ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Grado

Sistema generador de interfaces de usuario sobre la plataforma OSGi

AUTOR: Ángel Fernández Saura
DIRECTOR(ES): Rafael Toledo Moreo
CODIRECTOR: José Santa Lozano

Octubre / 2012

Autor	Ángel Fernández Saura
E-mail del Autor	Angelfs89@gmail.com
Director(es)	Rafael Toledo Moreo
E-mail del Director	Rafael.toledo@upct.es
Codirector(es)	José Santa Lozano
Título del PFC	Sistema generador de interfaces de usuario sobre la plataforma OSGi
Descriptores	Gestión dinámica de interfaz | arquitectura software de a bordo| ordenador de a bordo | telemática en el vehículo | sistemas inteligentes de transporte.

Resumen

En la actualidad la dispersión de los sistemas informáticos es cada vez mayor y para ello existen tecnologías que centralizan la información y los sistemas agrupándolos para mejorar su gestión.

OSGi se presenta como una de estas tecnologías, una pasarela de servicios que gestiona distintos sistemas bajo un mismo software facilitando la comunicación y el intercambio de información entre módulos, todo sobre el lenguaje de programación Java ampliamente extendido, presente en multitud de dispositivos.

La pasarela de servicios OSGi proporciona todos los sistemas base para la comunicación y gestión de sensores y componentes, pero a la hora de desarrollar un interfaz gráfico es obligatorio implementar en cada aplicación específica los componentes visuales, no pudiendo gestionar todos los elementos y aspectos visuales desde un mismo módulo que permitiría unificar el aspecto gráfico del sistema.

Este proyecto recoge esa necesidad, añadiendo esa característica a la plataforma OSGi a través de un componente denominado *middleware HMI* que generará de forma dinámica (en tiempo de ejecución) el interfaz gráfico apropiado en base a una descripción normalizada del interfaz gráfico requerido. Este módulo hará la función de intermediario entre los módulos que ofrecen servicios y la interfaz gráfica final apreciada por el usuario.

Titulación	Pasarela a grado en ingeniería de telecomunicación
Intensificación	Ingeniería telemática
Departamento	Departamento de Electrónica, Tecnologías de Computadoras y Proyectos.
Fecha de Presentación	10-2012

ÍNDICE

ÍNDICE **3**
LISTA DE FIGURAS **5**
1.- INTRODUCCIÓN **7**
1.1.- PROBLEMÁTICA 7
1.2.- INTRODUCCIÓN A LA ESTRUCTURA GENERAL DEL SISTEMA 7
1.3.- OBJETIVOS 9
1.4.- ESTRUCTURA DEL PROYECTO 10
2.-TECNOLOGÍAS UTILIZADAS **11**
2.1.- JAVA 11
2.2.- ESTÁNDAR XML EN JAVA 12
2.3.- PASARELA DE SERVICIOS OSGI 14
2.4.- SDK'S PARA LA SÍNTETIS Y RECONOCIMIENTO DE VOZ 15
3.- ESTRUCTURA Y COMPONENTES DEL SISTEMA **18**
3.1.- MARCO DE TRABAJO 18
3.2.- DESCRIPCIÓN DE LA ESTRUCTURA GLOBAL 19
3.3.- ESTRUCTURA DE LOS ELEMENTOS DEL SISTEMA 20
4.- TRATAMIENTO DE DATOS CON XML **23**
4.1.- DESCRIPCIÓN XML DEL INTERFAZ GRÁFICO 23
4.2.- PROCESADO DE INTERFACES GRÁFICAS 24
5.- INTERFAZ DE LAS APLICACIONES CON EL MIDDLEWARE HMI **25**
5.1.- INTERFACES SOBRE OSGI 25
5.2.- SERVICIOS SOBRE OSGI 25
5.3.- REGISTRO Y ADHESIÓN A UN SERVICIO 26
5.4.- MENSAJES DE INTERCAMBIO 28
6.- RECONOCIMIENTO DE VOZ **31**
6.1.- DISEÑO DE LA INTERFAZ DE VOZ 31
6.2.- DESARROLLO DEL COMPONENTE 32
6.3.- FUNCIONAMIENTO BÁSICO DE LA SOLUCIÓN 35
7.- EXPLOTACIÓN DEL SISTEMA EN EL PROYECTO CENIT OASIS **39**
7.1.- CLIENTE IMS 39
7.2.- INTEGRACIÓN CON MIDDLEWARE HMI 39
7.3.- SERVICIOS INTEGRADOS 41
8.- CONCLUSIONES **44**
8.1.- PRINCIPALES CONTRIBUCIONES 44
8.2.- MEJORAS IDENTIFICADAS EN LA PLATAFORMA 45
8.3.- LÍNEAS FUTURAS 45

APÉNDICE 1: MANUAL DE USUARIO 46
A1.1.- CONCEPTOS BÁSICOS 46
A1.2.- ESTRUCTURA DE LA APLICACIÓN 47
A1.2.1.- VENTANA PRINCIPAL 47
A1.2.2.- BARRAS DE HERRAMIENTAS 47
A1.2.3.- PANEL PRINCIPAL 51
APÉNDICE 2: IMPLEMENTACIÓN DE LA APLICACIÓN 55
A2.1.- MÉTODOS PAINT, REPAINT Y REMOVE 55
A2.2.- MENSAJES GENERADOS POR EL SISTEMA HMI 62
A2.3.- INTEGRACIÓN DE PANELES EXTERNOS 63
A2.4.- VENTANA DE SELECCIÓN DE FICHERO 63
A2.5.- MENSAJES DE INFORMACIÓN Y DE ERROR 64
APÉNDICE 3: CREACIÓN DE UN NUEVO MÓDULO 68
A3.1.- SUSCRIPCIÓN AL MÓDULO HMI 68
A3.2.- DEFINICIÓN DEL INTERFAZ DEL MÓDULO 69
A3.3.- COMUNICACIÓN CON EL MÓDULO HMI 72
A3.3.1.- CONEXIÓN INICIAL 72
A3.3.2.- RESPUESTA DEL MIDDLEWARE 72
A3.4.- IMPLEMENTACIÓN DEL NUEVO MÓDULO 74
APÉNDICE 4: DISTRIBUCIÓN *STANDALONE* DE LA PLATAFORMA 75
A4.1.- DEFINIENDO UN PRODUCTO HMI 75
A4.2.- CREANDO EL *PRODUCT CONFIGURATION* 75
A4.3.- CONFIGURACIÓN DEL PRODUCTO 77
A4.4.- EXPORTANDO HMI 79
BIBLIOGRAFÍA 82

Lista de Figuras

Figura 1.- Comunicación esencial aplicación-middleware HMI **8**
Figura 2.- Esquema de módulos en el framework OSGi **20**
Figura 3.- Diagrama UML del módulo HMI. **21**
Figura 4.- Diagrama de clases de HMI. **22**
Figura 5.- Mensajes comunicación inicial módulo-sistema HMI **28**
Figura 6.- Diagrama de bloques sintetizador de voz **32**
Figura 7.- Diagrama de clases del módulo ASR **33**
Figura 8.- Diagrama de clases módulo TTS **33**
Figura 9. Pantalla principal del *middleware* HMI **35**
Figura 10. Captura de la unidad de a bordo con el sintetizador desactivado y el reconocedor en ejecución. **36**
Figura 11- Interfaz principal de la OBU. **40**
Figura 12- Interfaz de servicio VSL **41**
Figura 13- Interfaz de servicio de planificación de rutas **42**
Figura 14- Interfaz de servicio de incidencias **42**
Figura 15- Servicios integrados **43**
Figura 16- Ventana principal del sistema HMI **47**
Figura 17- Barra de herramientas superior **47**
Figura 18- Ventana instalación/desinstalación de módulos **48**
Figura 19- Ventana de desinstalación de módulos **48**
Figura 20- Barra de herramientas inferior **49**
Figura 21- Teclado en pantalla **49**
Figura 22- Teclado de letras mayúsculas **50**
Figura 23- Teclado de números y caracteres especiales **50**
Figura 24- Menú principal con módulos activos **51**
Figura 25- Panel del módulo “Calculator” **52**
Figura 26- Panel del módulo “Voice Synthesizer” **52**
Figura 27- Panel del módulo “Picture SlideShow” **53**
Figura 28- Ventana selección de fichero **53**
Figura 29- Mensaje número de imágenes **54**
Figura 30- Panel del módulo “Web Browser” **54**
Figura 31.- Elemento Gráfico Botón. **56**
Figura 32.- Elemento Gráfico Checkbox. **57**
Figura 33.- Elemento Gráfico ComboBox. **57**
Figura 34.- Elemento Gráfico Label. **58**
Figura 35.- Elemento Gráfico Image **58**
Figura 36.- Elemento Gráfico Password **59**
Figura 37.- Elemento Gráfico TextField **59**

Figura 38.- Elemento Gráfico RadioButton **60**
Figura 39.- Ventana de selección de ficheros y directorios. **63**
Figura 40.- Mensaje sin formato **64**
Figura 41.- Mensaje de pregunta **64**
Figura 42.- Mensaje de aviso **64**
Figura 43.- Mensaje de información **64**
Figura 44.- Mensaje de error **64**
Figura 45.- Opciones “Si” o “No”. **65**
Figura 46.- Opciones “Si”, “No”, “Cancelar”. **65**
Figura 47.- Opciones “Aceptar”, “Cancelar” **65**
Figura 48.- Opciones por defecto **65**
Figura 49.- Ventana campo de escritura **66**
Figura 50.- Ventana lista desplegable **66**
Figura 51.- Ventana botones opciones personalizables **67**
Figura 52.- Ventana principal Reproductor MP3 ..... **71**
Figura 53- Ventana configuración de producto **76**
Figura 54- Editor de producto. **77**
Figura 55- Ventana dependencias producto **78**
Figura 56- Ventana Editor / configuración **78**
Figura 57- Ventana Lanzamiento **79**
Figura 58- Asistente exportación de producto **80**
Figura 59- Ficheros carpeta compilación **81**

1.- Introducción

1.1. Problemática

En la actualidad la dispersión de los sistemas informáticos es cada vez mayor y para ello existen tecnologías que centralizan la información y los sistemas agrupándolos para mejorar su gestión.

OSGi se presenta como una de estas tecnologías, una pasarela de servicios que gestiona distintos sistemas bajo un mismo software facilitando la comunicación y el intercambio de información entre módulos, todo sobre el lenguaje de programación Java ampliamente extendido, presente en multitud de dispositivos.

La pasarela de servicios OSGi proporciona todos los sistemas base para la comunicación y gestión de sensores y componentes, pero a la hora de desarrollar un interfaz gráfico es obligatorio implementar en cada aplicación específica los componentes visuales, no pudiendo gestionar todos los elementos y aspectos visuales desde un mismo módulo que permitiría unificar el aspecto gráfico del sistema.

Este proyecto recoge esa necesidad, añadiendo esa característica a la plataforma OSGi a través de un componente denominado *middleware HMI* que generará de forma dinámica (en tiempo de ejecución) el interfaz gráfico apropiado en base a una descripción normalizada del interfaz gráfico requerido. Este módulo hará la función de intermediario entre los módulos que ofrecen servicios y la interfaz gráfica final apreciada por el usuario.

1.2. Avance de la estructura general del sistema

Cumpliendo con las necesidades descritas en el apartado anterior, se ha desarrollado una aplicación denominada *middleware HMI* sobre la pasarela de gestión de servicios OSGi. El componente se encarga de la generación dinámica de elementos gráficos del lenguaje Java encargándose también de los eventos que vaya generando el usuario.

El *middleware* desarrollado se encarga de encontrar e integrar todas las aplicaciones destinadas a utilizar la generación de interfaces gráficas ofrecidas por este componente, mostrándolas en un menú principal para que sea más sencillo al usuario lanzar cada una de las aplicaciones adaptadas.

Para que una aplicación sea integrada en este sistema debe cumplir una serie de requisitos previos necesarios para la comunicación con el componente principal que se encuentran explicadas en el capítulo 5 de esta memoria (Comunicación con el módulo HMI).

La estructura con la que se ha construido este *middleware* permite a los módulos integrados en la plataforma evitar la programación relacionada con los elementos gráficos y todo lo relacionado con la interacción usuario-máquina. Cada módulo o *bundle* integrado en el sistema deberá contar sólo con un archivo XML (*Extensible Markup Language*) que

contendrá la información necesaria para la construcción de la interfaz gráfica inicial y contará con los métodos del *middleware HMI* para la representación de mensajes de advertencia o error, reproducción de mensajes de texto por voz además de la modificación o creación de elementos gráficos *on the go (no es necesario reiniciar la aplicación).*

Por tanto, el componente abordado en este proyecto se preocupa solamente de la generación de interfaces gráficas y del tratamiento de eventos generados por el usuario generando así una interfaz gráfica homogénea ya que todos los elementos gráficos se construyen con el mismo patrón de diseño.

Como avance y resumen, en la Figura 1 que se muestra a continuación se encuentra un ejemplo de la comunicación mínima que debe ocurrir entre un módulo de ejemplo (una calculadora) y el sistema HMI. Los pasos que sigue la comunicación son los siguientes:

- El módulo que quiere generar una interfaz gráfica se registra al servicio del componente HMI.
- Le envía un documento en formato XML con la descripción de la interfaz gráfica inicial.
- Al generar el usuario un evento a través de la aplicación gráfica éste es comunicado al módulo (calculadora) que es el que contiene la lógica necesaria para procesar la información del cálculo.
- El módulo se comunica con el *middleware HMI* para actualizar los elementos gráficos y mostrar el resultado de la operación *sumar a+b* al usuario.

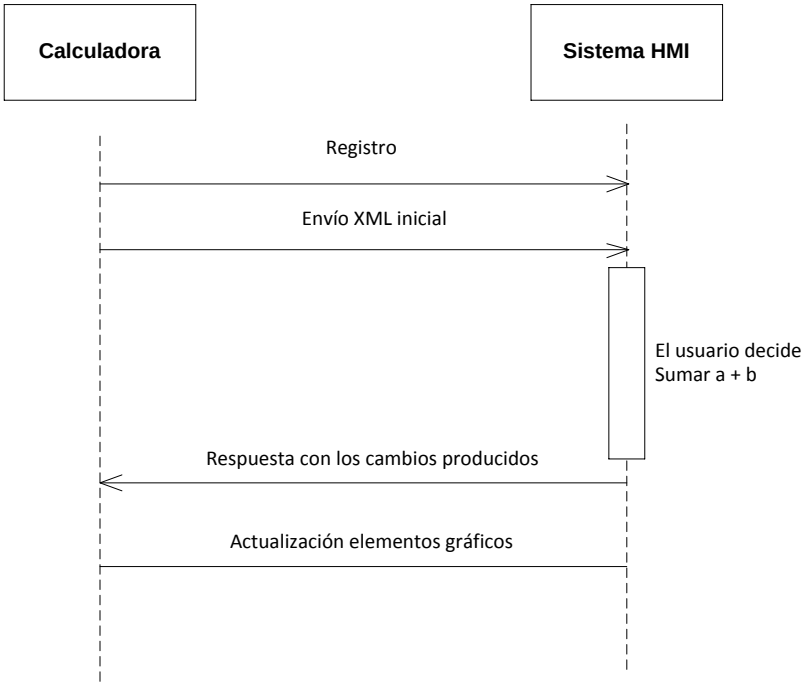

Figura 1.- Comunicación esencial aplicación-middleware HMI

1.3. Objetivos

Este proyecto fin de grado está basado en la implementación de un componente HMI ofreciendo el soporte gráfico del que carece el *framework OSGi*. Se centralizará en la generación dinámica de interfaces gráficas para cada uno de los módulos compatibles obviando la lógica de procesado de cada una de las operaciones, haciendo únicamente de pasarela gráfica entre el usuario y las aplicaciones.

El *middleware HMI* a desarrollar contará por tanto con las funciones necesarias para la creación, modificación y eliminación de elementos gráficos básicos como botones, cajas de texto, listas desplegables o mensajes de error o advertencia.

Se debe implementar mediante Java utilizando el entorno de desarrollo de código abierto *Eclipse*, que contiene nativamente la pasarela de servicios OSGi.

Los objetivos fijados en el proyecto son los siguientes:

- Desacoplar la generación de interfaces gráficas del diseño de las aplicaciones vehiculares.
- Ofrecer un soporte de interfaz hombre-máquina dinámica.
- Fomentar el desarrollo de módulos software comunicados mediante interfaces de comunicación según estándares W3C.
- Explotar las capacidades de la tecnología Java en el software de propósito general para vehículos.
- Implementación y evaluación funcional de los diseños realizados.
- Integración con el proyecto CENIT OASIS.
- Adaptación de un sistema de reconocimiento de voz.

Para la creación de una HMI, se partirá de un archivo XML que fijará la aplicación cliente. Este fichero se parseará mediante un intérprete XML basado en Java.

También se debe incluir un asistente de instalación/desinstalación de aplicaciones en el *framework* OSGi, así como un modulador de voz que según sea oportuno reproduzca los cambios en la pantalla.

Debido a que este proyecto está enfocado a la industria automovilística, deberá contar con un teclado táctil en pantalla cuyo uso debe ser transparente a las aplicaciones que se ejecutan en el *framework*.

Como añadido, se debe implementar un navegador para la plataforma.

Para finalizar, la plataforma OSGi con todos los componentes desarrollados deben ser ejecutados en modo *standalone* sobre un entorno Linux.

1.4. Estructura del proyecto

El documento se encuentra organizado según los siguientes capítulos:

- **2.- Tecnologías utilizadas:** En este apartado se mencionan las principales tecnologías utilizadas en el desarrollo del proyecto así como un resumen de los conceptos tenidos en cuenta a lo largo de su desarrollo.
- **3.- Estructura y componentes del sistema:** Para tener una visión más clara de cuáles son los elementos gráficos que pueden ser creados a partir de esta *sistema HMI*, en este capítulo se da una explicación detallada de cada uno de ellos incluyendo también los mensajes de control que pueden ser generados.
- **4.- Tratamiento de datos con XML:** La comunicación inicial entre el *sistema HMI* y una aplicación del *framework* OSGi, será iniciada con un fichero XML, que contendrá los elementos de partida para crear la interfaz gráfica dinámica. El documento XML se "parseará" para convertir el contenido en objetos Java, más accesibles desde el *middleware HMI.* En este capítulo se propondrán las diferentes tecnologías existentes, así como la seleccionada en el desarrollo del proyecto.
- **5.- Interfaz de las aplicaciones con el middleware HMI:** Para la generación e interpretación de una interfaz gráfica de usuario, se necesitan mensajes de forma bidireccional, es decir, en este apartado se explicará la comunicación *middleware HMI-aplicación* y *aplicación-middleware HMI*, así como los mensajes relacionados.
- **6.- Reconocimiento de voz:** Para cumplir con la normativa vigente, el componente HMI debía contar con un sistema que permitiera al conductor del vehículo comunicarse con el sistema evitando fijar la vista en la pantalla. El funcionamiento del reconocimiento de voz se explicará en este apartado.
- **7.- Explotación del sistema en el proyecto CENIT OASIS:** Una de las principales metas de este proyecto fin de grado es la integración con aplicaciones de desarrollo avanzado. En este apéndice se detallará la integración con un componente de mapas denominado OASIS.
- **8.- Conclusiones:** En este apartado se describen las conclusiones sacadas después de la realización del proyecto fin de grado junto con una serie de características adicionales que podrían ser implementadas en el futuro.

Apéndice 1.- Manual de usuario: Todo usuario que quiera utilizar el *sistema HMI* deberá seguir unas directrices básicas de configuración. En este capítulo se abordará este tema, incluyendo algunos ejemplos.
Apéndice 2.- Implementación de la aplicación: Debido a las numerosas funciones que componen el *middleware HMI*, aquí se enumeran cada una de ellas y su funcionalidad.
Apéndice 3.- Creación de un nuevo módulo: Para facilitar la integración de un nuevo módulo en el *framework*, se explica el procedimiento que hay que seguir para la correcta implementación de una aplicación compatible con OSGI y con el s*istema HMI.*
Apéndice 4.- Distribución *STANDALONE* de la plataforma: El sistema final, que incluye la pasarela OSGi y sus módulos debe ejecutarse sin necesidad de intervención del usuario. Todo ello es posible a través de Linux y con el entorno de desarrollo eclipse.

2.- Tecnologías utilizadas

La aplicación desarrollada en este proyecto se ha implementado en lenguaje Java, el que es utilizado tanto por JAXB como por OSGi.

El api JAXB se encargará de convertir los documentos XML de las aplicaciones del *framework* en objetos Java, facilitando el tratamiento de la información. Esta interfaz de programación creará los objetos a partir de unas clases Java pre-cargadas, haciendo su utilización más rápida.

Por otro lado, también será necesario tener una pasarela de servicios que se encargue de organizar las aplicaciones de forma modular en un entorno común, esta función se implementa utilizando el *framework* OSGI.

En los siguientes apartados se detalla en profundidad cada una de estas tecnologías, que son imprescindibles para el desarrollo de este proyecto.

2.1.- Java

Java es un lenguaje de programación orientado a objetos, desarrollado por *Sun Microsystems* a principios de los años 90. Su sintaxis toma muchos elementos propios de otros lenguajes como C ó C++, pero no incluye herramientas de bajo nivel que éstos si integran como puede ser la manipulación directa de punteros.

Entre diciembre de 2006 y mayo de 2007, *Sun Microsystems* liberó la mayor parte de sus tecnologías bajo la licencia *"GNU GPL"*, de forma que prácticamente todo el Java de Sun es ahora software libre.

Las características principales que nos ofrece Java frente a otro lenguaje de programación son:

- Orientado a objetos: Es una forma de diseñar el software de forma que los distintos tipos de datos que se usen estén unidos a sus operaciones. Así, los datos y las funciones se combinan en entidades llamadas "objetos", que permiten la reutilización de software entre proyectos.
- Independencia de la plataforma: Los programas escritos en este lenguaje pueden ejecutarse igualmente en cualquier hardware. Para ello, se compila el código fuente escrito en Java, para generar un código conocido como *bytecode*, que sería un paso intermedio entre el código fuente y el código máquina que entiende el equipo destino. Este *bytecode* es ejecutado en la máquina virtual (JVM), un programa escrito en el código nativo de la plataforma destino, que lo interpreta y ejecuta. Hay implementaciones del compilador de Java que convierten el código fuente directamente en el código nativo de la máquina, como GCJ (GNU compiler for Java). Esto elimina la etapa intermedia de generación del *bytecode* pero sólo podrá ejecutarse en la arquitectura en la que ha sido compilado.

- El recolector de basura: El problema causado por un bloque de memoria que no es liberado en memoria tras su utilización es resuelto mediante esta característica del lenguaje. El compilador determina cuándo se crean los objetos y el entorno en tiempo de ejecución *Java runtime* es el responsable de gestionar el ciclo de vida de éstos. Cuando un objeto deja de ser referenciado, el recolector de basura lo borra, liberando la memoria que ocupaba previniendo así posibles fugas de memoria.

El diseño de este lenguaje, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática, apareciendo en dispositivos móviles, sistemas empotrados, en el navegador web, en sistemas de servidor o en aplicaciones de escritorio.

2.2.- Estándar XML en Java

XML, siglas en inglés de *Extensible Markup Language* (Lenguaje de Marcas Extensible), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). Permite unificar la forma de representar los datos, permitiendo el intercambio de información entre aplicaciones o sistemas que pueden estar desarrollados en distintos lenguajes de programación o en equipos de distinta arquitectura.

Es una simplificación y adaptación del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. En este proyecto juega un papel muy importante, ya que en el campo de la telemática vehicular es interesante utilizar un lenguaje que permita definir sus propias etiquetas, haciendo que en la lectura de un DTD (definición de tipo de documento) se pueda entender la función de un parámetro conociendo sólo su nombre especificado.

La existencia de varios tipos de *parsers* de documentos XML hace necesario que se tengan en cuenta cuál es la función que se quiere que cumpla en la aplicación principal. Por ejemplo, si los documentos XML van siempre a tener la misma estructura o si ésta va a cambiar de forma dinámica. A continuación se detallan tres de los *parsers* más importantes:

- SAX: Su principal característica es que es un *parser* dirigido por eventos. Cuando se lee un elemento o un documento se genera un evento. Su principal ventaja es que es eficiente en cuanto a tiempo y memoria empleados en el análisis de un documento XML, pero no permite modificar el documento ni volver a un elemento anterior (evento anterior).

- DOM, en cambio genera una estructura en forma de árbol del documento XML y los almacena en memoria, facilitando la manipulación y modificación del documento. Es más lento que SAX, ya que éste no tiene que volcar todo el contenido del fichero en memoria, sino sólo una parte (elemento).
- JAXB: Su funcionamiento se asemeja bastante al utilizado por DOM, sólo que JAXB no parte de "cero" sino que contiene la estructura del documento XML "pre-cargada". Todo esto es posible mediante la especificación de un DTD, con el que se generarán las clases, que añadidas al código fuente de la aplicación, permitirán un acceso rápido al documento XML. Por ello, JAXB combina lo mejor de las dos tecnologías antes propuestas, un acceso rápido al documento y menor impacto en cuanto a consumo de memoria.

JAXB simplifica el acceso de un programa Java a un documento XML representando éste en el mismo lenguaje de programación, que proporciona a los desarrolladores de aplicaciones Java una forma rápida y conveniente para enlazar y vincular esquemas XML a representaciones Java.

El API posee métodos para desorganizar (unmarshal) documentos XML en árboles de contenido (generados en código Java), para después utilizar los mismos y generar mediante el método organizar (marshal) instancias XML de las que fueron creados.

Las clases generadas JAXB describen sólo la relación real definida en los esquemas fuentes. El resultado de lo anterior son datos XML altamente portables que unido a un código Java portable puede ser usado para crear flexibles y ligeras aplicaciones.

Las características que nos ofrece el API son las siguientes:

- JAXB usa tecnología Java y XML.
- Garantiza datos válidos.
- Es rápida y fácil de usar.
- Puede restringir datos.
- Es personalizable y extensible.

Los datos que se representan usando XML se pueden pasar entre aplicaciones porque la estructura de los datos se puede especificar en un esquema, lo que permite que un analizador de sintaxis valide y procese los datos.

En el lenguaje XML no se hace uso de etiquetas, como lo hace HTML (HyperText Markup Language), se hace uso de un DTD (Document Type Definition) que nos permite definir etiquetas propias, y éstas a su vez describen los datos. Con el acceso fácil a los datos XML que proporciona JAXB, solamente se necesita programar aplicaciones que realmente utilizarán éstso datos, en vez gastar el tiempo en escribir código para formatear los datos.

2.3.- Pasarela de servicios OSGi

El *framework* OSGi es un sistema modular y una plataforma de servicios para el lenguaje de programación Java, que implementa lo que podríamos llamar "Modularidad dinámica", esto quiere decir que podemos instalar, actualizar, iniciar, parar y desinstalar componentes (Aplicaciones) sin la necesidad de reiniciar el sistema. El registro de servicios permite a los *bundles* (módulos) detectar la incorporación o desaparición de un nuevo servicio y actuar en consecuencia.

La modularidad incorpora características muy importantes a Java:

- División del trabajo: Se pueden asignar tareas a diferentes módulos para que se implementen en paralelo. El desarrollador que trabaje en un *bundle* tendrá que conocer el funcionamiento de su propio módulo pero no del resto.
- Abstracción: Sólo necesitamos saber cómo funciona una parte del sistema, que es en lo que se va a centrar nuestro módulo.
- Reusabilidad: Facilita el desarrollo que un componente del sistema pueda ser reutilizado, aunque tenga que sufrir alguna modificación.
- Sencillez en el mantenimiento y la reparación: Los módulos dañados pueden ser borrados, reparados y añadidos de nuevo, sin que afecte al sistema.

La fuente de los problemas en el Java tradicional es el *classpath* (ruta donde se encuentran los archivos a compilar o ejecutar) global. OSGi resuelve estos conflictos aportando a cada módulo su propio *classpath*, separado del de los otros módulos.

Por la naturaleza de los módulos, se necesita compartir clases entre ellos. OSGi define unas reglas sobre cómo las clases deben ser compartidas, usando un mecanismo de importaciones y exportaciones explícitas.

Un *bundle* o módulo en OSGi no es más que un archivo JAR (Java Archive), que incluye un archivo de metadatos, que lo incorporará en la plataforma. Este *metadata* toma el nombre de "MANIFEST.MF" y consiste en:

- El nombre del *bundle*: Se especifica un nombre simbólico que es usado por OSGi para identificar a ese módulo.
- La versión del *bundle*.
- La lista de importaciones y exportaciones.
- Opcionalmente, la versión mínima de Java que el *bundle* necesita para ser ejecutado.
- Información extra sobre el desarrollador del módulo, declaración de copyright, dirección de contacto, etc.

En la actualidad existen varios *frameworks* que usan OSGi. En este proyecto se usa *Equinox* que es nativo en el entorno de desarrollo de código abierto *Eclipse*, utilizado en la implementación del *sistema HMI*.

2.4.- SDK's para la síntesis y reconocimiento de voz

En la elección de la API para la síntesis de voz se empezó valorando alternativas como *Festival o FreeTTS,* pero se quedaron descartadas por diferentes motivos. La primera es un sistema de síntesis de voz bajo licencia de software libre, implementado en lenguaje C++ que dispone de librerías tanto en dicho lenguaje como *Java* para el desarrollo de aplicaciones. Aunque esta opción es multilingüe, actualmente sólo soporta inglés y español, estando el primero de ellos bastante más avanzado que el segundo. Este aspecto la descartó, ya que las soluciones como las ofrecidas por *Loquendo* mejoraban esta alternativa como se explica más adelante. La segunda, *FreeTTS*, es un sintetizador que implementa la *Java Speech API* y es distribuida con licencia BSD. Esta opción fue la elegida en un principio pero suponía una opción muy sencilla debido a que actualmente sólo soporta inglés como lenguaje.

Para el servicio de conocimiento de voz hay dos vertientes ampliamente diferenciadas, sistemas independientes del locutor y sistemas independientes del lenguaje. La elección se centró en una solución independiente del locutor, ya que este tipo de reconocedores ofrecen independencia de los usuarios que pueden hacer uso de ella, aspecto requerido en este trabajo. Por este motivo, soluciones como *Dragon NaturallySpeaking SDK* quedaron automáticamente descartadas, ya que aunque esta alternativa es muy completa y eficiente, dispone de una herramienta de dictado, es multiplataforma y multilingüe, entre otras características, esta solución es independiente del lenguaje pero no del locutor. Sin embargo, *Verbio*, es una solución independiente del locutor que soporta la creación de gramáticas propias, ya que es dependiente del lenguaje. Esta lleva implementadas gramáticas básicas con las que se puede trabajar y aunque esta solución ofrece robustez frente al ruido, está orientada a entornos telefónicos, lo que podia suponer un inconveniente para este proyecto, ya que el entorno vehicular que es sobre el que se desarrolla este es muy particular.

Hasta este punto las diferentes opciones que se barajan u ofrecen una API para desarrollar el servicio de síntesis u ofrecen una API para implementar el de reconocimiento pero no los dos. Con esta nueva idea de encontrar alguna SDK que pudiera ofrecer los dos y así obtener la ventaja de trabajar con una única SDK siempre y cuando no empeorara las opciones anteriores se incorporan las dos siguientes propuestas: *Microsoft Speech SDK* y *Loquendo SDK*.

- *Microsoft Speech SDK*. Esta opción ofrece tanto una interfaz de síntesis como de reconocimiento del habla. En el reconocimiento hace uso de gramáticas para así limitar el conjunto sobre el que debe decidir y mejorar el rendimiento de este, mientras que en el sintetizador entre sus características principales destaca la aplicación de una prosodia adecuada en cada situación, lo que reporta una expresión oral similar a la empleada por el ser humano, además de concatenar los segmentos de la señal para producir una voz continua. Esta alternativa aunque muy completa quedó descartada ya que está altamente acoplada a sistemas operativos Win32 y la unidad de a bordo emplea sistema UNIX.

- *Loquendo SDK*. Kit de desarrollo software tanto para los servicios de síntesis como de reconocimiento de voz. Entre todas las características que se describen en el siguiente apartado destacar su robustez, naturalidad y expresividad en el resultado final ofrecido.

2.4.1 Loquendo SDK

En todo el abanico de posibilidades que se ha comentado en la sección anterior y que se estuvo barajando para la implementación de los dos servicios requeridos en este interfaz como *Microsoft Speech SDK*, *Verbio*, *FreeTTS* o *Festival* entre otros, *Loquendo SDK* fue la opción elegida. Sus principales características y, por consiguiente, lo que decantó esta opción sobre las demás, fue la calidad del servicio y la total posibilidad de personalización y adaptación de su API a nuestro proyecto, además de otras características técnicas que veremos a continuación.

Un aspecto importante y determinante para la implementación de un interfaz de estas características es la obtención de resultados de voz agradable y natural. En este, la API de *Loquendo* ofrece voces específicas de alta calidad, integradas con bases de datos pre-grabadas que contienen las expresiones más comunes.

Otra característica destacada es que la tecnología para la síntesis que ofrece *Loquendo* es compatible con los alfabetos fonéticos *SAMPA*. Definidos estos como alfabetos fonéticos legibles por ordenador mediante caracteres ASCII, este tipo son utilizados por ejemplo por los productores de mapas *TeleAtlas* y *Navtec* por lo que la compatibilidad del sintetizador de *Loquendo* con estos puede utilizarse en futuros proyectos por ejemplo para realizar la lectura de callejeros de una manera fácil y eficiente.

2.4.2 Loquendo ASR

Loquendo ASR es el software de reconocimiento de voz elegido para implementar una de las dos partes de las que está compuesto este interfaz, el reconocimiento de voz. La tecnología que utiliza este es una combinación de Redes Neurales y Modelos Ocultos de Markov de Densidad Continua, lo que garantiza óptimas prestaciones incluso con vocabularios extensos.

Como modalidades de reconocimiento este software puede utilizar tanto gramáticas como modelos estadísticos del lenguaje. En este proyecto, las gramáticas serán utilizadas para establecer los posibles comandos a reconocer por dicho servicio.

Destaca también que es independiente del hablante, por lo que no requiere formación previa para poder ser utilizado y, además, presenta una gran robustez frente al ruido, aspecto indispensable para poder implementarlo en un proyecto de entorno vehicular, donde el ruido es uno de los factores más determinantes.

Opcionalmente, *Loquendo ASR* soporta la funcionalidad de verificación del hablante, permitiendo en futuros proyectos añadir un nivel de seguridad al incorporar esta característica.

2.4.3 Loquendo TTS

Para la implementación de la otra parte del interfaz, el sintetizador de voz, se eligió *Loquendo TTS*. Las voces sintéticas de Loquendo, naturales e inteligibles, poseen la expresividad de las voces humanas, lo que conlleva que su utilización no sea invasiva para los individuos del vehículo. Esto repercute en un ambiente de confort en su utilización en este.

Entre sus especificaciones técnicas destacan sus bajos requisitos de memoria para el motor vocal y para las voces empleadas, aspecto necesario para poder ser integrado en unidades de a bordo vehiculares, ya que, aunque cada vez más estos dispositivos poseen características técnicas muy avanzadas, todavía existen algunas limitaciones como es la memoria de la que disponen.

Otras características que se pueden destacar de esta API son su capacidad multilingüe y multivoz con más de 30 idiomas y 70 voces disponibles. Esta amplia gama de idiomas disponibles permite que en futuros trabajos se puedan incorporar nuevos de ellos sin tener que realizar cambios significativos en la implementación de este proyecto.

3.- Estructura y componentes de la plataforma

3.1.- Marco de trabajo

Este proyecto se encuentra enfocado en el terreno de la telemática vehicular, un marco ahora en gran evolución en el que se están centrando la mayoría de los fabricantes de automóviles, puesto que proporciona al usuario mayor comodidad en el uso de aplicaciones esenciales en los vehículos como la radio, el GPS o el sistema de manos libres. Todas estos "sistemas" necesitaban una interfaz gráfica en la que pudieran ser mostrados al usuario final, por ello surge este proyecto, para cubrir esa necesidad, implementando el elemento que faltaba en este sector del desarrollo de aplicaciones, una HMI (*Human Multimedia Interface*) proporcionando a los sistemas o módulos integrados en la plataforma OSGi un interfaz gráfico común.

A la hora de desarrollar un proyecto de estas características, surgen varias preguntas en cuanto a las funcionalidades que debe ofrecer, cómo debe presentarse la interfaz gráfica o cuál va a ser la estructura de los elementos gráficos que la van a componer finalmente.

De todas las características implementadas en este *middleware HMI* cabe destacar algunas como prescindir de un teclado físico e implementar uno virtual ya que la mayoría de los sistemas de a bordo incluyen ya una pantalla táctil. Otra función también contemplada es que normalmente los requisitos que son buscados por los programadores en una aplicación de generación de interfaces dinámica pueden ser la fácil integración de un nuevo módulo al *sistema HMI*, existencia de métodos que faciliten la representación de elementos gráficos o el uso de un lenguaje de metadatos estandarizado como XML.

Todo esto, nos lleva a un proyecto fin de grado que busca facilitar el desarrollo de aplicaciones OSGi, ya que el desarrollador se centrará únicamente en la lógica de su módulo, así como una mayor aceptación del usuario en cuanto a sistemas que se centren en esta plataforma, ya que se encontrará con un sistema gráfico compacto, fácil de usar, que cuenta con aplicaciones de todo tipo y a un coste bajo.

Durante este capítulo se abordarán la estructura de clases que configura el *sistema HMI*, explicando su función en el sistema. También se introducirán los conceptos básicos relacionados con los elementos gráficos generados por el *middleware HMI*, incluyendo también algún ejemplo de la estructura de éstos en los documentos XML que son enviados al sistema desarrollado para generar la interfaz gráfica inicial.

En esta sección de la memoria del proyecto también se incluirá una descripción de lo que es un parseador en XML, presentando al lector la solución que ha sido escogida entre tantas en el desarrollo de este proyecto, proponiendo las características más destacadas que hicieron que al final fuera seleccionada.

3.2.- Descripción de la estructura global

La aplicación desarrollada en este proyecto, se encuentra integrada en el marco del *framework* OSGi. Debido a que en la plataforma pueden encontrarse instalados varios módulos que no se corresponden con los que están relacionados con este proyecto, se especificarán los que han sido implementados o los que dependen del *middleware HMI* para su correcto funcionamiento.

Hay varias características que posee el *sistema HMI* que hacen necesario la adicción de módulos al *framework* OSGi. Sin incluir con los que ya cuenta la plataforma, el *middleware HMI* necesita el módulo encargado de la modulación de voz, el "*Voice synthesizer*" para poder escuchar los comandos de voz por los altavoces del equipo en el que se ejecute, ya que es éste el que contiene toda la lógica relacionada con la carga de voces para que luego serán reproducidas.

El resto de módulos, que aunque no son necesarios para poder iniciar el *bundle* desarrollado, han sido implementados e integrados en el *framework* OSGi para demostrar que este *sistema HMI* puede incluir aplicaciones tan diversas como pueden ser una calculadora, un navegador web o un módulo que se encargue de mostrar imágenes a partir de una ruta seleccionada por el usuario.

Partiendo de una aplicación básica que puede imprimir mensajes en pantalla, se puede llegar a un módulo que parta de una interfaz gráfica inicial a partir de un documento XML dado, que sea mostrado en el menú principal del *middleware HMI* con un nombre y un icono personalizados, esto último añadiendo sólo dos parámetros al archivo MANIFEST del *bundle.*

A partir de un módulo nuevo que implemente el servicio de generación de interfaces gráficas que implementa el *sistema HMI* se podrá generar desde elementos gráficos como botones, cajas de texto, listas de selección… hasta una ventana para seleccionar la ruta de un fichero o un directorio, o añadir un JPanel (ventana gráfica en Java) a la pantalla personalizada propia del *bundle*.

En la siguiente Figura se muestra la organización de los módulos que han sido implementados e integrados en la plataforma OSGi durante el desarrollo de este proyecto. Como puede observarse los *bundles* integrados en la plataforma OSGi se comunican con el *sistema HMI* para la generación de cada interfaz gráfica mediante los métodos de la interfaz asociada al servicio de interfaces gráficas que ofrece el *middleware* desarrollado. Como se trata de una comunicación bidireccional, el *bundle HMI* también se deberá comunicar con los módulos que estén usando su servicio, para ello dispone de un método llamado "processHMICmdEvent" que tienen que estar desarrollado en los módulos asociados con el *middleware HMI*, ya que ahí es donde se encuentra toda la lógica que determina a cada módulo la evolución de la interfaz gráfica cuando el *sistema HMI* le informa de que ha habido un cambio en la que se encontraba activa.

Como se ha comentado antes, el módulo "Calculator" es el que se encarga de implementar la función de una calculadora en el *framework* OSGi, su funcionamiento y su

comunicación con la aplicación principal ya se explicaron en el apartado 1.2 de esta memoria.

El *bundle* que recibe el nombre de "Picture SlideShow", se encarga de, como su nombre indica, mostrar una tira de imágenes al usuario. Se partirá de una interfaz gráfica sin imágenes cargadas, pudiendo abrir nuevas imágenes a través del método "logicCommandFileChooser" de la aplicación HMI, que se encarga de mostrar una ventana de selección de un directorio del que se cargarán las imágenes. Tras haber sido añadidas las imágenes, se dispondrán de dos botones para ir avanzando por la secuencia.

Este *framework* OSGi "personalizado" en el que nos estamos basando en este apartado, también hay dos módulos que se podría decir que son los más importantes después del módulo HMI. Son el "Web Browser" o navegador web y el "*Voice synthesizer",* que ya fue presentado antes. Su característica más destacada relacionada con la disposición de los módulos en el *framework* OSGi es que dependen de otro módulo para su correcto funcionamiento, como ocurría con el *middleware HMI*. El navegador web necesita la existencia del módulo JDIC, que es el que realmente construye el panel que integrará el navegador. Lo mismo ocurre con el sintetizador de voz, que depende del Speech Module.

A continuación se muestra de forma gráfica lo que se ha ido comentando durante este apartado:

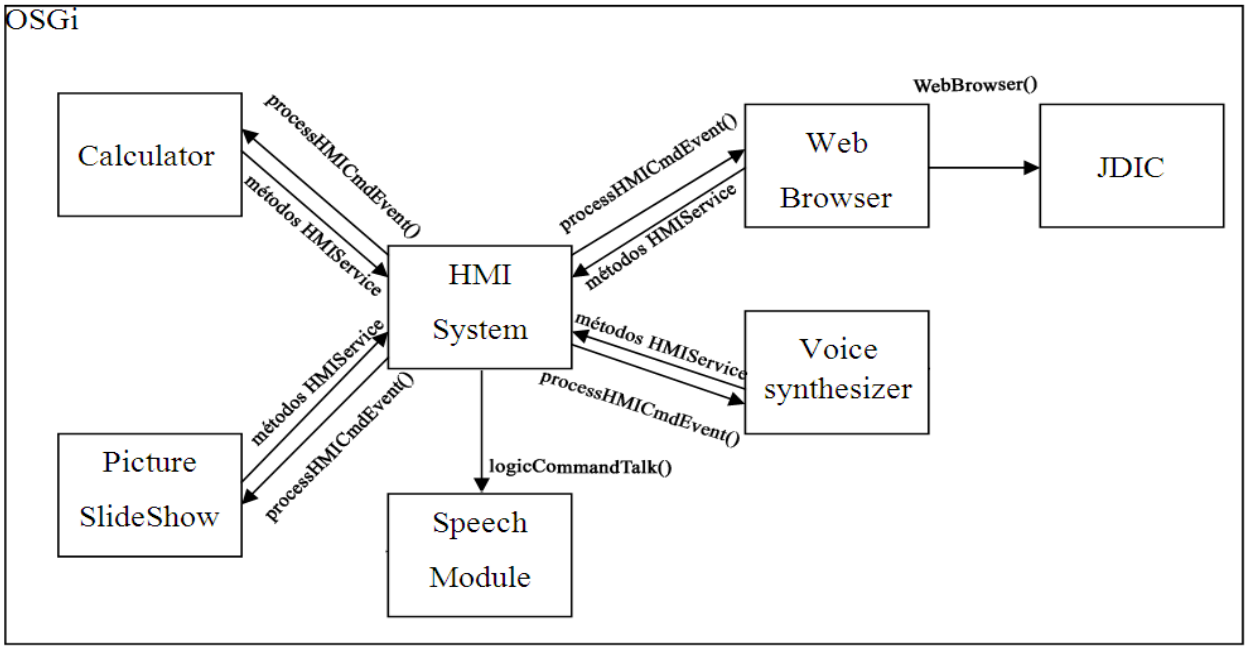

Figura 2.- Esquema de módulos en el framework OSGi.

3.3.- Estructura de los elementos del sistema

Para una mejor comprensión de la forma de utilizar de la aplicación de generación de interfaces dinámicas, a continuación se muestra un diagrama UML de las clases presentes en este *bundle* así como las relaciones entre ellos. Los métodos pertenecientes a cada uno no han sido incluidos, a posteriori se detallarán los del paquete "org.hmi":

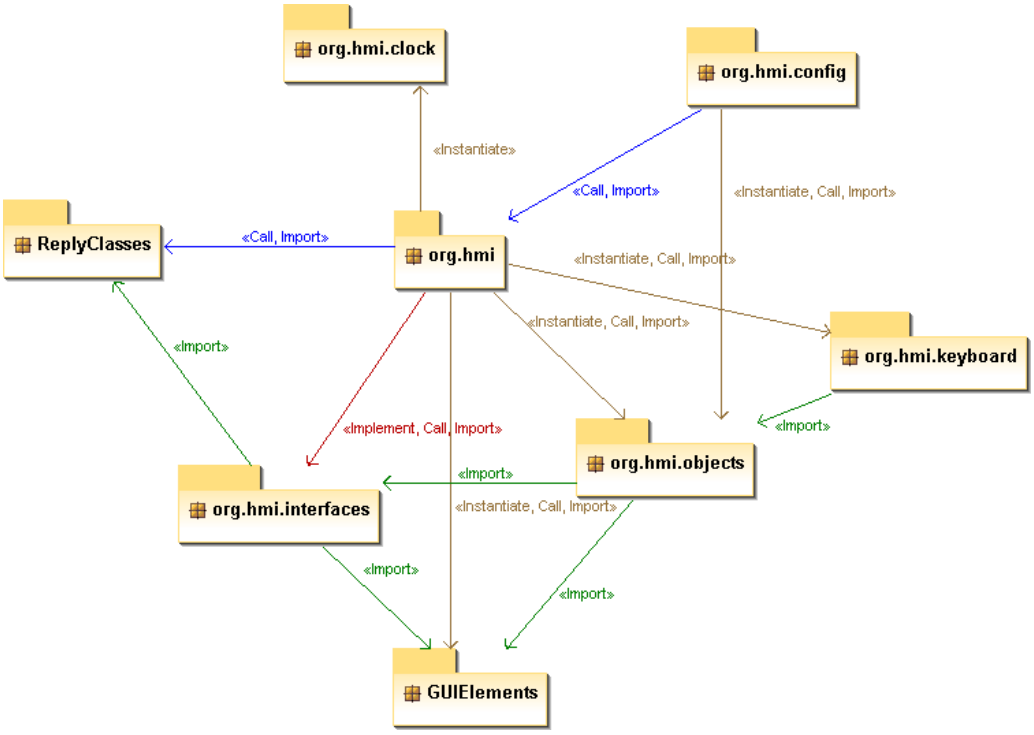

Figura 3.- Diagrama UML del módulo HMI.

Como puede apreciarse en la ilustración anterior, la arquitectura del *bundle* está formada por una asociación de paquetes centralizado, siendo "org.hmi" el principal. Formando parte del *package* "org.hmi", se encuentra la clase *Activator* que es la que se encarga de crear el servicio que ofrece el *sistema HMI*, y de registrar el servicio relacionado con el sintetizador de voz. También se halla la clase *hmi* desde la que se coordina todo el proceso de creación de las interfaces gráficas de los distintos módulos.

A continuación se especifica la función del resto de los paquetes incluidos en el *sistema HMI:*

- Org.hmi.clock: Crea los elementos encargados de mostrar la hora y la fecha del sistema en la aplicación principal.
- Org.hmi.config: Muestra una ventana para la instalación o desinstalación de *bundles* incluidos en el *framework* OSGi.
- Org.hmi.keyboard: Añade el teclado a la aplicación HMI.
- Org.hmi.objects: Incluye todos los elementos externos necesarios en la aplicación (Elementos del menú principal, paneles con imágenes de fondo).
- Org.hmi.interfaces: Contiene las interfaces públicas del *bundle*.
- GUIElements y ReplyClasses forman parte de Org.hmi.objects. Son las clases generadas por el api JAXB, contienen la estructura de los objetos de los elementos XML usados en la comunicación del *sistema HMI* con el resto de módulos.

Como se ha comentado antes, la funcionalidad del *middleware HMI*, se encuentra principalmente en la clase *hmi*. En la siguiente Figura se muestra el diagrama de clases de la clase *hmi* y de la interfaz *HMIService*. Sólo han sido representados los parámetros y métodos más importantes. En el apéndice 2 de esta memoria se detallarán las características de cada uno de ellos:

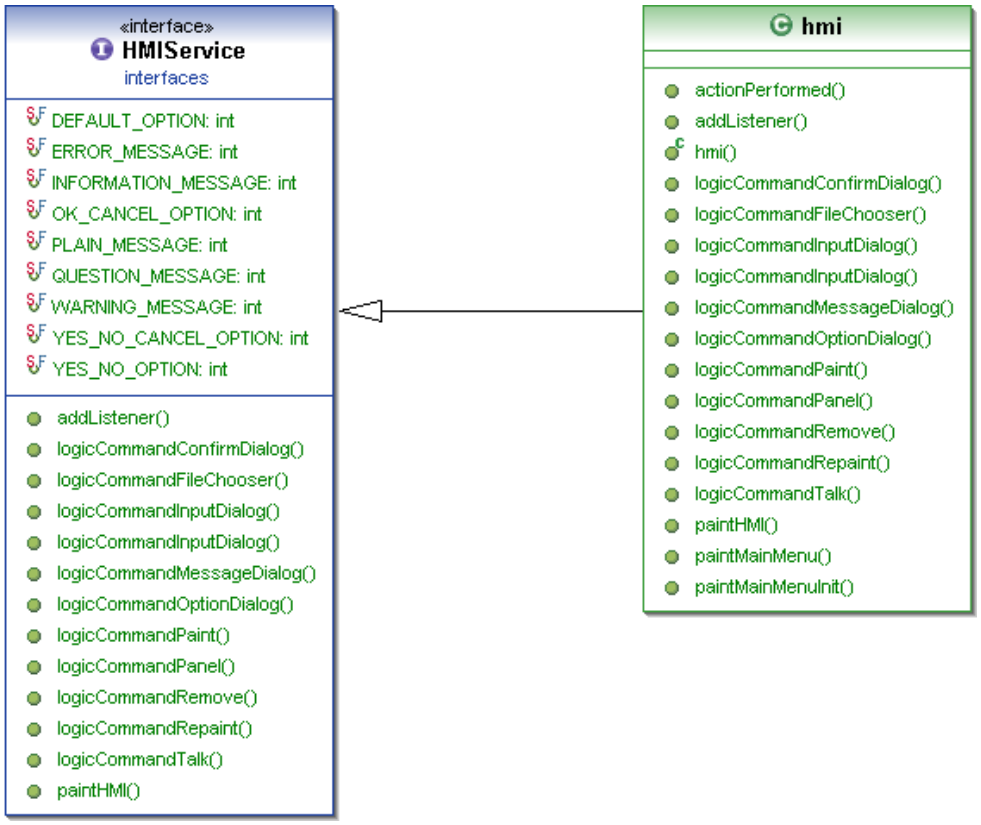

Figura 4.- Diagrama de clases de HMI.

El *middleware* de generación de interfaces gráficas dinámicas desarrollado publica un servicio en la plataforma de servicios del *framework OSGi*.

La interfaz que se publica en el servicio de "hmi" recibe el nombre de HMIService. Volviendo de nuevo a la Figura 4, las variables que se muestran al principio son las que están relacionadas con los métodos "logicCommandInputDialog()", "logicCommandConfirmDialog()", "logicCommandMessageDialog()" y "logicCommandOptionDialog ()", que son los que se encargan de mostrar mensajes de información en la aplicación HMI. Su función se detallará en el Apéndice 2 de esta memoria.

Todos estos métodos integrados en la interfaz permiten la comunicación de los módulos con el *bundle HMI*, preocupándose de la generación de interfaces gráficas dinámicas sólo el *middleware* desarrollado.

El método addListener() es a través del cual las aplicaciones del *framework* OSGi se añaden como *listeners*, para que puedan ser referenciados a la hora de devolver la información de vuelta al módulo que haya solicitado la generación de una interfaz gráfica al módulo HMI.

4.- Tratamiento de datos con XML

Un *parser* o analizador sintáctico lee el documento XML y verifica que es XML bien formado, dependiendo del utilizado, se puede comprobar que el XML sea válido. Otra de sus funcionalidades es que los documentos XML analizados también pueden ser incorporados a cualquier aplicación. Como se explicará en el apartado 4.2, en la tecnología escogida en este proyecto, esto es posible mediante la definición de un DTD, a partir del cual se generarán unas clases en lenguaje Java que permitirán validar el documento y agilizar el acceso al mismo.

4.1.- Descripción XML del interfaz gráfico

A la hora de elegir un metalenguaje apropiado, habrá que tener en cuenta las características que precisa la aplicación que se va a desarrollar. Teniendo en cuenta las características del sistema a implementar, la tecnología escogida será la más adecuada. En este proyecto XML fue el metalenguaje utilizado, sobre todo porque se trata de un lenguaje extendido a nivel global, ya que es uno de los más usados en internet. Además, como se resumió en el apartado de introducción de esta sección, es un lenguaje que nos permite dar un formato a los datos basado en etiquetas, que serán definidas en un DTD, facilitando el acceso a los datos. Los principales fundamentos en los que se basa XML son los siguientes:

- Es extensible: Después de diseñado y puesto en producción, es posible entender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan *bugs* y se acelera el desarrollo de aplicaciones.
- Si alguien externo a la creación de un documento XML decide usarlo, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones. Se pueden comunicar aplicaciones de distintas plataformas, sin que importe el origen de los datos
- Se transforman datos en información, pues se le añade un significado concreto y se asocian a un contexto, con lo cual tenemos flexibilidad para estructurar documentos.

Después de confirmar que XML es la tecnología apropiada, el siguiente paso sería buscar un "parseador" o *parser* (programa que permite trabajar con documentos XML), que vendría a ser una aplicación implementada en un lenguaje de programación, en este proyecto sería Java, que contiene todos los elementos para parsear (identificar los elementos de un documento XML). En el siguiente apartado se explican tres APIs más importantes relacionadas con esta tecnología, en la que se incluye la que ha sido usada en este proyecto.

4.2.- Procesado de interfaces gráficas

En el apartado anterior se definen las APIs más importantes en cuanto al tratamiento de documentos XML en Java, cada una tiene una aplicación específica sobre una aplicación a desarrollar. Lo que ha llevado a elegir a JAXB como tecnología final, es que proporciona un acceso rápido a los documentos permitiendo también su posterior lectura desde memoria, ya que también hace un volcado del documento, eso sí sólo permite trabajar con documentos XML que posean una estructura estática.

SAX y DOM proporcionan validación del documento XML, pero para conseguirlo, el desarrollador debe añadir el código extra necesario, lo cual puede ser complicado y conllevar a errores. JAXB, como contrapartida, puede mejorar la estructura y la validación del documento XML con clases Java generadas a partir de un esquema DTD. Como DOM, una aplicación JAXB genera una estructura que es volcada en memoria, pero DOM genera un árbol de contenido específico para cada documento nuevo consumiendo más memoria. Uno de los puntos fuertes de JAXB es que los métodos de acceso que se generan a partir del esquema tienen nombres específicos, lo que hace que las interfaces generadas sean más intuitivas que las de DOM.

JAXB por tanto cuenta con lo mejor de DOM y de SAX, ya que permite un acceso rápido a los documentos XML por tener unas clases pre-compiladas y ya cargadas en memoria. También se reduce el consumo de memoria al utilizar JAXB, con el único inconveniente de que se tendrá que conocer de antemano la estructura que va a tener el documento XML recibido y tener las clases apropiadas ya integradas en el programa. En este proyecto este punto no es importante, puesto que sólo se disponen de dos tipos de documentos XML, el enviado de un módulo a la aplicación HMI, cuya estructura también se utiliza como un método estandarizado para intercambiar los elementos gráficos, y el documento de respuesta que contiene todos los elementos gráficos que posee la interfaz propia del módulo incluyéndose también en este documento los eventos causados por el usuario.

Teniendo en cuenta lo anterior, JAXB se hace más apropiado para programas cuyos documentos XML no vayan a cambiar su estructura de forma dinámica, como ocurre en este proyecto. En cambio, si la aplicación a desarrollar necesita documentos dinámicos, se podrá elegir SAX por su acceso rápido a los documentos XML o DOM por cargar estos documentos en memoria, facilitando su posterior acceso.

5.- Interfaz de las aplicaciones con el middleware HMI

5.1.- Interfaces sobre OSGi

OSGi cuenta con una plataforma de servicios para permitir la comunicación entre los módulos incluidos en la plataforma. Cada servicio estará relacionado con una interfaz en lenguaje Java que formará parte del *bundle* que exporta el servicio al resto de módulos.

Los módulos incluidos en la plataforma OSGi que quieran importar este servicio para la posterior utilización de los métodos especificados en la interfaz, deberán registrarse al servicio con el *Service Registry* (Plataforma de servicios).

Cada interfaz asociada a un servicio contendrá unos métodos públicos al resto de módulos del sistema que se registren, desarrollándose estos métodos en las clases Java contenidas en el *bundle* que exporta el servicio.

La principal interfaz a implementar es "org.osgi.framework.BundleActivator", que obliga a declarar dos métodos:

- Start: Método que se invoca al arrancar el Bundle.
- Stop: Método que se invoca al parar el Bundle.

La plataforma de servicios facilita la comunicación entre los módulos de la plataforma OSGi, permitiendo transparencia a través de las interfaces.

5.2.- Servicios sobre OSGi

En el apartado anterior se da una visión global de la función de los servicios en el *framework* OSGi y su relación con las interfaces de Java, pero ¿Qué es un servicio realmente?

Los servicios en OSGi están incluidos en lo que antes se mencionaría como *Service Registry* o plataforma de servicios, que permite a los dispositivos o módulos del sistema mantenerse al día de las actualizaciones más recientes. Esta plataforma *middleware*, abierta y basada en Java, también juega un papel fundamental en la vinculación de muchos dispositivos diferentes y permite la entrega y gestión de nuevas funcionalidades.

La plataforma de servicios de OSGi hace posible ampliar la vida de productos tan diversos como los incluidos en un coche o en un teléfono móvil mediante la actualización dinámica del software integrado. Esto se consigue al menor coste para los usuarios finales y sin interrupciones en el servicio.

Para los desarrolladores su ventaja está justificada: se les permite centrarse en sus aplicaciones, ya que la plataforma de servicios OSGi elimina largas fases de prueba y eliminación de errores en una gran cantidad de sistemas operativos distintos.

El *middleware* implementado ofrece varios servicios al resto de módulos del sistema, entre los que se encuentran:

- Teclado: Al estar enfocado al terreno de la telemática vehicular, se incluye la posibilidad de utilizar un teclado virtual integrado en la interfaz gráfica, permitiendo utilizar el sistema en equipos que carecen de un teclado físico y disponen de una pantalla táctil.
- Sintetizador de voz: Ofrece al resto de módulos de la plataforma un sistema mediante el cual se pueda avisar al usuario a través de textos hablados.
- Instalación y desinstalación de módulos: También se proporciona un sistema para añadir o suprimir *bundles* del *framework* OSGi, permitiendo añadir nuevas funcionalidades sin tener conocimientos en la plataforma.
- Gestión de mensajes de error y de información.
- Resto de servicios como son la creación de elementos gráficos a partir de objetos Java basados en elementos XML.

5.3.- Registro y adhesión a un servicio

Un servicio se asimila a un objeto básico en Java, sólo que es publicado en el registro de servicios de OSGi con el nombre de una o más interfaces Java.

Para comprender mejor cuál es el procedimiento a seguir a la hora de publicar o implementar un servicio, en este apartado se detallarán los pasos a seguir.

5.3.1.- Registro de un servicio

La publicación de un servicio en la plataforma de servicios es un ejemplo de la interacción con el *framework* OSGi, necesitándose siempre una referencia al *BundleContext* (contexto de ejecución bundle en el *framework OSGi*) para poder registrarlo. Un ejemplo de registro de un servicio es el que se efectúa en la clase *Activator* del *sistema HMI*:

```
//Se crea una referencia a la clase "hmi", que es la que contiene los
métodos declarados en la interfaz "HMIService".

hmiImpl=new hmi(context,"HMI application",speechSynthesizerService);

//Al contexto se le solicita que registre el servicio con la interfaz
"HMIService" y la referencia a la clase "hmi".

registration=context.registerService(HMIService.class.getName(),hmiImpl,n
ull);
```

Código 1. Registro de un servicio en la plataforma de servicios OSGi

Como se aprecia en el código anterior, básicamente el registro de un servicio en el *framework OSGI* se reduce a dos fases, la primera que sería crear una referencia a la clase que contiene los métodos desarrollados de la interfaz y una segunda que publica en el contexto un servicio de la interfaz "HMIService" con el código de la clase "hmi".

5.3.2.- Adhesión a un servicio

En el apartado anterior se registraba un servicio en la plataforma de servicios para conseguir que los métodos de la interfaz "HMIService" pudieran ser implementados por otros módulos en el *framework OSGi*. Aquí se explicará el código a incluir en un módulo para conseguir que pueda usar un servicio ya publicado. Para listar los servicios ya registrados en la plataforma, basta con escribir el comando "services" en la consola de OSGi. Un ejemplo sería el siguiente, en el que el servicio no está siendo usado por ningún módulo todavía:

```
{org.hmi.interfaces.HMIService}={service.id=28}
 Registered by bundle: org.HMI_1.0.0.qualifier [22]
        No bundles using service.
```

Código 2.- Servicio HMIService en la plataforma de servicios OSGi (1).

Cuando un módulo se une a un servicio ya registrado, en la consola se mostraría lo siguiente:

```
{org.hmi.interfaces.HMIService}={service.id=28}
 Registered by bundle: org.HMI_1.0.0.qualifier [22]
 Bundles using service:
      org.webBrowser_1.0.0.qualifier [21]
```

Código 3.- Servicio HMIService en la plataforma de servicios OSGi (2).

Como puede apreciarse, el módulo "webBrowser" que es el encargado de mostrar el navegador web en el *sistema HMI*, está usando el servicio de "hmi".

Para utilizar un servicio presente en el *framework*, los únicos pasos a seguir serían:

```
//Se le solicita al contexto que nos devuelva la referencia al servicio del sintetizador de voz.

speechSynthesizerServiceReference=context.getServiceReference(SpeechSynthesizerService.class.getName());

//Con la referencia obtenida, se pide el objeto que referencia a la interfaz buscada.

speechSynthesizerService=(SpeechSynthesizerService)context.getService(speechSynthesizerServiceReference);
```

Código 4.- Adhesión a un servicio de la plataforma de servicios OSGi.

Con lo explicado anteriormente, un módulo podrá ser integrado en el *framework* OSGi, pudiéndose comprobar si los pasos seguidos están bien realizados mediante la consola de comandos de OSGi utilizando el parámetro "services" como se ha indicado antes.

Así se conseguirá partir de un módulo o *bundle* que registre un servicio en el *framework*, en este proyecto el encargado sería el *middleware HMI*, para que posteriormente pueda ser usado por el resto de módulos de la plataforma OSGi, restando complicaciones a los desarrolladores de las aplicaciones.

5.4.- Mensajes de intercambio

Tras conocer los conceptos básicos relacionados con la comunicación entre módulos en la plataforma OSGi, en este apartado se detalla cuál es el procedimiento de comunicación incluyendo los mensajes que son intercambiados.

En una primera fase, el *bundle HMI* registra el servicio "HMIService" en la plataforma de servicios, lo que posibilita que el resto de módulos puedan asociarse a él. El requisito necesario a incluir en la aplicación que quiere ofrecer el servicio es contar con una interfaz que posea los métodos que quieren ser compartidos con el resto de aplicaciones o módulos, para posteriormente registrar un servicio que se encuentre relacionado con dicha interfaz, este proceso se detallaría en el apartado 5.3.1 (Registro de un servicio).

Tras comprobar que el servicio se encuentra correctamente registrado en la plataforma de servicios como se explicó en el apartado 5.3.2 (adhesión a un servicio), el módulo interesado se unirá a este servicio contando con los métodos propios de la interfaz que haya sido vinculada a éste.

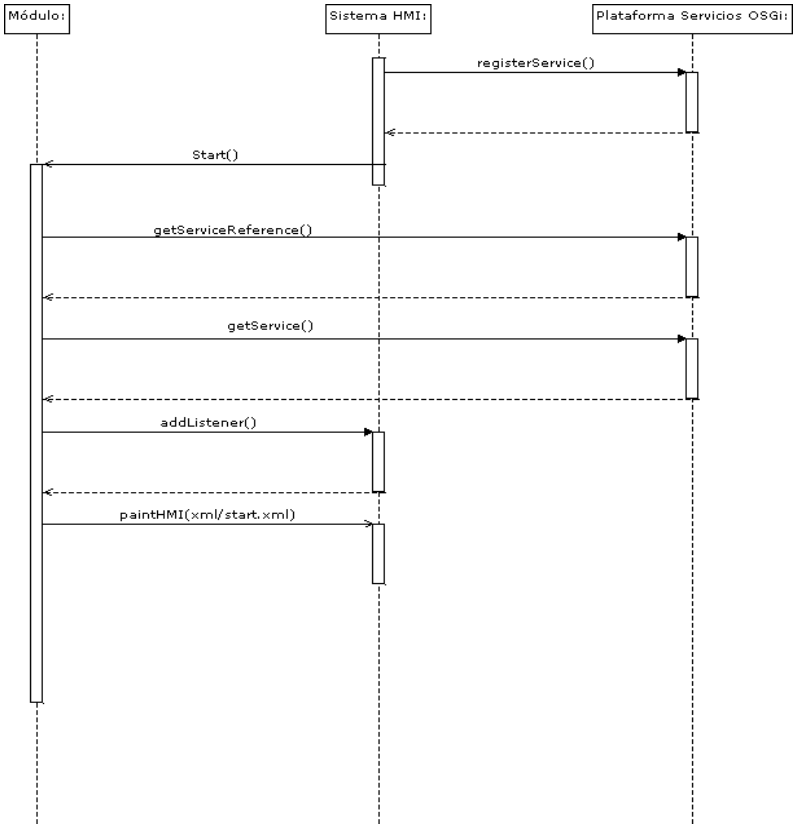

Figura 5.- Mensajes comunicación inicial módulo-sistema HMI.

Los pasos que sigue la comunicación de los módulos con el *framework* OSGi se muestra en la Figura anterior. Cabe destacar que como puede apreciarse, los módulos integrados en el *framework* OSGi que pretender usar el servicio de creación de interfaces que ofrece el *middleware HMI* se encuentran en un modo inactivo, esperando a que sean iniciados por el usuario a través del menú principal del sistema HMI. Tras esto el módulo seleccionado se registra al servicio del *middleware* desarrollado en este proyecto, pasando su referencia para el sistema pueda identificarlo para posteriores notificaciones (método addListener()). Tras esto, el *bundle* pasa al sistema de creación de interfaces dinámica una referencia al documento XML que se usará para crear la ventana inicial de programa.

Cada uno de los métodos que ofrece el *middleware HMI* cuenta con una forma estandarizada para los mensajes de envío y respuesta que se generarán entre el *bundle* desarrollado en este proyecto y el resto de módulos. Estos métodos reciben como parámetros de entrada objetos que proceden de dos documentos XML que han sido adaptados para el módulo HMI en particular.

La característica de utilizar documentos XML como "archivos" de información para crear las interfaces gráficas se consigue mediante el API JAXB, que fue explicada en el apartado 2.2, que permite la creación de clases Java a partir de unos documentos llamados "esquemas" o DTDs, que contienen la estructura de los documentos XML que serán procesados por el sistema HMI.

Los esquemas en los que se basará el *middleware HMI* para la interpretación de los documentos XML de intercambio entre los módulos del *framework* OSGi son los siguientes:

- GUIElements: Documento que contiene los parámetros necesarios para la generación de elementos gráficos de un módulo del *framework* OSGi al módulo HMI.
- ReplyClasses: Contiene los elementos utilizados entre la aplicación HMI y el resto de módulos como respuesta a una acción del usuario.

Cada uno de los parámetros contenidos en estos esquemas contienen a su vez elementos necesarios en la aplicación HMI para poder crear correctamente cada uno de los elementos gráficos demandados y también para dar una respuesta estandarizada al resto de módulos. A continuación se listan los más importantes:

- Nombre: identificará cada uno de los elementos gráficos generados
- Posiciones relativas en los ejes horizontal y vertical.
- Dimensiones del objeto a crear.
- Fuente, si se trata de un elemento que contiene texto.
- Parámetro "File", que permitirá cargar imágenes en objetos de tipo JButton (Botones gráficos de la librería swing de Java).
- Texto: contendrá los caracteres a representar en el objeto a incluir en la interfaz gráfica.

La estructura de uno de los elementos que formarían parte de un documento DTD a partir del cual serían generadas las clases Java sería como sigue:

```
<xsd:complexType name="TypeLabel">     //Nombre del elemento.
   <xsd:sequence>                     //Parámetros del elemento.
         <xsd:element name="x" type="xsd:int"/>
         <xsd:element name="y" type="xsd:int"/>
         <xsd:element name="w" type="xsd:int"/>
         <xsd:element name="h" type="xsd:int"/>
         <xsd:element name="text" type="xsd:string"/>
         <xsd:element name="font" type="xsd:int"/>
   </xsd:sequence>
   <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
```

Código 5.- Elemento "TypeLabel" del DTD "GUIElements.xsd".

Teniendo en cuenta las reglas de diseño de documentos DTD planteadas, se puede conformar un documento ".xsd" que es el que contendrá todos los elementos relacionados con la comunicación XML entre módulos del *framework* OSGi.

Para la generación de las clases Java a partir del documento ya terminado, basta con ejecutar el siguiente comando en un entorno "Windows":

```
xjc c:\file.xsd –p ejemplo
```

Código 6.- Comando generación clases Java

En el código anterior, el archivo file.xsd será el documento DTD que contenga todos los elementos XML que se quieren añadir a la aplicación a desarrollar como objetos Java.

Algo que se debe mencionar es que el compilador permite o acepta ciertas opciones adicionales al momento de generar las clases para nuestro esquema; en específico, podemos observar la existencia de un parámetro adicional, -p ejemplo, que define el paquete donde se van a generar las clases.

6.- Reconocimiento de voz

Los sistemas integrados en vehículos comerciales como el desarrollado han incorporado durante años botoneras en el panel central del salpicadero y controles en el volante para el control del sonido o el manos libres.

Con el aumento de las funcionalidades del control del vehículo se hace indispensable el desarrollo de interfaces de voz para minimizar al máximo la distracción y pérdida de concentración que supone utilizar los dispositivos táctiles convencionales.

Por ello es por lo que se ha implementado una funcionalidad de control mediante voz que permite lanzar e interactuar con aplicaciones del sistema *middleware HMI*.

La interfaz de voz desarrollada ha sido implementada sobre el *framework OSGi*, el cual utiliza *Java* como lenguaje de programación.

Por otra parte, para el desarrollo interno de los servicios del interfaz, reconocimiento y síntesis de voz, se optó por la utilización de *Loquendo SDK* para implementarlos, ya que además de otras muchas características, su expresividad y la calidad del servicio ofrecido lo destacaron entre las demás opciones.

6.1.- Diseño de la interfaz de voz

La idea con la que se diseña este interfaz de voz ofrece una solución completa para la provisión de servicios tanto de reconocimiento como de síntesis de voz en una unidad de a bordo. En la Figura 6 se muestra el diagrama de bloques general de la estructura del sintetizador de voz.

En la parte superior del diagrama se muestran las aplicaciones y el tipo de comunicación que estas realizan con el sistema HMI de la unidad de a bordo. En este caso la comunicación entre ambas es bidireccional ya que independientemente que registren o soliciten servicios a la plataforma de servicios OSGi, estas aplicaciones realizan una comunicación con el sistema HMI para poder ser mostradas gráficamente en el mismo.

En la parte inferior del diagrama está representado el interfaz de voz. Este está compuesto por cuatro módulos, los dos superiores representan los servicios de reconocimiento y síntesis de voz, en donde esta implementada toda la lógica relacionada con dichos servicios, *ASR Service* y *TTS Service*. Estos son ofrecidos por los *bundles org.service.asr* y *org.service.tts,* descritos en el siguiente punto. Mientras tanto, en la parte inferior del diagrama, se muestran dos módulos que forman parte de la librería necesaria de Loquendo. Estos contienen las clases y requisitos indispensables para la síntesis y el reconocimiento del habla, por lo que los módulos que representan los servicios dependen cada uno de uno de ellos para su correcto funcionamiento. Destacar también de esta representación del interfaz que el módulo de reconocimiento de voz, *ASR Service,* tiene una comunicación bidireccional con el HMI ya que será este quien en un primer lugar solicite al módulo *ASR Service* su servicio con una determinada gramática, para luego una vez reconocida la orden dada, si es que se encuentra definida dentro de la gramática,

devolver el resultado del proceso para su posterior ejecución. En cambio en el servicio de síntesis de voz, *TTS Service*, la comunicación con el HMI del sistema es unidireccional ya que tras la solicitud de su ejecución, este módulo cargará las voces y demás requisitos necesarios para su funcionamiento pudiendo así poder reproducir el mensaje deseado por la salida de audio del dispositivo, no teniendo que volver a comunicarse con el sistema.

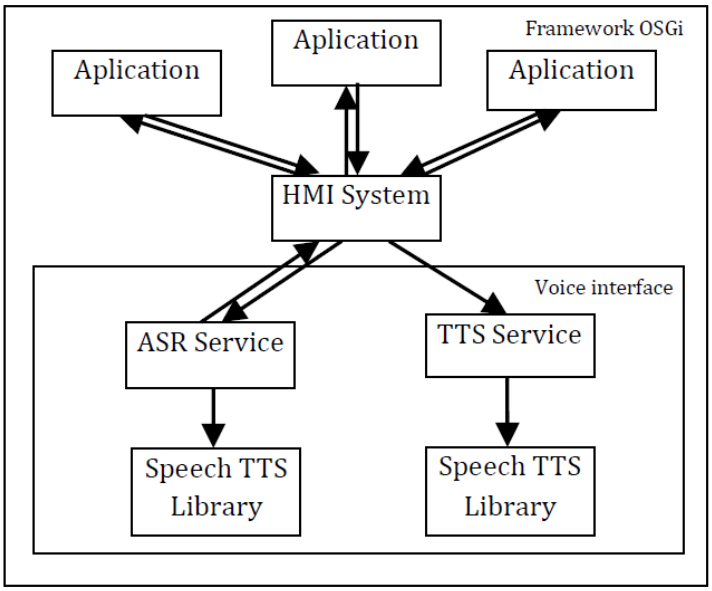

Figura 6.- Diagrama de bloques sintetizador de voz.

6.2.- Desarrollo del componente

Este apartado se centra en la explicación en detalle de la manera de implementar esta solución modular, dinámica y genérica por la que se ha optado para la creación del interfaz de voz y así comprender mejor su funcionamiento. La estructura de este apartado comienza por la descripción de la funcionalidad y de los métodos de cada una de las clases que forman los módulos de esta solución, terminando con un diagrama formado por todos ellos para entender las relaciones que los unen.

- Módulo *org.service.asr*. Este módulo se encarga del servicio de reconocimiento de voz. Está formado por dos paquetes: *org.service.asr* y *org.service.asr.impl*, como se muestra en la Figura 7. En el primero de ellos se encuentra la interfaz *IASR* que contiene los métodos que serán publicados por este módulo en forma de servicio en la plataforma del *framework OSGi* para que puedan ser accedidos por el resto de módulos. Por otro lado, el segundo paquete está compuesto por la implementación de varias clases necesarias para el funcionamiento del servicio como es la reserva del micrófono, necesaria para su posterior uso. Entre estas clases se encuentra *ASRImpl* que contiene toda la lógica de funcionamiento del módulo, transparente e invisible a los demás. El resto de librerías y clases necesarias para el funcionamiento de este servicio están agrupadas en un módulo aparte, *org.loqjapi.asr*. Con esta separación se consigue que en futuras actualizaciones ofrecidas por *Loquendo* para mejorar su tecnología no sea necesaria la

modificación del módulo donde se ha implementado el servicio propiamente dicho, simplemente se modificará este módulo extra. Además, esta división aporta al proyecto parte de la modularidad y escalabilidad que se pretende que tenga. En siguientes apartados se comenta el contenido de este módulo extra que contiene las librerías.

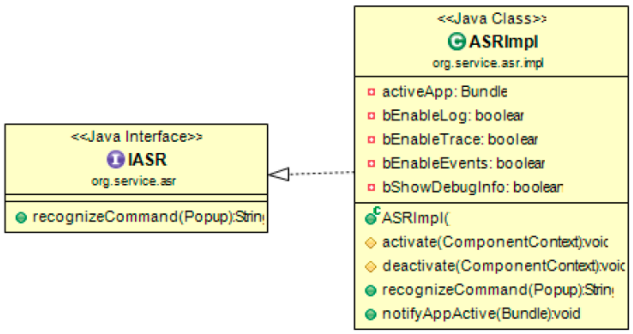

Figura 7.- Diagrama de clases del módulo ASR

- Módulo *org.service.tts*. Este módulo es el encargado del servicio de síntesis de voz. Al igual que él anterior, está compuesto por dos paquetes: *org.service.tts* y *org.service.tts.impl*, como se observa en la Figura 8. El primero está formado por la interface *ITTS* cuyo contenido son los métodos ofrecidos para que los demás módulos puedan hacer uso del servicio. El segundo está compuesto por la clase *TTSImpl* encargada del funcionamiento interno de este. Este, a diferencia del primero, no es publicado por el módulo por lo que no es visible a los demás, ya que el funcionamiento interno de este servicio solo es de interés para él. Al igual que el módulo de reconocimiento de voz, este no contiene en su interior las librerías y clases necesarias para su funcionamiento, encontrándose todas estas en un módulo aparte para conseguir los objetivos mencionados anteriormente de modularidad y escalabilidad.

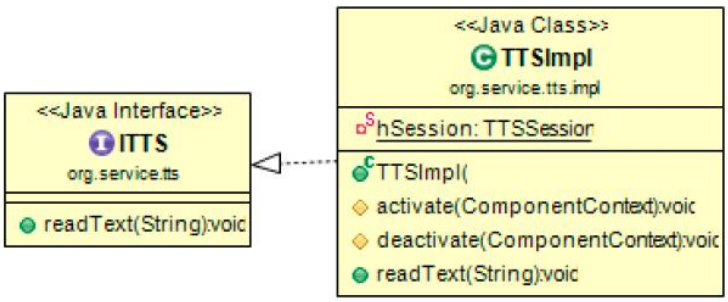

Figura 8.- Diagrama de clases del módulo TTS

- Módulos *org.loqjapi.asr* y *org.loqjapi.tts*. Estos módulos son los que contienen las librerías y clases necesarias para el correcto funcionamiento de los servicios de voz. La explicación de cada una de estas librerías no se realizará en esta memoria por considerar que queda fuera del ámbito de este trabajo ya que tratan aspectos de bajo nivel y su explicación no es relevante para el entendimiento de este.
- Módulo *org.observer*. La función de este módulo es aunque muy sencilla, imprescindible para el funcionamiento de este proyecto. Este módulo se limita a ofrecer una interfaz denominada *IAppActiveListener* cuyo contenido es un único método denominado *notifyAppActive()*. Este permite que el módulo de reconocimiento de voz permanezca informado de cuál es la actual aplicación en ejecución o, en ausencia de esta, indicar que el menú principal del HMI es el que está en primer plano. Esta funcionalidad es imprescindible para que el reconocimiento de comandos se realice de una manera contextual, lo que facilita que el HMI y cada aplicación del sistema tenga una serie de comandos específicos y personalizados, proporcionando modularidad al servicio de reconocimiento de voz, ya que en caso contrario todos los posibles comandos deberían estar en una misma gramática. Esto produciría problemas de eficiencia y escalabilidad, ya que por cada nueva aplicación que tuviera comandos de voz, habría que añadirlos a esta gramática común, haciendo que la comprensión y modificación de esta fuera muy compleja por la cantidad de información que albergaría, sobretodo en sistemas con muchas aplicaciones y con muchos comandos. Además, esta gramática global también incorporaría problemas tras la eliminación de una aplicación, ya que habría que modificarla para eliminar los comandos de la aplicación desinstalada. Por otra parte, la actualización o modificación de comandos seria también un proceso arduo sobre todo en gramáticas muy grandes. Pero para poder realizar el reconocimiento de comandos de forma contextual, no solo se requiere conocer cuál es la aplicación que está en primer plano sino que se debe proporcionar algún método para la creación de gramáticas y para el procesamiento de comandos de una manera genérica, dinámica y contextual a cada aplicación, lo que nos lleva al siguiente módulo.
- Módulo *org.i.voice*. Este módulo, al igual que el anterior, contiene un interfaz que es ofrecida para su uso. Este interfaz, *IVoice*, ofrece dos métodos: *getGrammar()* y *processCommand()*. El primero será el encargado de generar, si es la primera vez, u obtener, si ya se ha ejecutado antes, la gramática especifica de la aplicación en ejecución. El segundo método, por su parte, es el encargado de, una vez obtenido un resultado en el reconocimiento de voz, ejecutar el comando solicitado. Este interfaz tiene que ser implementado por toda aplicación que desee tener comandos de voz, ya que será en la implementación de estos dos métodos donde residirá la contextualización particular de cada una de ellas. Con este módulo conseguimos que la implementación de este interfaz de voz se realice de una forma sencilla y dinámica, ya que la incorporación a una nueva aplicación de comandos de voz solo requiere implementar dicha interfaz, haciendo transparente a las aplicaciones el proceso interno realizado por el servicio de reconocimiento.

- Módulos HMI y Aplications. Estos módulos han sido utilizados como referencia para validar la propuesta, por ello se explica brevemente su función en este proyecto, ya que queda fuera de este TFG el entendimiento de su funcionamiento interno. El sistema HMI es el encargado de la generación dinámica de interfaces graficas para las aplicaciones software integradas en el sistema. Las aplicaciones, por su parte, son necesarias para validar la propuesta de este proyecto, haciendo uso del interfaz de voz. Las aplicaciones usadas como ejemplo han sido una calculadora y un visor de imágenes, denominados *Calculator* y *Slide Show* respectivamente.

6.3.- Funcionamiento básico de la solución

En este apartado se muestra y explica un ejemplo de ejecución y uso del interfaz de voz, empezando por la parte visible *(front-end)* de este y acabando con los detalles más significativos a nivel de código que se emplea en su funcionamiento *(back-end)*.
En la Figura 18 se muestra la pantalla principal del *middleware* HMI instalado en la OBU.

Figura 9. Pantalla principal del *middleware* HMI

Se pueden observar diversos botones e iconos de los cuales solo se comentaran los relacionados con el interfaz de voz, ya que los demás quedan fuera del ámbito de este proyecto.

Los botones e iconos relacionados con este interfaz son los siguientes:

-  Este es el encargado de activar o desactivar el sintetizador de voz, en verde indica que está activado y en tono grisáceo que esta función no está disponible.

-  Este botón es el encargado de ejecutar el reconocedor de voz cuando se le pulsa. En ese momento aparece un *pop-up* que indica que se está ejecutando dicha función esperando recibir alguna palabra o frase. Cuando finaliza, el *pop-up* desaparece. A diferencia del anterior, este botón no cambia indicando si está activo o no, sino que en este caso esta función siempre esta desactivada y cuando es pulsado dicho botón se ejecuta y aparece el *pop-up*.

- 

Este icono representa que el reconocimiento de comandos se está ejecutando. Este es el pop-up que se lanza cuando se ejecuta y sirve para indicar cuándo podemos decir nuestra orden hablada.

En la Figura 19 se presenta la captura de la unidad de a bordo con el sintetizador desactivado y el reconocedor de comandos en ejecución. En esta se puede observar lo comentado en los puntos anteriores.

Figura 10. Captura de la unidad de a bordo con el sintetizador desactivado y el reconocedor en ejecución.

Una vez explicada la parte visible del interfaz, queda comentar los aspectos más significativos que realiza internamente este.

En este se explica cómo realiza el interfaz a nivel de código sus tareas más significativas, como la contextualización de los comandos de voz por parte del ASR en función de la aplicación en ejecución. Todo ello se muestra en el siguiente fragmento de código:

```
public String recognizeCommand(Popup popup) {
...
   String [] serviceReference = (String[])activeApp.getRegisteredServices()[0].getProperty("objectClass");
   for (int cont=0;cont<serviceReference.length;cont++){
         if(serviceReference[cont].toString().matches("es.umu.i.voice.IVoice")){
           service_found = true;
           voice = (IVoice)activeApp.getBundleContext().getService(activeApp.getRegisteredServices()[0]);
           grammar = voice.getGrammar();
           break;
         }
   }
   if (service_found){
   ...
   if (ordenOK) result = voice.processCommand(result);
   }
   else result ="Esta aplicación no tiene implementados comandos de voz";
   return result;
   }
public void notifyAppActive(Bundle app) {
   activeApp = app;
   if (bShowDebugInfo) System.out.println("BUNDLE ACTIVE: "+activeApp.toString());
}
```

Código 7.- Funcionamiento básico reconocimiento de voz.

En este código se observa cómo en el comienzo de la ejecución del reconocedor de voz se solicitan todos los servicios que tiene registrada la aplicación activa. Esto se realiza con la función *activeApp.getRegisteredServices*. Una vez se conoce que servicios ofrece dicha aplicación, el ASR busca si alguno de ellos es *es.umu.i.voice.IVoice*, servicio necesario para el reconocimiento de comandos. Si la búsqueda es positiva se observa que el reconocedor de voz solicita la gramática particular de dicha aplicación llamando a la función *getGrammar()*. En este punto, aunque no se muestra, el reconocedor realiza una serie de acciones necesarias para su correcto funcionamiento como es: cargar dicha gramática, iniciar una sesión, cargar un RO (*Recognize Object*), habilitar el micrófono, etc. Tras esta serie de acciones, si se ha obtenido como resultado una orden definida en la

gramática, se procederá a la ejecución de la función *processCommand()* donde está implementado el comportamiento de cada una de estas.

Al final del código se observa la función *notifyAppActive()* que es la encargada de indicarle al ASR qué aplicación es la activa en cada momento. Esta función pertenece a la interfaz *IAppActiveListener* contenida en el paquete *es.umu.observer*.

7.- Explotación del sistema en el proyecto CENIT OASIS

El *middleware* desarrollado ha sido elegido para implementar la interfaz gráfica de un proyecto real denominado CENIT OASIS, encapsulando en el menú principal cada una de las interfaces gráficas de los servicios que éste ofrece. A continuación se explica en qué consiste este proyecto.

El proyecto Operación de Autopistas Seguras, Inteligentes y Sostenibles (OASIS), primer proyecto español en abordar el diseño de la autopista del futuro, se enmarca dentro del programa CENIT, cuyo objetivo principal es la financiación de grandes proyectos integrados de colaboración público-privada en investigación industrial de carácter estratégico.

En OASIS se ha optado por realizar un despliegue de los diferentes servicios haciendo uso de IMS (explicado más adelante), el cual ofrece una serie de ventajas, como son la gestión de sesión, seguridad, calidad de servicio, ofrece movilidad de las comunicaciones a nivel de aplicación, etc. En el caso de OASIS, su uso ha estado motivado fundamentalmente por la capacidad de negociación de parámetros para el inicio de sesión, lo que nos permite establecer una sesión con un servicio previa negociación de una serie de parámetros.

Es posible realizar un despliegue IMS sin contar con el soporte de una operadora de telefonía utilizando una implementación *open source* del núcleo IMS. En el caso de OASIS se ha utilizado OpenIMS.

7.1- Cliente IMS

IMS es un estándar definido dentro del 3GPP (3rd Generation Partnership Project), para la gestión de servicios ofrecidos bien por las propias operadoras de red, o bien por terceros proveedores de servicios. Se introdujo por primera vez en 2002 en la versión 5 de las especificaciones de 3GPP. IMS juega un papel importante en las especificaciones de 3GPP, de modo que en todas y cada una de las versiones desde que se introdujo sigue evolucionado y mejorando para adaptarse a los diferentes cambios en las tecnologías de acceso a la red.

IMS pretende ser una arquitectura de servicios estandarizada basada en IP, que permita trabajar tanto a usuarios móviles como usuarios fijos haciendo uso de las redes de voz y datos actuales. Pretende servir para todo tipo de servicios, tanto actuales como futuros, que se puedan prestar por Internet, permitiendo a los usuarios que se desplazan poder utilizarlos de igual forma que cuando se encuentran en ubicaciones fijas. IMS hace que el mundo IP de Internet converja con la telefonía móvil celular y permite además que los operadores de ISP puedan controlar y facturar el uso de los servicios ofrecidos.

7.2- Integración con middleware HMI

Dentro del proyecto OASIS, se ha establecido que el acceso a cada uno de los servicios integrados en esta plataforma sea a través del *middleware HMI*. Esto se ha debido a la dificultad que supone para el usuario de un sistema de a bordo el manejo de diferentes

ventanas independientes para cada servicio disponible. Para ello desde la Universidad de Murcia se ha desarrollado un servicio OSGi (denominado como Renderizado Dinámico de Interfaces) que presenta un sistema HMI único que integra las nuevas aplicaciones del protecto CENIT OASIS dentro de un interfaz unificado. Este módulo se puede ver reflejado en la siguiente Figura.

Sobre este interfaz gráfico genérico se integraron las diferentes aplicaciones que permitían interactuar con los diferentes servicios y visualizar la información enviada desde la infraestructura. El interfaz hombre-máquina consta de una pantalla principal, en la que se puede activar cualquiera de los tres servicios *(VSL, Planificación de Rutas* o *Incidencias*) y acceder a la pantalla de visualización de cada uno. La aplicación permite seleccionar una cuarta opción (*Servicios Integrados*) que muestra en una misma pantalla información de todos los servicios que se encuentren activados. La pantalla también permite visualizar si las comunicaciones con los servicios se realizan a través de WiFi o 3G, y ofrece la opción de cambiar de tecnología en caso de que ambas se encuentren disponibles. En condiciones normales, será el router móvil quien seleccione automáticamente la tecnología a emplear cuando ambas estén disponibles, en función de la calidad de la señal recibida, estando esta funcionalidad disponible en el interfaz de la OBU (*On Board Unit*) por motivos puramente demostrativos.

Figura 11- Interfaz principal de la OBU.

7.3- Servicios Integrados

En este apartado pasamos a describir las características principales del funcionamiento del interfaz de la plataforma.

7.3.1- Servicio VSL

Este servicio es el encargado de mostrar la velocidad recomendada para un tramo concreto de vía, el cual se va actualizando dependiendo del posicionamiento del vehículo y del estado de la vía.

En el interfaz de esta aplicación, como se puede ver en la siguiente Figura, consta de un mapa done se puede ver la situación del vehículo, y de un panel en el lado izquierdo que muestra la velocidad recomendada para la zona en la que nos encontramos. En caso de que el servicio de sintetizado de voz se encuentre activo, cuando haya un cambio en la velocidad recomendada, ésta será notificada al usuario. Debajo de la velocidad recomendada el interfaz presenta los botones que permiten la conexión y desconexión del servicio, respectivamente.

Figura 12- Interfaz de servicio VSL.

7.3.2- Servicio Planificación de rutas

En la siguiente Figura podemos observar el interfaz para el servicio de planificación de rutas. Se puede ver la localización del vehículo en el mapa, junto con la ruta planificada, marcada por una línea en rojo. Debajo del mapa, podemos observar los botones de conexión y desconexión del servicio, respectivamente.

A continuación podemos ver un cuadro de texto llamado “Siguiente paso”, el cual nos indica cual es la siguiente acción que se tiene que llevar a cabo; esta acción será leída por el sintetizador en caso de que se encuentre activo. A la derecha podemos ver tres campos que nos informan de la velocidad media, del tiempo al destino y la distancia a destino respectivamente.

Figura 13- Interfaz de servicio de planificación de rutas.

7.3.3- Servicio Incidencias

A continuación podemos observar la interfaz para el servicio de incidencias. Dicho servicio presenta un mapa que nos indica la posición del vehículo, junto con las incidencias que se vayan presentando en la vía.

En el lado izquierdo, contamos con un panel de configuración para indicar el tipo de eventos que queremos consultar además del radio de acción de la consulta y la frecuencia con la que queremos consultar. La última opción de todas, nos permitirá mostrar o esconder el texto relacionado con la incidencia.

También contamos con una barra de progreso que nos indica el tiempo restante que queda para el próximo chequeo de incidencias. Y finalmente, contamos con los botones para establecer la conexión y desconexión del servicio.

Figura 14- Interfaz de servicio de incidencias.

7.3.4- Servicios Integrados

En dicha aplicación se muestra la información combinada de los servicios que se encuentren activos. En el caso de la siguiente Figura, se muestra la información de los tres servicios involucrados en OASIS.

Como se puede ver aparece la ruta que está siguiendo el vehículo, con un trazado en rojo. Del mismo modo, aparece sobre el icono verde del vehículo la velocidad variable(VSL) recomendada y por último, sobre la vía se puede observar una incidencia en la vía que indica que existe congestión.

Figura 15- Servicios integrados.

Con dicha aplicación se pretende integrar la información proveniente de los tres servicios, para que el usuario evite distracciones y pueda observar desde una única ventana la información unificada.

8.- Conclusiones

8.1.- Principales contribuciones

En este proyecto se ha implementado y diseñado un módulo middleware sobre OSGi que permite la integración directa de nuevas aplicaciones vehiculares sin la necesidad de programar una interfaz de usuario independiente. El modelo de interfaz se define mediante un esquema XML normalizado que facilita las tareas de despliegue y fomenta la cohesión del sistema gráfico de cara al usuario.

El lenguaje de desarrollo escogido es Java, un lenguaje portable, permitiendo la ejecución de la plataforma en cualquier dispositivo como sistemas de a bordo, que es en lo que basa la creación de todos los elementos propios de la interfaz gráfica.

En lo relacionado con los objetivos propuestos para este proyecto, se han cumplido en su totalidad, implementando incluso la tecnología desarrollada en un prototipo real, que cuenta con una pantalla táctil y que al tratarse de un sistema compacto podría incluirse como sistema de a bordo.

OSGi es una tecnología muy a tener en cuenta, por la multitud de características que ofrece al usuario final, y por ello será una de las tecnologías más extendidas durante los próximos años.

El software desarrollado al estar implementado sobre lenguaje Java permite su ejecución en sistemas de a bordo de todo tipo, ya que este lenguaje de programación ofrece independencia sobre el sistema operativo al ejecutarse sobre una máquina virtual.

Al implementar un nuevo sistema de reconocimiento y síntesis de voz se mejora la comunicación con el sistema.

Tras comprobar la integración del *middleware HMI* con el proyecto CENIT OASIS se ha llegado a las siguientes conclusiones:

- IMS se posiciona como una solución estable y madura como plataforma para el despliegue de servicios.
- OSGi se establece como una opción madura para el desarrollo y despliegue de aplicaciones modulares, como así ha quedado demostrado en el desarrollo del software de a bordo.

En lo referente al interfaz gráfico desarrollado sobre OSGi, se ha visto que esta tecnología es claramente útil en el vehículo, en donde la OBU hace las veces de pasarela de servicios para el conductor y los pasajeros. Además, ahora con la integración del nuevo sistema de voz, se hace posible la comunicación directa usuario - máquina.

Como conclusión final, el *middleware HMI* con las diferentes mejoras incorporadas cumple con la finalidad para la cual fue diseñado, dar soporte gráfico común a un conjunto de aplicaciones software sobre la pasarela de servicios OSGi.

8.2.- Mejoras identificadas en la plataforma

Durante el desarrollo de este proyecto se han identificado varias ampliaciones que podrían incorporar nuevas funcionalidades al *bundle* implementado de generación de interfaces gráficas:

- Implementación de un sistema de pestañas: Podría ser interesante que el sistema contara con un menú principal en el que las aplicaciones activas pudieran estar incluidas en pestañas independientes, agilizando su acceso.
- Integración de elementos flash: En una aplicación de carácter, en su mayor parte, visual, incluir elementos animados, haciendo que el producto final quedara con una mejor presentación al usuario.
- Adicción de nuevos elementos gráficos: Nuevos elementos aportarían nuevas funcionalidades a la aplicación HMI, facilitando a un desarrollador de un módulo del *framework* OSGi la creación de la interfaz gráfica.
- Arranque al inicio del sistema: Esta característica permitiría el arranque de la plataforma OSGi desde el inicio del sistema.
- Ejecución del reconocimiento de voz en segundo plano, para no tener que accionar un botón cada vez que se quiera utilizar.
- Añadir la personalización del interfaz gráfico reconociendo mediante voz al usuario.

8.3.- Líneas futuras

El *middleware* desarrollado en este proyecto fin de grado admite la integración de nuevas funcionalidades, siendo implementadas y añadidas al código fuente permitiendo un avance y una adaptación del sistema en el futuro.

Una nueva línea de trabajo relacionada con el f*ramework* OSGi podría ser el desarrollo de servicios vehiculares destinados a la mejora de la seguridad y el confort sobre la plataforma: comunicación entre vehículos para evitar accidentes, mejorar el sistema de avisos de la aplicación o simplemente hacerla más intuitiva de cara al usuario final.

OSGi cuenta con una característica denominada “Servicios declarativos” que permite el registro y la adhesión de servicios a través de la declaración de un componente que será el encargado de comunicarse con la plataforma de servicios. Una línea futura podría incluir la adaptación del *middleware HMI* a esta función del *framework OSGi.*

Una posible línea en la que se podría trabajar sería la comunicación de este con sistemas ADAS para alertar de situaciones peligrosas detectadas informando de ello al conductor o pasajeros del vehículo. Un ejemplo podría ser que sistemas que detecten conducción irregular como desplazamientos laterales constantes comuniquen esta situación peligrosa al conductor y al resto de pasajeros utilizando para ello la interfaz de voz, minimizando así el tiempo de reacción empleado para corregir esta situación peligrosa.

APÉNDICE 1: Manual de usuario

En este apartado se ofrece al usuario un manual completo en el que se explica detalladamente el uso y el funcionamiento de cada una de las funciones de la aplicación.

A1.1.- Conceptos básicos

En este apartado se le da al usuario una serie de conceptos útiles para entender este manual y conocer los aspectos más necesarios del uso de esta aplicación:

- Sintetizador de voz: Programa que se encarga de convertir texto escrito a voz.
- Java: Lenguaje de programación basado en objetos, las aplicaciones están escritas compiladas típicamente en *bytecode*, que será interpretado por la máquina virtual.
- Máquina virtual: Software que emula a una computadora y que permite ejecutar programas como si fuera una computadora real. En lo relacionado con Java, es la que permite la independencia de la plataforma en la que vaya a ser ejecutado el código (potabilidad).
- OSGi: Su objetivo es definir las especificaciones abiertas de software que permita diseñar plataformas compatibles que puedan proporcionar múltiples servicios. Fue pensado principalmente para su aplicación en redes domésticas y por ende en la llamada Domótica o informatización del hogar.
- Módulo: Estos módulos (llamados *bundles*) no son más que jars con algunas clases especiales propias de OSGi y un fichero de manifiesto con campos especiales, que entiende OSGi. Los *bundles* pueden colaborar entre ellos, indicando qué paquetes Java exportan a otros módulos y qué paquetes Java de otros módulos necesitan. Una plataforma que implemente OSGi (Equinox en este proyecto) es capaz de cargar, arrancar o parar estos módulos en tiempo de ejecución, sin necesidad de tirar la aplicación y volver a arrancarla. También es capaz de saber qué módulos dependen de cuales para pararlos si les faltan dependencias o arrancarlos sólo cuando todas sus dependencias están cargadas.
- Archivo "Manifest.mf": Contiene todos los parámetros de configuración que son necesarios en un módulo que se encuentre en el *framework* OSGi. Cuenta con parámetros como las clases que importa y exporta el *bundle*, su nombre en la plataforma o los módulos que necesita para ser ejecutado.
- Interfaz gráfica: También conocida como GUI, proporciona un entorno visual sencillo que permite la comunicación del usuario con el sistema operativo de una máquina o computador.
- *Middleware*: software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida.

A1.2.- Estructura de la aplicación

A1.2.1.- Ventana principal

Proporciona a la plataforma OSGi una interfaz gráfica inicial desde la que el usuario pueda seleccionar fácilmente aplicaciones del *framework* que se correspondan con el sistema HMI. También se podrán desinstalar e instalar aplicaciones desde esta ventana o desactivar el sintetizador de voz.

En la siguiente imagen se muestra el aspecto de la ventana principal de la aplicación de generación de interfaces, que contiene un el menú principal donde aparecen las aplicaciones integradas en el *framework* OSGi que pretender utilizar el sistema HMI.

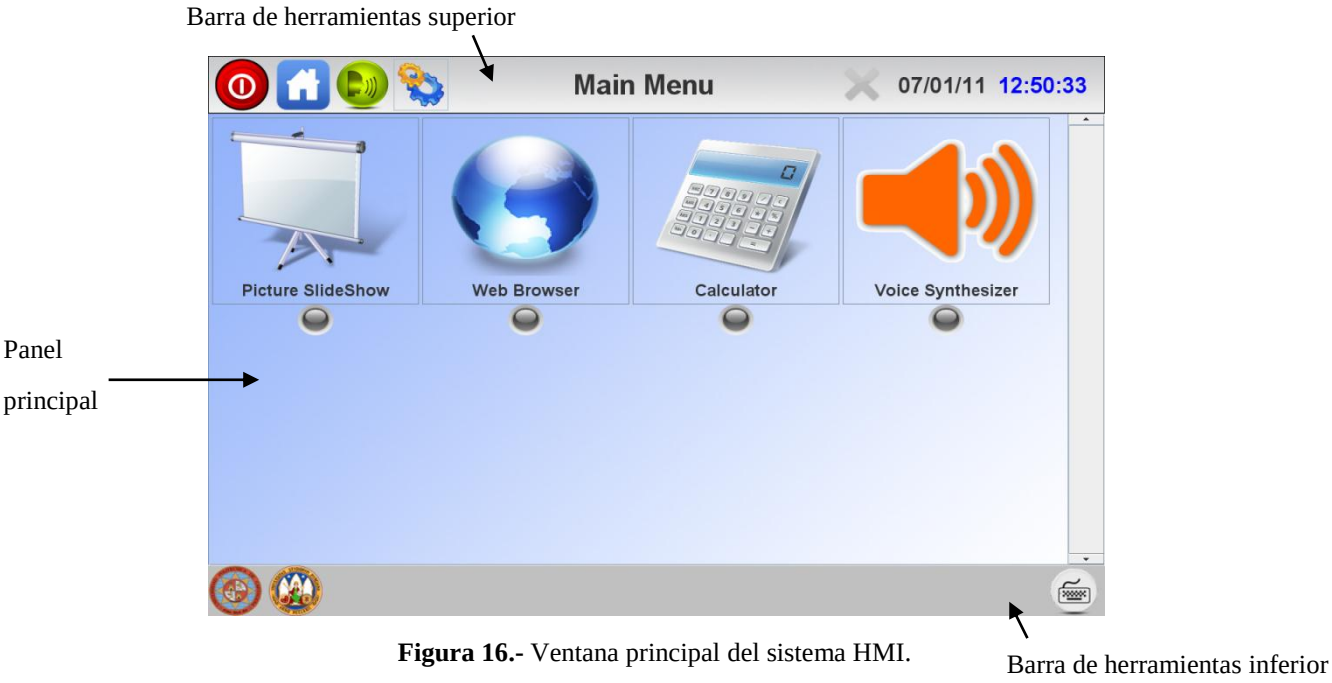

Figura 16.- Ventana principal del sistema HMI.

A1.2.2.- Barras de herramientas

En este apartado se explicarán las funciones de cada una de las barras de herramientas en las que se divide la interfaz de la aplicación principal así como lo que el usuario puede encontrar en cada una de ellas. Su función es integrar las principales opciones de interacción con el usuario del sistema HMI en forma de iconos y etiquetas para facilitar su uso.

Barra de herramientas superior

Esta parte de la ventana principal se compone de los siguientes elementos:

Figura 17.- Barra de herramientas superior.

-  : Permite apagar la aplicación.
-  : Cuando una aplicación es iniciada desde el menú principal, este botón permite volver a la pantalla de inicio, dejando la aplicación iniciada en segundo plano.
-  : Permite activar o desactivar el sistema del sintetizador de voz. Desactivado aparecerá en color gris  .
-  : Muestra el menú que permite instalar o desinstalar aplicaciones del *framework* OSGi. La ventana que aparece tras pulsarlo es la siguiente:

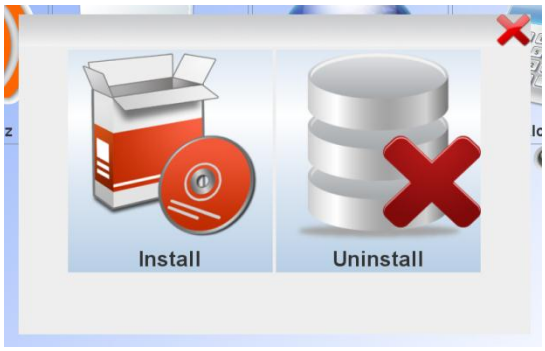

Figura 18.- Ventana instalación/desinstalación de módulos.

  - Install: Permite añadir nuevos módulos al *framework* OSGi. Serán incluidos en el menú principal del *middleware HMI* siempre y cuando cumplan las siguientes características:
    - Tener los atributos Bundle-HMI y Bundle-image en su archivo "manifest.mf": Contienen el nombre y la ruta del icono con que será mostrado en el menú principal del *middleware HMI*.
    - Contener un documento XML válido para la interfaz gráfica inicial.
  - Uninstall: Muestra la siguiente ventana que permite la eliminación de los módulos del *framework* OSGi que se encuentren relacionados con el sistema HMI. Aparecerá una lista en la que podrán ser seleccionados uno por uno los módulos referenciados por su atributo "Bundle-HMI", pulsando en el botón "uninstall" para eliminarlos de la plataforma OSGi.

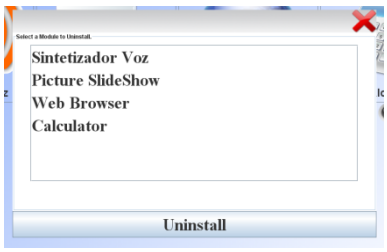

Figura 19.- Ventana desinstalación de módulos.

- 

: Etiqueta que contiene el nombre de la aplicación activa, si el usuario se encuentra en el menú principal, el mensaje que se muestra es el que contiene la imagen, en cambio si se encuentra en una interfaz gráfica de un módulo externo se muestra el nombre especificado en el parámetro Bundle-HMI en el archivo "manifest.mf" de ese módulo.

-  : Botón que permite al usuario cerrar una aplicación activa, por defecto en el menú principal se encuentra desactivado (ninguna aplicación abierta). Su icono en modo activado, se muestra a continuación  .

- 

: Etiquetas que muestran la fecha y hora del sistema.

Barra de herramientas inferior

Contiene únicamente un botón que es el que permite mostrar el teclado en pantalla, tanto en el menú principal como en el resto de aplicaciones.

Figura 20.- Barra de herramientas inferior.

Al presionar este botón la ventana principal es actualizada mostrando el teclado. Destacar que la imagen ha sido tomada desde la interfaz del módulo "Calculator", cuya función es una calculadora básica.

Figura 21.- Teclado en pantalla.

En la anterior imagen pueden observarse dos elementos que no aparecían en la ventana principal de la aplicación HMI, que son:

- Barra de scroll: Debido a que el panel del teclado se solapa en parte con el panel principal de la aplicación, este scroll permite que sean seleccionados todos los elementos contenidos en éste para poder permitir su escritura.
- Panel del teclado: El que se muestra en la Figura anterior es el que aparece por defecto al pulsar sobre el botón de teclado. Contiene botones que permiten lo siguiente:
    - Caps ON : Activa las letras mayúsculas.
    - Caps OFF : Desactiva las letras mayúsculas.
    - .123 : Muestra el panel que contiene los números y caracteres especiales.
    - : Permite borrar caracteres escritos.

Los paneles que mostrarán al pulsar los botones anteriormente citados son:

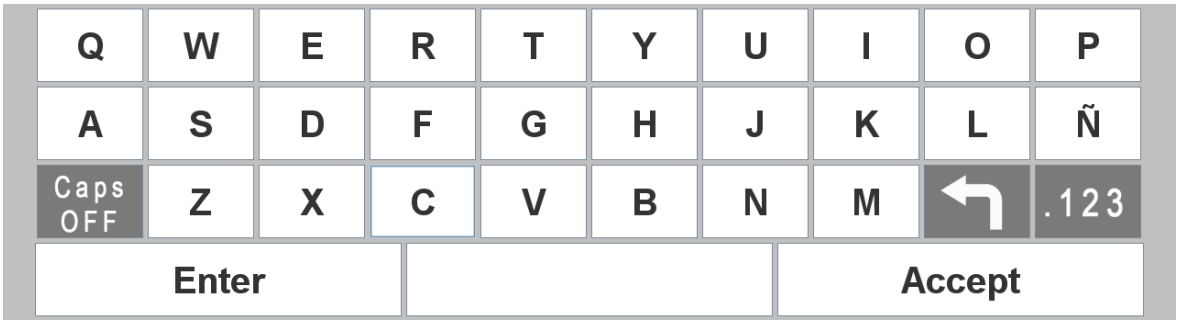

Figura 22.- Teclado de letras mayúsculas.

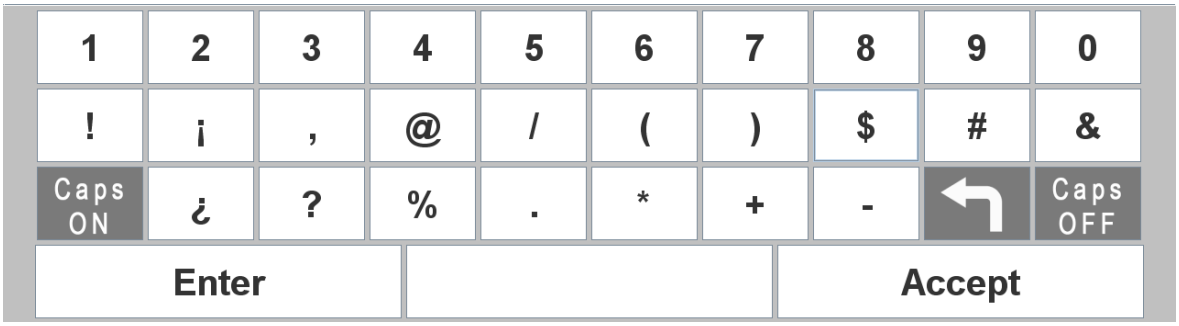

Figura 23.- Teclado de números y caracteres especiales

A1.2.3.- Panel principal.

Es una parte de la ventana principal que contiene todos los elementos gráficos generados por la aplicación HMI, el método para generar cada uno de estos objetos que compondrán este panel principal se explicará en el apéndice 2 de esta memoria. Cada aplicación, al ser iniciada desde el menú principal se comunica con el sistema HMI a través de un archivo XML que es el que contiene todos los parámetros necesarios para construir cualquier elemento gráfico desde cero. Sobre este panel principal aparecerán los posibles mensajes de error que puedan surgir en ejecución como datos no válidos en el caso de la calculadora.

El teclado, que aparecerá al pulsar el botón correspondiente actuará únicamente sobre este panel gráfico, desactivándose el resto de elementos para evitar que pueda volverse al menú principal o cerrar la aplicación activa.

A través de este panel, el sistema HMI se comunica con el usuario de forma gráfica, facilitando el uso del *framework* OSGi. Cada aplicación irá actualizando su propia interfaz a través de las funciones ofrecidas del *middleware HMI*.

Otra de las características que ofrece el menú principal de la aplicación desarrollada es que cuenta con un sistema multi-tarea, que se muestra en funcionamiento en la siguiente Figura mostrando varias aplicaciones iniciadas en segundo plano. Será posible cerrar una aplicación que se encuentre en este estado, accediendo a la aplicación en particular y pulsando el botón “cerrar”, explicado en el apartado A1.2.2. Para dejar una de estas aplicaciones en segundo plano bastará con pulsar el botón “home”, volviendo al menú principal. Para conocer qué módulos se encuentran iniciados en segundo plano, basta con mirar el icono que se encuentra debajo de la imagen de cada módulo, indicando si está naranja que la aplicación está activada.

Figura 24.- Menú principal con módulos activos.

Para mostrar al usuario la diversidad de aplicaciones que pueden presentarse en una plataforma de este estilo, a continuación se muestran las interfaces que son generadas inicialmente por los módulos desarrollados en este proyecto a parte del sistema HMI. Entre estos módulos puede encontrarse una calculadora, un sintetizador de voz, un sistema para cargar y mostrar imágenes e incluso hasta un navegador web.

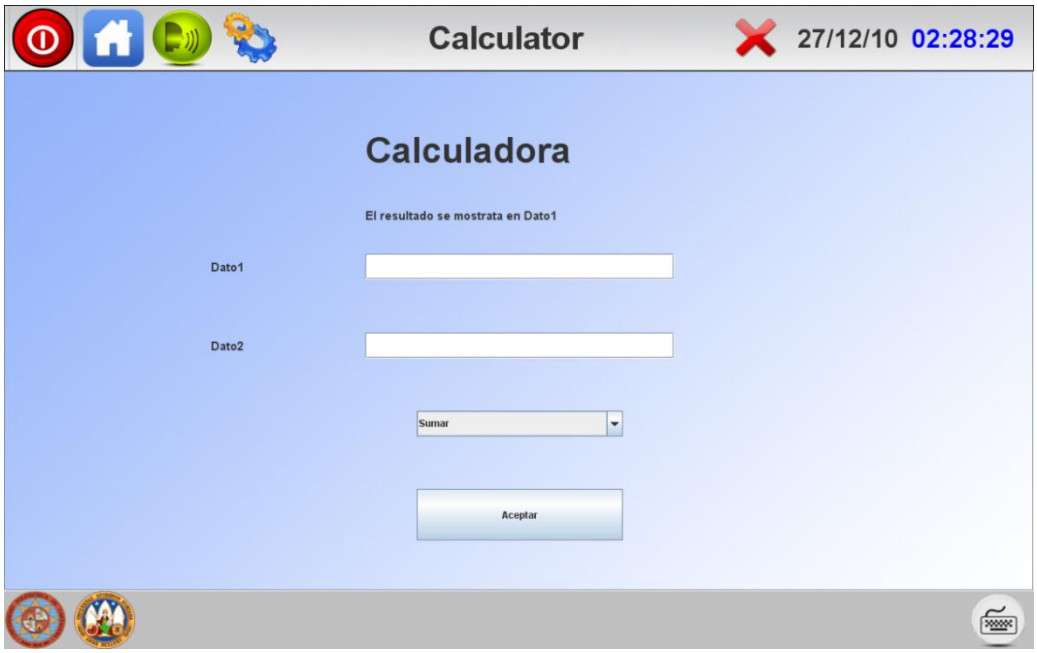

Figura 25.- Panel del módulo "Calculator".

La captura mostrada corresponde al *bundle* que realiza la función de calculadora, entre sus funciones permite sumar, restar, multiplicar o dividir dos números, los contenidos en los campos dato 1 y dato 2, mostrando el resultado en la caja de texto dato1 tras ser calculado.

Figura 26.- Panel del módulo "Voice Synthesizer".

Ofrece al usuario un sistema para testear el correcto funcionamiento del sintetizador de voz, característica que ofrece el sistema HMI.

Los caracteres escritos en el campo de texto serán convertidos a mensajes hablados por el sintetizador del *middleware HMI* tras pulsar el botón "speak".

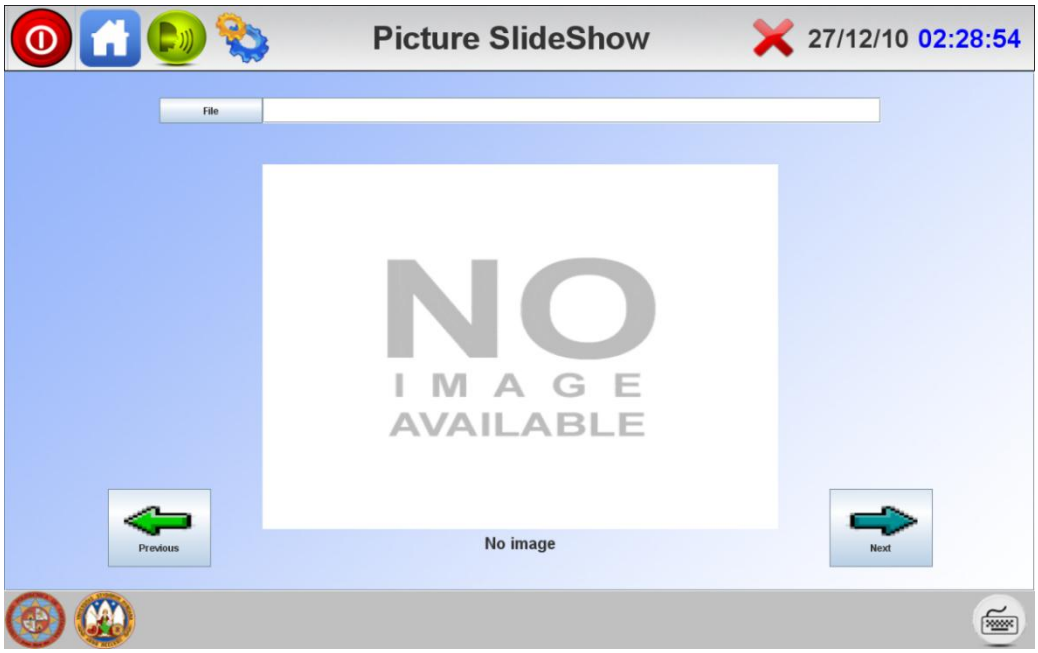

Figura 27.- Panel del módulo "Picture SlideShow".

Ya que el sistema HMI está destinado a un dispositivo de a bordo en el campo de la telemática vehicular, se ha desarrollado un módulo que permita visualizar imágenes contenidas en las carpetas del sistema. Al principio la aplicación se encontrará sin imágenes cargadas, pudiendo ser visualizadas pulsando el botón "file" que abrirá una ventana de selección de carpeta.

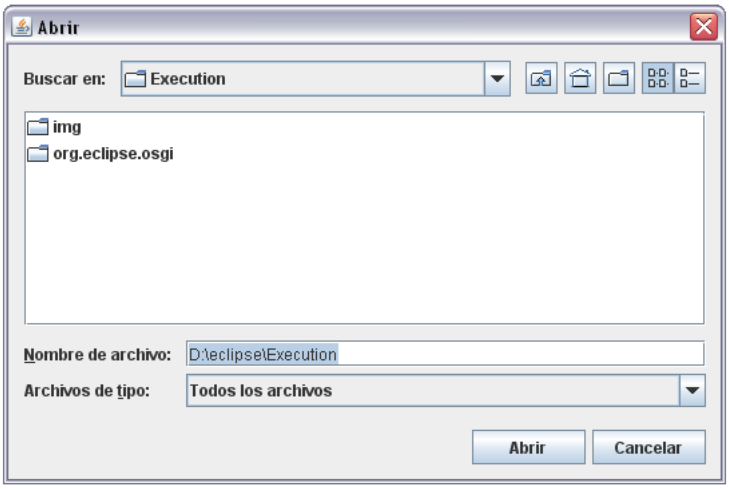

Figura 28.- Ventana selección de fichero.

Tras seleccionar la carpeta, si contiene imágenes con las extensiones de archivo válidas, mostrará un mensaje de advertencia con el número de imágenes encontradas.

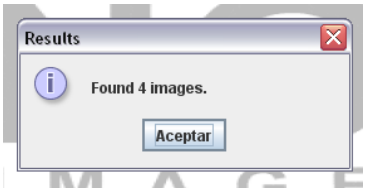

Figura 29.- Mensaje número de imágenes.

Si se ha cargado más de una imagen, los botones de "Previous" y "Next" permitirán visualizar el resto de las imágenes.

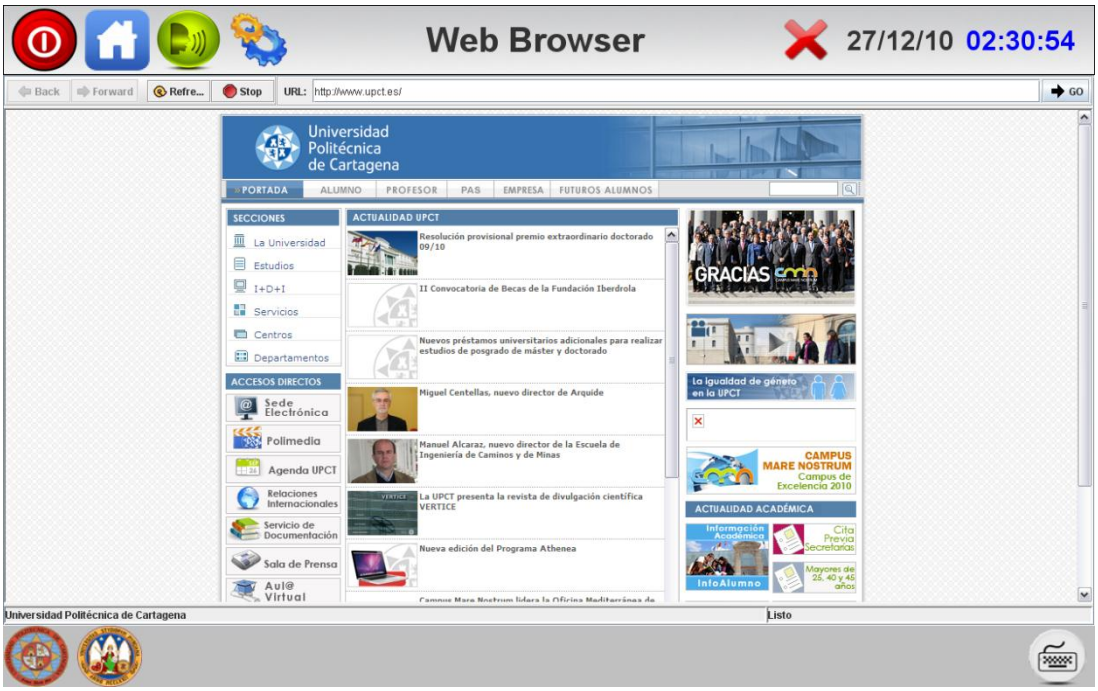

Figura 30.- Panel del módulo "Web Browser".

El navegador se basa en un proyecto de código abierto, el JDIC, que integra la funcionalidad del navegador predeterminado del sistema operativo en una aplicación desarrollado en Java. Sus principales características es que soporta Flash, Java, CSS, permitiendo mostrar todo tipo de páginas web de forma correcta.

En el campo URL se escribirá mediante el teclado virtual proporcionado por la aplicación, si el equipo no dispone de un teclado físico, la dirección web a visitar. Los botones "stop", "refresh", "Back", "Forward" permiten controlar la navegación entre sitios web.

APÉNDICE 2: Implementación de la aplicación

Anteriormente se ha explicado cuáles son las opciones disponibles en el sistema HMI para crear las interfaces gráficas para cada módulo de la plataforma OSGi. Cada ventana personalizada puede ser elaborada a partir de una diversidad de elementos gráficos que serán creados a partir de las funciones incluidas en la aplicación de generación de interfaces gráficas desarrollada.

Para que el lector conozca cuál es la finalidad de cada uno de los métodos incluidos en el *middleware* desarrollado, en este apéndice se detallará la finalidad de cada uno de ellos proporcionando ejemplos de código así como los elementos XML relacionados con cada objeto gráfico.

A2.1.- Métodos paint, repaint y remove

Estas funciones son las que incluyen el código utilizado por el sistema HMI para crear cada uno de los elementos gráficos incluidos en la interfaz gráfica de cada módulo.

Estos objetos gráficos se basan en las librerías gráficas "swing" y "awt", incluidas en el lenguaje de programación Java. Estas librerías contienen elementos básicos para la creación de interfaces gráficas, como ventanas, botones, cajas de texto, métodos de escucha, etiquetas, listas desplegables, etc. La aplicación desarrollada proporciona al *framework* OSGi un sistema de generación de interfaces gráficas de forma dinámica a partir de los objetos contenidos en estas librerías, creando elementos que se adapten a las características que presenta un sistema de a bordo de un automóvil, campo en el que se basa este proyecto.

En la aplicación desarrollada la generación de elementos gráficos ha sido agrupada en tres conjuntos de métodos relacionados con la generación de objetos gráficos Java, que serán incluidos en la interfaz gráfica de cada *bundle*, a continuación se detalla la función de cada uno de ellos:

- Métodos Paint: Crean un objeto gráfico básico inicial a partir de un objeto propio del lenguaje Java, que mantiene la estructura especificada en los esquemas XML.
- Métodos Repaint: El sistema HMI es capaz de generar interfaces gráficas que serán actualizadas en tiempo de ejecución (dinámicas), estos métodos consiguen esta característica, cambiando parámetros como dimensiones, texto incrustado, imágenes, fuente del texto…, de un objeto gráfico creado e incluido en la ventana central de la aplicación HMI,
- Método Remove: Este método permite suprimir un elemento gráfico incluido en la interfaz a partir de una referencia al mismo. Es indispensable a la hora de especificar a la aplicación HMI qué objetos deben no aparecer en la interfaz para mostrar un comportamiento dinámico al usuario.

A2.1.1.- Métodos Paint

La finalidad de los métodos "Paint" es incluir elementos gráficos "nuevos" en la interfaz gráfica del módulo que lo solicite. Estos objetos Java son creados a partir de las librerías gráficas de Java, como se indicó en el apartado A2.1 de esta memoria. Cada elemento gráfico es independiente del resto a partir de un nombre específico que le asigna la aplicación HMI, esto es necesario ya que a la hora de contestar al módulo "cliente" del servicio HMI, se debe especificar exactamente cuál es cada elemento para actuar en consecuencia de la forma más correcta posible.

En realidad, a la hora de utilizar el método "Paint" puede parecer que es sólo una función, se debe a que en la declaración de las funciones "`logicCommandPaint`" se ha incluido lo que en Java se conoce como método "Sobrecargado": Varios métodos comparten el mismo nombre, pero distinta lógica. En ejecución se comprueba cuál es el tipo del parámetro de entrada de la función solicitada, y esto es lo que determina qué método es el utilizado.

Para facilitar la comprensión de la funcionalidad de cada uno de estos métodos, a continuación se muestra la función a explicar, y los objetos Java basados en una estructura creada a partir de un esquema XML, que generarían cada uno de los elementos gráficos:

```
public void logicCommandPaint()
```

Código 8.- Método "`logicCommandPaint()`"

TypeButton

Crea un botón, que se basa en los siguientes parámetros de configuración:

- **Name**: Nombre con el que será referenciado en la aplicación HMI, todos los elementos gráficos generados cuentan con este campo.
- **Coordenadas x, y**: Posición del elemento gráfico a crear en la ventana gráfica.
- **Parámetros w, h**: Determinan las dimensiones del elemento gráfico a crear.
- **Text**: Aquí se especificará el texto que deberá contener el botón.
- **File**: Es posible que el botón deba incluir una imagen, aquí deberá aparecer la ruta del fichero.
- **wImage y hImage**: Determinan las dimensiones de la imagen contenida en el botón.

```
<button name="button1">
      <x>40</x>
      <y>80</y>
      <w>20</w>
      <h>10</h>
      <text>Aceptar</text>
</button>
```

Código 9.- Elemento ejemplo botón.

Figura 31.- Elemento Gráfico Botón.

TypeCheckbox

Crea un "Checkbox", que se basa en los siguientes parámetros de configuración:

- **Name**: Nombre con el que será referenciado en la aplicación HMI, todos los elementos gráficos generados cuentan con este campo.
- **Coordenadas x, y**: Posición del elemento gráfico a crear en la ventana gráfica.
- **Parámetros w, h**: Determinan las dimensiones del elemento gráfico a crear.
- **Text**: Aquí se especificará el texto que deberá contener el elemento gráfico.
- **Selected:** Campo en el que se especifica un valor de tipo booleano ("true" o "false"). Si es el valor "true" el que aparece en este parámetro, la caja de selección aparecerá activada.

```
<checkbox name="checkbox1">
      <x>10</x>
      <y>80</y>
      <w>15</w>
      <h>5</h>
      <selected>true</selected>
      <text>CheckBox</text>
</checkbox>
```

Código 10.- Elemento ejemplo Checkbox.

Figura 32.- Elemento Gráfico Checkbox.

TypeComboBox

Crea un "ComboBox", que se basa en los siguientes parámetros de configuración:

- **Name**: Nombre con el que será referenciado en la aplicación HMI, todos los elementos gráficos generados cuentan con este campo.
- **Coordenadas x, y**: Posición del elemento gráfico a crear en la ventana gráfica.
- **Parámetros w, h**: Determinan las dimensiones del elemento gráfico a crear.
- **Item:** Aquí se incluirá la lista de los elementos que compondrán la lista de selección, teniendo que aparecer como mínimo uno.

```
<comboBox x="40" y="65" h="5" w="20"
name="operacion">
      <item>
            <text>Sumar</text>
      </item>
      <item>
            <text>Restar</text>
      </item>
      <item>
            <text>Multiplicar</text>
      </item>
      <item>
            <text>Dividir</text>
      </item>
</comboBox>
```

Código 11.- Elemento ejemplo ComboBox

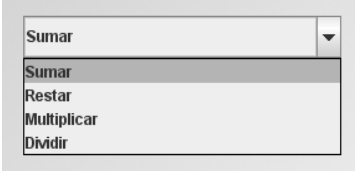

Figura 33.- Elemento Gráfico ComboBox.

TypeLabel

Crea un "Label", que se basa en los siguientes parámetros de configuración:

- **Name**: Nombre con el que será referenciado en la aplicación HMI, todos los elementos gráficos generados cuentan con este campo.
- **Coordenadas x, y**: Posición del elemento gráfico a crear en la ventana gráfica.
- **Parámetros w, h**: Determinan las dimensiones del elemento gráfico a crear.
- **Text**: Aquí se especificará el texto que deberá contener el elemento gráfico.
- **Font:** Tamaño de letra que será utilizado para mostrar el texto.

```
<label name="label1">
      <x>35</x>
      <y>10</y>
      <w>30</w>
      <h>10</h>
      <font>45</font>
      <text>Calculadora</text>
</label>
```

Código 12.- Elemento ejemplo Label.

Figura 34.- Elemento Gráfico Label.

TypeImage

Crea una imagen, que se basa en los siguientes parámetros de configuración:

- **Name**: Nombre con el que será referenciado en la aplicación HMI, todos los elementos gráficos generados cuentan con este campo.
- **Coordenadas x, y**: Posición del elemento gráfico a crear en la ventana gráfica.
- **Parámetros w, h**: Determinan las dimensiones del elemento gráfico a crear.
- **Text**: Aquí se especificará el texto que deberá contener el elemento gráfico.
- **File**: Ruta de la imagen a mostrar.
- **wImage y hImage**: Determinan las dimensiones de la imagen.

```
<image name="image">
      <x>25</x>
      <y>15</y>
      <w>50</w>
      <h>80</h>
      <text>No image</text>
      <file>images/noimg.gif</file>
      <wImage>50</wImage>
      <hImage>70</hImage>
</image>
```

Código 13.- Elemento ejemplo Image.

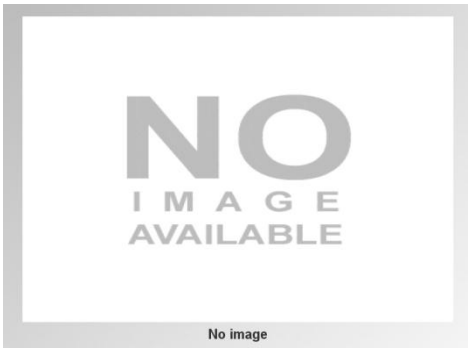

Figura 35.- Elemento Gráfico Image.

TypePassword

Crea un campo de escritura de contraseñas, que se basa en los siguientes parámetros de configuración:

- **Name**: Nombre con el que será referenciado en la aplicación HMI, todos los elementos gráficos generados cuentan con este campo.
- **Coordenadas x, y**: Posición del elemento gráfico a crear en la ventana gráfica.
- **Parámetros w, h**: Determinan las dimensiones del elemento gráfico a crear.
- **Text**: Aquí se especificará el texto que deberá contener el elemento gráfico.

```
<password name="pass1">
      <x>5</x>
      <y>65</y>
      <w>15</w>
      <h>5</h>
      <text></text>
</password>
```

Código 14.- Elemento ejemplo Password.

Figura 36.- Elemento Gráfico Password.

TypeTextField

Crea un campo de escritura, que se basa en los siguientes parámetros de configuración:

- **Name**: Nombre con el que será referenciado en la aplicación HMI, todos los elementos gráficos generados cuentan con este campo.
- **Coordenadas x, y**: Posición del elemento gráfico a crear en la ventana gráfica.
- **Parámetros w, h**: Determinan las dimensiones del elemento gráfico a crear.
- **Text**: Aquí se especificará el texto que deberá contener el elemento gráfico.

```
<textField name="dato1">
      <x>35</x>
      <y>35</y>
      <w>30</w>
      <h>5</h>
      <text></text>
</textField>
```

Código 15.- Elemento ejemplo TextField.

Figura 37.- Elemento Gráfico TextField.

TypeRadioButton

Crea un "RadioButton", que se basa en los siguientes parámetros de configuración:

- **Name**: Nombre con el que será referenciado en la aplicación HMI, todos los elementos gráficos generados cuentan con este campo.
- **Coordenadas x, y**: Posición de los "items" que componen el "RadioButton" a crear en la ventana gráfica.
- **Parámetros w, h**: Determinan las dimensiones "items" que componen el "RadioButton".
- **Item:** Aquí se incluirá la lista de los elementos que compondrán la lista de selección, teniendo que aparecer como mínimo uno.
- **Default:** El elemento que contenga este campo a true aparecerá como seleccionado por defecto.

```
<radioButton name="radioButton1">
      <elementRadio >
            <text>Sumar</text>
            <x>5</x>
            <y>65</y>
            <w>15</w>
            <h>5</h>
      </elementRadio>
      <elementRadio default="true">
            <text>Multiplicar</text>
            <x>5</x>
            <y>75</y>
            <w>15</w>
            <h>5</h>
      </elementRadio>
</radioButton>
```

Código 16- Elemento ejemplo RadioButton.

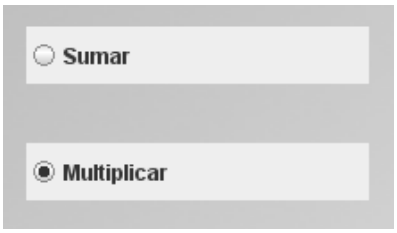

Figura 38.- Elemento Gráfico RadioButton.

A2.1.2.- Métodos Repaint

Como ocurría con los métodos "paint", las funciones "repaint" son métodos sobrecargados. La estructura de los objetos que se pasan como parámetros es la misma que la explicada en el apartado A2.1.1 (Métodos Paint) de esta memoria. Ahora este método comprueba qué tipo de parámetro de entrada recibe, buscando un elemento similar en la interfaz gráfica, actualizándolo con los nuevos parámetros.

Con este método el desarrollador de un módulo que haga uso del sistema HMI, evita tener que suprimir un elemento de la ventana para posteriormente volver a incluirlo con pequeñas variaciones gráficas. También agiliza la forma en la que los elementos gráficos son mostrados en pantalla, ya que es más costoso (computacionalmente hablando) eliminar un elemento e incluir uno nuevo que actualizar uno ya existente.

```
public void logicCommandRepaint()
```

Código 17.- Método "`logicCommandRepaint()`"

A2.1.3.- Método Remove

Esta función (que forma parte del *middleware HMI)*, es la encargada de suprimir elementos gráficos de la ventana gráfica personalizada de cada *bundle* que se encuentre utilizando la aplicación de generación de interfaces gráficas.

El parámetro de entrada que recibe tiene una estructura dedicada sólo a eliminar elementos gráficos, que es la siguiente:

- **Name:** Es el nombre del elemento gráfico a suprimir. Cuando se llamaba a la función "logicCommandPaint" del sistema HMI pasándole como parámetro un elemento XML, se creaba un objeto gráfico que se incluía en la interfaz gráfica. El parámetro "name" del elemento XML es el que se utilizará para referenciar al objeto y será el utilizado en este campo para eliminar completamente este objeto Java de la interfaz.
- **Type:** Dentro del mismo tipo los nombres de los elementos tienen que ser únicos, pero puede haber nombres iguales entre distintos tipos de elementos. Este campo comprueba qué tipo de elemento gráfico quiere ser eliminado, eliminando cualquier tipo de confusión. Entre los tipos, pueden aparecer los siguientes:
  - button: para eliminar botones.
  - label: para eliminar etiquetas.
  - textField: para eliminar campos de escritura de texto.
  - password: para eliminar campos de escritura de contraseña.
  - checkbox: para eliminar cajas de marcado.
  - radioButton: para eliminar listas de selección tipo "radioButton".
  - comboBox: para eliminar listas de selección desplegables.

La función encargada de suprimir elementos gráficos de la interfaz HMI es la siguiente:

```
public void logicCommandRemove()
```

Código 18.- Método "`logicCommandRemove()`"

A2.2.- Mensajes generados por el sistema HMI

La finalidad del *middleware HMI* es generar interfaces gráficas dinámicas a partir de elementos XML que recibe de los módulos interesados en usar su servicio.

Un sistema gráfico es desarrollado para facilitar la interacción del usuario con la plataforma, en este caso el *framework OSGi*, pero no sólo se debe implementar una funcionalidad de creación y representación de elementos gráficos, ya que es el mismo usuario el que mediante los sistemas de entrada como son el teclado o el ratón el que hará funcionar el sistema de acuerdo a sus objetivos.

El sistema HMI generará mensajes de respuesta a los módulos que se encuentren usando su servicio usando una estructura basada en un esquema XML. Dependiendo de la fase en la que se esté ejecutando la aplicación, los mensajes son los siguientes:

1. **Fase inicial:** Tras representar todos los elementos gráficos especificados en el documento XML que es un módulo envía al *middleware HMI*, éste responde con un objeto tipo "Root", que contiene todos el contenido del documento XML parseado en objetos Java.
2. **Mensajes respuesta:** A partir de la ventana gráfica inicial generada, el usuario generará interacciones con la interfaz, produciendo mensajes de respuesta dirigidos al módulo involucrado. Cada mensaje será enviado a través del método "processHMICmdEvent()", que deberá implementar cada módulo. Esta función recibirá como parámetro un elemento con los siguientes campos:
    a. **Name:** Nombre del elemento gráfico implicado.
    b. **Type:** Tipo del objeto recibido.
    c. **Value:** La aplicación HMI, al detectar un evento (p.e. pulsar un botón), comprueba si ha habido algún cambio en los elementos gráficos, si es así crea uno de estos objetos Java y pone el nuevo valor en este campo.

Con la estructura de elementos antes mostrada, los mensajes de respuesta del *middleware HMI* se simplifican, por lo que el módulo que decida utilizar esta aplicación para generar su interfaz gráfica, sólo se deberá encargar de crear un documento XML inicial que contenga los elementos que compondrán la interfaz gráfica de inicio e implementar el método "processHMICmdEvent()" para poder procesar correctamente los mensajes de respuesta generados por la aplicación HMI.

Para comprender mejor su funcionamiento, en el apéndice 3 de esta memoria (Creación de un nuevo módulo), se incluye código de ejemplo de cómo deben ser tratados este tipo de mensajes.

A2.3.- Integración de paneles externos

Un sistema enfocado al terreno de la telemática vehicular puede contar con aplicaciones que desarrollen funciones muy diferentes. Esto a la hora de tenerlo en cuenta se traduce a que deben contemplarse elementos gráficos muy diversos.

Dado que es difícil aunar todos los tipos de objetos gráficos en una sola aplicación, el *middleware HMI* proporciona al desarrollador la posibilidad de integrar un panel externo a la interfaz gráfica, pasando a ser el panel principal del módulo.

Entre las aplicaciones desarrolladas, se encuentra el navegador web, que hace uso de esta característica, integrando un panel externo, que es generado por un módulo que proviene de un proyecto de código abierto (JDIC).

La función encargada de esta tarea es la siguiente:

```
public void logicCommandPanel()
```

Código 19.- Método "`logicCommandPanel()`"

A2.4.- Ventana de selección de fichero

El *middleware HMI* también es capaz de crear una ventana gráfica que permite seleccionar un fichero del sistema de archivos.

Para utilizar esta función, el método implicado es la que aparece a continuación:

```
public void logicCommandFileChooser()
```

Código 20.- Método "`logicCommandFileChooser()`"

El parámetro que recibe este método es un valor de tipo entero, que permite seleccionar cuál va a ser la forma en que se listen los directorios y los archivos:

- `JFileChooser.DIRECTORIES_ONLY:` Muestra sólo directorios.
- `JFileChooser.FILES_AND_DIRECTORIES:` Muestra directorios y archivos.
- `JFileChooser.FILES_ONLY:` Muestra sólo archivos.

La ventana que será generada al llamar a esta función será la siguiente:

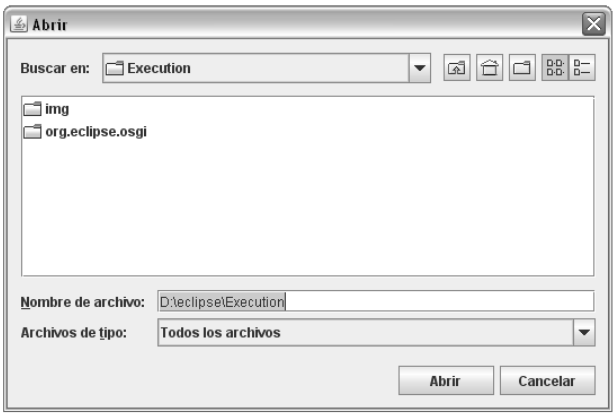

Figura 39.- Ventana de selección de ficheros y directorios.

A2.5.- Mensajes de información y de error

En este apartado se abordará el tratamiento de los mensajes de información y de error que pueden aparecer en el sistema HMI. Se incluirá una imagen de muestra, el código necesario para generar una de estas ventanas y los mensajes que se generan de respuesta en la aplicación de generación de interfaces gráficas.

A2.5.1.- Mensaje de información

El método relacionado con esta función es el siguiente:

```
public void logicCommandMessageDialog(String name,String menssage, String title, int type);
```

Código 21.- Método "`logicCommandMessageDialog()`"

Parámetros de entrada:

- **Name:** Nombre con el que será referenciada la ventana del mensaje.
- **Message:** Texto que aparecerá para informar al usuario.
- **Title:** Título de la ventana a crear.
- **Type:** Tipo de mensaje: 0 –Mensaje de error, 1 –Mensaje de información, 2 –Mensaje de aviso, 3 –Mensaje de pregunta, -1 –Mensaje sin formato.

En función de los parámetros introducidos, las ventanas que pueden ser generadas son las siguientes:

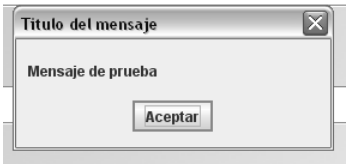

Figura 40.- Mensaje sin formato.

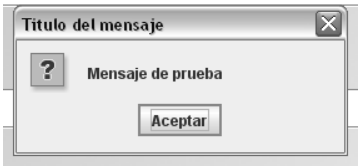

Figura 41.- Mensaje de pregunta.

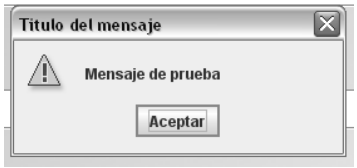

Figura 42.- Mensaje de aviso.

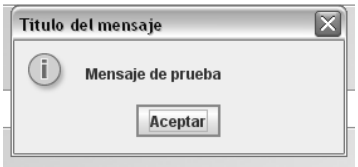

Figura 43.- Mensaje de información.

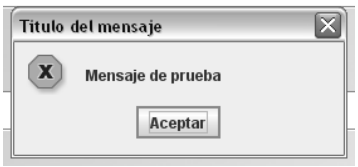

Figura 44.- Mensaje de error.

A2.5.2.- Mensaje de Confirmación

El método relacionado con esta función es el siguiente:

```
public void logicCommandConfirmDialog(String name,String menssage, String
title, int type);
```

Código 22.- Método "`logicCommandConfirmDialog()`"

Parámetros de entrada:

- **Name:** Nombre con el que será referenciada la ventana del mensaje.
- **Message:** Texto que aparecerá para informar al usuario.
- **Title:** Título de la ventana a crear.
- **Type:** Tipo de mensaje: 0 –Opciones "Si" o "No", 1 –Opciones "Si", "No", "Cancelar", 2 –Opciones "Ok", "Cancelar", -1 –Opciones por defecto.

En función de los parámetros introducidos, las ventanas que pueden ser generadas son las siguientes:

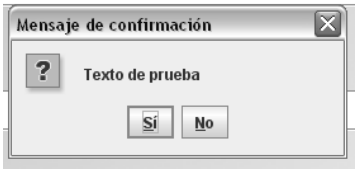

Figura 45.- Opciones "Si" o "No".

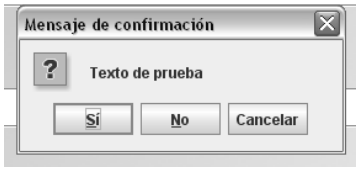

Figura 46.- Opciones "Si", "No", "Cancelar".

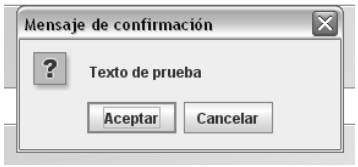

Figura 47.- Opciones "Aceptar", "Cancelar".

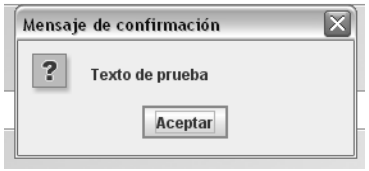

Figura 48.- Opciones por defecto.

A2.5.3.- Mensaje con campo de escritura

El método relacionado con esta función es el siguiente:

```
public void logicCommandInputDialog(String name,String menssage);
```

Código 23.- Método "`logicCommandInputDialog()`"

Parámetros de entrada:

- **Name:** Nombre con el que será referenciada la ventana del mensaje.
- **Message:** Texto que aparecerá para informar al usuario.

En función de los parámetros introducidos, las ventanas que pueden ser generadas son las siguientes:

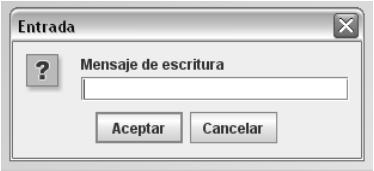

Figura 49.- Ventana campo de escritura.

A2.5.4.- Mensaje con lista de selección

El método relacionado con esta función es el siguiente:

```
public void logicCommandInputDialog(String name,String menssage, String
title,int type,Object[] values,Object selectedValue);
```

Código 24.- Método "`logicCommandConfirmDialog()`"

Parámetros de entrada:

- **Name:** Nombre con el que será referenciada la ventana del mensaje.
- **Message:** Texto que aparecerá para informar al usuario.
- **Title:** Título de la ventana a crear.
- **Type:** Tipo de mensaje: 0 –Mensaje de error, 1 –Mensaje de información, 2 –Mensaje de aviso, 3 –Mensaje de pregunta, -1 –Mensaje sin formato.
- **Object[]:** Contiene la lista de elementos que compondrán la lista desplegable. Para declarar este parámetro sería:

```
Object[] valores={"1","2","3","4"};
```

Código 25.- Declaración objeto Java Object[].

- **Object:** Elemento que aparecerá seleccionado por defecto en la lista desplegable.

En función de los parámetros introducidos, las ventanas que pueden ser generadas son las siguientes:

Figura 50.- Ventana lista desplegable.

A2.5.5.- Mensaje con botones de opciones personalizables

El método relacionado con esta función es el siguiente:

```
public void logicCommandOptionDialog(String name,Object menssage, String
title, int typeOption, int typeMessage,Object[] values, Object
selectedValue);
```

Código 26.- Método "`logicCommandOptionDialog()`"

Parámetros de entrada:

- **Name:** Nombre con el que será referenciada la ventana del mensaje.
- **Message:** Texto que aparecerá para informar al usuario.
- **Title:** Título de la ventana a crear.
- **typeOption:** Tipo de mensaje: 0 –Opciones "Si" o "No", 1 –Opciones "Si", "No", "Cancelar", 2 –Opciones "Ok", "Cancelar", -1 –Opciones por defecto.
- **typeMessage:** Tipo de mensaje: 0 –Mensaje de error, 1 –Mensaje de información, 2 –Mensaje de aviso, 3 –Mensaje de pregunta, -1 –Mensaje sin formato.
- **Object[]:** Contiene la lista de elementos que compondrán la lista desplegable. Para declarar este parámetro sería:

```
Object[] valores={"Si","No","Aceptar","Cancelar"};
```

Código 27.- Declaración objeto Java Object[].

- **Object:** Elemento que aparecerá seleccionado por defecto en la lista desplegable.

En función de los parámetros introducidos, las ventanas que pueden ser generadas son las siguientes:

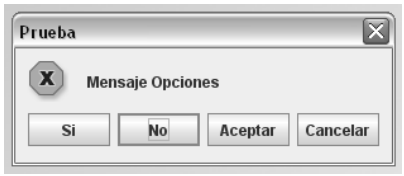

Figura 51.- Ventana botones opciones personalizables.

Apéndice 3: Creación de un nuevo módulo

Una aplicación que requiera el entorno OSGi para su ejecución y que necesite una plataforma de generación de interfaces gráficas como la desarrollada en este proyecto debe ser implementado cumpliendo una serie de parámetros y especificaciones que permitirán que la conexión y la utilización del servicio del *middleware* HMI sea la correcta.

En este apartado se creará un módulo que hará uso del módulo HMI implementado en este proyecto. Particularmente, se ha decidido implementar un reproductor de archivos MP3, que será de gran utilidad en el campo de sistemas de a bordo en vehículos.

Dividiendo la fase de implementación del nuevo módulo en dos grandes bloques, quedaría la dedicada a la conexión con el *middleware HMI* y su integración en el entorno OSGi, y la enfocada al desarrollo del algoritmo principal de la nueva aplicación.

La primera fase del desarrollo de este nuevo módulo englobaría la conexión con el *middleware HMI* y las funciones implicadas.

En una segunda parte se incluiría la descripción del algoritmo utilizado y códigos de ejemplo para comprender el funcionamiento "HMI-módulo", "módulo-HMI".

A3.1.- Suscripción al módulo HMI

Un módulo, para ser integrado en el sistema HMI y poder usar sus funciones para crear elementos gráficos para interactuar con el usuario, deberá adherirse al servicio que ofrece el *middleware* de generación de interfaces gráficas para tal fin.

En un primer paso se deberá crear un proyecto OSGi en el IDE (entorno de desarrollo integrado) escogido. Este proyecto fin de grado ha sido implementado utilizando "eclipse" y será el que será usado durante este apéndice.

Tras este proceso se creará automáticamente un fichero de código llamado "Activator.java" que es el punto de entrada en la ejecución de los *bundles* en OSGi. En el desarrollo del nuevo *bundle* este fichero es el encargado de registrar servicios en el *framework* OSGi y adherirse a servicios que ya estén incluidos en la plataforma de servicios (*Service reference*).

El código para registrarse al servicio que ofrece el *middleware HMI* es el siguiente:

```
/**
 * Variables que contienen las referencias a la plataforma de
 * servicios y al servicio del HMI.
 */
HMIService hmiService;
ServiceReference reference;

/**
 * Se obtiene la referencia del servicio.
 */
reference=context.getServiceReference(HMIService.class.getName());
if(reference == null){
      System.out.println("MP3Player: Reference is null");
}
else{
```

```
		/**
		 * Se obtiene el servicio.
		 */
		hmiService=(HMIService)context.getService(reference);

		/**
		 * Se instancia la clase que contiene en algoritmo que deberá
		 * implementar la clase HMIListener.
		 */
		player=new CargarAudioYReproducir(hmiService);
		if(hmiService != null){

			/**
			 * Se añade la clase como "listener" de eventos de la
			 * aplicación HMI.
			 */
			hmiService.addListener(player);

			/**
			 * El método encargado de pintar la interfaz inicial es
			 * llamado.
			 */
			hmiService.paintHMI("xml/start.xml");
		}
		else{
			System.out.println("MP3Player: HMIService is null");
		}
	}
```

Código 28.- Adhesión a un servicio.

A3.2.- Definición del interfaz del módulo

Cada módulo que se añada al sistema tendrá que contener un documento XML base que será el utilizado por la aplicación HMI para elaborar la interfaz gráfica inicial.

Los elementos incluidos en este documento deberán cumplir una serie de reglas establecidas por un DTD, como contener un nombre único para que puedan ser identificados posteriormente o el tamaño del objeto gráfico a crear.

Teniendo en cuenta las necesidades que presenta un módulo de reproducción de MP3 se ha creado un documento XML que creará la interfaz gráfica que permitirá la interacción con el usuario.

Un ejemplo de un botón que es usado en una aplicación de este tipo sería como sigue:

```
<button name="play">
	<x>13</x>
	<y>70</y>
	<w>12</w>
	<h>25</h>
	<file>images/play.gif</file>
	<wImage>12</wImage>
	<hImage>25</hImage>
</button>
```

Código 29.- Elemento XML Button.

Como puede apreciarse el botón tomará el nombre de “play” en la aplicación HMI y será dibujado con las coordenadas “x” e “y” cuyos valores representan el porcentaje respecto a la ventana. Ocurre lo mismo con los parámetros “w” y “h” que se refieren a la anchura y a la altura del componente gráfico. El resto de elementos son los encargados de incluir una imagen en el botón, con unas dimensiones especificadas.

También se incluye lo que en java se conoce como un “Slider”, que es el encargado de controlar el volumen del reproductor de MP3. Para crear un objeto gráfico de este tipo, se debería incluir un elemento nuevo en el documento XML como el siguiente:

```
//(valor)Esto es un comentario.
<slider name="slide1">
      <x>10</x>
      <y>19</y>
      <w>80</w>
      <h>80</h>
      // Orientación del slider.
      <orientation>vertical</orientation>
      // Valor inicial
      <beginInt>0</beginInt>
      // Valor final
      <endInt>100</endInt>
      //(true)Muestra lo que se conoce como "Ticks" o marcas.
      <paintTicks>true</paintTicks>
      //(true)Muestra los valores si se encuentra.
      <paintLabels>true</paintLabels>
      // Espaciado entre marcas.
      <minTick>1</minTick>
      // Si el valor es 10, se mostrarán los valores
      // 0,10,20,30 en el slider.
      <maxTick>10</maxTick>
      //Valor inicial
      <startValue>100</startValue>
      //Panel en el que se incluye el componente gráfico.
      <panel>panel1</panel>
</slider>
```

Código 30.- Elemento XML Slider.

Un elemento “Slide” tiene parámetros comunes con el resto de elementos que pueden aparecer en el documento XML inicial como el nombre, pero contiene varios parámetros específicos que son necesarios para construir un elemento gráfico de este tipo.

El elemento panel que contiene especifica el JPanel en el que será incluido. Esta característica permite la independencia entre elementos gráficos de distintos paneles.

Para crear un nuevo panel que será incluido en la ventana principal se consigue con el elemento XML siguiente:

```
<frame name="main">
      //Imagen de fondo de la ventana principal.
      <imgBack>images/SkyBackground.jpg</imgBack>
      <panel name="panel1">
            <x>45</x>
            <y>20</y>
            <w>10</w>
            <h>45</h>
            //Color fondo.
            <color>white</color>
            <border>2</border>
            <borderColor>gray</borderColor>
      </panel>
</frame>
```

Código 31.- Elemento XML Panel.

El elemento panel debe incluirse en el elemento padre "frame", que es el que contiene las propiedades de la ventana principal como la imagen de fondo.

La estructura del resto de elementos que pueden incluirse en la ventana gráfica del módulo, puede encontrarse en el Apéndice 2 apartado 1 (métodos paint).

Con los elementos especificados anteriormente y con otros nuevos añadidos la ventana del módulo de reproducción de archivos MP3 podría tener un aspecto parecido al siguiente:

Figura 52.- Ventana principal Reproductor MP3.

A3.3.- Comunicación con el módulo HMI

Cuando se quiere añadir una interfaz gráfica a un módulo del *framework* OSGi se tienen varias alternativas. Al usar el *middleware* desarrollado en este proyecto se evaden varias cuestiones relacionadas con la implementación gráfica ya que es el sistema de representación el que las contempla.

En este apartado se explicarán alguna de las funciones que se utilizan en un módulo básico que use el sistema HMI, aclarando cada uno de los parámetros que intervienen así como los elementos gráficos que se generarían.

A3.3.1.- Conexión inicial

El módulo nuevo a incorporar al *framework OSGI* dispone de varias funciones declaradas en "HMIService" para comunicarse con el *middleware HMI*. Para poder disponer de estos métodos previamente debe haberse adherido el módulo nuevo al servicio en la plataforma de servicios y haber sido añadido como oyente del sistema HMI.

Volviendo al tema de la implementación de un reproductor de MP3 sólo sería necesario utilizar las funciones de "repaint()","fileChooser()" y "messageDialog()" que ofrece el *middleware HMI*, a continuación se explica el resultado producido por cada una de estas funciones:

- logicCommandRepaint(Object x): actualiza un elemento existente en la ventana gráfica con los nuevos valores indicados por "x". En el módulo implementado cambia por ejemplo la imagen del botón "play" a "pause" cuando un archivo es reproducido.
- logicCommandFileChooser(int x): muestra una ventana en la que el usuario podrá elegir un directorio o un fichero. Permite elegir el directorio que contiene los archivos de audio para ser reproducidos más tarde.
- logicCommandMessageDialog(): informa al usuario de un dato no encontrado o de algún evento producido. Informa al usuario, por ejemplo, si no se han encontrado archivos con extensión ".mp3" en el directorio especificado.
- logicCommandRemove(TypeRemove remove): Aunque este método no es necesario para la ejecución del reproductor, podría utilizarse para eliminar el un botón de configuración inicial, que sólo debería aparecer al principio del programa.
- logicCommandRemoveAll(): elimina todos los elementos gráficos de la ventana gráfica principal. Permitiría actualizar el diseño del reproductor desde cero, añadiendo nuevos componentes gráficos eliminando todos los elementos actuales de la ventana sólo con un comando.

A3.3.2.- Respuesta del middleware

En un sistema de generación de interfaces gráficas se necesita una comunicación bidireccional entre ambas partes para gestionar los eventos que pueden aparecer producidos por la interacción del usuario con el *middleware*.

El sistema HMI se comunica con un módulo del *framework* OSGi mediante los métodos publicados en la interfaz "HMIListener", que contiene dos métodos:

- ProcessAppBackground(Boolean active): a través de este método el sistema HMI informa al módulo si se encuentra en primer o segundo plano. Suele ser útil a la hora de actualizar la ventana gráfica o no, dependiendo de en qué estado se encuentre el módulo. Permite que el módulo del reproductor no actualice la ventana si se encuentra en "segundo plano", ya que la aplicación activa en el sistema HMI no sería el reproductor de mp3.
- ProcessHMICmdEvent(TypeRoot root): es el que envía al módulo la información actualizada de los elementos gráficos de la ventana principal cada vez que se produce un evento (un botón es pulsado o un elemento es seleccionado en una lista). El elemento root contiene un objeto de tipo TypeEvent que es el que contiene los eventos generados por el sistema HMI. Cuando se recibe una respuesta del *middleware HMI* por este método deberá comprobarse qué ha producido el evento. Para ello se muestra a continuación el código que tendría que incluir el módulo para comprobar si el evento ha sido generado por un elemento gráfico de tipo botón:

```
//Se lee elemento de tipo "TypeEvent" del elemento root que es el
//raíz.
Iterator<TypeEvent> eventIterator=root.getEvent().iterator();
TypeEvent eventObject = eventIterator.next();

//se obtienen los eventos de tipo "TypeOnPressed" que contienen los
//eventos generados por los botones de la aplicación gráfica.
Iterator<TypeOnPressed>
onPressedIterator=eventObject.getOnPressed().iterator();
while(onPressedIterator.hasNext()){
      TypeOnPressed onPressedObject = onPressedIterator.next();

      //Se comprueba si el evento ha sido generado por el botón
      //"play".
      if(onPressedObject.getElement().equals("play")){
            if(todo_ok){
            if(!run){
                  PLAY();
            }
            else{
                  PAUSE();
                  pausado=true;
            }
            statePButton();
            }
      }
}
```

Código 32.- Gestión Evento Sistema HMI.

A3.4- Implementación del nuevo módulo

El desarrollo de una aplicación tendrá éxito si el algoritmo ha sido implementado correctamente evitando fallos o una ejecución del módulo no deseada.

Tras completar la conexión del sistema HMI de generación de interfaces gráficas con el módulo a desarrollar y teniendo en cuenta cuál es la forma de utilizar las funciones que ofrece el *middleware* y sabiendo gestionar los mensajes de respuesta que se pueden generar, falta completar el módulo con el algoritmo que será el que determine cuál es la función que se desea que ofrezca esta aplicación a la plataforma OSGi.

Como en este apéndice la función escogida que desempeñe el módulo es un reproductor de MP3, se deberá incluir una nueva clase que será la que incluya el algoritmo planteado.

Los métodos más importantes que deberá contener serán los encargados de la reproducción de un archivo MP3:

- PLAY(): reproduce el archivo.
- PAUSE(): pausa la reproducción, permitiendo reanudarla más tarde.
- STOP(): para la reproducción.
- OPEN(): carga un archivo nuevo al sistema de reproducción.
- NEXT(): carga el siguiente archivo.
- PREV(): carga el archivo anterior.

La implementación de cada uno de los métodos puede encontrarse en el archivo "CargarAudioYReproducir.java" del módulo org.mp3Player.

Apéndice 4: Distribución *STANDALONE* de la plataforma

Aunque el sistema HMI es totalmente funcional depende del entorno de desarrollo eclipse para su ejecución y por tanto no podría ser utilizado como una aplicación más instalada en el sistema operativo. El objetivo de este capítulo es coger una aplicación OSGi y convertirla en un sistema preparado para ser instalado basado en OSGi.

A4.1- Definiendo un producto HMI

Hasta ahora el componente HMI ha sido ejecutado directamente usando configuraciones de ejecución de eclipse, en inglés *Launch configurations*. Una configuración de producto (*product configuration*) incluye toda la información necesaria para crear un completo sistema aislado de eclipse. Incluye la lista de los módulos o *bundles* y características así como la información sobre pantallas gráficas, iconos del ejecutable además de otras opciones.

*Product configurations (*configuraciones de producto*)* no están definidas en OSGi y por tanto no forman parte de la aplicación desarrollada. Son añadidos que permiten describir el sistema que va a crearse como aplicación nativa del sistema operativo. A continuación, en los siguientes apartados se explicará las dos configuraciones de ejecución para arrancar el cliente (HMI) y el *back end* (*product configuration*) para convertirlos en *product definitions*.

A4.2- Creando el *Product Configuration*

Primero se va a crear el *product configuration*. En las siguientes secciones se explicará guiadamente los pasos para configurar todo correctamente.

- Ir a File > New > Project > General > Project para crear un nuevo proyecto llamado HMIBackEnd.
- Seleccionar el nuevo proyecto org.equinoxosgi.hmi.product.backend e ir a File > New > Product Configuration para arrancar el asistente como se muestra en la Figura 34.

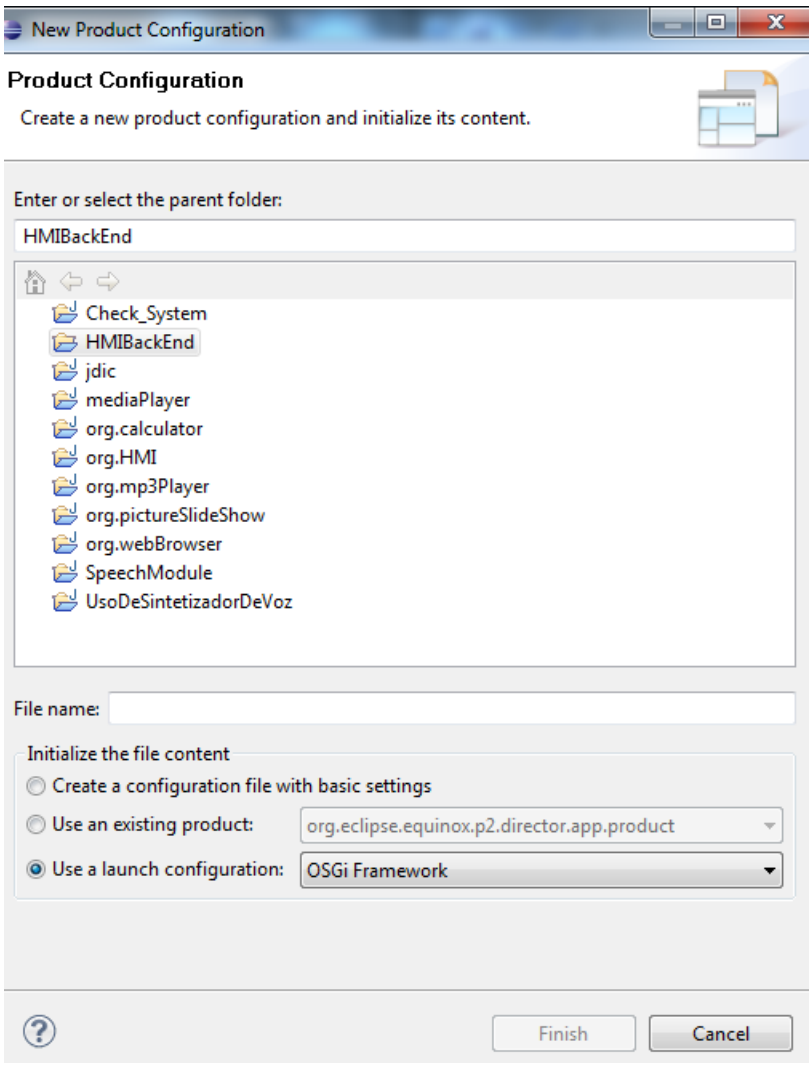

Figura 53- Ventana configuración de producto.

- Completa los campos como aparece en la imagen configurando el nombre del *product definition* como backend.product.
- En configuración de ejecución (*launch configuration*) elige la configuración del entorno OSGi en el que se encuentran los módulos.

El asistente lee la configuración de arranque y la usa para construir la definición del producto. En particular, coge la lista de los módulos y cualquier configuración además de los argumentos de la línea de comandos. La nueva configuración de producto abre un editor, similar al que se muestra en la Figura 35.

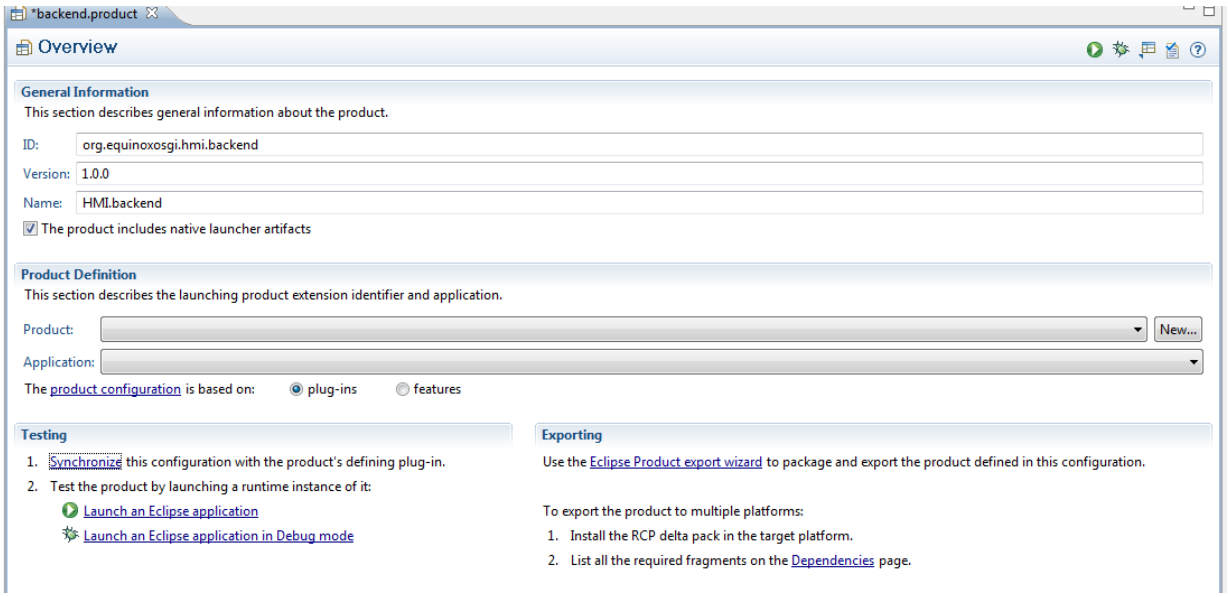

Figura 54- Editor de producto.

A4.3- Configuración del producto

Como con el editor de módulos, el editor del *product configuration* aporta información de todos ficheros relacionados en un mismo lugar. La información de configuración está agrupada en varias pestañas integradas en el mismo editor. La pestaña *overview* mostrada en la Figura 35 muestra la información general del producto.

Esta pestaña da acceso al ID, versión y nombre del producto. Ninguno de estos campos son obligatorios para ejecutar el producto pero cuando se compila o se exporta, el ID y la versión son muy útiles.

- Configura el ID como org.equinoxosgi.hmi.backend.

Tener en cuenta los dos botones de radio del final de esta sección. Permiten seleccionar como se van a listar los contenidos del producto, como bundles (plug-ins) o una lista de características. En este proyecto se han desarrollado *bundles* por lo que se seleccionará la primera opción.

A4.3.1- La ventana de dependencias

Aquí se incluyen la lista de los *bundles* o módulos que serán incluidos en la aplicación *standalone*.

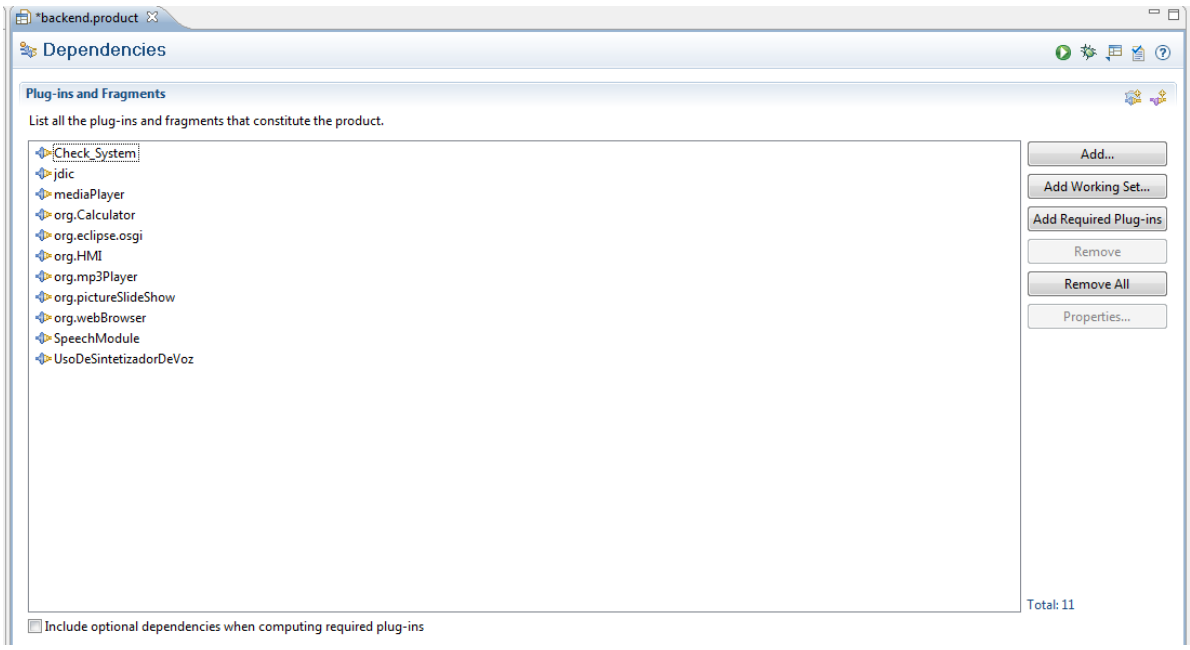

Figura 55- Ventana dependencias producto.

A4.3.2- La ventana de configuración

Aquí se encuentan la lista de los bundles que van a ser incluidos o que se encuentran implicados por el contenido de la página de dependencias. Por cada módulo listado, se puede confgurar si va a ser arrancado automáticamente y en qué orden. Auto-arranque signifia que un módulo asociado es instalado e iniciado a la misma vez cuando el sistema es ejecutado. El nivel de arranque ayuda a definir el orden de inicio.

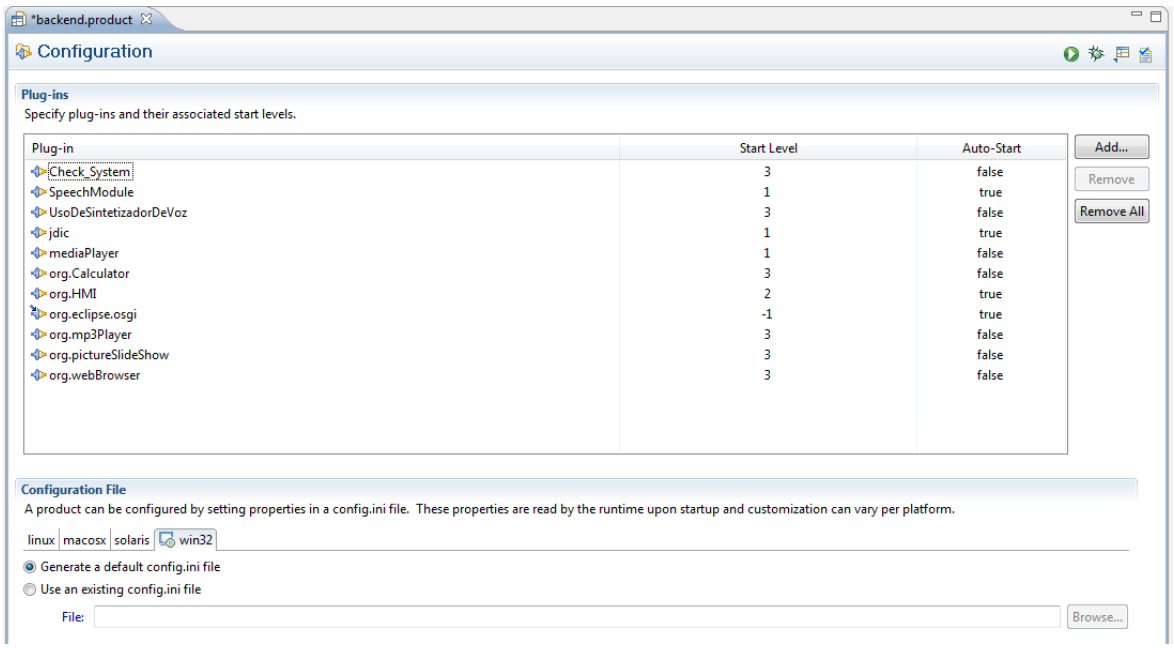

Figura 56- Ventana Editor / configuración.

A4.3.3- La ventana de lanzamiento

El ejecutable es el programa que los usuarios finales inician cuando quieren iniciar el sistema HMI. Un ejemplo sería HMI.exe en Windows. Esto nos permite tener un ejecutable más ligero que contiene todos los módulos y componentes necesarios para ejecutar nuestro sistema.

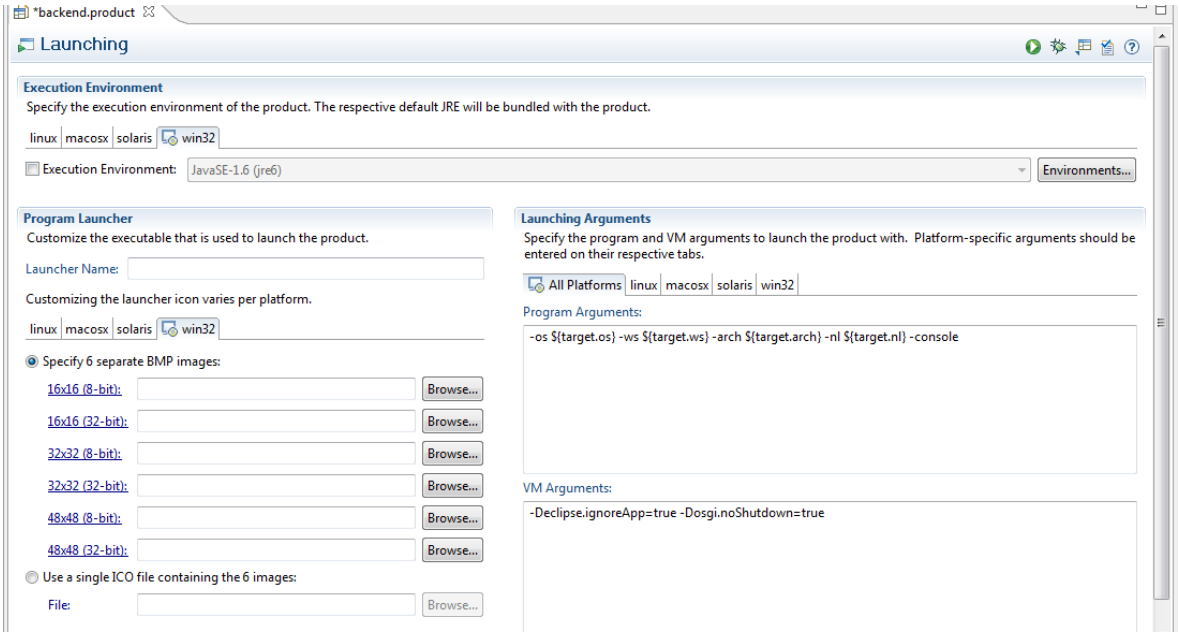

Figura 57- Ventana Lanzamiento.

La caja de texto *Launcher Name* permite especificar el nombre del ejecutable a crear. No se debe incluir la extensión .exe ya que esa información es dependiente de la plataforma.

Debajo del *Launcher Name* hay una serie de campos para incluir los iconos al ejecutable. Como se soportan varias plataformas, cada una cuenta con diferentes formatos y tamaños de imagen. Se deberán incluir las imágenes del sistema operativo para el cual se va a portar la aplicación. Durante la exportación, el *plug-in development environment (PDE)* crea un ejecutable que se comporta exactamente como un ejecutable estándar de Eclipse pero es renombrado y diseñado con los iconos que se especifican.

A4.4- Exportando HMI

Para conseguir el sistema HMI fuera del workspace (entorno de desarrollo), hay que exportarlo. Para conseguirlo, se deben identificar qué partes de los proyectos van a ser agrupadas en los correspondientes *binary bundles*.

Para exportar el sistema a un ejecutable, vamos a file > export… y en el asistente elegimos plug-in development > Eclipse product y pulsamos en siguiente mostrándose una ventana similar a la incluida en la Figura 39.

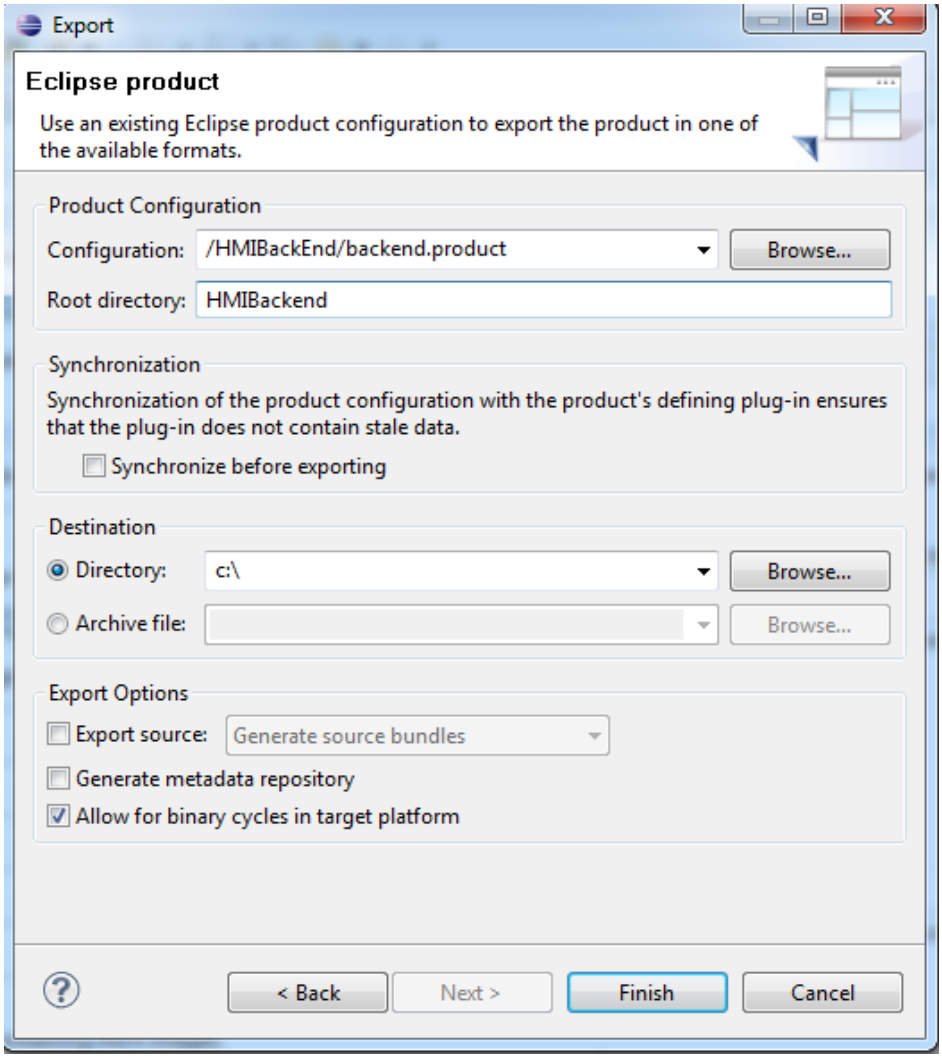

Figura 58- Asistente exportación de producto.

- Primero debemos asegurarnos de que *backend.product* está seleccionado.
- Completamos el *Root directory*, que es el directorio sobre el que se va a incluir los ficheros resultantes de la exportación.

Después de que pulse sobre finish, PDE empezará a exportar. El proceso se ejecutará en *background* por lo que se puede continuar usando Eclipse mientras que la exportación concluye. Primero, compila el código del *workspace* acorde a la configuración descrita anteriormente. El asistente de exportación recopila el código compilado y los elementos requeridos de la plataforma objetivo y los exporta a la ubicación especificada.

Cuando el proceso termina, c:\HMIBackend contiene una plataforma OSGi con todos los módulos que se inicia fuera del workspace. Ahora se puede ejecutar el .exe que se encuentra en dicha carpeta y comprobar cómo se ejecuta el **servidor stand-alone basado en Equinox.**

En la siguiente Figura se muestra qué ficheros se generan con este proceso en la carpeta HMIBackend:

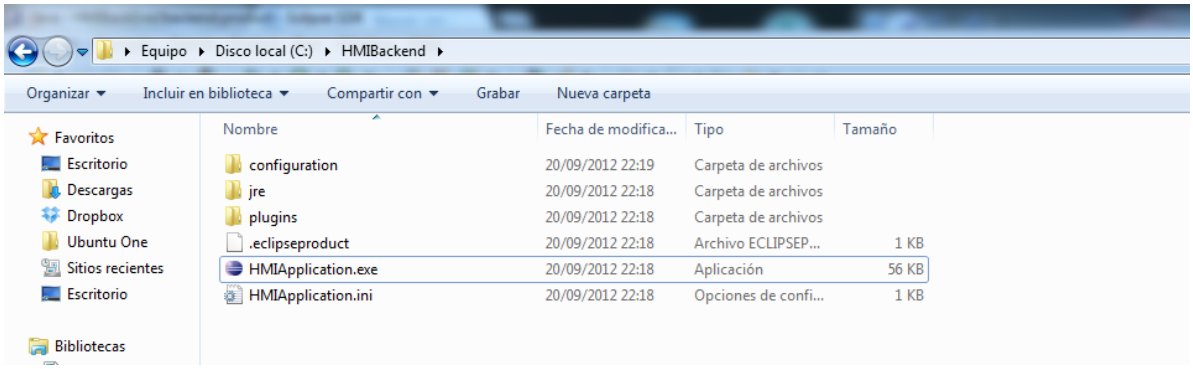

Figura 59- Ficheros carpeta compilación.

Bibliografía

Las referencias sobre las cuales ha sido desarrollado este proyecto han sido las siguientes:

[1] OSGi Alliance. http://www.osgi.org.

[2] Bartlett, Neil. OSGi In Practice, DRAFT ed. January 2009.

[3] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, Extensible Markup Language (XML) W3C Recommendation, Third Edition, February 2004.

[4] Sekhar Vajjhala, Joe Fialli, The Java™ Architecture for XML Binding (JAXB) 2.0 Final Release, April 19 2006.

[5] Y. Li, F.-Y. Wang, and Z. Li. OSGi-based service gateway architecture for intelligent automobiles. In IEEE Intelligent Vehicles Symposium, pages 861–865, Las Vegas, USA, June 2005.

[6] Eckel, Bruce, Thinking in Java, 4rd Edition, 2006.

[7] OSGi and Equinox Creating Higly Modular Java Systems.