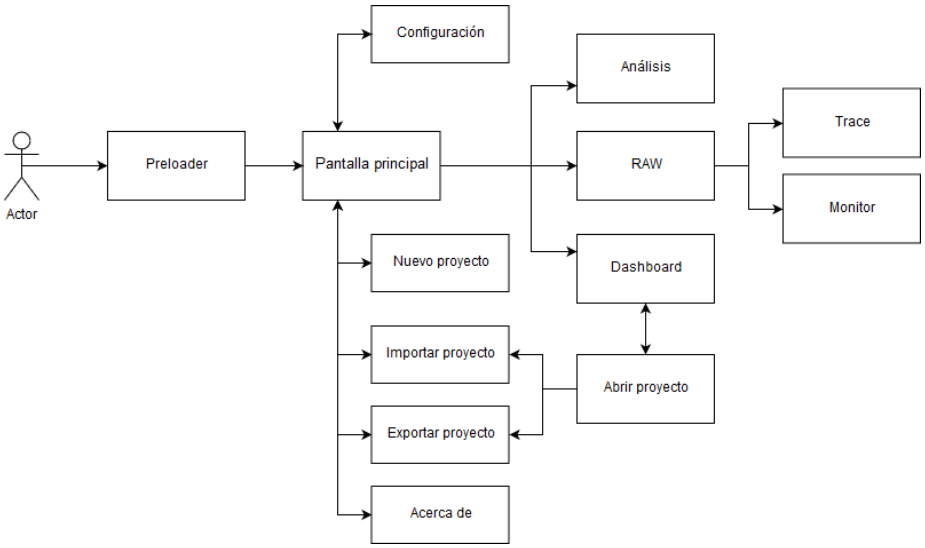

Figura C.10: Diagrama de navegabilidad

.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este anexo se describe la documentación técnica de programación, incluyendo la preparación del entorno de desarrollo, la estructura de la aplicación y las configuraciones necesarias.

D.2. Estructura de directorios

El proyecto tiene la siguiente estructura:

- */documentacion/:* Documentación del proyecto.
- */documentacion/img/:* Imágenes utilizadas en la documentación.
- */documentacion/javadoc/:* Documentación *Javadoc*.
- */hardware/:* Esquemas del *hardware* realizado en formato para KiCAD.
- */hardware/gerber/:* Ficheros *gerber* para la producción del hardware.
- */src/:* Código fuente de la aplicación Java
- */resources/:* Recursos de la aplicación.

D.3. Manual del programador

El siguiente manual tiene como objetivo servir de referencia para futuros programadores. Se explica como montar el entorno de desarrollo, las dependencias necesarias así como su compilación y ejecución.

Entorno de desarrollo

Para el proyecto se necesita tener instaladas las siguientes dependencias:

- Java JDK 8
- JavaFX
- Git
- SceneBuilder
- Eclipse

Java JDK 8

Es uno de los lenguajes más utilizados hoy en día. El uso del lenguaje Java nos permite la compatibilidad de la aplicación en distintas plataformas. Se puede obtener del siguiente enlace:

https://www.oracle.com/technetwork/java/javase/\downloads/jdk8-downloads-html

JavaFX

Se trata de un conjunto de productos y tecnologías para la creación de interfaces gráficas en Java. Para la ejecución de estas aplicaciones solamente es necesaria la instalación de JRE en la máquina.

Se puede obtener del siguiente enlace: https://www.oracle.com/technetwork/java/javafx/overview/index.html

Git

Para utilizar el repositorio en el que se encuentra la aplicación, es necesario el uso de Git. Este programa nos permitirá clonar el repositorio a nuestra máquina y empezar a trabajar con ello.

Podemos instalarlo en Windows habilitando el WSL (Windows Subsystem for Linux), y ejecutando el comando *apt-get install git* con permisos de *root*.

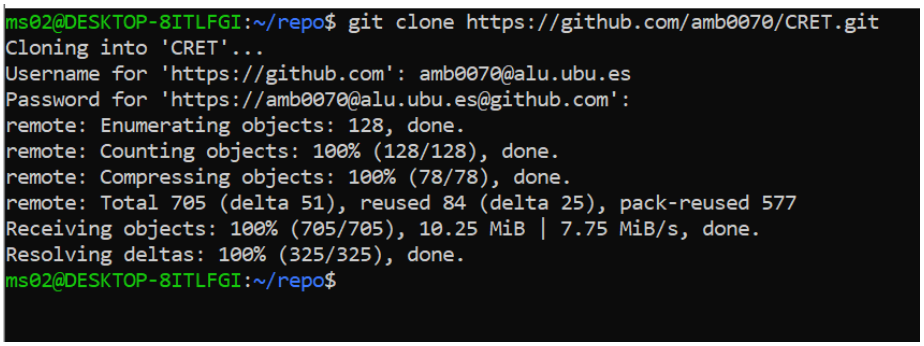

Figura D.1: Comando Git sobre WSL.

SceneBuilder

Se trata de una interfaz gráfica creada para el desarrollo de interfaces gráficas en JavaFX. Sigue el modelo drag and drop, y toda la información es almacenada en un fichero FXML, un formato especial que extiende de XML.

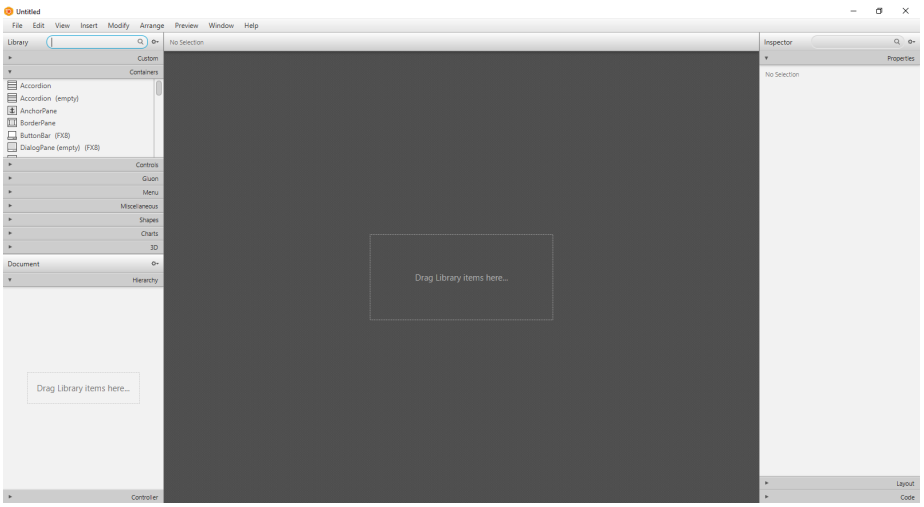

Figura D.2: SceneBuilder 10.0

Eclipse

Para el desarrollo del código Java, se ha utilizado la IDE Eclipse. En ella se han importado las librerías correspondientes, e instalado plugins para el soporte de JavaFX. De esta manera, es posible llamar a SceneBuilder desde Eclipse.

Obtención del código fuente

El código fuente de la aplicación está hospedado en un repositorio Git en GitHub. Para obtener una copia de este código, se pueden seguir los siguientes pasos:

Abrir el WSL en Windows.

Posicionarse en el directorio en el que se desea copiar el código. (Si deseamos acceder al sistema de ficheros de Windows desde WSL, la ruta será /mnt/c).

Introducir el siguiente comando:

git clone https://github.com/amb0070/CRET.git

Una vez finalizada la descarga del repositorio, se podrá acceder al código fuente y a los recursos del proyecto.

Importar proyecto a Eclipse

Una vez descargado el proyecto, abrimos Eclipse.

Hacemos *click* en *File >Import...*

Seleccionamos *Projects from File System or Archive*, y seleccionamos el directorio donde hemos clonado el repositorio.

De esta manera, queda importado el proyecto a nuestro *workspace* de Eclipse.

Importar librerías necesarias

Para importar las librerías necesarias y utilizadas por el proyecto, podemos seguir los siguientes pasos:

Sobre la carpeta del proyecto, en el árbol izquierdo, hacer click derecho.

Seleccionar sub-menú *Build Path >Configure Build Path.*

Hacemos *click* en el botón *Add External JARs...* y seleccionamos los ficheros .jar de las librerías.

D.4. Compilación, instalación y ejecución del proyecto

Compilación

La compilación del proyecto se realiza desde Eclipse.

Para exportarlo como un fichero .jar, podemos hacer click derecho sobre el árbol izquierdo del proyecto, y *click* en *Export...*.

Seleccionamos *Java >Runnable JAR file.*

Incluimos las librerías en el fichero .jar, y exportamos.

MAVEN

Ejecución

No es necesario realizar una instalación de la aplicación. Simplemente tener instalado en la máquina el JRE.

La ejecución de la aplicación no requiere de ningún elemento externo, pero si que es necesario tener el *hardware* conectado a la máquina para su funcionamiento.

De igual manera, el proyecto puede ser ejecutado desde Eclipse, pero para su funcionamiento es necesario el *hardware* que se ha diseñado.

D.5. Producción del *hardware*

A continuación se expone el procedimiento para la producción del *hardware* desarrollado.

El proyecto puede ser abierto con KiCAD de forma que tenemos acceso a la modificación de los esquemas y del diseño de la PCB.

El flujo básico de trabajo en KiCAD, a rasgos muy generales sería el siguiente:

- **Esquema**: Desarrollo del esquema del circuito. En éste esquema indicaríamos los componentes a utilizar y su conexión entre ellos.

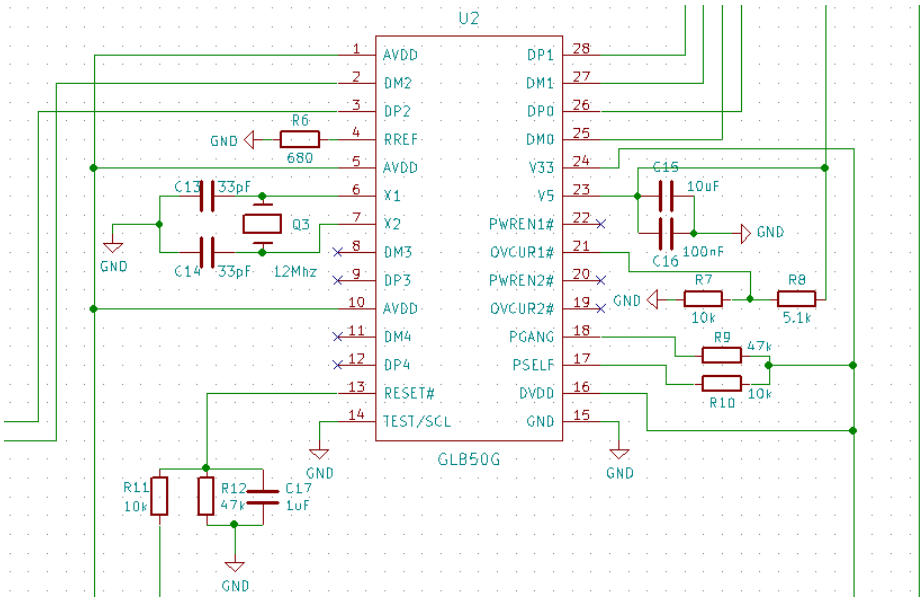

Figura D.3: Ejemplo del diseño de uno de los elementos en el esquema.

- **Asignación de huellas**: El siguiente paso en el desarrollo sería asignar a cada uno de los elementos del esquema el *footprint*, es decir, la forma y tamaño del componente que se va a incluir en la *PCB*.

  Para ello, podemos hacer *click* en el siguiente botón:

Figura D.4: Icono para asignar las huellas al esquema.

En la siguiente ventana tenemos una relación entre el nombre del componente y el *footprint* asignado.

1	C1 -	100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
2	C2 -	100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
3	C3 -	100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
4	C4 -	100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
5	C5 -	100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
6	C6 -	100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
7	C7 -	10uF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
8	C8 -	0,1uF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
9	C9 -	0,1uF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
10	C10 -	22pF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
11	C11 -	22pF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
12	C12 -	100nF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
13	C13 -	33pF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder

Figura D.5: Relación entre los componentes y su *footprint*.

Analizando el primer elemento marcado en la lista, podemos obtener la siguiente información:

- *C1*: Hace referencia al nombre del elemento.
- *100nF*: Hace referencia a la capacidad del elemento. En este caso al ser un condensador, la medida está indicada en nano faradios.
- *Capacitor SMD:* Hace referencia al tipo de componente. En este caso un condensador para montaje sobre superficie (*SMD - Surface Mount Device*).
- *1206*: Este dato hace referencia a la tecnología de montaje superficial. Se podría decir que nos indican el tamaño del encapsulado del componente.
- *Pad1.42x.75mm HandSolder*: En este caso, nos indica el tamaño de los *pads*, es decir, de la superficie en la que es soldado el componente. En este caso tienen un tamaño un poco más grande de lo normal para facilitar la soldadura a mano (*HandSolder*).

- **Netlist:** Una vez concluido el diseño del esquema y su asignación de huellas, deberíamos de generar el fichero *netlist*. Este fichero será el que posteriormente se importe al diseñador de *PCBs* y el cual contiene todos los elementos del esquema y sus conexiones.

  En *KiCAD* podemos generarlo con el siguiente botón:

Figura D.6: Icono para generar el fichero *netlist*.

Importamos dicho fichero en el editor de *PCBs* y obtendremos algo parecido a esto:

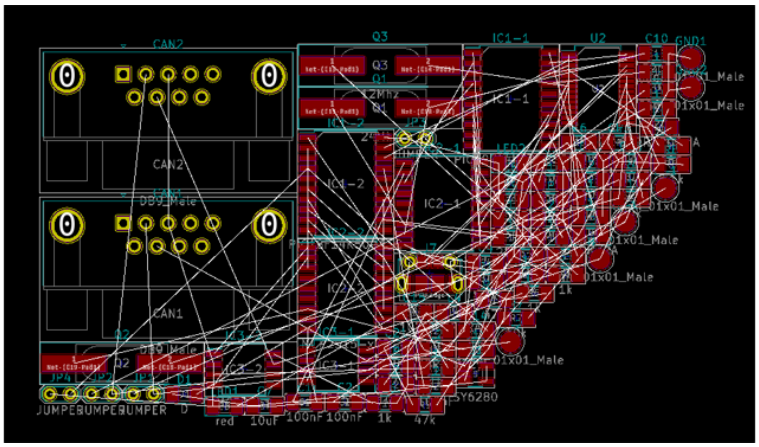

Figura D.7: Diseño inicial de la *PCB*

Insertando las vías y *enrutando* todo el diseño, obtenemos el siguiente resultado:

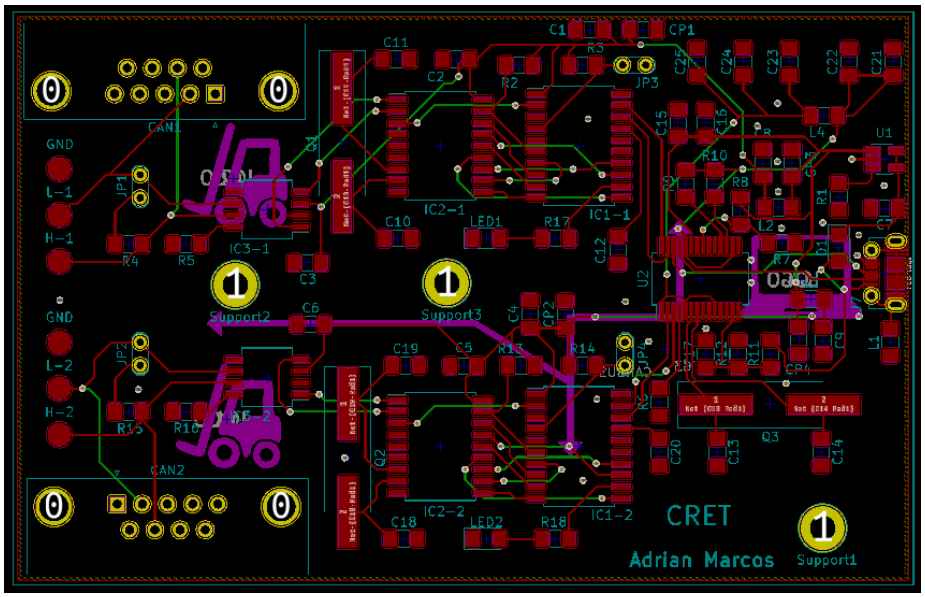

Figura D.8: Diseño final de la *PCB*.

Con el diseño concluido, generamos los ficheros *Gerber* para su producción.

En este caso, los prototipos han sido producidos en: https://jlcpcb.com/.

El listado de componentes necesarios para la producción del *hardware* son los siguientes:

Cantidad	Componente	Elemento
2	PIC18F14K50T-I/SO	IC1-1, IC1-2
2	MCP2515T-I/SOCT-ND	IC2-1, IC2-2
1	GL850G	U2
1	SY6280	U1
2	MCP2551-I/SN-ND	IC3-1, IC3-2
2	CRYSTAL 24.0000MHZ 18PF SMD	Q1, Q2
2	FERRITE BEAD 600 OHM 1206 1LN	L1, L2, L3, L4
2	LED RED DIFFUSED 1206 SMD	LED1, LED2
1	CAP CER 1UF 10V X7R 1206	C17
2	CAP CER 33PF 50V C0G/NP0 1206	C13, C14
1	CONN RCPT USB2.0 MICRO B SMD R/A	J7
2	CONN D-SUB PLUG 9POS R/A SOLDER	CAN1, CAN2
4	CAP CER 22PF 10V C0G/NP0 1206	C10, C11, C18, C19
3	CAP CER 10UF 10V X7R 1206	C7, C15, C22
2	CAP CER 0.1UF 10V X7R 1206	C8, C9
15	CAP CER 100UF 6.3V X5R 1206	C(1-6,12,16,20,21,23-25),CP(3,4)
2	CAP CER 4.7UF 10V X5R 1206	CP1, CP2
2	RES 47K OHM 5 % 1/4W 1206	R9, R12
1	RES 5.1K OHM 5 % 1/4W 1206	R8
3	RES 10K OHM 0.1 % 1/8W 1206	R7, R10, R11
1	CRGCQ 1206 680R 1 %	R6
2	RES SMD 120 OHM 5 % 1/2W 1206	R4, R15
8	RES 1K OHM 1 % 1/2W 1206	R2,R3,R5,R13,R14,R16-R18
1	RES SMD 4.7K OHM 1 % 1/4W 1206	R1
1	DIODE 1206	D1

Todos estos componentes han sido comprados en https://www.digikey.es/ aunque pueden ser obtenidos en otras tiendas.

El formato de todas las resistencias, condensadores y ferritas ha sido SMD 1206, debido a su reducido tamaño pero a la vez suficiente para soldarlo a mano.

Una vez disponemos de la PCB y de los componentes, es necesario realizar un paso previo antes de soldarlos. Es necesario programar los PIC con el *bootloader* para posteriormente cargar el *firmware* en los mismos.

Para ello, teniendo en cuenta el *datasheet* del microcontrolador, podemos realizar la siguiente conexión:

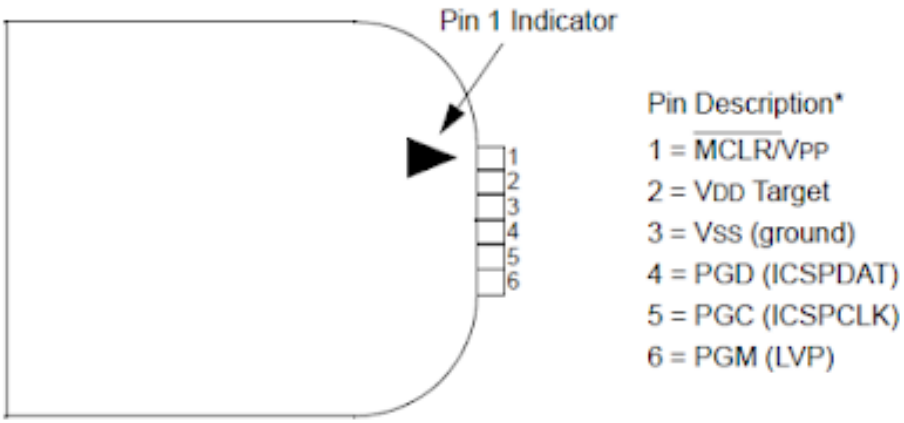

Figura D.9: *Pinout* del *PICKit3*.

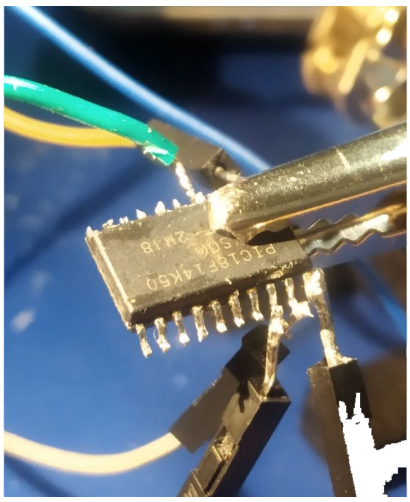

Figura D.10: Cargando *firmware* en el PIC.

Una vez detectado el dispositivo por el equipo, es necesario conectarse a el, e indicar el fichero .hex que queremos cargar, en este caso, el *bootloader* referenciado en la página oficial.

Es importante configurar el programador para que alimente el microcontrolador, sino la carga del *bootloader* no será posible. Para ello, hay que acceder al modo avanzado del programador.

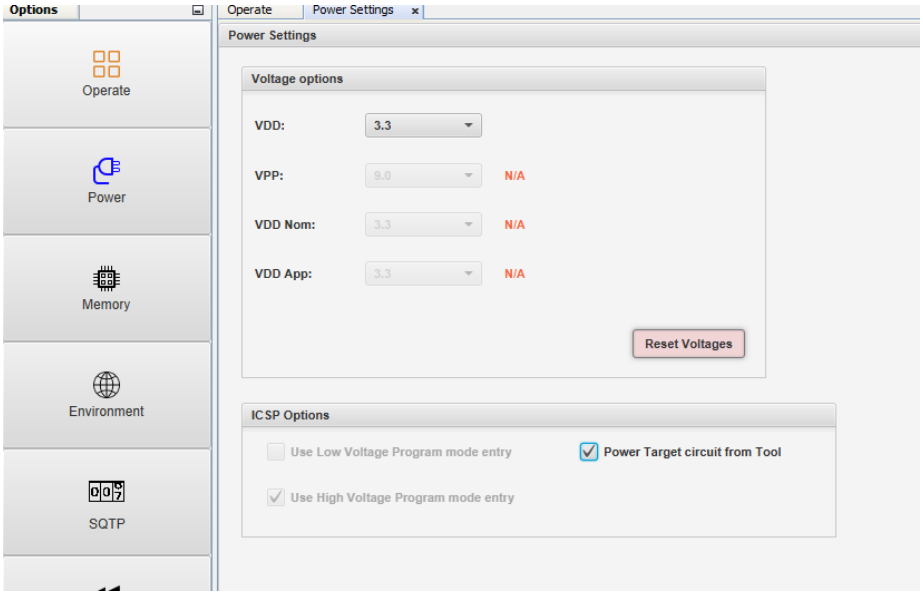

Figura D.11: Configuración avanzada de *MPLAB*.

Seleccionamos la opción para que el programador alimente el controlador.

Ya podemos grabar el *bootloader* en el microcontrolador.

Una vez realizados estos pasos, ya se pueden soldar todos los componentes en la placa.

Para hacerla funcionar es necesario un paso más. Cargar el *firmware*.

Para ello, conectamos la placa al USB del equipo, con el *jumper* JP3 o JP4 (cada uno es para su microcontrolador), y ejecutamos la siguiente utilidad:

mphidflash - https://github.com/ApertureLabsLtd/mphidflash

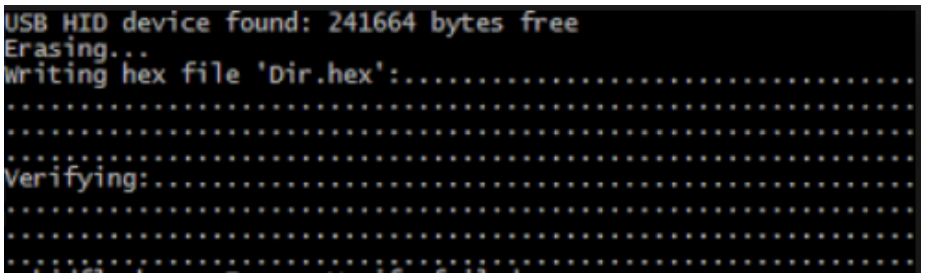

Figura D.12: Salida del comando *mphidflash*.

Una vez completada la carga, podemos comprobar el dispositivo y estaría listo para su uso.

Apéndice E

Documentación de usuario

E.1. Introducción

En el siguiente manual se definen los requerimientos de la aplicación, como instalarla y ejecutarla.

E.2. Requisitos de usuarios

Los requisitos mínimos para la utilización de la aplicación son los siguientes:

Hardware para conectarse al bus CAN (usbtin) Java ....

E.3. Instalación

No es necesaria una instalación de la aplicación. Simplemente la utilización de una versión de Java compatible.

E.4. Manual del usuario