Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Display multifunción para visualizar datos NMEA 0183 mediante dispositivos de bajo coste Rasberry Pi / Arduino

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Joan Batiste Canet Collado

Tutor: José Vicente Soler Bayona

2016 – 2017

Agradecimientos

En primer lugar, agradecer a mi tutor José Vicente Soler Bayona por el apoyo recibido para llevar a cabo este proyecto y por sus consejos que me han facilitado el camino para realizarlo. Además como profesor me ha aportado los conocimientos informáticos que me han servido directamente para el desarrollo de este trabajo y esta memoria.

También he de mencionar al resto de profesores que a lo largo de los cursos de la carrera han puesto a nuestra disposición sus conocimientos y experiencia para que nuestra formación fuera lo más completa posible.

Por último agradecer a todas las personas que me han mostrado su apoyo y ánimos para poder avanzar. Una mención especial a mi familia que sin ellos no habría sido posible el llegar aquí. Y a mi pareja que ha sido un gran pilar que me ha ayudado a visualizar el camino, bueno más que un pilar una viga maestra.

Resumen

El presente trabajo de fin de grado trata sobre el desarrollo de una aplicación de escritorio para la visualización de datos NMEA 0183. La aplicación se llama nmea-Display.

Su objetivo es la presentación de datos obtenidos de un dispositivo emisor que se comunica mediante el protocolo estándar NMEA 0183. Estos datos se muestran en la aplicación pudiendo personalizar aquellos que se visualizan.

La implementación se ha realizado con vista a su funcionamiento en un dispositivo de bajo coste como lo es la Raspberry pi 3.

Se ha utilizado para el desarrollo el lenguaje de programación Java para la lógica y la tecnología de JavaFX para la interfaz.

Palabras clave: Java , JavaFX , NMEA 0183, Raspberry pi , Scene Builder, MVC.

Abstract

The current final degree Project addresses about the development of a desktop application to visualize NMEA 0183 data. The application its named nmea-Display.

The application is focused on showing data obtained by an emisor that comunicates by the NMEA 0183 standard protocol. This data is displayed in the application and you could customize the ones that are displayed.

The implementation has been focused on its operation with a low-cost device like a Raspberry pi 3.

It has been used the Java programming language for the logic of the application and the JavaFx technology for the user interface.

Keywords: Java , JavaFX , NMEA 0183, Raspberry pi , Scene Builder, MVC.

Tabla de contenidos

1. Introducción

1.1 Motivación

En el mercado de la navegación náutica existen numerosas empresas que ofrecen soluciones de recogida y emisión de datos del ambiente y de posición. También ofrecen soluciones para la visualización de los datos recogidos. El principal problema es que el coste para adquirir dichos dispositivos es elevado.

El motivo principal para la creación de esta aplicación es conseguir una alternativa de bajo coste para la visualización de datos NMEA0183.

1.2 Objetivos

El principal objetivo de este trabajo de fin de grado es el diseño y desarrollo de una aplicación para visualizar datos NMEA 0183 en un dispositivo de bajo coste.

Con la memoria se pretende dar a conocer el proceso seguido en el desarrollo y explicar las herramientas utilizadas.

2. Especificación de requisitos

2.1 Introducción

Este apartado es una Especificación de Requisitos Software (ERS) para la aplicación de visualización de datos NMEA 1083 con el nombre de nmea-Display. Se examinarán y definirán los requisitos necesarios para la misma.

Para esta Especificación de Requisitos de Software (ERS) se han seguido las recomendaciones definidas por el estándar ANSI/IEEE 830, 1998. Al seguir este estándar la estructura del documento resultante es clara y facilita la definición de las características esperadas.

2.1.1 Propósito

El propósito de este apartado es el de definir los requisitos que serán utilizados posteriormente para el diseño, desarrollo e implementación de la aplicación.

Una buena definición de los requisitos es importante. Cuanto más avanzado esté el diseño de la aplicación, mayor será el esfuerzo a realizar para implementar algún cambio y, sobretodo, si ya está en fase de implementación.

2.2 Descripción general

El proyecto constará de una aplicación que se ejecutará en un dispositivo estilo Raspberry pi la cual mediante un puerto USB recibirá la información a filtrar y mostrar.

2.2.1 Perspectiva del Producto

Como se ha comentado anteriormente, la aplicación será ejecutada en un dispositivo tipo Raspberry pi. Deberá disponer de un puerto USB accesible por el que un dispositivo emisor se comunicará con la aplicación y le suministrará los datos a mostrar.

2.2.1.1 Funciones del producto

Mostrar la información recibida en la pantalla.

La información se actualizará conforme se reciben datos.

Se podrá añadir y eliminar pantallas de presentación de datos.

Gestión del aspecto visual de la aplicación.

2.2.2 Características de los usuarios

Esta aplicación será utilizada por un solo tipo de usuario.

A continuación se realizará su descripción.

Usuario Principal

El usuario que utilizará la aplicación no necesitará de ningún tipo de registro, la aplicación no gestiona usuarios.

Podrá visualizar los datos presentados por la aplicación.

También podrá decidir según sus necesidades el número de pantallas que mostrará la aplicación, así como qué tipo de información se asignará a cada una de ellas.

Por último podrá elegir el aspecto de la aplicación entre los disponibles.

2.2.3 Restricciones

La comunicación con el aparato emisor deberá ser mediante el protocolo estándar NMEA 0183.

La aplicación debe funcionar con hardware limitado (dispositivo tipo Raspberry pi 3)

2.2.4 Suposiciones y dependencias

Se supone en este proyecto que la comunicación con el dispositivo emisor no se ve interrumpida y que dicho aparato emisor funciona correctamente.

El usuario que utiliza el software deberá de disponer de privilegios de administrador para poder administrar el sistema y lanzar la aplicación.

Se da por supuesto que se dispone de una pantalla táctil para poder interactuar con la aplicación así como responder a la entrada por ratón.

2.3 Requisitos funcionales

2.3.1 Configurar pantallas

Tabla 1 RF: Configurar el número de display

Función	Configurar el número de display a mostrar
Introducción	Establecer el número y el tipo de display que mostrará la aplicación
Entrada	Pulsar sobre el botón Display de la vista Inicio.
Proceso	Si la aplicación acaba de iniciarse se generan los display por defecto. En caso de que se hayan configurado nuevos displays el programa cargará cada uno de ellos y los mostrará en la vista display.
Salida	Sin salida, es un proceso automático que se ejecuta sin mostrarse al usuario mientras se accede a la vista Display.

Tabla 2 RF: Editar datos que mostrar en pantalla

Función	Editar datos que mostrar en pantalla
Introducción	Editar el tipo de datos que se muestran en la pantalla
Entrada	Se selecciona el tipo de dato que se quiere mostrar en el apartado de configurar pantalla.
Proceso	La aplicación guarda el tipo de dato seleccionado y lo asocia a la pantalla que se estaba editando.
Salida	El dato que ahora muestra esa configuración de pantalla es el elegido. Por otra parte al visualizar la pantalla se mostrará el dato elegido en la configuración.

2.3.2 Visualizar pantallas

Tabla 3 RF: Mostrar pantalla con datos

Función	Mostrar pantalla con datos
Introducción	Cargar las pantallas con los datos configuradas.
Entrada	Pulsar sobre el botón Display de la vista Inicio.
Proceso	La aplicación carga las pantallas configuradas y las visualiza en la pantalla.
Salida	La aplicación cambia a la pantalla de display y muestra el primero de los disponibles.

Tabla 4 RF: Cambiar a la siguiente pantalla de datos

Función	Cambiar a la siguiente pantalla de datos
Introducción	Cambiar a la siguiente pantalla de datos.
Entrada	Pulsar sobre el botón SIG. de la vista Display.
Proceso	La aplicación carga la pantalla siguiente de la lista de pantallas configuradas.
Salida	La aplicación cambia la pantalla visible a la siguiente de la lista.

Tabla 5 RF: Cambiar a la anterior pantalla de datos

Función	Cambiar a la anterior pantalla de datos
Introducción	Cambiar a la anterior pantalla de datos.
Entrada	Pulsar sobre el botón Prev. de la vista Display.
Proceso	La aplicación carga la pantalla anterior de la lista de pantallas configuradas.
Salida	La aplicación cambia la pantalla visible a la anterior de la lista.

2.3.3 Visualizar datos

Este requisito sería la misma tabla para cada uno de los datos a mostrar.

Tabla 6 RF: Mostrar dato Temperatura

Función	Mostrar dato de Temperatura
Introducción	Representar el dato temperatura.
Entrada	Se recibe una nueva trama de temperatura.
Proceso	La aplicación lee la trama nueva y actualiza el dato de la temperatura.
Salida	El dato temperatura representado en la pantalla cambia al nuevo valor recibido

2.3.4 Gestionar interfaz

Tabla 7 RF: Cambiar *modo interfaz*

Función	Cambiar modo interfaz
Introducción	Establecer el aspecto visual de la aplicación
Entrada	Pulsar sobre el checkBox modo noche de la pantalla configuración
Proceso	Si la aplicación se encuentra en “modo día” cambia a “modo noche” y viceversa.
Salida	El aspecto visual de la aplicación cambia al modo activado.

2.4 Requisitos no funcionales

La interfaz debe tener una respuesta rápida menos de un segundo y no ha de bloquearse.

La interfaz debe responder a entrada táctil y al ratón.

La visualización de los datos debe actualizarse conforme llegan al aparato.

2.5 Requisitos de hardware

Debe ser compatible con una pantalla táctil,

Debe disponer de un USB de entrada

Debe de haber una fuente de alimentación continua para evitar el apagado del aparato.

3. Análisis

3.1 Introducción

En este tercer bloque se va a realizar un análisis para la aplicación. Mediante varios apartados se va a utilizar el lenguaje UML a representar de manera visual y detallada los diagramas de contexto con los actores (que son las personas que van a interactuar con la aplicación), así como los casos de uso con su diagrama y posterior explicación de cómo interactúa el actor con la aplicación. Por último, se mostrará el diagrama de clases que muestra la relación de las diferentes clases que conforman nuestra aplicación.

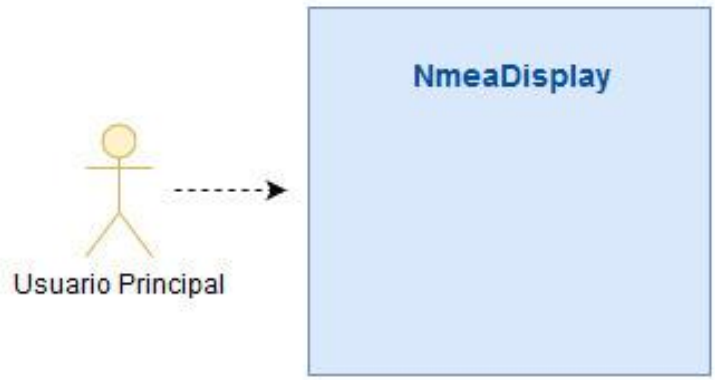
Con este conjunto de diagramas se puede conocer el funcionamiento general de la aplicación para su mejor entendimiento y posterior desarrollo.

3.2 Diagrama de contexto.

Este diagrama representa una visión global del sistema y sus actores principales.

Como único actor tenemos al Usuario Principal que será el usuario que manejará la aplicación.

Ilustración 1 - Diagrama de contexto.



3.3 Casos de uso

3.3.1 Introducción

Con el diagrama de casos de uso se pretende mostrar las posibles interacciones del actor con el sistema y el comportamiento del mismo. Posteriormente se procederá a explicar cada uno de ellos con más detalle.

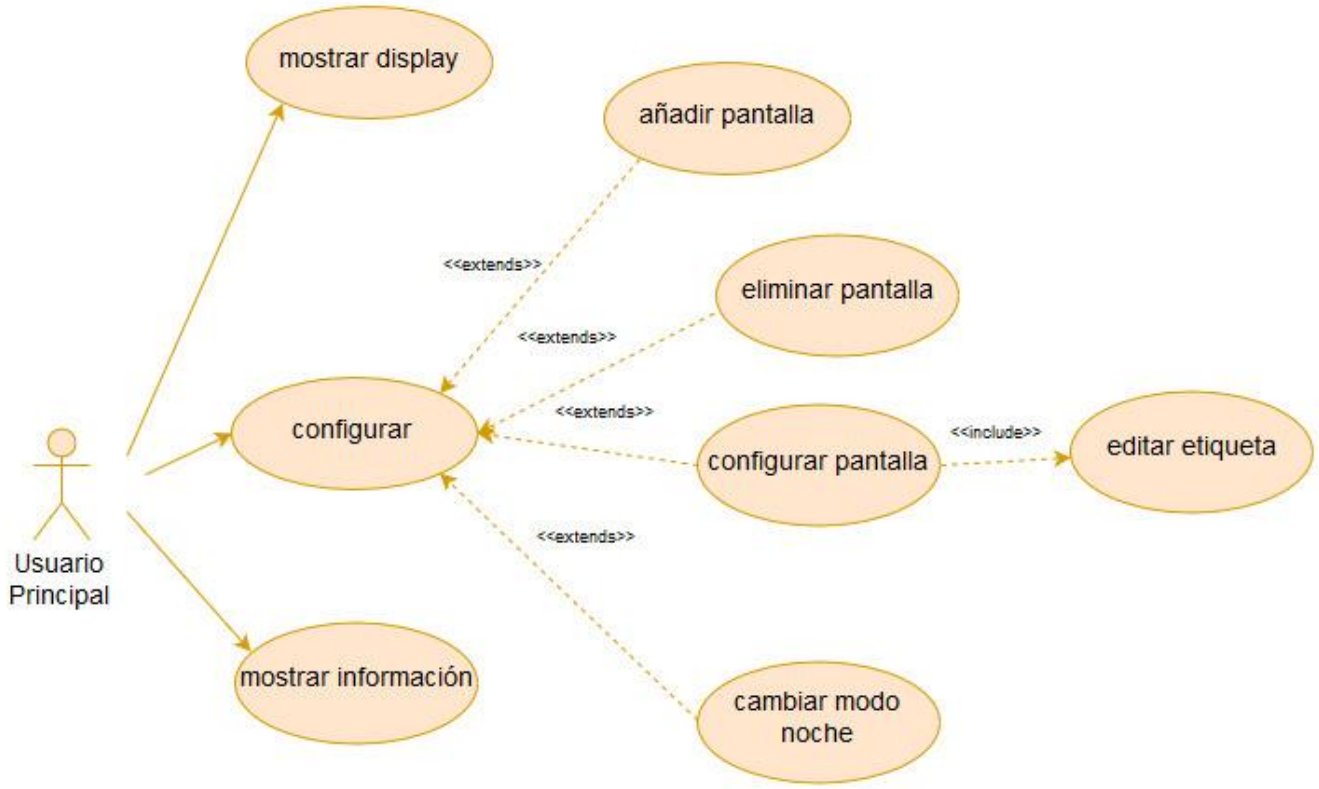
La figura del diagrama representa al actor que se ve involucrado en la acción representada.

Para su representación se ha utilizado UML y las etiquetas utilizadas son las siguientes:

- <<include>> Esta etiqueta implica **obligatoriedad**, es decir, que necesariamente se **debe** dar el caso de uso al que apunta la flecha discontinua. En el diagrama adjunto sería que para editar una pantalla se debe editar una etiqueta de la misma.
- <<extends>> Esta etiqueta implica **condición**, el caso al que apunta puede darse o no. Por ejemplo al configurar se **puede** añadir una pantalla nueva o no añadir ninguna.

3.3.2 Diagrama de casos de uso

Ilustración 2 - Diagrama de casos de uso.



3.3.3 Casos de uso del usuario principal

En este apartado se describe más detalladamente cada caso de uso que ha sido representado en la sección anterior.

Se explica de manera general qué actor participa en el caso de uso, así como las precondiciones para poder llevarse a cabo y el modo de realizarlo.

Tabla 8 caso de uso: Mostrar display

Caso de uso	Mostrar display
Descripción	Se muestra la pantalla con los displays para mostrar la información.
Actor	Usuario Principal.
Precondición	El usuario se encuentra en la pantalla principal (Home) y el aparato emisor debe estar en funcionamiento.
Flujo principal	1) El usuario pulsa sobre el botón Display de la pantalla Home. 2) Se muestra una nueva vista, con los display que están configurados en la aplicación.
Flujo alternativo	Sin flujo alternativo.
Pos condición	El usuario visualiza la pantalla con el display que muestra la información.

Tabla 9 caso de uso: Configurar

Caso de uso	Configurar
Descripción	Se muestra la pantalla configuración
Actor	Usuario Principal.
Precondición	El usuario se encuentra en la pantalla principal (Home).
Flujo principal	1) El usuario pulsa sobre el botón Configuración de la pantalla Home. 2) Se muestra una nueva vista, con las opciones de configuración disponibles.
Flujo alternativo	1) El usuario desde cualquier vista pulsa sobre el menú Herramientas. 2) El usuario pulsa sobre el submenú configuración.
Pos condición	El usuario visualiza la pantalla de configuración.

Tabla 10 caso de uso: Añadir pantalla

Caso de uso	Añadir pantalla
Descripción	Se añade una pantalla al conjunto de displays de la aplicación.
Actor	Usuario Principal.
Precondición	El usuario se encuentra en la pantalla de Settings.
Flujo principal	1) El usuario pulsa sobre el botón añadir. 2) Se añade un nuevo elemento a la lista de display.
Flujo alternativo	Sin flujo alternativo.
Pos condición	El usuario visualiza que se ha añadido un nuevo elemento a la lista de display.

Tabla 11 caso de uso: Eliminar pantalla

Caso de uso	Eliminar pantalla
Descripción	Se elimina una pantalla al conjunto de displays de la aplicación.
Actor	Usuario Principal.
Precondición	El usuario se encuentra en la pantalla de Settings. Debe de haber como mínimo dos pantallas configuradas.
Flujo principal	1) El usuario pulsa sobre el botón eliminar. 2) Se añade un nuevo elemento a la lista de display.
Flujo alternativo	Sin flujo alternativo.
Pos condición	El usuario visualiza que se ha eliminado un elemento de la lista de display.

Tabla 12 caso de uso: Configurar pantalla

Caso de uso	Configurar pantalla
Descripción	Se muestra el panel de edición del display seleccionado.
Actor	Usuario Principal.
Precondición	El usuario se encuentra en la vista de Settings.
Flujo principal	1) El usuario pulsa sobre un elemento de la lista de display. 2) Se muestra el panel de edición del display seleccionado.
Flujo alternativo	Sin flujo alternativo.
Pos condición	El usuario visualiza que ha aparecido el panel de edición para editar el display seleccionado.

Tabla 13 caso de uso: Editar etiqueta

Caso de uso	Editar etiqueta
Descripción	El usuario configura qué tipo de información se mostrará en el display seleccionado.
Actor	Usuario Principal.
Precondición	El usuario debe encontrarse en la vista Settings. El usuario debe haber seleccionado un elemento en la lista.
Flujo principal	1) El usuario pulsa sobre uno de los comboBox del panel de edición y elige qué información se mostrará de una lista. 2) El ComboBox editado se queda con el valor elegido.
Flujo alternativo	El usuario puede realizar esta acción con cada uno de los comboBox que aparecen en el panel.
Pos condición	El usuario visualiza que ha aparecido el panel de edición para editar el display seleccionado.

Tabla 14 caso de uso: Mostrar información

Caso de uso	Mostrar información
Descripción	El usuario muestra la información de la aplicación.
Actor	Usuario Principal.
Precondición	No hay precondiciones.
Flujo principal	1) El usuario pulsa sobre el menú Settings 2) El usuario pulsa sobre el elemento info 3) Aparece una ventana con la información de la aplicación. 4) El usuario pulsa sobre el botón aceptar.
Flujo alternativo	No hay flujo alternativo.
Pos condición	El usuario vuelve a la pantalla en la que se encontraba antes de mostrar la información.

Tabla 15 caso de uso: Cambiar modo noche

Caso de uso	Cambiar modo noche
Descripción	El usuario configura qué tipo interfaz ha de mostrar la aplicación.
Actor	Usuario Principal.
Precondición	El usuario debe encontrarse en la vista Settings.
Flujo principal	1) El usuario pulsa sobre el checkBox modo noche. 2) La estética de la aplicación cambia según esté seleccionado o no el elemento.
Flujo alternativo	No hay flujo alternativo.
Pos condición	El usuario visualiza que la apariencia de la aplicación ha cambiado según el modo elegido.

3.4 Diagrama de clases

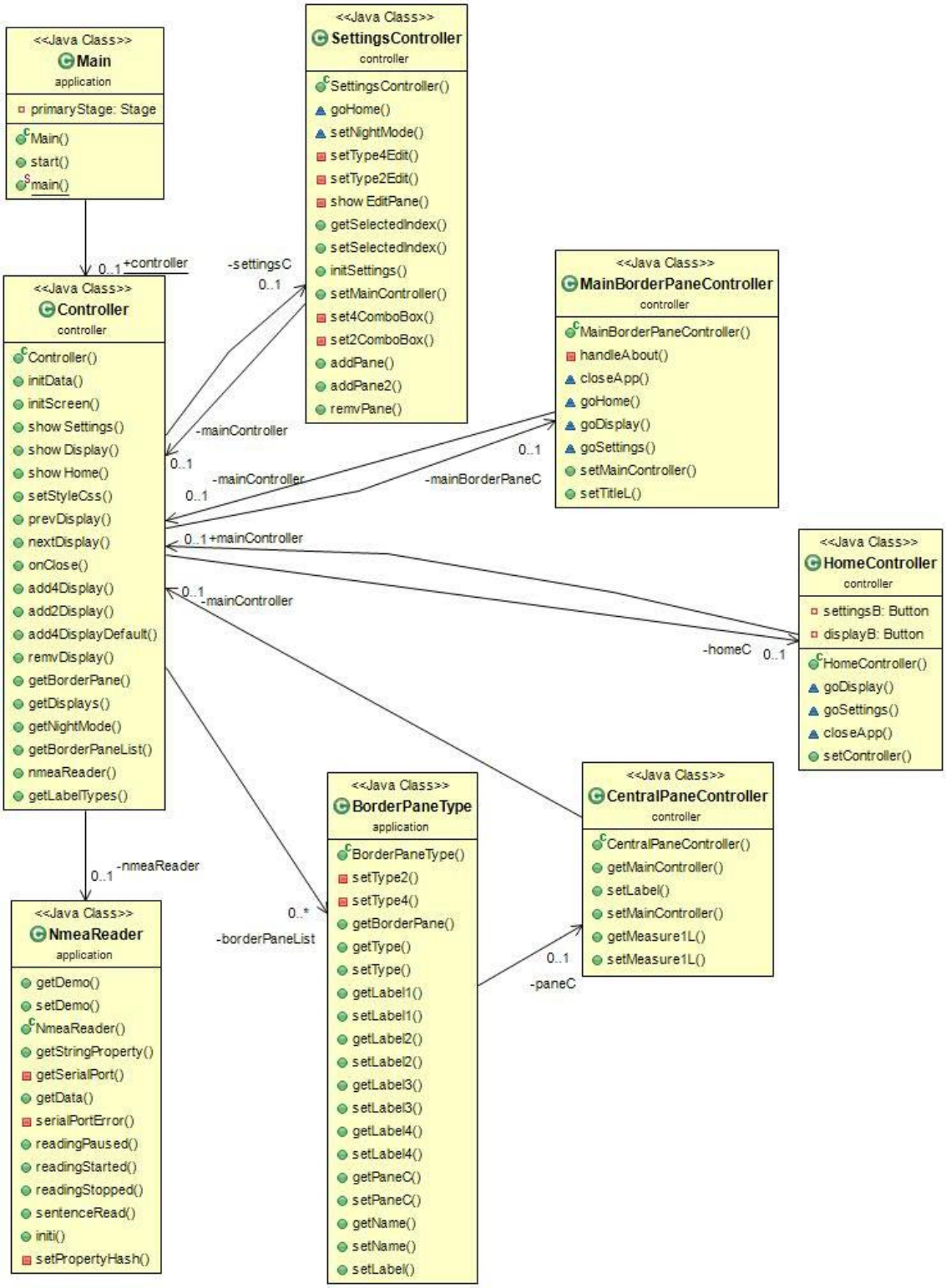
En esta sección se muestra a continuación el diagrama de clases.

El diagrama de clases muestra una vista general de las clases que conforman el proyecto y las relaciones que hay entre ellas mediante referencias e instancias de objetos.

Se puede destacar que la clase central es la clase Controller que tiene una instancia de cada una del resto de las clases que conforman la aplicación. Esto es debido a la arquitectura elegida para el desarrollo de la aplicación.

También destacar que la clase principal “Main” es simplemente un iniciador de la aplicación y un nexo con la clase Controller, por lo que solo se asocia con esa clase y no con el resto.

Ilustración 3 - Diagrama de clases.



4. Diseño

En este apartado se expondrá con mayor nivel de detalle la estructura de la aplicación. Esto facilitará posteriormente la implementación de la misma.

4.1 Arquitectura de la aplicación

A la hora de desarrollar una aplicación es muy importante pensar y analizar bien el tipo de estructura que tendrá la misma. Un buen tipo de estructura nos facilita el desarrollo, el entendimiento y el mantenimiento del software desarrollado.

La aplicación que se ha desarrollado está enfocada a mostrar información que le llega de una fuente emisora, el usuario puede navegar por la aplicación y elegir qué datos y pantallas ver.

Analizando los requisitos e investigando tecnologías, se llegó a la idea de utilizar una arquitectura que siga el patrón Modelo Vista Controlador (MVC) que permite una gestión de la interfaz separada de los datos, todo ello controlado por una clase central.

El patrón de diseño Modelo Vista Controlador MVC es un patrón muy común. También es uno de los aplicados en el grado por lo que sería una buena forma de enfocar la arquitectura del proyecto.

Patrón de diseño Modelo Vista Controlador.

El diseño modelo vista controlador (MVC) es un patrón arquitectónico que permite separar en diferentes capas la interfaz de usuario, la lógica de negocio y los datos que manejará la aplicación.

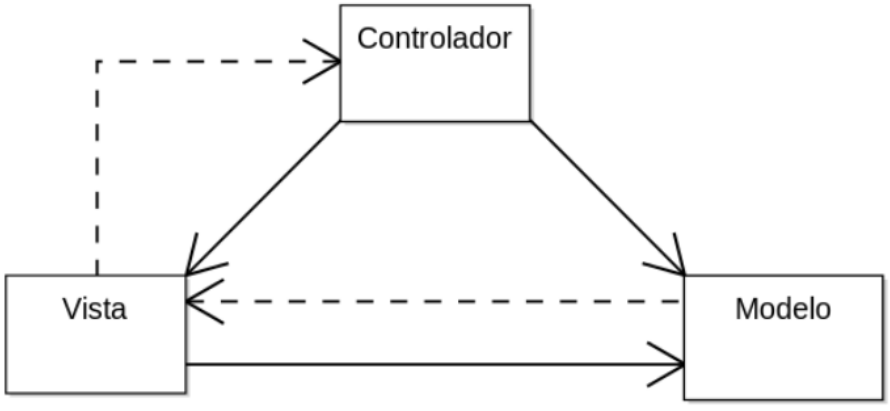
Como se describe anteriormente, se compone de tres partes diferenciadas:

- **Modelo**
  Es la parte que se encarga de la gestión, manejo y preparación de los datos que será de utilidad para el funcionamiento correcto y esperado de la aplicación.

- **Vista**
  Es la parte de presentación de la aplicación también conocida como la interfaz de usuario. Es la encargada de mostrar por la pantalla cada elemento de la aplicación y permitir la interactuación del usuario. También es la encargada de mostrar la información de salida de la aplicación.

- **Controlador**
  Es la parte encargada de gestionar los eventos generados por las acciones del usuario y por el modelo. Sirve de enlace entre la vista y el modelo, haciendo que una interacción en la vista por parte del usuario se transmita al modelo para que pueda obtener, transformar o guardar los datos asociados a la acción del usuario y también reflejar los cambios efectuados por el modelo en la vista.

Ilustración 4 – Esquema Modelo Vista Controlador



4.2 Estructura de la aplicación

La aplicación consta de tres paquetes que recogen las diferentes clases que la forman.

Tabla 16 Estructura de la aplicación: Paquete Application

Paquete Application	(Modelo)
Main.java	Clase principal que tiene la función de lanzar la aplicación.
BorderPaneType4.java	Clase se encarga de la gestión de los datos y atributos de las pantallas para visualizar los datos disponibles.
NmeaReader.java	Clase que gestiona la lectura manejo de los datos recibidos del emisor.

Tabla 17 Estructura de la aplicación: Paquete View

Paquete view	(Vista)
MainBorderPane.fxml	Archivo que define la interfaz base de la aplicación.
CentralPaneEmpty.fxml	Archivo que define la interfaz base de una pantalla para mostrar datos.
Home.fxml	Archivo que define la interfaz de la vista Inicio.
Settings.fxml	Archivo que define la interfaz de la vista Configuración.
dia.css	Archivo que define el estilo visual de los elementos de la aplicación.
noche.css	Archivo que define el estilo visual de los elementos de la aplicación.

Tabla 18 Estructura de la aplicación: Paquete Controller

Paquete controller	(Controlador)
Controller.java	Clase que sirve de nexo entre los modelos de datos y los controladores de la interfaz
CentralPaneController.java	Clase que gestiona la interacción del usuario con los elementos de la aplicación en la vista de visualización de datos.
HomeController.java	Clase que gestiona la interacción del usuario con los elementos de la en la vista de Inicio
MainBorderPaneController.java	Clase que gestiona la interacción del usuario con los elementos de la interfaz general (Barra de menú)
SettingsController.java	Clase que gestiona la interacción del usuario con los elementos de la vista de Configuración.

5. Desarrollo e implementación

En esta sección se va a explicar las herramientas y la forma en la que se ha trabajado e implementado la aplicación con respecto a los análisis anteriores.

Se explicará el entorno de trabajo con el conjunto de herramientas de las que se han hecho uso para el desarrollo de la aplicación.

También se explicará qué tecnologías han sido elegidas para la implementación explicando sus principales características y detallando algunas importantes.

5.1 Entorno de desarrollo.

En este apartado se describe el entorno de desarrollo que ha sido utilizado, tanto el hardware como las herramientas software más relevantes.

Es importante un buen entorno de desarrollo ya que facilita mucho la implementación ya sea con ayudas y atajos del software utilizado como la facilidad de acceso, análisis y pruebas de todos los componentes relacionados.

5.1.1 Hardware

El hardware en el caso de este proyecto viene delimitado por el enfoque a un dispositivo de bajo coste como la Raspberry pi / Arduino.

Por otra parte también se limita a las herramientas disponibles y asequibles por parte del desarrollador.

5.1.1.1 *Ordenador Portátil*

Para el desarrollo principal se ha utilizado un ordenador portátil (en este caso un Toshiba Satellite L850-1RZ) que permite una movilidad para poder desarrollar en casi cualquier lugar, ya sea una biblioteca, en casa o para poder compartir el estado del desarrollo con el tutor en su despacho.

Sus características principales son las siguientes:

- Windows 10 Pro 64x
- Disco SSD 120GB
- 8GB de memoria RAM
- Pantalla de 15.6”
- Gráficos AMD Radeon™ HD 7670M

5.1.1.2 *Raspberry pi 3*

En cuanto al hardware objetivo se ha elegido una Raspberry pi 3 debido a que tiene una relación calidad/potencia/precio que nos resulta útil para este proyecto. Los dispositivos Raspberry pi cuentan con una comunidad muy grande detrás y la información disponible en la red es abundante. Por eso mismo en caso de dudas, problemas o errores se puede encontrar información o publicar en muchos de los foros que están enfocados al dispositivo y su entorno.

También cuenta con muchos accesorios fácilmente instalables como ratones y teclados inalámbricos, y la pantalla táctil que se ha utilizado para poder visualizar e interactuar con la aplicación también se puede alimentar con una fuente de 5v y un adaptador micro USB (como por ejemplo una batería externa).

5.1.2 Software

Aquí, se comenta el software utilizado para la implementación de la aplicación.

El ordenador de desarrollo funciona bajo el sistema operativo Windows 10.

La Raspberry pi funciona bajo el sistema operativo Raspbian.

5.1.2.1 Raspbian.

Se ha utilizado el sistema operativo Raspbian y se ha procedido a su instalación desde Windows con la aplicación NOOBS.

5.1.2.2 Eclipse.

Eclipse es un IDE muy popular y muy completo para el desarrollo en Java.

A lo largo de la carrera se nos han presentado diferentes programas para el desarrollo e implementación de programas de software.

Entre ellos empezamos con BlueJ, Eclipse, Netbeans…

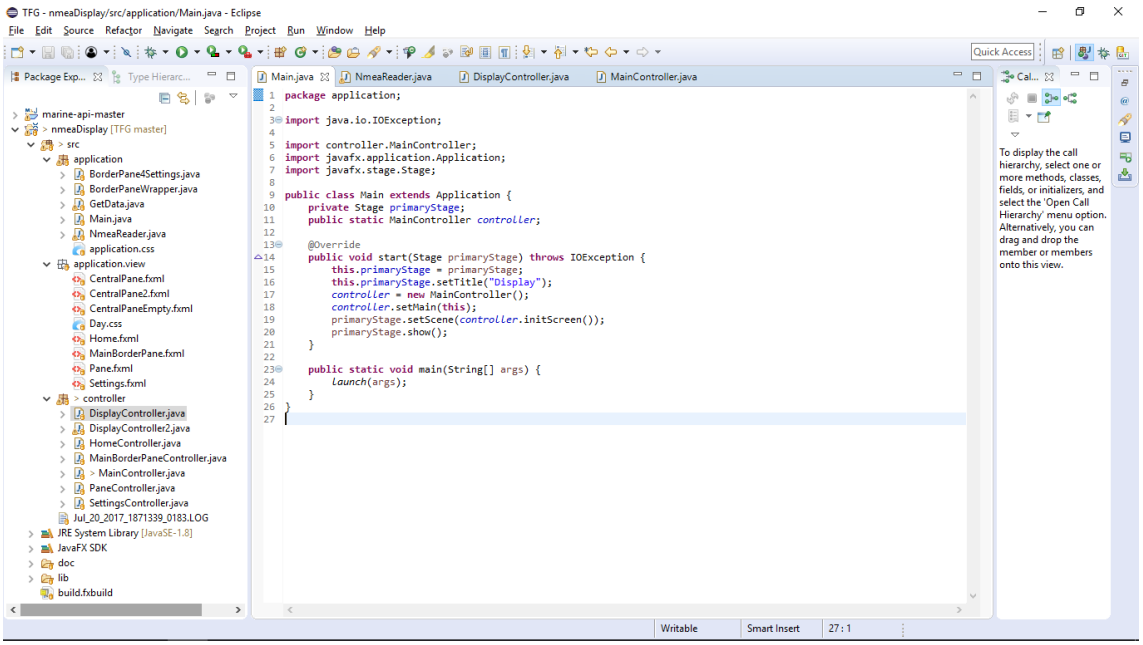
De todos los presentados, eclipse es el programa con el que más tiempo he trabajado, por ello lo he elegido para el desarrollo de este proyecto.

Contiene un gran número de herramientas y funciones que ayudan al desarrollador en su tarea. Incluye un editor de código fuente, también un compilador para poder ejecutar el proyecto. Además contiene una herramienta de depuración de código que permite ejecutar de manera controlada el código para ver el flujo del mismo, encontrar errores y controlar la estructura general de la aplicación en cada momento del desarrollo.

Otras características que aporta que han ayudado a la elección de este IDE son la capacidad de autocompletar código, el autoformato para mayor legibilidad y estructuración, la navegación entre las llamadas a funciones, herencia y definición de las mismas así como las sugerencias de tipos de objetos.

Por último resaltar la gran versatilidad que consigue este IDE con la posibilidad de instalación de complementos que le permiten ganar funcionalidades específicas tanto para el desarrollo como para la integración de Git y el control de versiones.

Ilustración 5 -Captura del Eclipse.



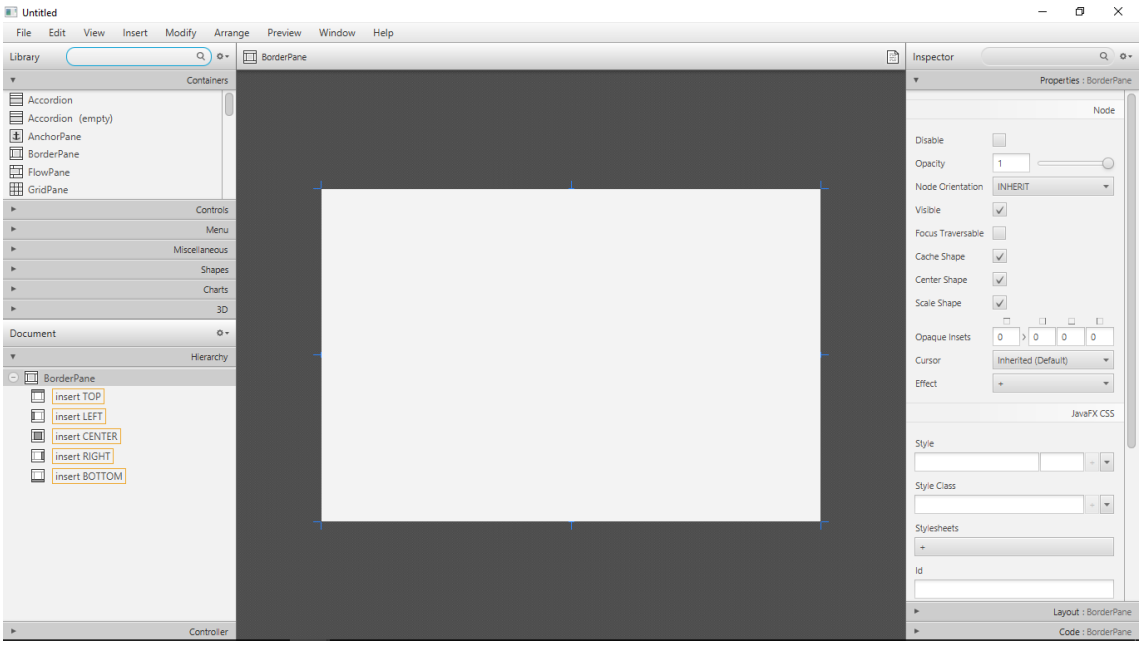
5.1.2.3 *E(fx)clipse.*

Complemento de eclipse que permite el manejo de los archivos utilizados para el desarrollo de aplicaciones con JavaFX tales como archivos *.fxml , *.css y manejo de su sintaxis específica. También integra la utilización del programa Scene builder para la interfaz gráfica.

5.1.2.4 *JavaFX Scene Builder 2.0*

Programa que permite la creación de la interfaz gráfica de una manera muy intuitiva. Evita la tediosa tarea de definir y configurar una interfaz de usuario mediante código. Su funcionalidad consiste en arrastrar los elementos que se quieren colocar y después editar sus atributos como sus medidas, color, identificador para luego acceder a dicho elemento en el código. El archivo generado es un .fxml que también se puede editar desde eclipse con código y utiliza el lenguaje xml.

Ilustración 6 - Captura de Scene Builder.



5.1.2.5 objetaid Class Diagram

Complemento de eclipse para la creación del diagrama de clases. Ha sido elegido por su facilidad de uso, ya que se crea el diagrama, se arrastran las clases que se quiere mostrar y se relacionan entre ellas automáticamente. Se puede editar qué elementos se muestran (atributos, métodos, tipos de retorno etc). Por último, permite exportar el diagrama en forma de imagen.

5.1.2.6 Git

Git es un software controlador de versiones diseñado por Linus Torvalds.

Permite tener un historial de los cambios realizados en un servicio en la nube como gitLab. Gracias al uso de git se puede tener un control de qué versión del código se está utilizando, y concede volver hacia atrás en el historial para deshacer cambios no deseados.

También es útil para poder trabajar en el proyecto desde varios dispositivos y tener todos los cambios sincronizados y evitar conflictos.

5.1.2.7 GitLab

Servicio web utilizado para el control de versiones. Permite alojar tus proyectos en la nube y poder acceder a ellos desde cualquier parte con cualquier gestor de Git.

Es un servicio gratuito (aunque existe versión de pago) y de código abierto.

5.1.2.8 Git for Windows

Para el control de versiones del código como se menciona anteriormente se ha utilizado la tecnología git.

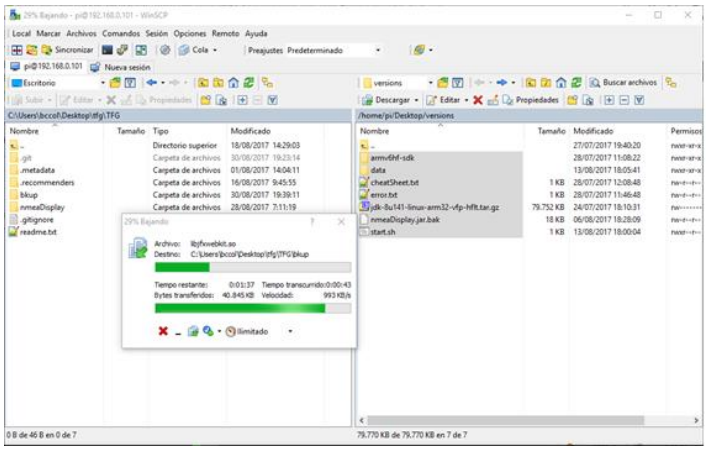
Con el programa **Git for Windows** se pueden realizar todas las funciones que proporciona git desde una consola en Windows.

Este programa nos permite tener constancia de los cambios realizados en el código y así controlar que los cambios sean correctos antes de subir el código al repositorio de GitLab. Por otra parte, permite tener un historial de los cambios realizados por si hay que revertir alguna modificación.

5.1.2.9 WinScp

Aplicación cuya función principal es la transferencia de archivos entre sistemas informáticos en nuestro caso el entorno de desarrollo (ordenador portátil) y el entorno de pruebas (Raspberry pi 3). Utiliza el protocolo SFTP (SSH File Transfer Protocol) para la comunicación entre dispositivos y la transferencia de archivos.

Ilustración 7 - Captura de WinSCP.



5.1.2.10 Putty

Cliente que utiliza el protocolo ssh para comunicar dos equipos. Se caracteriza por su sencillez para configurar y ser software libre. Posibilita el manejo por consola de la Raspberry pi desde el ordenador portátil.

5.1.2.11 WebApp moqups

Aplicación web que admite la creación de mockups de manera rápida, sencilla y exportarlos en diferentes formatos.

https://app.moqups.com/

5.2 Tecnologías de Implementación

Para este proyecto se ha elegido utilizar el lenguaje de programación de Java junto con la tecnología de JavaFX. La comunicación que se recibe del aparato emisor es la definida por el estándar nmea 0183. Scene Builder

5.2.1 Java

En el grado hemos aprendido a programar en este lenguaje. Además, **Java** es independente de la plataforma y permite desarrollar una vez un programa en un sistema operativo (Windows 10 en este caso) y ejecutarlo en otros compatibles con la máquina virtual de java (JVM).

Java es un lenguaje de propósito general, no está enfocado a una función concreta, permite desarrollar aplicaciones con muchos propósitos diferentes, captura de datos, conexión con bases de datos, conexión con servicios web, aplicaciones multimedia, visores de datos etc.

Al ser uno de los más utilizados en el mundo cuenta con gran cantidad de complementos y librerías que facilitan el desarrollo de aplicaciones.

Además, es un lenguaje de programación orientado a objetos (OO). Un objeto puede definirse como un paquete que contiene el "comportamiento" (el código, las funciones y la lógica) y el "estado" (datos) de parte del programa.

5.2.2 JavaFX

JavaFX es una tecnología de software que, al combinarse con Java en forma de API, permite crear y desplegar aplicaciones.

Facilita el diseño de la interfaz de la aplicación y proporciona herramientas para añadir funcionalidades y modificar el aspecto visual con la compatibilidad con CSS.

JavaFX tiene como principales características la integración de contenidos gráficos y multimedia. Además tiene compatibilidad nativa con dispositivos táctiles.

5.2.3 Xml

Como hemos comentado, para el desarrollo de la interfaz se utiliza el programa Scene Builder que nos genera archivos *.fxml. Estos trabajan con el lenguaje xml.

El lenguaje XML, de las siglas en inglés de *eXtensible Markup Language* que se traduce al español como "Lenguaje de Marcado Extensible", es un meta-lenguaje que permite definir tus marcas (etiquetas) para estructurar la información que contiene el archivo y poder así utilizarla en tu programa.

5.2.4 Css

JavaFX proporciona compatibilidad con CSS para editar el aspecto visual de la interfaz desarrollada.

CSS, de las siglas en inglés *Cascading Style Sheets* traducido como “Hojas de Estilo en Cascada”, es un lenguaje de diseño gráfico que describe estilos de los elementos presentes en un documento estructurado, en este caso *.fxml.

Con CSS se puede definir para cada elemento con qué características será representado. Por ejemplo color, tamaño, fuente del texto, separación entre elementos etc.

5.2.5 Api utilizadas

5.2.5.1 Java Marine API.

Java Marine API es una API diseñada para el acceso a datos provenientes del estándar **NMEA 0183**.

Mediante su uso es posible atender una entrada de tramas NMEA 0183 y extraer los datos que contiene para su uso.

5.3 Visión general de implementación

Como se ha comentado anteriormente se ha aplicado el patrón Modelo Vista Controlador. Es muy útil porque nos permite separar las capas de presentación con la de lógica.

5.3.1 Capa presentación

Esta capa es la encargada del diseño de la interfaz de usuario.

Para su implementación se ha de diseñar cada una de las pantallas que ha de tener el programa para obtener una base de trabajo.

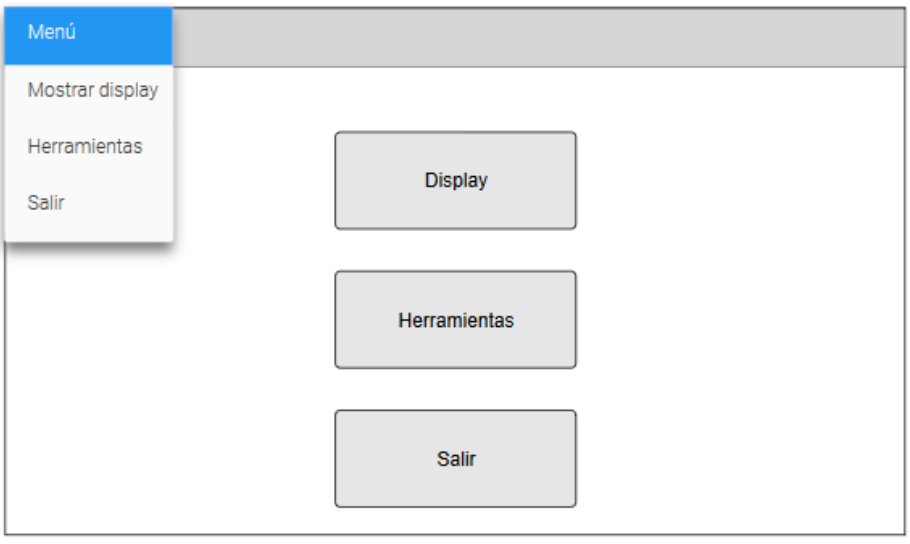
De este modo, se han utilizado mockups (diseñados con la herramienta de la webApp **moqups**) que representan el diseño inicial que se ha de implementar.

Para la implementación se ha manejado la herramienta SceneBuilder 2.0 que genera archivos *.fxml editables tanto por el programa como por código.

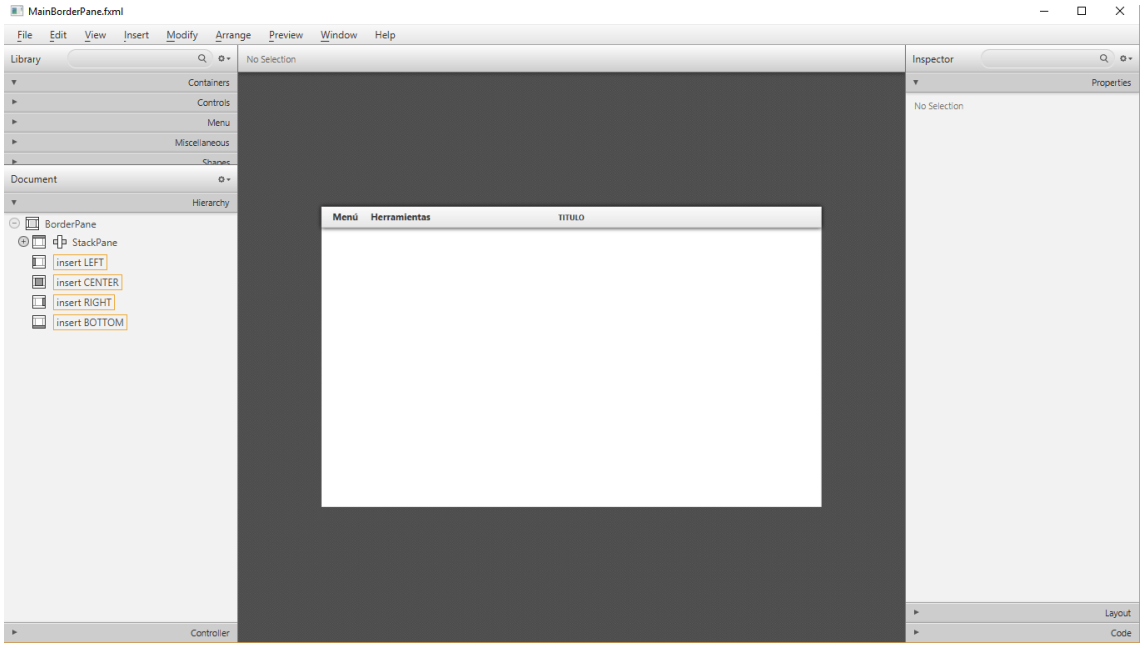
Pantalla principal.

En un principio se iba a representar la pantalla de inicio como la principal. Pero se ha separado, se ha utilizado la pantalla principal como esqueleto de la aplicación con los menús superiores y sobre ella, se añaden las diferentes pantallas que se mostrarán.

Mockup:



Scenebuilder:



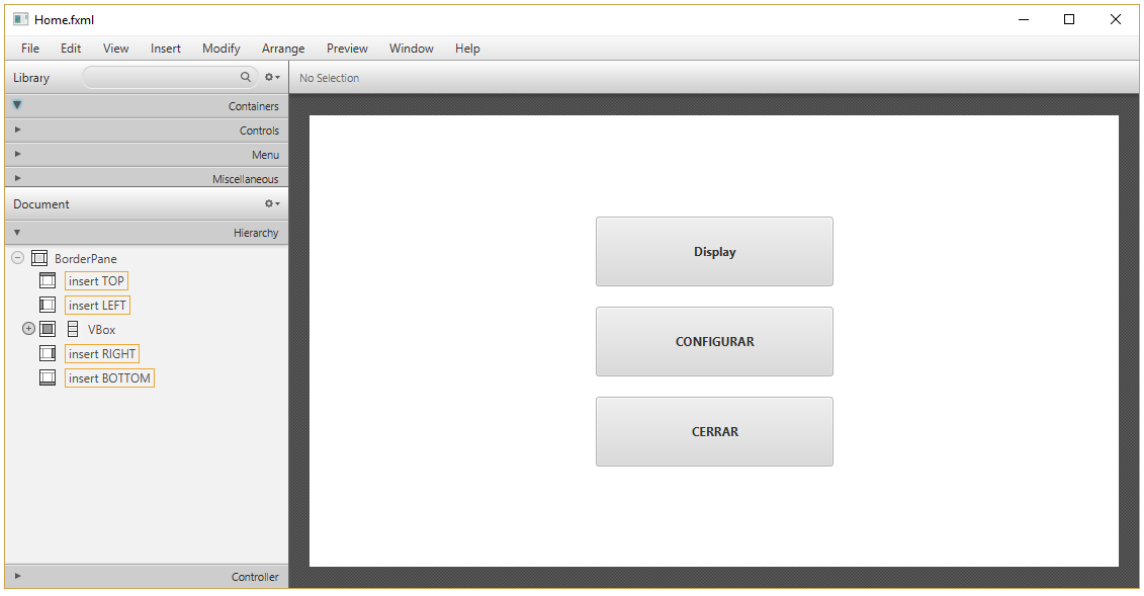
Pantalla de inicio.

Pantalla que contiene permite el acceso a las otras pantallas y el cierre de la aplicación.

Mockup:

Está contenido en el mockup de Pantalla de inicio

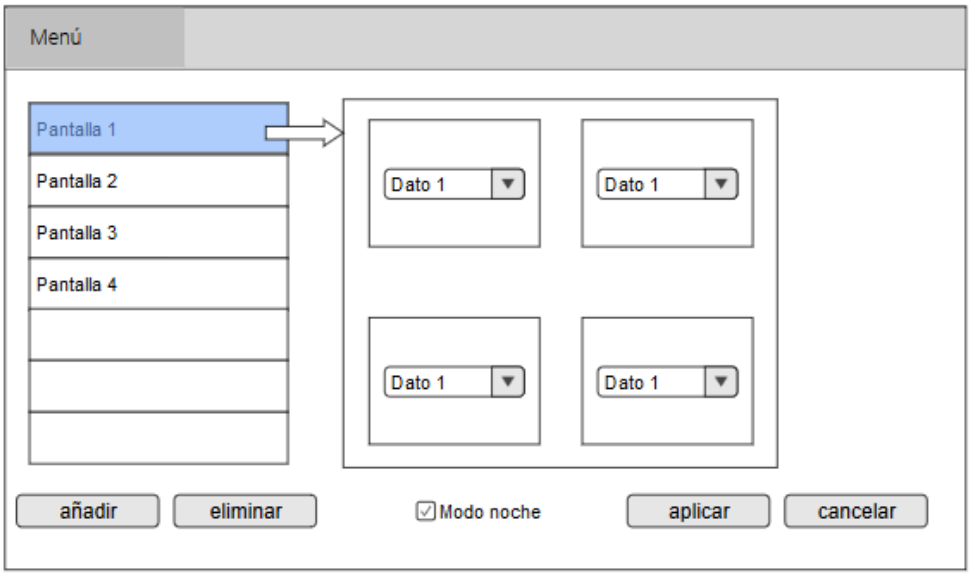
Scenebuilder:



Pantalla de configuración.

Pantalla que contiene los elementos para la configuración de las pantallas de muestra de datos y más herramientas.

Mockup:

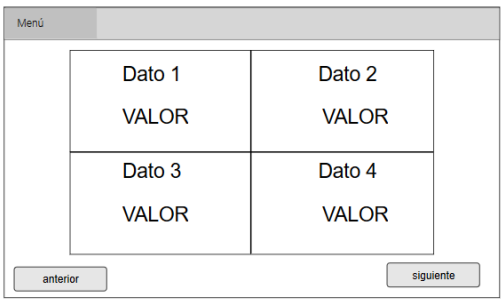


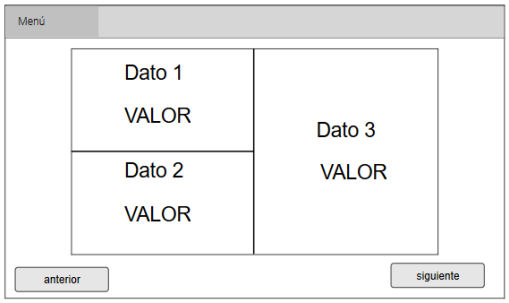
Scenebuilder:

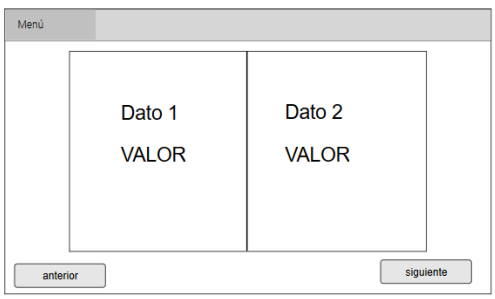
Pantalla de visualización de datos.

Pantalla que contiene los datos a visualizar y los botones de navegación entre pantallas. Se han implementado la de dos y cuatro datos.

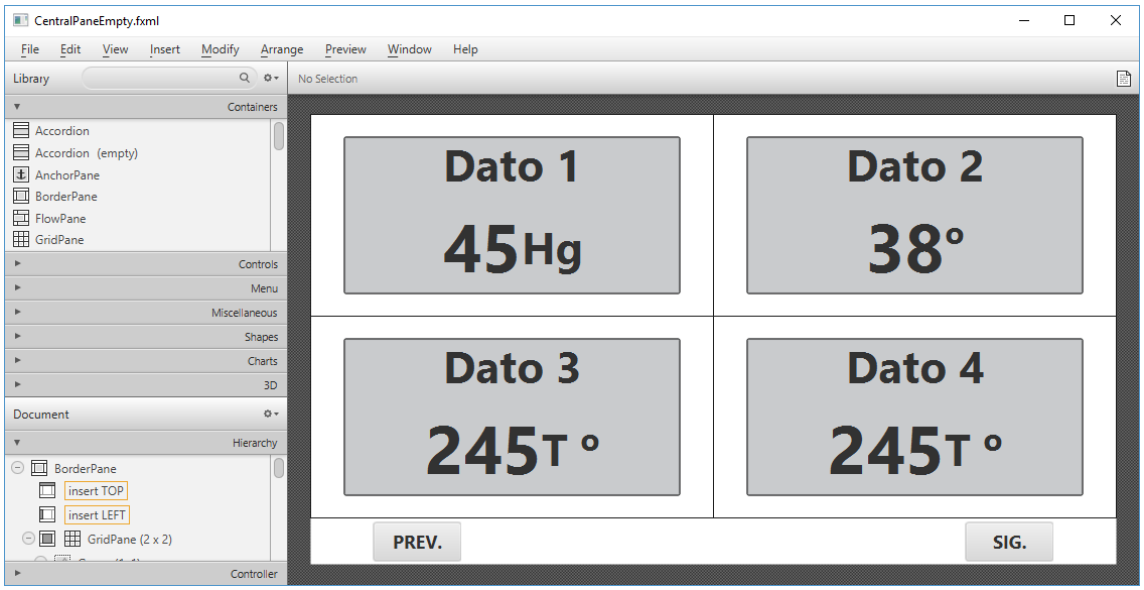
Mockup:







Scenebuilder:



5.3.2Capa lógica

Nuestra aplicación tiene una clase central que se encarga de la lógica de la aplicación.

Esa clase es CoCntroller.java.

Por una parte esta clase controlador gestiona:

1. **Inicialización de la aplicación**

- initScreen()

  Este método es llamado al inicio de la aplicación. Es el encargado de cargar la ventana principal (un BorderPane), poner en el centro la pantalla Inicio, aplicar el estilo a la aplicación y devolver la escena configurada al método Main para que la muestre al inicio.

```
public Scene initScreen() throws IOException {
    FXMLLoader loader = new FXMLLoader();
    loader.setLocation(Main.class.getResource("view/MainBorderPane.fxml"));
    mainLayout = loader.load();
    if (mainBorderPaneC == null)
        mainBorderPaneC = loader.getController();
    mainBorderPaneC.setMainController(this);
    if (scene == null)
        scene = new Scene(mainLayout);
    setStyleCss();
    showHome();
    return scene;
}
```

2. El cambio de pantallas de la aplicación

Con los métodos:

- showSettings()
- showDisplay()
- showHome()

Dichos métodos cargan el archivo *.fxml correspondiente a cada pantalla, extraen su clase controladora, le pasa la instancia del controlador, si es necesario invoca algún método de inicialización y, por último, muestra la pantalla en la parte central de la aplicación.

```
public void showSettings() throws IOException {
    FXMLLoader loader = new FXMLLoader();
    loader.setLocation(Main.class.getResource("view/Settings.fxml"));
    BorderPane settings = loader.load();
    settingsC = loader.getController();
    settingsC.setMainController(this);
    settingsC.initSettings();
    mainLayout.setCenter(settings);
    mainBorderPaneC.setTitleL("CONFIGURACIÓN");
}
```

3. La navegación entre pantallas de datos.

Con los métodos:

- prevDisplay()
- nextDisplay()

Cada método muestra la pantalla de visualización de datos siguiente o anterior.

Para ello, se asegura que la posición en la lista de pantallas disponibles esté dentro de los límites y, si no lo está, pone el índice al principio o al final de la lista.

```
public void prevDisplay() {
    pos -= 1;
    if (pos < 0) {
        pos = borderPaneList.size() - 1;
    }
    mainLayout.setCenter(borderPaneList.get(pos).getBorderPane());
    mainBorderPaneC.setTitleL(borderPaneList.get(pos).getName().toUpperCase());
}
```

4. La gestión de pantallas

- add4DisplayDefault(String, int)
- remvDisplay(int)

Con estos métodos se añade y se elimina una pantalla de datos a la lista.

En el caso de la eliminación se renombran de nuevo las pantallas para que coincidan con el orden de aparición.

```
public void remvDisplay(int index) {
    borderPaneList.remove(index);
    displays.remove(index);
    for (int i = 0; i<displays.size();i++) {
        String name = "Display "+i;
        displays.set(i, name);
        borderPaneList.get(i).setName(name);
    }
}
```

5.3.3 Capa de datos

En esta capa se obtienen los datos a mostrar y también se guardan las pantallas configuradas.

La clase NmeaReader implementa la interfaz SentenceListener para el manejo de las tramas NMEA 0183 recibidas.

1. **Obtención de puerto USB para lectura.**

Con el método getSerialPort() de la clase NmeaReader se escanea los puertos del dispositivo en busca de una trama NMEA 0183.

```
71  private SerialPort getSerialPort() {
72      try {
73          Enumeration<?> e = CommPortIdentifier.getPortIdentifiers();
74          while (e.hasMoreElements()) {
75              CommPortIdentifier id = (CommPortIdentifier) e.nextElement();
76
77              if (id.getPortType() == CommPortIdentifier.PORT_SERIAL) {
78
79                  SerialPort sp = (SerialPort) id.open("puertoEjemplo", 30);
80
81                  sp.setSerialPortParams(4800, SerialPort.DATABITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
82
83                  InputStream is = sp.getInputStream();
84                  InputStreamReader isr = new InputStreamReader(is);
85                  BufferedReader buf = new BufferedReader(isr);
86
87                  System.out.println("Escaneando puerto " + sp.getName());
88
89                  for (int i = 0; i < 5; i++) {
90                      try {
91                          String data = buf.readLine();
92                          if (SentenceValidator.isValid(data)) {
93                              System.out.println("NMEA data found!");
94                              return sp;
95                          }
96                      } catch (Exception ex) {
97                          ex.printStackTrace();
98                      }
99                  }
100                 is.close();
101                 isr.close();
102                 buf.close();
103             }
104         }
105         System.out.println("NMEA no encontrado...");
106
107     } catch (Exception e) {
108         e.printStackTrace();
109     }
110     return null;
111 }
```

Estándar comunicación en serie NMEA 0183

Typical Baud rate	4800
Data bits	8
Parity	None
Stop bits	1
Handshake	None

1. Primero se obtiene la lista de identificadores de puertos en la línea 73.
2. Por cada identificador obtenido se comprueba que es uno de tipo PORT_SERIAL que corresponde con el USB.
3. Se abre el puerto para la lectura de datos. Línea 79.
4. Después se configura el puerto a los estándares para el NMEA 0183, línea 81

5. Después se inicializan los elementos para la lectura del puerto. Línea 83-85
6. Se comprueba varias veces si los datos recibidos son tramas válidas NMEA0183 con la función isValid(data) que proporciona la API nma marine
7. En caso de ser válida la trama se devuelve el puerto abierto en la línea 94. En caso de no ser válido ninguno de los puertos se devuelve null línea 110.

2. Lectura de datos

Una vez se ha obtenido el puerto válido se pasa a configurar el objeto encargado de leer las tramas (SentenceReader reader)

```
InputStream stream = null;
sp = getSerialPort();
if (sp != null) {
    stream = sp.getInputStream();
}

reader = new SentenceReader(stream);
reader.addSentenceListener(this);
reader.start();
```

De esta manera se inicializa un nuevo SentenceReader y se le pasa como parámetro el flujo de datos que se recibe del puerto USB obtenido anteriormente.

Con el método addSentenceListener() se le pasa la clase que va a gestionar el manejo de las tramas leídas, por lo que, como entrada, recibe la instancia de la propia clase que va a ser la encargada de ello.

Por último se inicia la lectura con el método start().

3. Gestión de datos leídos

La implementación de la interfaz SentenceReader nos obliga a implementar las siguientes funciones:

Las tres primeras serían para gestionar los cambios en el estado de lectura. Se han implementado simplemente indicando el estado por consola para el desarrollo.

```
void readingPaused();
void readingStarted();
void readingStopped();
```

La implementación más importante es la siguiente.

```
void sentenceRead(SentenceEvent event);
```

A este método se le llama cada vez que el objeto SentenceReader recibe una trama. La API se encarga de hacer la llamada después de comprobar que la trama recibida es válida utilizando para ello un patrón que forman las siguientes expresiones regulares.

Si la trama no incorpora un checksum:

"^[$|!]{1}[A-Z0-9]{3,10}[,][\\x20-\\x7F]*(\\r|\\n|\\r\\n|\\n\\r){0,1}$"

Si la trama incorpora un checksum:

"^[$|!]{1}[A-Z0-9]{3,10}[,][\\x20-\\x7F]*[*][A-F0-9]{2}(\\r|\\n|\\r\\n|\\n\\r){0,1}$"

Con esta implementación hemos de proporcionar la funcionalidad de la clase para el manejo de tramas recibidas.

El primer paso a realizar es el de analizar qué trama es la que nos ha llegado:

```
switch (event.getSentence().getSentenceId())
```

Con esta línea de código se evalúa la ID de la trama recibida y según de qué tipo sea se procederá al manejo de los datos.

Por ejemplo:

```
case "MWD":
    MWDSentence mwd = (MWDSentence) event.getSentence();
    Platform.runLater(() -> {
        // Velocidad del viento
        vViento.setValue(Double.toString(mwd.getWindSpeed()));
        // Dirección del viento
        dViento.setValue(Double.toString(mwd.getTrueWindDirection()));
    });
    break;
case "VTG":
    VTGSentence vtg = (VTGSentence) event.getSentence();
    Platform.runLater(() -> {
        // Velocidad en nudos
        velocidad.setValue(Double.toString(vtg.getSpeedKnots()));
    });
    break;
```

En caso que la Id de la trama sea “MWD” se transforma la trama a una trama de tipo MWDSentence que es la que gestiona las tramas MWD.

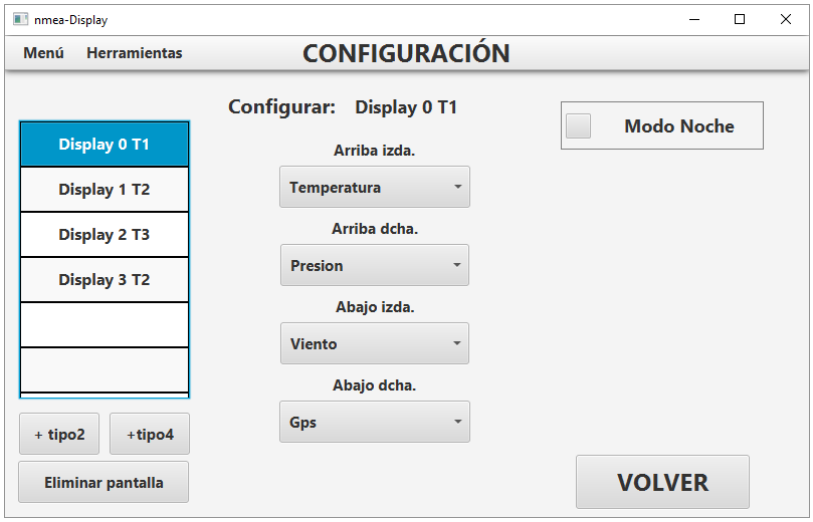
Después con la función Platform.runLater se gestiona el contenido y se asocia a las *property* que guardan y enlazan el dato con lo que se mostrará en la interfaz.

En caso que la trama lleve asociada más de un tipo de dato, se asociará cada uno de ellos con la *property* apropiada como podemos ver en la imagen con la trama de tipo “MWD” que asocia los datos de velocidad del viento y el de dirección del viento.

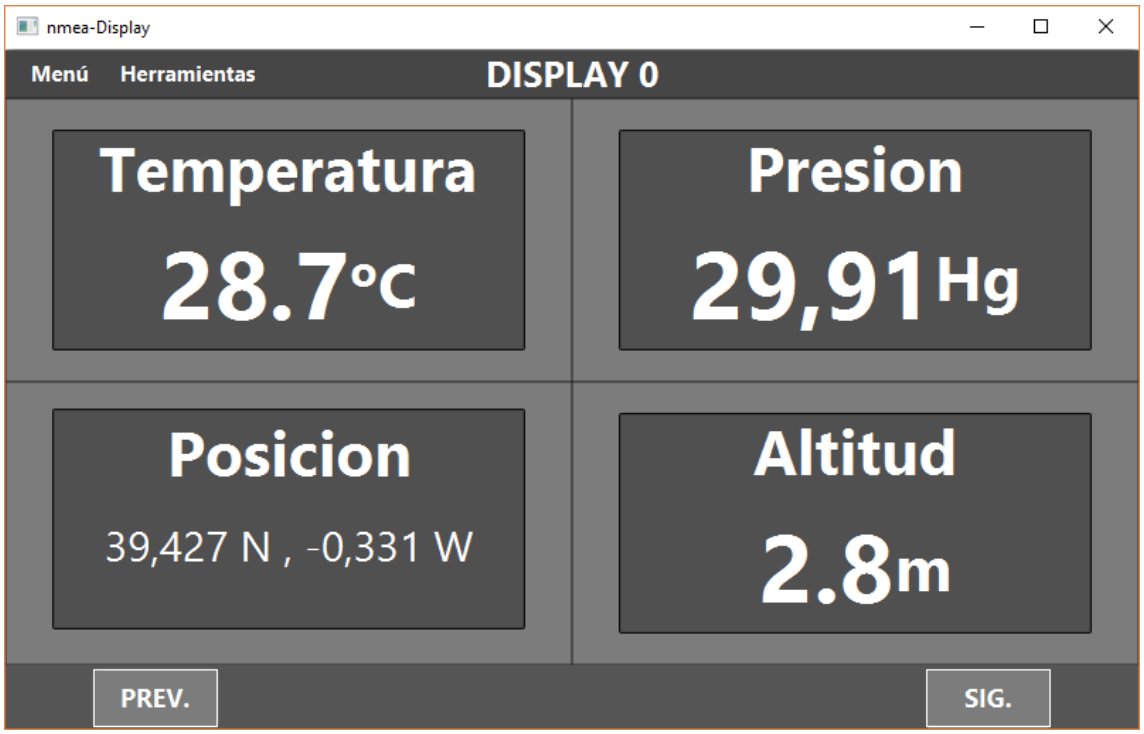
5.3.4Capturas de pantalla de la aplicación

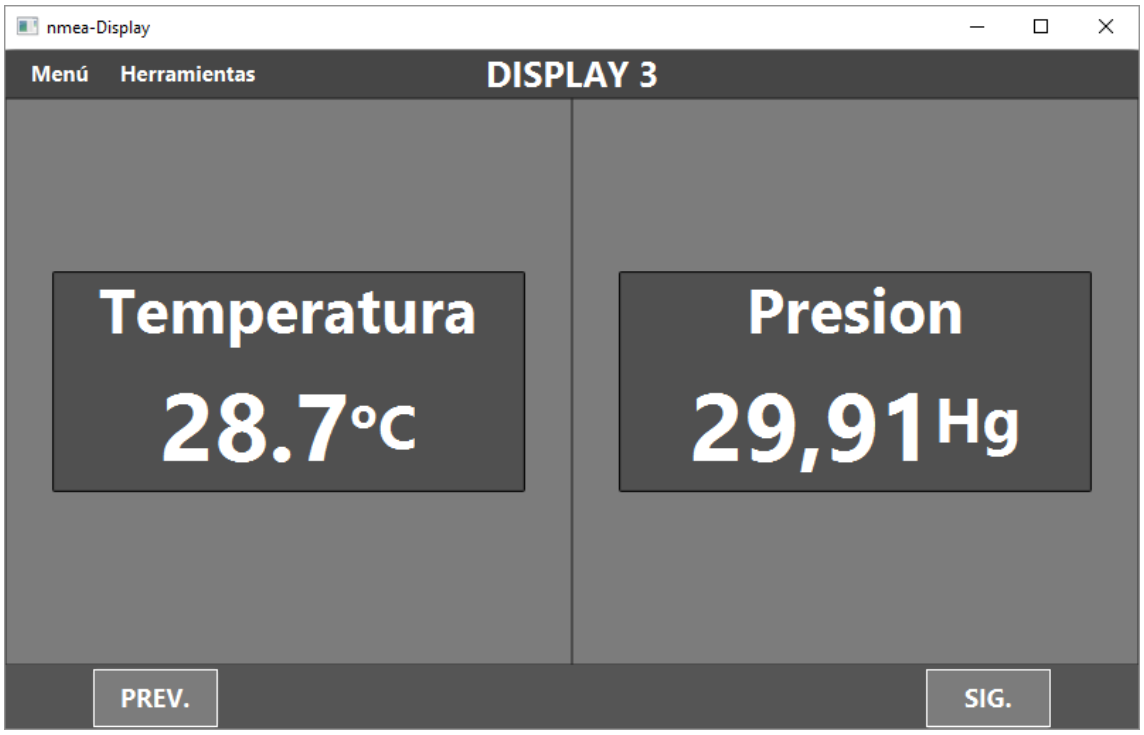
Pantalla de inicio

Configuración



Display (modo noche)

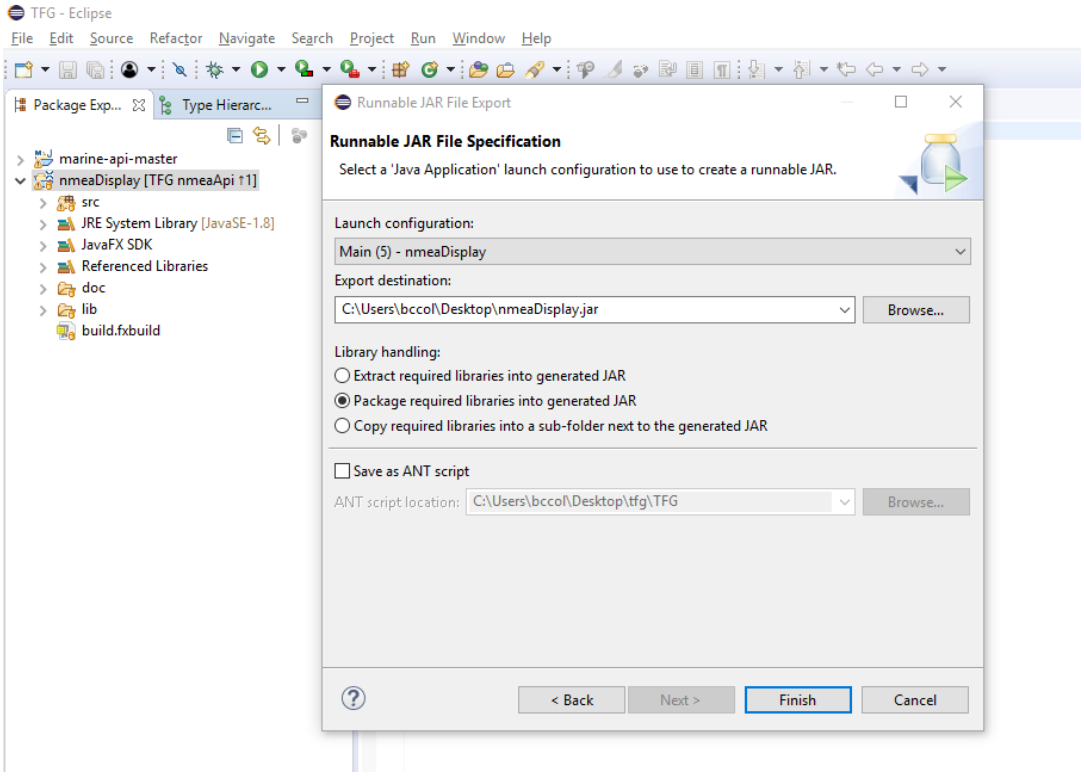




6. Despliegue y pruebas

Una vez realizado el desarrollo de la aplicación se ha utilizado la función de Eclipse de exportar la aplicación a un archivo .jar ejecutable el cual contiene las librerías necesarias para su ejecución. El archivo .jar es ejecutable en aquellos sistemas compatibles con java.

Ilustración 8 - Importación de la aplicación.



Al empaquetar las librerías necesarias se facilita la portabilidad de la aplicación ya que no es necesario instalar las librerías en el dispositivo que vaya a hacer uso de la aplicación.

Mediante el programa WinScp se ha establecido conexión y se ha transferido el archivo a la Raspberry pi. Luego se le ha dado permisos de ejecución.

Las pruebas se han ido realizando en los dos entornos de desarrollo.

En el portátil de desarrollo con eclipse, su compilador y la herramienta de debug se ha ido probando los cambios que poco a poco se iban implementando. La herramienta de debug nos permite inspeccionar el estado de los objetos y del

programa en tiempo de ejecución, pudiendo comprender el origen de los posibles fallos que vayan ocurriendo a lo largo del desarrollo.

Por otra parte, se ha ido probando también en la Raspberry pi para ver que el comportamiento fuera el mismo que en el portátil y estudiar los posibles problemas específicos a la plataforma que puedan surgir.

7. Conclusiones

7.1 Conclusiones generales

El trabajo final de grado ha sido una experiencia enriquecedora para mí, tanto como en el aporte de nuevos conocimientos así como a nivel personal, teniendo un valor importante para mi futuro.

Esta herramienta para desarrollar aplicaciones JavaFX era desconocida para mí hasta la realización de este proyecto ya que en el la asignatura de Interfaces Persona Computador (11556) utilizamos una herramienta ya en fase de abandono por parte de sus desarrolladores.

He aprendido cómo encontrar información valiosa sobre tecnologías a utilizar, así como a desenvolverme con tecnología nueva para mí.

También he obtenido conocimientos de cómo calcular la estimación de trabajo necesaria y el tiempo adecuado para realizar tareas. Esto es importante ya que ayuda a la planificación de las nuevas implementaciones futuras.

Ha sido un trabajo duro con sus problemas y resoluciones, cuya elaboración me ha ayudado en mi formación como informático y profesional en el sector.

8. Anexos

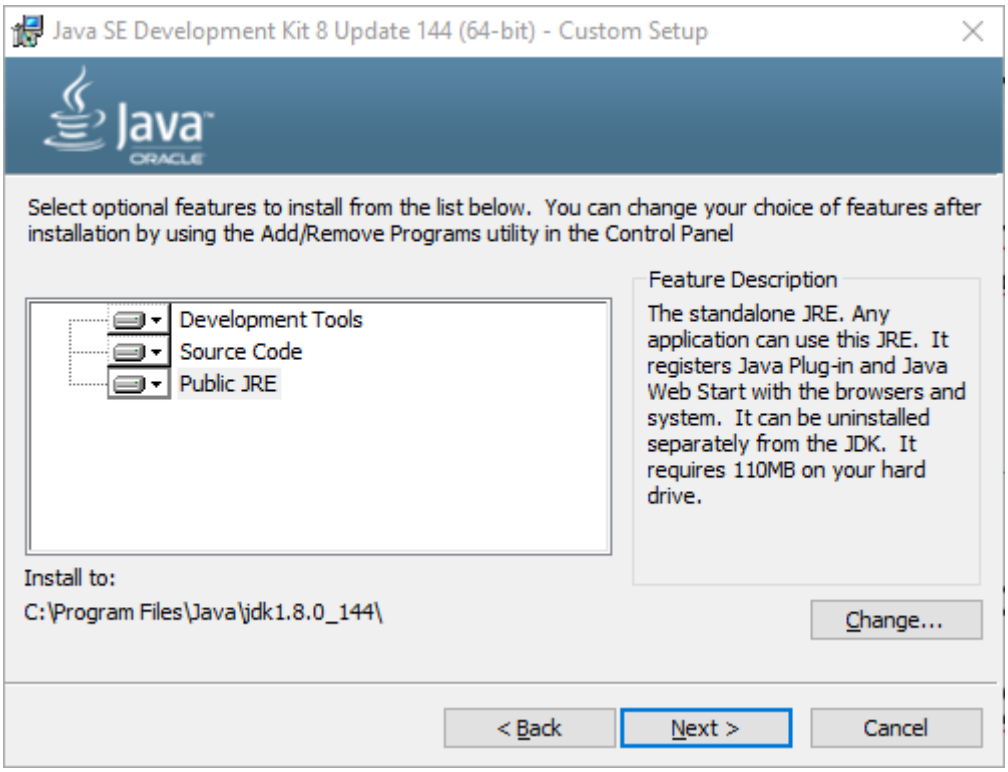
8.1 Configuración del entorno para JavaFX

Java

Para la utilización y el desarrollo con la tecnología JavaFX es necesario tener java instalado con la última versión del JDK (Java Development Kit).La instalación de JDK incluye JavaFX de base para poder desarrollar con esta tecnología.

La instalación es sencilla, se ha de descargar la versión acorde al sistema operativo (Windows 10 x64 en mi caso) el instalador de la página oficial de descargas y se procede seguir los pasos del instalador.

Ilustración 9 - Instalador JDK.



Eclipse

Una vez instalado Java JDK se procede a instalar el IDE Eclipse también de su página oficial.
Elegimos la versión normal para desarrolladores Java. Incluye las funcionalidades básicas de Eclipse así como Git integrado que será de gran utilidad.

Simplemente hay que seguir los pasos que indica el instalador.

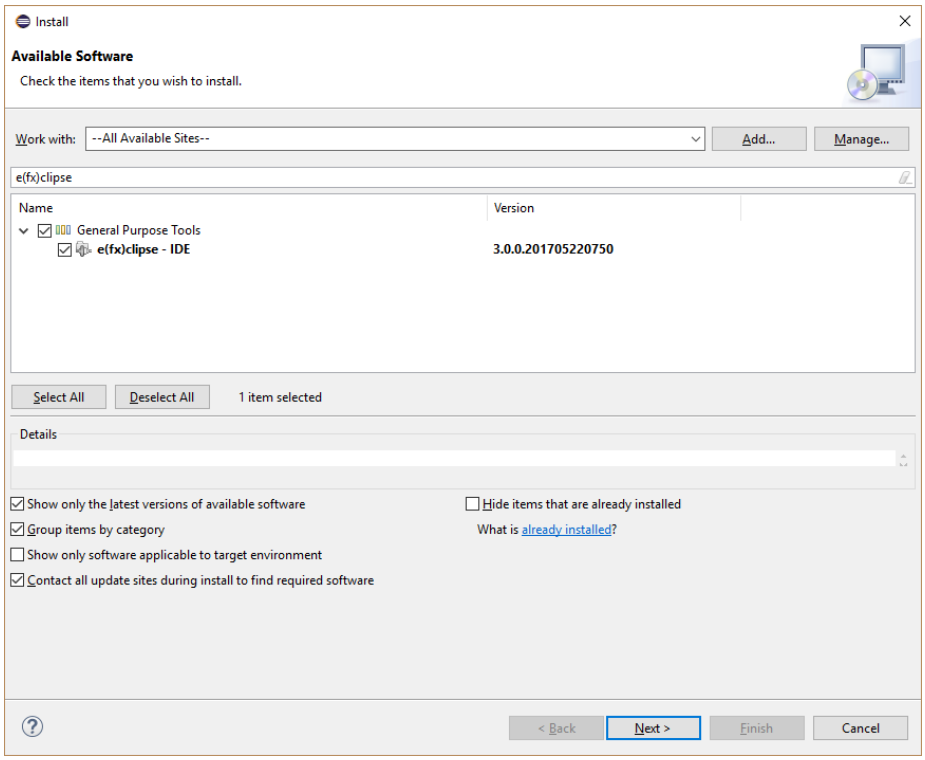
Ilustración 10 -Versión de Eclipse.

E(fx)clipse

El siguiente paso para poder trabajar con JavaFX es instalar el complemento E(fx)clipse. Para ello seguimos las instrucciones de la documentación de eclipse.

Menú Help -> Install new Software -> seleccionamos --All avaliable Sites y buscamos e(fx)clipse. Seleccionamos el elemento que aparece, le damos a next y seguimos las instrucciones en pantalla.

Ilustración 11 - Diagrama de contexto.



Scene Builder

El programa Scene Builder está desarrollado y mantenido por OpenJFX Project, además de disponible en forma de código fuente. Para poder utilizarlo hay varias opciones:

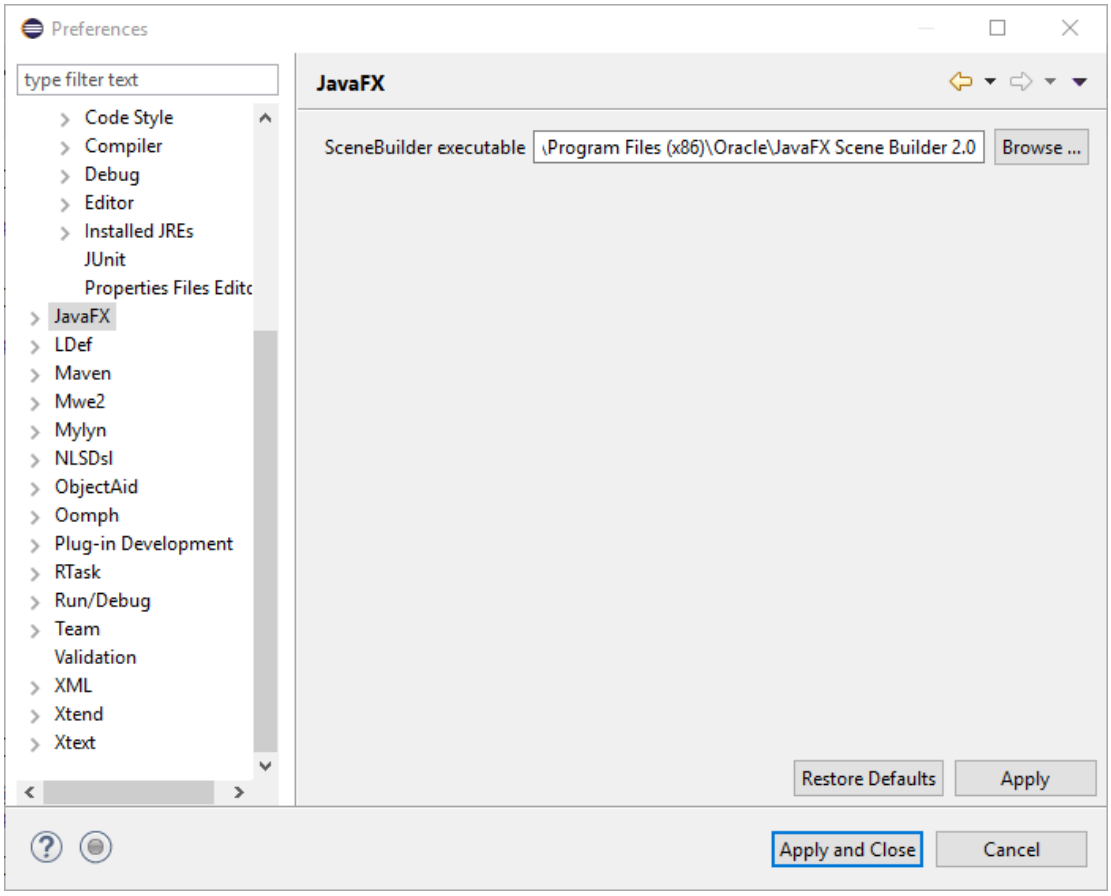
Compilarlo uno mismo para la máquina en que se vaya a utilizar o bien descargarte una versión ya compilada e instalable que esté a disposición pública.

En este caso se ha descargado el instalador ya compilado de la página oficial de Gluon.

Una vez instalado se ha de configurar la ruta del ejecutable de Scene builder para que pueda ser utilizada por el proyecto.

Para ello vamos al menú Window -> Preferences -> JavaFX e introducimos la ruta del ejecutable en el campo SceneBuilder executable.

Ilustración 12 - Configuración ruta SceneBuilder.



Con este último paso ya tenemos configurado nuestro entorno para la utilización de JavaFX.

8.2 Configuración rapberry pi para JavaFX

Desde la versión JDK de Java para los dispositivos incrustados arm eliminó la herramienta javaFx por lo que para poder utilizarla hay que instalarla manualmente. Para ello nos basamos en una de las muchas guías que encontramos en internet.

Primero instalar el jdk para dispositivos embebidos.

```
sudo apt-get install oracle-java8-jdk
```

En este trabajo se ha utilizado la siguiente de Gluon:

http://gluonhq.com/products/mobile/javafxports/get/

Se ha de descargar la versión JavaFX Embedded SDK.

Y para su instalación se ha seguido su guía en el apartado 2.1.4:

http://docs.gluonhq.com/javafxports/#anchor-1

Básicamente se ha de copiar los archivos a las ubicaciones indicadas para que el entorno java tenga acceso a ellos cuando sea lanzada la aplicación y pueda utilizar las librerías JavaFX

Archivo comprimido	**Ubicación destino**
armv6hf-sdk/rt/lib/ext/jfxrt.jar	jre/lib/ext/
armv6hf-sdk/rt/lib/arm/*	jre/lib/arm/
armv6hf-sdk/rt/lib/javafx.platform.properties	jre/lib/
armv6hf-sdk/rt/lib/javafx.properties	jre/lib/
armv6hf-sdk/rt/lib/jfxswt.jar	jre/lib/

8.3 Solución de problemas Raspberry pi

Orientación de la pantalla

Un problema encontrado ha sido que la orientación normal de la pantalla era incompatible para dejar el aparato apoyado en una superficie lisa. Esto se debe a que la entrada de alimentación queda en la parte inferior por lo que el conector estorba con la posibilidad de romperlo así como la entrada de alimentación.

Se ha solucionado cambiando la configuración de arranque modificando la línea siguiente del archivo en la ruta /boot/config.txt.

```
display_rotate=2
```

Lentitud en la respuesta del ratón

Para solucionar este error se ha de añadir el siguiente código al archivo "cmdline.txt" que se encuentra en la ruta: /boot/cmdline.txt

```
sbhid.mousepoll=0
```

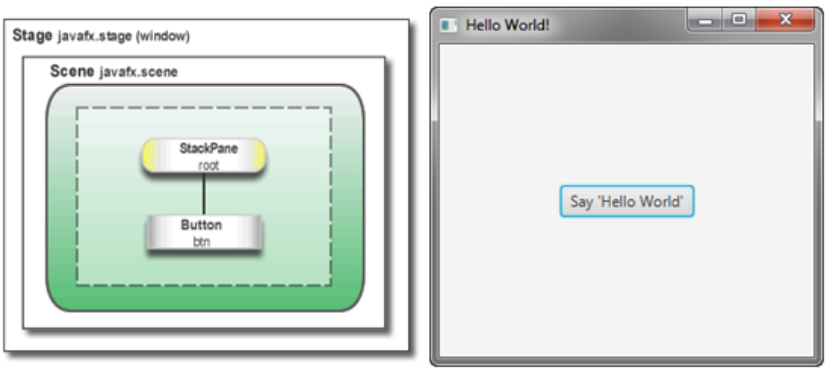
Comportamiento erróneo con las pulsaciones del ratón y táctil.

Al probar el dispositivo ocurrió que no respondía bien la aplicación a las interacciones con el ratón y mediante la pantalla táctil.

La solución es iniciar la aplicación en modo root.

8.4 Estructura base de una aplicación JavaFX

La estructura de interfaz de JavaFX está diseñada en forma de árbol y cada elemento que la compone cuelga de uno superior.



La estructura principal de una aplicación JavaFX suele seguir el siguiente esquema (De la raíz hacia las hojas del árbol).

1 Una ventana principal. (STAGE)

> **Stage** hace referencia a la ventana principal de la aplicación en el que se muestra el título y los botones de cerrar, redimensionar y minimizar.

2 Una base para el contenido.(SCENE)

> **Scene** es la base en la que se anclan los elementos de la aplicación.

3 Un elemento contenedor para los elementos de control(Pane)

> **StackPane** es un elemento contenedor que define el diseño de la distribución de los elementos internos.

4 Elementos de control

> **Button** es un elemento de control que representa a un botón y tiene la funcionalidad básica.

Con JavaFX tenemos contenedores que permiten anclar en su interior a los elementos de control de interfaz (Botones, etiquetas, menús etc.).

JavaFX tiene un hilo principal para el manejo y renderizado de la interfaz. Si se necesita hacer alguna tarea que requiera tiempo para realizar su función y ésta modifica la interfaz, se debe separar del hilo principal ya que si no bloquearía la aplicación. Para ello se utiliza para hace un hilo separado del principal el método Task facilitado por la API y a la hora de cambiar la interfaz disponemos de Platform.runLater(() -> {} que conecta el hilo en el que se ejecuta con el hilo principal de la interfaz cuando esté disponible para la actualización requerida.

8.5 Protocolo estándar NMEA 0183

El NMEA 0183 es un estándar que combina las especificaciones eléctricas y de datos entre aparatos electrónicos marinos como los receptores GPS.

Ha sido definido por la National Marine Electronics Association (NMEA)

Para la parte eléctrica sigue el estándar RS-422 que especifica las características de los aparatos emisores.

Para la parte de datos utiliza ASCII para la comunicación de datos en serie y define las tramas transmitidas por el emisor que pueden ser recibidas por varios receptores.

Las características de la comunicación en serie son las siguientes:

Typical Baud rate	**4800**
Data bits	8
Parity	None
Stop bits	1
Handshake	None

Se transmite a una tasa de 4800 baudios, cada trama tiene 8 bit, no tiene bit de paridad para corrección de comunicación, tiene un bit de parada que especifica el final de la trama no tiene protocolo de "handshaking" que coordina la comunicación entre dos dispositivos.

La estructura de las tramas es la siguiente:

Caracteres reservados:

ASCII	Hex	Dec	Use
<CR>	0x0d	13	Carriage return
<LF>	0x0a	10	Line feed, end delimiter
!	0x21	33	Delimitador de trama encapsulada (varias tramas seguidas)
$	0x24	36	Delimitador de inicio de trama
*	0x2a	42	Delimitador del checksum de la trama
,	0x2c	44	Delimitador de campo de datos
\	0x5c	92	Delimitador de bloque
^	0x5e	94	Delimitador para los caracteres representados en HEX con la ISO/IEC 8859-1 (ASCII)
~	0x7e	126	Reservado

Para los datos se utilizan todos los caracteres ASCII entre 0x20 (espacio) y 0x7e (~) menos los reservados de la tabla anterior.

Todas las tramas contienen como máximo 82 caracteres con los delimitadores incluidos

El primer carácter de la trama es para el delimitador '$' o '!'

Los siguientes 5 son para el identificador de trama. 2 para identificar la fuente y 3 el tipo de trama.

Sigue los campos de datos delimitados por comas, si un campo está vacío se dejan dos comas seguidas.

Después de los datos, si la trama tiene checksum, se encuentra un asterisco '*' seguido del resultado del mismo.

Para finalizar la trama se colocan los delimitadores <CR><LF>

Un ejemplo de trama:

$ inicio de trama, GP(dispositivo GPS), ZDA Tiempo y hora. Luego datos hora+fecha+horaLocal+minutosLocal y por último checksum.

$GPZDA , hhmmss.ss ,dd , mm , yyyy , xx , yy *CC

$GPZDA , 092642 , 20 , 07 , 2017 , 00 , 00 *42

El checksum es el resultado de hacer la operación XOR a cada carácter de la trama sin contar los elementos de principio '$' y final '*'. El resultado es un valor hexadecimal que permite la comprobación de que la trama ha sido recibida correctamente.

En caso que el resultado del checksum calculado sea diferente al recibido, la trama será inválida pero no hay un protocolo en el estándar para volver a recibirla, por lo que esta se pierde.

9. Glosario de términos

A

API:

Siglas en inglés de “Aplication Programming interface” sus siglas en español son “Interfaz de Programación de Aplicaciones”. Es un conjunto de métodos accesibles de una biblioteca que son utilizados por otro software. Se interactúa con esos métodos que procesan las peticiones y ofrecen una respuesta

B

Biblioteca:

Conjunto de implementación de funciones para su uso por otro software.

C

Checksum:

Sistema de verificación que permite la identificación de cambios accidentales en una secuencia de datos. El valor del checksum se transmite con la información para que el receptor pueda verificar la integridad de los mismos.

Complemento:

Programa adicional que modifica la funcionalidad de otro. Se integra con el programa objetivo.

I

IDE - Integrated *Development Environment*:

Siglas en ingles de Entorno de Desarrollo Integrado.
Entorno que contiene herramientas que facilitan el desarrollo de software, como por ejemplo, un editor de código fuente, autocompletado de código, compilador de software etc.

Interfaz (de usuario):

Espacio por el cual el usuario interactúa con un programa y visualiza su salida. Sirve para la comunicación usuario-máquina.

Interfaz (Clase interface):

Plantilla que contiene los métodos sin funcionalidad desarrollada cuya lógica debe ser definida por las clases que la implementen.

M

Mockup:

Boceto rápido que se realiza para representar la interfaz y funcionalidad de un programa.

P

Patrón de diseño:

Solución arquitectónica a problemas comunes en el desarrollo de software. Son soluciones que han sido probadas y documentadas que se pueden aplicar a problemas con características similares.

S

Ssh:

Siglas en inglés de Secure Shell. En español intérprete de órdenes seguro. Protocolo de acceso remoto que permite el manejo telemático mediante una consola de comandos de un equipo así como la copia de datos de manera segura.

Sftp:

Siglas en inglés de SSH File Transfer Protocol. Protocolo de transferencia segura de archivos en español. Permite el manejo de archivos remotos de manera segura.

T

Trama:

Unidad de envío de datos. Es una serie sucesiva de datos definida entre caracteres que delimitan el inicio y el final de la misma.

W

Webapp:

Aplicación alojada en la red que permite su manejo local como si estuviera instalada de manera local. Se utiliza a través de un navegador.

10. Referencias y bibliografía

Bibliografía.

[1] EBBERS, H. (julio 2014) ***Mastering JavaFX 8 Controls*.** McGraw-Hill Education**. ISBN** 978007c1833783

[2] **IEEE Std 830-1998**, Recommended Practice for Software Requirements Specifications [Última consulta: agosto de 2017]
[Fuente: http://www.cse.msu.edu/~chengb/RE-491/Papers/IEEE-SRS-practice.pdf]

[3] **JavaFX doc**[Consulta en línea][Última consulta: agosto 2017]
[Fuente: http://docs.oracle.com/javase/8/javafx/api/]

[4] **JavaFX CSS Reference Guide** [Consulta en línea] [Última consulta: agosto de 2017]

[5] **JavaFx** [Consulta en línea] [Última consulta: junio de 2017]
[Fuente: https://stackoverflow.com/questions/38359076/how-can-i-get-javafx-working-on-raspberry-pi-3/38363136#38363136 (Comentario de José Pereda del 13 de Julio de 2016)]

[6] **e(fx)clipse** [consulta en línea] [Última consulta: 10 de junio de 2017]
[Fuente: https://www.eclipse.org/efxclipse/install.html#for-the-ambitious]

[7] **css** [Consulta en línea][Última consulta: agosto de 2017]
[Fuente: https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada]

[8] **xml**] [Consulta en línea] [Última consulta: agosto de 2017]
[Fuente: https://es.wikipedia.org/wiki/Extensible_Markup_Language]

[9] **nmea data**[Consulta en línea] [Última consulta: agosto de 2017]
[Fuente: http://www.gpsinformation.org/dale/nmea.htm#nmea]

[10] **nmea** [Consulta en línea] [Última consulta: agosto de 2017]
[Fuente: https://en.wikipedia.org/wiki/NMEA_0183]

[11] **Java marine API doc** [Consulta en línea] [Última consulta: agosto de 2017]
[Fuente: https://ktuukkan.github.io/marine-api/0.10.0/javadoc/]

[12] **Noobs guía** [Consulta en línea] [Última consulta: junio de 2017]
[Fuente: https://raspberryparatorpes.net/instalacion/noobs-paso-a-paso-instalar-el-sistema-operativo-en-la-raspberry-pi/]

Referencias.

objectAid

[Fuente: http://www.objectaid.com/home]

gitLab

[Fuente: https://about.gitlab.com/]

SceneBuilder

[Fuente: http://gluonhq.com/products/scene-builder/]

Winscp

[Fuente: https://winscp.net/eng/docs/lang:es]

Putty

[Fuente: http://www.putty.org/]

Moqups

[Fuente: https://app.moqups.com/]

Eclipse

[Fuente: https://www.eclipse.org/downloads/]

JDK

[Fuente: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html]

Portátil

[Fuente: http://www.toshiba.es/discontinued-products/satellite-l850-1rz/]

Git for Windows

[Fuente: https://git-scm.com/download/win]

moqups:

[Fuente: https://app.moqups.com/]